



USER MANUAL

# BCM1250/BCM1125/BCM1125H

## User Manual for the BCM1250, BCM1125, and BCM1125H

## REVISION HISTORY

<i>Revision</i>	<i>Date</i>	<i>Change Description</i>
1250-UM100-R	06/25/01	Initial release.
1250-UM101-R	01/02/02	<b>New Sections:</b>  Section : "Interrupts" on page 47  Section : "ZBbus Cycle Count and Compare" on page 58.  Section : "Reduced Cache Size" on page 94.  Section : "Cache Configuration Register" on page 99.  Section : "DDR FCRAMs" on page 123.  Section : "HyperTransport Read Restrictions" on page 200.  Section : "HyperTransport Bounce Space" on page 213.  Section : "HyperTransport Differences from Revision 1.03 Specification" on page 224.  Section : "HyperTransport Differences from Revision 0.17 Specification" on page 222.  Section : "HyperTransport Target Done Counter" on page 236.  Section : "TCP Checksum Checking" on page 282.  Section : "Flow Control In Encoded Packet FIFO Modes" on page 293.  Section : "Restrictions When Resetting the Interface" on page 300.  Section : "Burst Mode" on page 370.  Section : "Early Chip Select" on page 372.  Section : "Transport Protocol Reset" on page 405.  Section : "Extended Protocol" on page 406.  Section : "BSRMODE - Holding Boundary Scan Active" on page 434.  <b>New Tables:</b>  Table 15, Table 48, Table 126, Table 127, Table 138, Table 139, Table 157, Table 203, Table 212 and Table 214.

---

<b>Revision</b>	<b>Date</b>	<b>Change Description</b>
1250_1125-UM100-R	10/21/02	<p>This list summarizes the major changes between 1250-UM101 and 1250_1125-UM100. The 1250_1125-UM100CB version of this manual has change bars indicating all changes between the older and newer versions.</p> <ul style="list-style-type: none"> <li>• Section 1: Updates to describe BCM1125/H</li> <li>• Section 2: Updates to describe BCM1125/H.</li> <li>• Section 3: Additional Clarifications and BCM1125/H descriptions <ul style="list-style-type: none"> <li>• New section: Error Conditions</li> </ul> </li> <li>• Section 4: Additional Clarifications and BCM1125/H descriptions <ul style="list-style-type: none"> <li>• Expanded section: Bus Watcher</li> <li>• New Section: Magic Decoder Ring for Using The Trace Buffer</li> </ul> </li> <li>• Section 5: Updates for BCM1125/H <ul style="list-style-type: none"> <li>• New register: Level 2 Cache Settings Register</li> </ul> </li> <li>• Section 6: Additional Clarifications and BCM1125/H descriptions <ul style="list-style-type: none"> <li>• New Section: Memory Access Sequencing</li> <li>• New Section: Example CHannel and Chip Select Configurations</li> <li>• Updated guidelines: Timing Parameter Guidelines</li> </ul> </li> <li>• Section 7: Additional Clarifications and BCM1125/H descriptions <ul style="list-style-type: none"> <li>• New Section: Unaligned Buffer Descriptor Format for Ethernet DMA</li> <li>• New Section: CRC and Checksum Generators</li> </ul> </li> <li>• Section 8: Additional Clarifications and BCM1125/H descriptions</li> <li>• Section 9: Additional Clarifications and BCM1125/H descriptions <ul style="list-style-type: none"> <li>• New Section: Prependded Header Frame Format</li> <li>• New Functionality: Destination Address Filtering</li> <li>• New Functionality: Receive DMA Channel Selection</li> <li>• New Functionality: Flow Control</li> </ul> </li> <li>• Section 16: Cross reference links added from register to defining table</li> <li>• Index: Expanded</li> </ul>

Broadcom Corporation

P.O. Box 57013

16215 Alton Parkway

Irvine, CA 92619-7013

© 2002 by Broadcom Corporation

All rights reserved

Printed in the U.S.A.

Broadcom® and the pulse logo® are trademarks of Broadcom Corporation and/or its subsidiaries in the United States and certain other countries. All other trademarks are the property of their respective owners.

---

## TABLE OF CONTENTS

<b>Section 1: Introduction</b> .....	1
<b>The SiByte Broadband Processor Family</b> .....	1
<b>The BCM1250</b> .....	2
<b>The BCM1125 and BCM1125H</b> .....	3
<b>Audience</b> .....	3
<b>Other Documentation</b> .....	4
<b>Terminology</b> .....	5
<b>Section 2: Signal Overview</b> .....	7
<b>BCM1250 Signal Groups</b> .....	7
<b>BCM1125/H Signal Groups</b> .....	8
<b>Section 3: System Overview</b> .....	9
<b>Introduction</b> .....	9
<b>Internal Registers</b> .....	11
<b>Coherence</b> .....	12
<b>Ordering Rules and Device Drivers</b> .....	14
<b>CPU Speculative Execution</b> .....	16
<b>Error Conditions</b> .....	17
Cache Error Exceptions .....	17
Bus Error Exceptions .....	18
<b>CPU to CPU Communication (BCM1250 Only)</b> .....	19
<b>External Interrupts</b> .....	19
<b>Overview of the ZBbus Protocol</b> .....	20
Arbitration.....	21
Address Phase.....	22
Response Phase.....	23
Data Phase .....	24
<b>Reset</b> .....	26
<b>Clocks</b> .....	31
<b>Memory Map</b> .....	34
<b>Section 4: System Control and Debug Unit</b> .....	41
<b>Introduction</b> .....	41

---

<b>System Control</b> .....	41
<b>Mailbox Registers</b> .....	46
<b>Interrupts</b> .....	47
HyperTransport Interrupts .....	48
The Full Interrupt Mapper .....	50
<b>Timers</b> .....	57
Watchdog Timers .....	57
General Timers .....	58
Timer Special Cases .....	58
ZBbus Cycle Count and Compare .....	58
Timer Registers .....	59
<b>System Performance Counters</b> .....	61
<b>Bus Watcher</b> .....	64
<b>Address Trapping</b> .....	67
<b>Trace Unit</b> .....	70
Trigger Events .....	70
Trigger Sequences .....	73
Using the Trace Buffer .....	76
Reading the Trace Buffer .....	79
Magic Decoder Ring For Using The Trace Buffer .....	81
Connections to the Trace Logic .....	83
Trace Example 1: All CPU0 Activity .....	84
Trace Example 2: Network Packet Headers .....	85
<b>Section 5: L2 Cache</b> .....	89
<b>Introduction</b> .....	89
<b>Normal Operation</b> .....	89
<b>Using the L2 Cache as Memory</b> .....	91
Standard RAM .....	92
Memory Locked in the L2 Cache .....	92
Comments on Using the L2 as Memory .....	93
Reduced Cache Size .....	94
<b>Cache Management Access</b> .....	94
Standard Management Mode Accesses (both ECC_diag address bits zero) .....	97
ECC Diagnostic Management Accesses (ECC_diag bits nonzero) .....	98



---

Cache Configuration Register .....	99
Example Startup Code to clear the L2 Cache.....	99
<b>Registers</b> .....	100
<b>Section 6: DRAM</b> .....	103
<b>Introduction</b> .....	103
A Comment on the term Bank.....	103
<b>Memory Controller Architecture</b> .....	104
Memory Access Sequencing .....	107
<b>Clock Ratios and Clocking Scheme</b> .....	107
<b>Memory Configurations</b> .....	109
Mapping .....	109
Channel Select.....	109
Chip Select.....	110
Example Channel and Chip Select Configurations .....	112
Row, Column and Bank Configuration .....	117
Choosing Interleave Parameters .....	120
Page Policy .....	122
Supported DRAMs and DIMMs.....	123
DDR SDRAMs.....	123
DDR FCRAMs .....	123
DIMMs .....	124
Larger Memory Systems .....	124
<b>ECC</b> .....	125
<b>SDRAM Timing</b> .....	125
<b>SDRAM Refresh</b> .....	126
<b>SDRAM Initialization and Commands</b> .....	126
<b>I/O Control</b> .....	128
<b>Timing Parameter Guidelines</b> .....	131
<b>Performance Monitoring Features</b> .....	134
<b>ZBbus Monitoring</b> .....	134
<b>Configuration Registers</b> .....	135
<b>Section 7: DMA</b> .....	147
<b>DMA Controllers</b> .....	147
<b>Data Buffers and Descriptors</b> .....	147

---

Unaligned Buffer Descriptor format for Ethernet DMA .....	154
<b>DMA Coherence and Cache Options .....</b>	<b>155</b>
<b>DMA Configurations .....</b>	<b>156</b>
<b>Ethernet and Serial DMA Engines .....</b>	<b>157</b>
Descriptor Count Watermarks .....	157
Completion Interrupts .....	158
Explicit Descriptor Interrupts.....	158
ASIC Mode Transfers .....	159
Option and Flag Bits for Ethernet MACs .....	171
Control and Flag Bits for Synchronous Serial Interface.....	174
<b>Data Mover .....</b>	<b>176</b>
Data Mover Operation .....	176
CRC and Checksum Generators.....	178
Checksum Generation.....	178
CRC Generation.....	179
Computation Sizes and Bandwidth .....	180
Examples.....	181
Data Mover Control Registers .....	184
Data Mover Descriptors.....	187
<b>Section 8: PCI Bus and HyperTransport Fabric .....</b>	<b>190</b>
<b>Introduction.....</b>	<b>190</b>
<b>PCI and HyperTransport Address Range.....</b>	<b>192</b>
Memory Mapped Devices.....	194
HyperTransport Expansion Space.....	194
Configuration Space.....	194
PCI I/O Space.....	195
The SouthBridge, VGA and Subtractive Decode.....	195
HyperTransport End Of Interrupt (EOI) Signaling Space .....	198
Legacy Interrupt Acknowledge (IACK) Space .....	199
PCI Full Access Space .....	199
Special HyperTransport Space.....	199
HyperTransport Read Restrictions .....	200
<b>Endian Policies .....</b>	<b>201</b>
Little Endian System: No Swaps .....	201





Big Endian System: Match Byte Lanes .....	202
Big Endian System: Match Bit Lanes .....	203
Viewing Endian Policy as an Optimization .....	204
<b>Accessing the SiByte from PCI Devices .....</b>	<b>205</b>
<b>Accessing the SiByte from HyperTransport Devices .....</b>	<b>210</b>
Force Isochronous Mode Address Range .....	212
Accessing the SiByte from a SiByte on a Double Hosted Chain.....	212
HyperTransport Bounce Space.....	213
<b>Performance of the PCI and HyperTransport Interfaces .....</b>	<b>213</b>
Accesses from the SiByte to the PCI or HyperTransport.....	214
Accesses from the HyperTransport to the SiByte .....	216
Accesses from the PCI to the SiByte .....	217
PCI Adaptive Retry .....	217
<b>Peer-to-Peer Accesses .....</b>	<b>219</b>
PCI Bus To HyperTransport Fabric.....	219
HyperTransport Fabric to PCI Bus.....	221
<b>PCI Arbiter .....</b>	<b>222</b>
<b>PCI Interrupts .....</b>	<b>222</b>
<b>HyperTransport Differences from Revision 0.17 Specification .....</b>	<b>222</b>
HyperTransport Differences from Revision 1.03 Specification.....	224
Ordering Rules.....	231
<b>Using the PCI in Device Mode.....</b>	<b>232</b>
<b>Configuration of PCI and HyperTransport .....</b>	<b>234</b>
HyperTransport Target Done Counter .....	236
Systems That Do Not Use HyperTransport.....	236
Configuration Header Descriptions .....	236
PCI Configuration Header.....	236
HyperTransport Configuration Header.....	245
System Reset Initialization of the HyperTransport Interface.....	256
Configuration Flags in the SriCmd Register.....	257
Timing Registers: SriRxDen, SriTxDen, SriRxNum and SriTxNum .....	257
Receive Pointer Margin Control in SriCmd Register.....	258
Transmit Pointer Initial Offset in the SriCmd Register .....	259
Error Control Register.....	259
Transmit Control Register.....	259

---

Buffer Control: TxBufCountMax and DataBufAlloc.....	260
HyperTransport Resets .....	260
<b>Section 9: Ethernet MACs .....</b>	<b>264</b>
<b>Introduction.....</b>	<b>264</b>
<b>Interface Overview.....</b>	<b>265</b>
<b>Protocol Engine and GMII/MII.....</b>	<b>267</b>
Ethernet Frame Format .....	268
Prepended Header Frame Format .....	270
Protocol Engine Configuration.....	271
Interface to PHY .....	271
<b>Transmitter Operation.....</b>	<b>272</b>
Transmitter Configuration .....	272
Transmit Path .....	274
<b>Receiver Operation.....</b>	<b>275</b>
Receiver Configuration .....	275
Receive Path .....	277
Destination Address Filtering.....	278
Receive DMA Channel Selection .....	281
Packet Type Identification .....	282
IPv4 Header Checksum.....	283
TCP Checksum Checking .....	283
Packets Dropped by the DMA Channel.....	283
<b>Flow Control.....</b>	<b>284</b>
<b>Interrupts.....</b>	<b>286</b>
Standard Interrupt Signaling.....	286
Split Interrupt Signaling .....	286
<b>Management Interface to PHY .....</b>	<b>287</b>
<b>RMON Counters.....</b>	<b>289</b>
<b>Packet FIFO Interfaces.....</b>	<b>292</b>
Flow Control In Encoded Packet FIFO Modes .....	294
<b>8-Bit Packet FIFO Operation .....</b>	<b>294</b>
8-Bit GMII Style Packet FIFO .....	295
8-Bit Encoded Packet FIFO.....	296
8-Bit SOP Flagged Packet FIFO .....	297



8-Bit EOP Flagged Packet FIFO .....	298
<b>16-Bit Packet FIFO Operation</b> .....	299
16-Bit GMII Style Packet FIFO.....	299
16-Bit Encoded Packet FIFO .....	300
<b>Restrictions When Resetting the Interface</b> .....	301
<b>MAC Registers</b> .....	302
<b>Section 10: Serial Interfaces</b> .....	321
<b>Introduction</b> .....	321
<b>Asynchronous Mode</b> .....	322
<b>Baud Rate Generators</b> .....	322
<b>Operation</b> .....	323
<b>Interrupts</b> .....	325
<b>Loopback</b> .....	326
<b>DUART Registers</b> .....	327
<b>Synchronous Mode</b> .....	337
Functional Overview .....	337
Input Line Interface .....	340
Input Using an External Enable .....	340
Input Using the Internal Sequencer .....	341
Output Line Interface .....	342
Output Using an External Enable .....	342
Output Using the Internal Sequencer .....	343
<b>Synchronous Serial Protocol Engine</b> .....	344
Operation in HDLC Mode.....	344
Framing Parameters .....	345
HDLC Transmitter .....	345
HDLC Receiver .....	347
Operation in Transparent Mode .....	349
Transmitter in Transparent Mode .....	350
Receiver in Transparent Mode .....	350
<b>Synchronous Interface Configuration</b> .....	351
DMA Configuration.....	351
FIFO Configuration .....	351
Protocol Engine Configuration .....	351

---

Line Interface Configuration .....	352
<b>Synchronous Serial Interrupts</b> .....	352
<b>Synchronous Serial Loopback</b> .....	352
<b>RMON Counters</b> .....	353
<b>Synchronous Serial Register Summary</b> .....	354
<b>Section 11: Generic/Boot Bus</b> .....	362
<b>Introduction</b> .....	362
<b>Overview</b> .....	362
<b>Configuring a Chip Select Region</b> .....	363
Address Range .....	363
Cacheable Access Blocking .....	364
Generic Bus Parity .....	364
Bus Width .....	365
Generic Bus Timing .....	365
Fixed Cycle Read Access .....	367
Fixed Cycle Write Access .....	368
Acknowledgement Read Access .....	369
Acknowledgement Write Access .....	370
Burst Mode .....	371
Early Chip Select .....	373
<b>Boot ROM Support</b> .....	373
<b>Generic Bus Errors</b> .....	374
<b>Drive Strength Control</b> .....	374
<b>Generic Bus Registers</b> .....	375
<b>Section 12: PCMCIA Control Interface</b> .....	384
<b>Introduction</b> .....	384
<b>Connecting a PCMCIA Slot</b> .....	384
Direct Connection of a Memory Only Card .....	385
Other PCMCIA Signals .....	389
<b>Using The PCMCIA Card</b> .....	390
Example PCMCIA Timings .....	391
Using the Power Outputs .....	393



---

<b>Section 13: GPIO</b> .....	397
<b>Introduction</b> .....	397
<b>The GPIO Pins</b> .....	397
<b>GPIO Registers</b> .....	399
<b>Other Pins That Can Be Used</b> .....	401
Serial Ports .....	401
PCI .....	401
MACs .....	401
PCMCIA Power Control Pins .....	401
<b>Section 14: Serial Configuration Interface</b> .....	404
<b>Introduction</b> .....	404
<b>SMBus Overview</b> .....	404
Transport Protocol .....	404
Transport Protocol Reset .....	406
SMBus Protocol .....	406
Extended Protocol .....	408
<b>Programming Model</b> .....	410
Using SMBus Protocols .....	410
Using Extended Protocols .....	412
<b>Direct Access</b> .....	413
<b>Booting Using an SMBus EEPROM</b> .....	413
Switching from SMBus Mode .....	414
<b>SMBus Registers</b> .....	416
<b>Section 15: JTAG and Debug</b> .....	422
<b>Introduction</b> .....	422
<b>TAP Controller</b> .....	422
BYPASS Instruction .....	425
IDCODE Instruction .....	425
WAFERID Instruction .....	425
IMPCODE Instruction .....	426
ADDRESS Instruction .....	426
DATA Instruction .....	426
CONTROL Instruction .....	426

---

EJTAGALL Instruction.....	426
EJTAGBOOT Instruction.....	426
NORMALBOOT Instruction.....	427
SCAN Instructions (0x26 - 0x38).....	427
SYSCTRL Instruction.....	427
TRACE Instruction.....	430
PERF Instruction.....	430
TRACECTRL and TRACECURCNT Instructions.....	431
PROCESSMON Instruction.....	432
Boundary Scan Register.....	432
BSRMODE - Holding Boundary Scan Active.....	435
<b>Processor and Probe Access.....</b>	<b>436</b>
Processor Accesses to the JTAG Space.....	438
Probe Accesses to the ZBbus.....	438
Address Register.....	439
Data Register.....	439
EJTAG Control Register.....	440
<b>Differences from EJTAG 2.5 (Feb. 22, 2000) Specification.....</b>	<b>442</b>
<b>Section 16: Reference.....</b>	<b>446</b>
<b>Internal Register Addresses by Function.....</b>	<b>446</b>
<b>BCM1250/BCM1125/H Internal Registers Ordered by Address.....</b>	<b>464</b>



---

## LIST OF FIGURES

Figure 1: BCM1250 Block Diagram .....	2
Figure 2: BCM1125/H Block Diagram .....	3
Figure 3: BCM1250 Signals.....	7
Figure 4: BCM1125/H Signals .....	8
Figure 5: Logical Block Diagram of BCM1250 and BCM1125/H .....	9
Figure 6: Internal Control and Status Register Alignment .....	11
Figure 7: Decision Tree for Memory Space Address Accesses .....	26
Figure 8: Clock Distribution Overview .....	33
Figure 9: Memory Map .....	35
Figure 10: Per-CPU Interrupt Mapper (replicated for each CPU; x = 0 or 1).....	51
Figure 11: Connections to Trace Logic.....	83
Figure 12: Level 2 Cache Way Disable Access Address .....	91
Figure 13: Cache Management Address.....	95
Figure 14: Memory Controller Block Diagram .....	105
Figure 15: Chip Select Options.....	110
Figure 16: Example Single Channel 128MB.....	112
Figure 17: Example 1GB with two chip selects on one channel.....	113
Figure 18: Example 1GB with two chip selects interleaved on one channel .....	114
Figure 19: Example 1GB with two chip selects interleaved across both channels.....	115
Figure 20: Example 2GB with two chip selects interleaved on one channel .....	116
Figure 21: Timing Relationships Set by DLLs .....	130
Figure 22: Nominal Windows at 133MHz for First Edge of DQS for Various Settings of [tCrD, tCrDh, tFIFO] ....	131
Figure 23: DMA Buffer.....	148
Figure 24: DMA Descriptor .....	149
Figure 25: Packet Spanning Three Buffers .....	150
Figure 26: DMA Descriptor Ring.....	151
Figure 27: DMA Descriptor Chain.....	153
Figure 28: Standard and Unaligned Buffer DMA Descriptors.....	154
Figure 29: Packet Reception Flow using DMA ASIC Mode.....	159
Figure 30: ASIC Mode Address Generation .....	160
Figure 31: Sending the Whole Packet in ASIC Mode.....	161
Figure 32: Sending a Packet Header in ASIC Mode .....	162

---

Figure 33: Example 1 - TCP checksum a packet.....	181
Figure 34: Example 2 - Preparing an iSCSI packet .....	182
Figure 35: Example 3 - Fragmenting an iSCSI packet.....	183
Figure 36: PCI and HyperTransport Organization .....	191
Figure 37: Address Ranges for CPU Access to PCI and HyperTransport.....	193
Figure 38: Little Endian System .....	201
Figure 39: Match Byte Lane Endian Policy .....	202
Figure 40: Match Bit Lane Endian Policy .....	203
Figure 41: PCI BAR0 Address Mapping Table .....	206
Figure 42: Default Host Mode Memory Map from PCI Bus Master.....	209
Figure 43: Memory Map From HyperTransport Device.....	210
Figure 44: Buffers Used for Accesses from the ZBbus to PCI and HyperTransport.....	214
Figure 45: Buffers Used for DMA Accesses from the PCI and HyperTransport .....	216
Figure 46: PCI Adaptive Retry Parameters.....	218
Figure 47: Buffers Used for PCI to HyperTransport Peer-to-Peer Accesses.....	220
Figure 48: Buffers Used for HyperTransport to PCI Peer-to-Peer Accesses.....	221
Figure 49: Configuration Space Address .....	234
Figure 50: HyperTransport Interface Clocks and FIFOs .....	256
Figure 51: Ethernet Interface Block Diagram.....	265
Figure 52: Ethernet Frame Format .....	268
Figure 53: Prepended Header Format.....	270
Figure 54: Transmit FIFO Thresholds.....	272
Figure 55: Receive FIFO Thresholds.....	275
Figure 56: Receive Address Filter.....	278
Figure 57: Receive Channel Selection.....	281
Figure 58: Selecting the Channel Offset .....	281
Figure 59: MDIO Flows .....	288
Figure 60: 8-bit Packet FIFO GMII Style .....	295
Figure 61: 8-Bit Packet FIFO Encoded Style .....	296
Figure 62: 8-Bit Packet FIFO SOP Style .....	297
Figure 63: 8-Bit Packet FIFO EOP Style .....	298
Figure 64: 16-Bit GMII Style Packet FIFO .....	299
Figure 65: 16-Bit Encoded Packet FIFO .....	300
Figure 66: UART Interrupt Generation.....	325
Figure 67: Synchronous Interface Block Diagram .....	338





---

Figure 68: Example Reception Using RIN as Active High Enable (sampling on the falling clock edge) .....	340
Figure 69: Example Reception Using RIN as Active High Sync (sampling on the falling clock edge) .....	342
Figure 70: Example Transmission Using TIN as Active High Enable (Driving/Sampling on Rising Clock Edge). 343	
Figure 71: Example Transmission Using TIN as Active High Sync (transition/sampling on rising clock edge).... 344	
Figure 72: Frame Address Matching .....	347
Figure 73: Synchronous Serial Loopback Connections.....	352
Figure 74: Fixed Cycle Read Access .....	367
Figure 75: Fixed Cycle Write Access.....	368
Figure 76: Acknowledge Read Access.....	369
Figure 77: Acknowledge Write Access .....	370
Figure 78: Generic Bus Burst Read.....	371
Figure 79: Generic Bus Burst Write .....	371
Figure 80: Example PCMCIA Slot Connection .....	385
Figure 81: Example Flash Card Timing Diagram .....	392
Figure 82: Single GPIO Pin Diagram.....	397
Figure 83: SMBus Signaling Start, Data Transfer and Stop .....	405
Figure 84: JTAG TAP State Machine .....	423
Figure 85: JTAG Boundary Scan Register Block .....	433
Figure 86: JTAG HyperTransport Output Boundary Scan Block .....	434
Figure 87: JTAG HyperTransport Input Boundary Scan Block.....	435
Figure 88: Example JTAG Probe Flowchart .....	437



---

## LIST OF TABLES

Table 1: ZBbus Agent IDs .....	20
Table 2: ZBbus Signals .....	20
Table 3: ZBbus Commands .....	22
Table 4: ZBbus Level 1 Cache Attributes .....	23
Table 5: ZBbus Byte Lane Assignments .....	24
Table 6: ZBbus Data Status Codes .....	24
Table 7: Operation of Different Reset Sources .....	27
Table 8: Static Configuration Options .....	27
Table 9: Core and HyperTransport Clock Settings .....	31
Table 10: Overview of BCM1250 Physical Address Map .....	34
Table 11: Address Map Details .....	36
Table 12: System Identification and Revision Register .....	42
Table 13: Part Revisions .....	42
Table 14: Manufacturing Information Register .....	43
Table 15: System Configuration Register .....	43
Table 16: Scratch Register .....	45
Table 17: Mailbox Registers .....	46
Table 18: Interrupt Mappings .....	47
Table 19: Interrupt Message Format for Writes to interrupt_idt_set Register .....	49
Table 20: Delivery of HyperTransport Interrupts .....	49
Table 21: Interrupt Registers .....	52
Table 22: Interrupt Sources .....	52
Table 23: Watchdog Timer Initial Count Registers .....	59
Table 24: Watchdog Timer Current Count Registers .....	59
Table 25: Watchdog Timer Configuration Registers .....	59
Table 26: General Timer Initial Count Registers .....	60
Table 27: General Timer Current Count Registers .....	60
Table 28: General Timer Configuration Registers .....	60
Table 29: ZBbus Count Register .....	61
Table 30: ZBbus Count Compare Registers .....	61
Table 31: System Performance Counter Configuration Registers .....	61
Table 32: System Performance Counters .....	62

---

Table 33: System Performance Counter Sources .....	62
Table 34: Bus Watcher Counters.....	64
Table 35: Bus Watcher Error Status Register.....	65
Table 36: Bus Watcher Error Status Debug Register .....	65
Table 37: Bus Watcher Error Data Registers.....	65
Table 38: Bus Watcher L2 ECC Counter Register.....	66
Table 39: Bus Watcher Memory and I/O Error Counter Register.....	66
Table 40: Address Trap Trigger Index Register .....	68
Table 41: Address Trap Trigger Debug Register .....	68
Table 42: Address Trap Trigger Address Register .....	68
Table 43: Address Trap Range Top Address Registers .....	68
Table 44: Address Trap Range Base Address Registers .....	69
Table 45: Address Trap Configuration Registers .....	69
Table 46: Trace Event Register .....	72
Table 47: Trace Sequence Control Registers.....	74
Table 48: Trace Control Register.....	76
Table 49: Trace Buffer Address/Control Bundle .....	77
Table 50: Trace Entry Format and Read Order .....	79
Table 51: Decode of some TIDs for system revision PERIPH_REV3.....	81
Table 52: Encoded Byte Enables for CPU Transactions .....	82
Table 53: Addresses for Memory Banks.....	92
Table 54: Management Address.....	95
Table 55: ECC Diagnostic Operations .....	98
Table 56: Level 2 Cache Tag Register .....	100
Table 57: Level 2 Cache Settings Register.....	100
Table 58: Clock Speed.....	107
Table 59: Percent Deltas from Popular DIMM Frequencies.....	108
Table 60: Mapping Physical Address To Memory Controller Address.....	109
Table 61: Address Bits Used by a Memory Channel .....	117
Table 62: Example for 128 MByte CS Region with 4K Rows, 1K Columns.....	118
Table 63: Example for 128 MByte CS Region with 4K Rows, 1K Columns, 64 Byte Interleave.....	118
Table 64: Example for 128 MByte CS Region with 4K Rows, 1K Columns, 128 Byte Interleave.....	119
Table 65: Example for 256 MByte Region with 4K Rows, 1K Columns, two CS, and 128 Byte Interleave ...	119
Table 66: Example for 512 MByte Region with 4K Rows, 1K Columns, four CS, and 128 Byte Interleave...	119
Table 67: Example for 128 MByte + 64 MB + 64 MB Mixed_CS Mode .....	120



Table 68: Supported SDRAMs .....	123
Table 69: Commands that can be Issued Through mc_dramcmd Register .....	126
Table 70: Adjustment Percentages and Multiplier for Values of DLL M .....	128
Table 71: First DQS Window Opening and Closing (Typical).....	132
Table 72: Memory Channel Configuration Register on BCM1250 .....	135
Table 73: Memory Channel Configuration Register on BCM1125/H.....	136
Table 74: Memory Clock Configuration Register.....	138
Table 75: DRAM Command Register .....	139
Table 76: DRAM Mode Register.....	139
Table 77: SDRAM Timing Register .....	140
Table 78: SDRAM Timing Register 2 .....	141
Table 79: Chip Select Start Address Register .....	141
Table 80: Chip Select End Address Register .....	141
Table 81: Chip Select Interleave Register .....	142
Table 82: Row Address Bits Select Register .....	142
Table 83: Column Address Bits Select Register.....	142
Table 84: Bank Address Bits Select Register.....	143
Table 85: Chip Select Attribute Register .....	143
Table 86: ECC Test Data Register .....	144
Table 87: ECC Test ECC Register .....	144
Table 88: Data Buffer Parameters.....	148
Table 89: Data Parameters .....	148
Table 90: Address Used for ASIC Mode Transfers .....	160
Table 91: Ethernet and Serial DMA Configuration Register 0 .....	163
Table 92: Ethernet and Serial DMA Configuration Register 1 .....	164
Table 93: Ethernet and Serial DMA Descriptor Base Address Register.....	166
Table 94: ASIC Mode Base Address.....	166
Table 95: Descriptor Count Register .....	166
Table 96: Current Descriptor A Debug Register.....	167
Table 97: Current Descriptor B Debug Register.....	167
Table 98: Current Descriptor Address Register.....	167
Table 99: Ethernet Receive Packet Drop Registers (Only if System Revision >= PERIPH_REV3).....	168
Table 100: DMA Descriptor First Doubleword .....	169
Table 101: DMA Descriptor Second Doubleword.....	169
Table 102: Unaligned Buffer Format DMA Descriptor First Doubleword.....	170

---

Table 103: Unaligned Buffer Format DMA Descriptor Second Doubleword .....	170
Table 104: Status Flags for Ethernet Receive Channel.....	171
Table 105: Option Flags for Ethernet Receive Channel .....	172
Table 106: Status Flags for Ethernet Transmit Channel.....	172
Table 107: Option Flags for Ethernet Transmit Channel .....	173
Table 108: Status Flags for Synchronous Serial Receive Channel .....	174
Table 109: Option Flags for Synchronous Serial Receive Channel.....	174
Table 110: Status Flags for Synchronous Serial Transmit Channel .....	174
Table 111: Option Flags for Synchronous Serial Transmit Channel.....	174
Table 112: Result in memory of appending the result CRC[31:0].....	179
Table 113: Example CRC configurations.....	180
Table 114: Data Mover Descriptor Base Address Register .....	184
Table 115: Debug Data Mover Descriptor Base Address Register.....	184
Table 116: Data Mover Descriptor Count Register .....	185
Table 117: Data Mover Current Descriptor Address.....	185
Table 118: Data Mover CRC Definition Registers (Only if System Revision >= PERIPH_REV3) .....	185
Table 119: Data Mover CRC/Checksum Definition Registers (Only if System Revision >= PERIPH_REV3).....	186
Table 120: Data Mover Channel Partial Result Registers (Only if System Revision >= PERIPH_REV3) .....	186
Table 121: Data Mover Descriptor First Doubleword.....	187
Table 122: Data Mover Descriptor Second Doubleword.....	188
Table 123: PCI Base Address Register Use.....	205
Table 124: Adaptive Retry Delay .....	218
Table 125: Error Routing Registers .....	229
Table 126: PCI CSR Access Rules.....	233
Table 127: PCI Interface Configuration Header (Type 0) .....	236
Table 128: PCI Command Register - Offset 4 Bits [15:0] .....	240
Table 129: PCI Status Register - Offset 4 Bits [31:16].....	240
Table 130: PCI Latency Timer - Offset 0C Bits [15:8].....	241
Table 131: PCI Cache Line Size - Offset 0C Bits [7:0] .....	241
Table 132: PCI Timeout Register - Offset 40 Bits [15:0].....	241
Table 133: PCI Feature Control Register - Offset 40 Bits [31:16].....	242
Table 134: PCI BAR0 Map Table Entry - Offset 44 – 80 .....	242
Table 135: PCI Additional Status and Control Register - Offset 88 Bits [31:0] .....	242
Table 136: PCI INTA Control Register - Offset 90 Bits [31:0] .....	243
Table 137: PCI Read Host Register - Offset 94 Bits [31:0].....	243



Table 138: PCI Adaptive Extend Register - Offset 98 Bits [31:0] .....	243
Table 139: PCI Bypass Control Register - RevId >= 3 Offset A8 Bits [31:0] .....	244
Table 140: HyperTransport Configuration Header (Type 1) .....	245
Table 141: HyperTransport Bridge Command Register - Offset 4 Bits [15:0] .....	247
Table 142: HyperTransport Bridge Primary (ZBbus) Status Register - Offset 4 Bits [31:16] .....	248
Table 143: HyperTransport Bridge Secondary (HT) Status Register - Offset 1C Bits [31:16] .....	248
Table 144: HyperTransport Bridge Control Register - Offset 3C Bits [31:16] .....	249
Table 145: HyperTransport Command Register - Offset 40 Bits [31:16] .....	249
Table 146: HyperTransport Link Control Register - Offset 44 Bits [15:0] .....	250
Table 147: HyperTransport Link Configuration Register - Offset 44 Bits [31:16] .....	251
Table 148: HyperTransport Link Frequency Register - Offset 48 Bits [15:8] .....	251
Table 149: HyperTransport SRI Command Register - Offset 50 Bits [31:16] .....	252
Table 150: HyperTransport Isochronous BAR - Offset 5C Bits [31:0] .....	252
Table 151: HyperTransport Isochronous Ignore Mask - Offset 60Bits [31:0] .....	253
Table 152: HyperTransport Error Control Register - Offset 68 Bits [23:0] .....	253
Table 153: HyperTransport Error Status Register - Offset 68 Bits [31:24] .....	254
Table 154: HyperTransport SRI Transmit Control Register - Offset 6C Bits [23:16] .....	254
Table 155: HyperTransport SRI Data Buffer Allocation Register - Offset 6C Bits [15:0] .....	254
Table 156: HyperTransport Additional Status Register - Offset 70 .....	255
Table 157: HyperTransport SRI Transmit Buffer Count Max Register - Offset C8 Bits [31:0] .....	255
Table 158: HyperTransport Diagnostic Receive CRC Expected - Offset DC .....	255
Table 159: HyperTransport Diagnostic Receive CRC Received - Offset F0 .....	255
Table 160: Ethernet Frame Fields .....	269
Table 161: Transmission Error Conditions .....	273
Table 162: Receiver Error Conditions .....	276
Table 163: Ethernet Type Mappings .....	282
Table 164: Back Pressure Methods in Half-Duplex Operation .....	284
Table 165: Pause Frame Options .....	285
Table 166: MAC to PHY Management Protocol .....	289
Table 167: RMON Counters .....	289
Table 168: BCM1125 Ethernet/Fifo Pin Usage .....	292
Table 169: BCM1250 Ethernet/Fifo Pin Usage .....	293
Table 170: Codes for GMII Packet FIFO Mode .....	295
Table 171: Codes for 8-Bit Encoded Bypass Mode .....	296
Table 172: Codes for 8-Bit SOP Packet FIFO .....	297

---

Table 173: Codes for 8-Bit EOP Bypass Mode.....	298
Table 174: Codes for 16-Bit GMII Style Packet FIFO .....	300
Table 175: Codes for 16-Bit Encoded Bypass Mode .....	300
Table 176: MAC Configuration Registers .....	302
Table 177: MAC Enable Registers.....	306
Table 178: MAC Transmit DMA Control Register .....	306
Table 179: MAC FIFO Threshold Registers.....	307
Table 180: MAC Frame Configuration Registers .....	308
Table 181: MAC VLAN Tag Registers .....	310
Table 182: MAC Status Registers.....	310
Table 183: MAC Status 1 Register .....	313
Table 184: MAC Debug Status Registers .....	313
Table 185: MAC Interrupt Mask Registers.....	314
Table 186: MAC FIFO Pointer Registers .....	314
Table 187: MAC Receive Address Filter Exact Match Registers.....	314
Table 188: MAC Receive Address Filter Mask Registers (Only if System Revision >= PERIPH_REV3).....	315
Table 189: MAC Receive Address Filter Hash Match Registers.....	315
Table 190: MAC Transmit Source Address Registers .....	315
Table 191: MAC Packet Type Configuration Registers .....	316
Table 192: MAC Receive Address Filter Control Registers .....	316
Table 193: MAC Receive Channel Select Map Registers.....	318
Table 194: MAC MII Management Interface Registers.....	318
Table 195: Serial Interface Signal Names .....	321
Table 196: Baud Rate Counter Values .....	322
Table 197: DUART Mode Registers .....	327
Table 198: DUART Second Mode Registers .....	327
Table 199: DUART Command Registers.....	328
Table 200: DUART Status Registers .....	328
Table 201: DUART Baud Rate Clock Registers .....	329
Table 202: DUART Full Interrupt Control Registers.....	329
Table 203: DUART Received Data Registers.....	329
Table 204: DUART Transmit Data Registers.....	330
Table 205: DUART Input Port Register.....	330
Table 206: DUART Input Port Change Status Register.....	330
Table 207: DUART Debug Access Input Port Change Register.....	331





---

Table 208: DUART Input Port Change Status Register for Channel A .....	331
Table 209: DUART Input Port Change Status Register for Channel B .....	331
Table 210: DUART Output Port Control Register.....	331
Table 211: DUART Per Channel Output Control Registers.....	332
Table 212: DUART Aux Control Register .....	332
Table 213: DUART Per Channel Aux Control Registers .....	332
Table 214: DUART Interrupt Status Register .....	333
Table 215: DUART Channel A Only Interrupt Status Register .....	333
Table 216: DUART Channel B Only Interrupt Status Register .....	333
Table 217: DUART Interrupt Mask Register.....	334
Table 218: DUART Channel A Only Interrupt Mask Register.....	334
Table 219: DUART Channel B Only Interrupt Mask Register.....	334
Table 220: DUART Output Port Set Register .....	335
Table 221: DUART Output Port Clear Register.....	335
Table 222: DUART Output Port RTS Register .....	335
Table 223: Synchronous Serial Interface Signal Names .....	339
Table 224: Synchronous Serial Interface GPIO Pins .....	339
Table 225: Sequencer Table Entries .....	341
Table 226: HDLC Frame Structure.....	344
Table 227: Option Flags for Synchronous Serial Transmit Channel .....	345
Table 228: Status Flags for Synchronous Serial Receive Channel.....	349
Table 229: Recommended Line Interface Settings for Loopback Mode 2.....	353
Table 230: RMON Counters .....	353
Table 231: Serial Mode Configuration Register.....	354
Table 232: Synchronous Serial Clock Source and Line Interface Mode Register .....	354
Table 233: Synchronous Serial Command Register (Write-only).....	355
Table 234: Serial Write Threshold Register.....	356
Table 235: Serial Transmit Read Threshold Register .....	356
Table 236: Serial Receive Read Threshold Register .....	356
Table 237: Serial Minimum Frame Size Register .....	356
Table 238: Serial Maximum Frame Size Register .....	357
Table 239: Serial DMA Enable Registers .....	357
Table 240: Synchronous Serial Status Register.....	357
Table 241: Serial Status Debug Register .....	358
Table 242: Serial Interrupt Mask Register.....	358

---

Table 243: Serial Address Mask Register.....	359
Table 244: Serial Address Match Register .....	359
Table 245: Sequencer Table Entries .....	359
Table 246: Serial RMON Counters .....	360
Table 247: Byte Lanes for the Generic Bus .....	363
Table 248: Generic Bus Timing Parameters.....	366
Table 249: Burst Cycle Summary .....	372
Table 250: Generic Bus Configuration for Each Boot Mode .....	373
Table 251: Generic Bus Region Configuration Registers .....	375
Table 252: Generic Bus Region Size Registers.....	375
Table 253: Generic Bus Region Base Address Registers .....	376
Table 254: Generic Bus Region Timing 0 Registers.....	376
Table 255: Generic Bus Region Timing 1 Registers.....	377
Table 256: Generic Bus Interrupt Status Register .....	377
Table 257: Generic Bus Error Data Register 0 .....	378
Table 258: Generic Bus Error Data Register 1 .....	378
Table 259: Generic Bus Error Data Register 2 .....	378
Table 260: Generic Bus Error Data Register 3 .....	378
Table 261: Generic Bus Error Address Register 0.....	379
Table 262: Generic Bus ErrorAddress Register 1 .....	379
Table 263: Generic Bus Error Parity Register.....	379
Table 264: Output Drive Control Register 0 .....	379
Table 265: Output Drive Control Register 1 .....	380
Table 266: Output Drive Control Register 2 .....	381
Table 267: Output Drive Control Register 3.....	381
Table 268: Source for PCMCIA Card Enable Signals.....	386
Table 269: PCMCIA 3.3V and 5V VCC Power Enable Truth Table.....	387
Table 270: PCMCIA VPP Power Enable Truth Table .....	388
Table 271: Example Flash Card AC Specs .....	391
Table 272: Example Generic Bus Timing parameters .....	393
Table 273: PCMCIA Configuration Register .....	394
Table 274: PCMCIA Status Register .....	395
Table 275: GPIO Pins and Alternate Uses .....	398
Table 276: GPIO Edge Clear Register.....	399
Table 277: GPIO Interrupt Type Register .....	399



Table 278: GPIO Read Register.....	399
Table 279: GPIO Input Invert Control Register.....	399
Table 280: GPIO Glitch Filter Select Register .....	400
Table 281: GPIO Direction Register .....	400
Table 282: GPIO Pin Clear Register .....	400
Table 283: GPIO Pin Set Register.....	400
Table 284: Other Pins that can be Used as General Inputs or Outputs .....	401
Table 285: Supported SMBus Transfer Types .....	406
Table 286: Command/Address Options .....	408
Table 287: Write Data Options .....	408
Table 288: Read Data Options .....	409
Table 289: SMBus Clock Frequency Registers.....	416
Table 290: SMBus Command Registers .....	416
Table 291: SMBus Control Registers .....	416
Table 292: SMBus Status Registers.....	417
Table 293: SMBus Data Registers .....	417
Table 294: SMBus Extra Data Registers.....	417
Table 295: SMBus Packet Error Check Registers.....	418
Table 296: SMBus Start and Command Registers SMBus Mode .....	418
Table 297: SMBus Start and Command Registers Extended Mode .....	419
Table 298: JTAG Signals.....	422
Table 299: JTAG Instructions .....	424
Table 300: JTAG Device ID Register .....	425
Table 301: JTAG Wafer ID Register .....	426
Table 302: System Control Scan Chain .....	428
Table 303: Performance Counter Scan Chain.....	430
Table 304: Trace Control Scan Chain .....	431
Table 305: Trace Current Count Scan Chain .....	431
Table 306: Ring Oscillator Scan Chain.....	432
Table 307: CPU and Probe Accesses .....	436
Table 308: JTAG Address Register Scan Chain .....	439
Table 309: Data Register Scan Chain .....	439
Table 310: EJTAG Control Register .....	440
Table 311: Internal Register Addresses by Function.....	446
Table 312: Internal Registers Ordered by Address .....	464



# Section 1: Introduction

## THE SiBYTE BROADBAND PROCESSOR FAMILY

The SiByte Broadband Processors form a family of high performance system-on-a-chip parts targeted at network centric tasks. Examples include:

- In-line packet processing
- Exception processing
- Switch control and management
- Higher layer switching and filtering
- Protocol conversion (VoIP Gateway)
- Network appliances (file servers, web cache, print servers)
- VPN access, firewalls, gateways

The different members of the family target different performance points and applications, while retaining software compatibility. This User Manual covers the dual-processor BCM1250, and the uni-processor BCM1125 and BCM1125H parts.

All the parts use the SB-1 CPU core. This is a high performance implementation of the standard MIPS64 instruction set architecture. The core supports a 4-issue enhanced skew pipeline and can dispatch up to two memory and two ALU (Integer, Floating point or MIPS-3D) instructions per cycle.

# THE BCM1250

The core of the BCM1250 is an on-chip multiprocessor (CMP) system consisting of two Broadcom SB-1 high performance MIPS64 CPUs, a shared 512K L2 cache and a DDR SDRAM memory controller. Three integrated 10/100/1000 Ethernet MACs enable easy interfacing to LANs. The three network interfaces can also be operated together to give two 16 bit wide interfaces that can run full-duplex at OC-48 rates. Two serial ports are provided for use as UARTs or for WAN connections at up to T3/OC-1 rates (55Mbit/s). High speed I/O is provided using the HyperTransport (formerly called "Lightning Data Transport" or "LDT") I/O fabric and a 66 MHz PCI (rev 2.2) local bus. To enable low chip count systems the BCM1250 includes a configurable generic bus that allows glueless connection of a boot ROM or flash memory and simple I/O peripherals. On-chip debug, trace and performance monitoring functions assist both hardware and software designers in debugging and tuning the system. The system can be run either big endian or little endian.

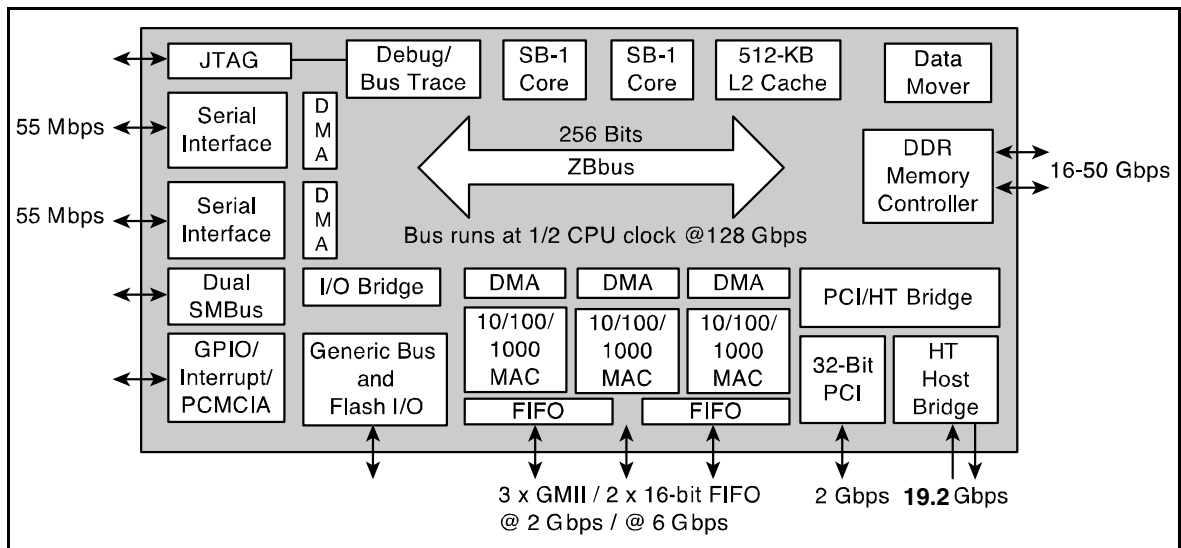


Figure 1: BCM1250 Block Diagram



## THE BCM1125 AND BCM1125H

The core of the BCM1125 and BCM1125H is a uniprocessor system consisting of a Broadcom SB-1 high performance MIPS64 CPU, a 256K L2 cache and a DDR SDRAM memory controller. Two integrated 10/100/1000 Ethernet MACs enable easy interfacing to LANs. The two network interfaces can also be operated together to give a 16 bit wide interface that can run full-duplex at OC-48 rates. Two serial ports are provided for use as UARTs or for WAN connections at up to T3/OC-1 rates (55Mbit/s). High speed I/O is provided on the BCM1125H using the HyperTransport (formerly called "Lightning Data Transport" or "LDT") I/O fabric. Both the BCM1125 and BCM1125H have a 66 MHz PCI (rev 2.2) local bus. To enable low chip count systems both parts include a configurable generic bus that allows glueless connection of a boot ROM or flash memory and simple I/O peripherals. On-chip debug, trace and performance monitoring functions assist both hardware and software designers in debugging and tuning the system. The system can be run either big endian or little endian.

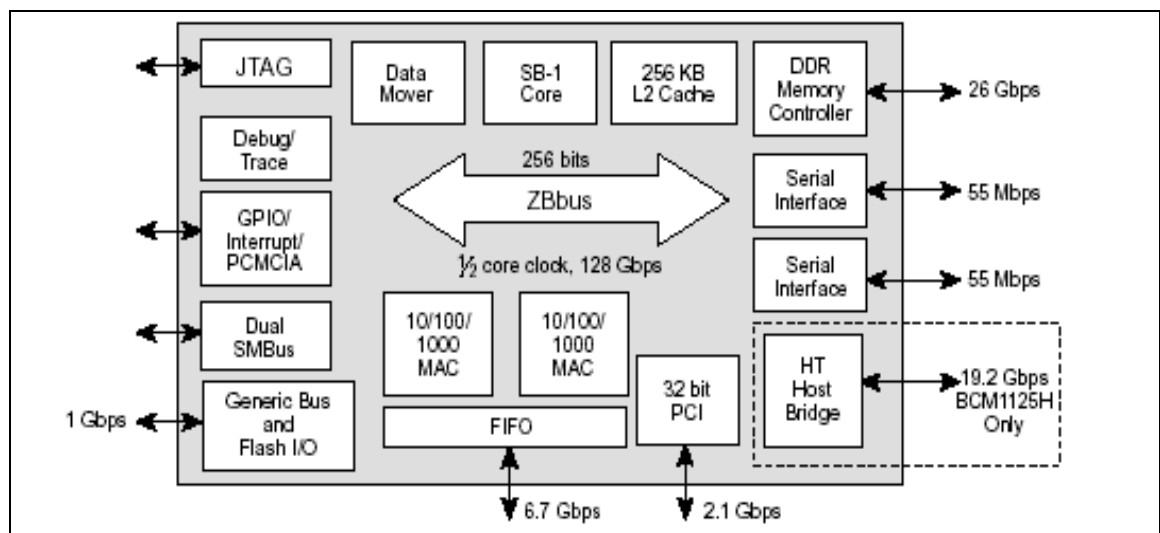


Figure 2: BCM1125/H Block Diagram

## AUDIENCE

This User Manual includes information that is needed when writing software for the BCM1250, BCM1125 or BCM1125H. It provides a system overview for Systems and Hardware Designers. This User Manual is common across the parts, reflecting their software compatibility. Each part has an individual Data Sheet containing detailed hardware information. (Note that some hardware details of the BCM1250 that were in previous versions of the User Manual are now found only in the Data Sheet.)

## OTHER DOCUMENTATION

Most of the example designs and Application Notes for the BCM1250 also apply to the BCM1125 and BCM1125H. Documentation useful to users of all the devices includes:

- *SB-1 Core Processor User Manual (SB-1-UM00-R)*
- *BCM1250 Board Routability Considerations Application Note (1250-AN101-R)*
- *BCM1250 Generic Bus Interface to ATA/ATAPI PIO Mode 3 (IDE) Hard Disk Application Note (1250-AN201-R)*
- *BCM1250 DRAM Support Application Note (1250-AN302-R)*
- *BCM1250 Power Supply Application Note (1250-AN4xx-R)*
- *BCM1250 Design Considerations for Fast Memory Systems Application Note (1250-AN5xx-R)*
- *BCM1250 Big Memory System Application Note (1250-AN6xx-R)*
- *BCM1250 Generic Bus Interface to 3-Volt Intel StrataFlash Memory (1250-AN7xx-R)*
- *HyperTransport I/O Link Specification Revision 1.03 from the HyperTransport Technology Consortium*
- *PCI Specification - Revision 2.2*
- *MIPS64 Architecture for Programmers: <http://www.mips.com/publications/index.html>*
  - *Volume I: Introduction to the MIPS64 Architecture (MD00083)*
  - *Volume II: The MIPS64 Instruction Set (MIPS MD00087)*
  - *Volume III: Privileged Resource Architecture (MIPS MD00091)*
  - *Volume IV-b: The MDMX ASE to MIPS64 (MIPS MD00095)*
  - *Volume IV-c: The MIPS-3D ASE to MIPS64 (MIPS MD00099)*
- *MIPS EJTAG Specification revision 2.5*
- *Corelis Application Note #00-117 BCM1250 EJTAG Connector*

Documentation specific to the BCM1250 device includes:

- *BCM1250 Data Sheet (1250-DS02-R)*
- *BCM91250A Evaluation Card for the BCM1250 Product Brief (91250A-PB03-R)*
- *BCM91250E PCI Evaluation Card for the BCM1250 Product Brief (91250E-PB01-R)*
- *BCM1250 Package Thermal Report (available on request)*

Documentation specific to the BCM1125 and BCM1125H includes:

- *BCM1125/H Data Sheet (1125H-DS01-R)*
- *BCM91125E PCI Evaluation Card for the BCM1125/H Product Brief (91125E-PB01-R)*
- *BCM1125/H Package Thermal Report (available on request)*



## TERMINOLOGY

**Numbers** used in data fields of this document follow the verilog convention of giving the field size in decimal, followed by a quote ('), a character representing the base (b for binary, d for decimal or h for hexadecimal) and the number. Thus the decimal number 250 in a 12 bit field will be written as 12'h0FA or 12'b000011111010.

**Block** or **Cache Block** is used to refer to 32 bytes of data with a base address aligned to a 32 byte boundary (the low 5 address bits are zero).

**Line** or **Cache Line** is used to refer to the 32 bytes of memory in a cache that holds a cache block.

**Physical Addresses** are written as 40 bit hexadecimal numbers spaced with underbars (\_) for legibility. For example 01\_2345\_6789.

**Virtual Addresses** are only meaningful within a CPU. They are written as 64 bit hexadecimal numbers spaced with underbars (\_) for legibility. For example 0123\_4567\_89AB\_CDEF.

**UNPREDICTABLE** operations or behaviors can give arbitrary results, that may differ from device to device or as a function of time on the same device. However, the system will continue to operate and any section that has been placed in an UNPREDICTABLE state can be restored to deterministic operation under software control.

**UNDEFINED** operations or behaviors can result in any outcome from no change in the state of the system to creating an environment in which the system no longer continues to operate. Following an UNDEFINED operation assertion of the reset signal may be required to restore deterministic operation.

**Register Definition Tables** include the register name and physical address in their titles. The table shows all bits that are implemented. Registers are all allocated a 64 bit field, any bits that are not implemented will return UNPREDICTABLE data if read. The default values show the register value after a system reset. If the default is given as "ext" the value following reset depends on an external pin setting. If no defaults are given the field is UNPREDICTABLE after reset.

**Reserved** register bits and fields are not used in the current device, but may be used in future versions. They must be written with zeros to get the documented behavior and the behavior is UNDEFINED if they are written with any other value. Future parts may have well defined operation with these bits set.

**Not Implemented** register bits and fields are not used in the current device and do not have any implementation supporting them. Reads will return UNPREDICTABLE values, writes will have no effect. There is no need to write these bits (for example if the top 32 bits of a register are Not Implemented then a store word can be used to set the lower bits).

**Read Only** registers and bit fields provide status information should only be read by the CPUs. Writes to these fields are ignored.

**Write Only** registers and locations should be written with data, but cannot be read back. Unless otherwise specified, reads will return UNPREDICTABLE data.

**PERIPH\_REV3** refers to the third major revision of the peripherals. These are used in the BCM1125, BCM1125H (All steppings) and later production versions of the BCM1250 (stepping C0 and later, also called "Pass 3"). This manual describes both the PERIPH\_REV2 and PERIPH\_REV3 revisions and tags information that only applies to PERIPH\_REV3. The revision number in the **system\_revision** register can be used by software to determine the available features. Note that code written for the earlier parts will work on PERIPH\_REV3 parts (provided that it correctly observes Reserved fields). An earlier version of the manual should be used for the prototype parts marked BCM12500 which lack some of the peripheral functionality.

**Broadcom Use Only** registers and operations are intended for use by Broadcom in testing the device. In most cases this manual does not describe these options in sufficient detail for them to be safely used. Incorrect settings of these registers or inappropriate use of operations may cause the system to behave in UNDEFINED ways, or require the system to be power cycled to restore operation.

## Section 2: Signal Overview

### BCM1250 SIGNAL GROUPS

The signal pins of the BCM1250 can be divided into functional groups, primarily related to the peripheral to which they are attached. Hardware designers should refer to the BCM1250 Data Sheet for full pinout and timing details.

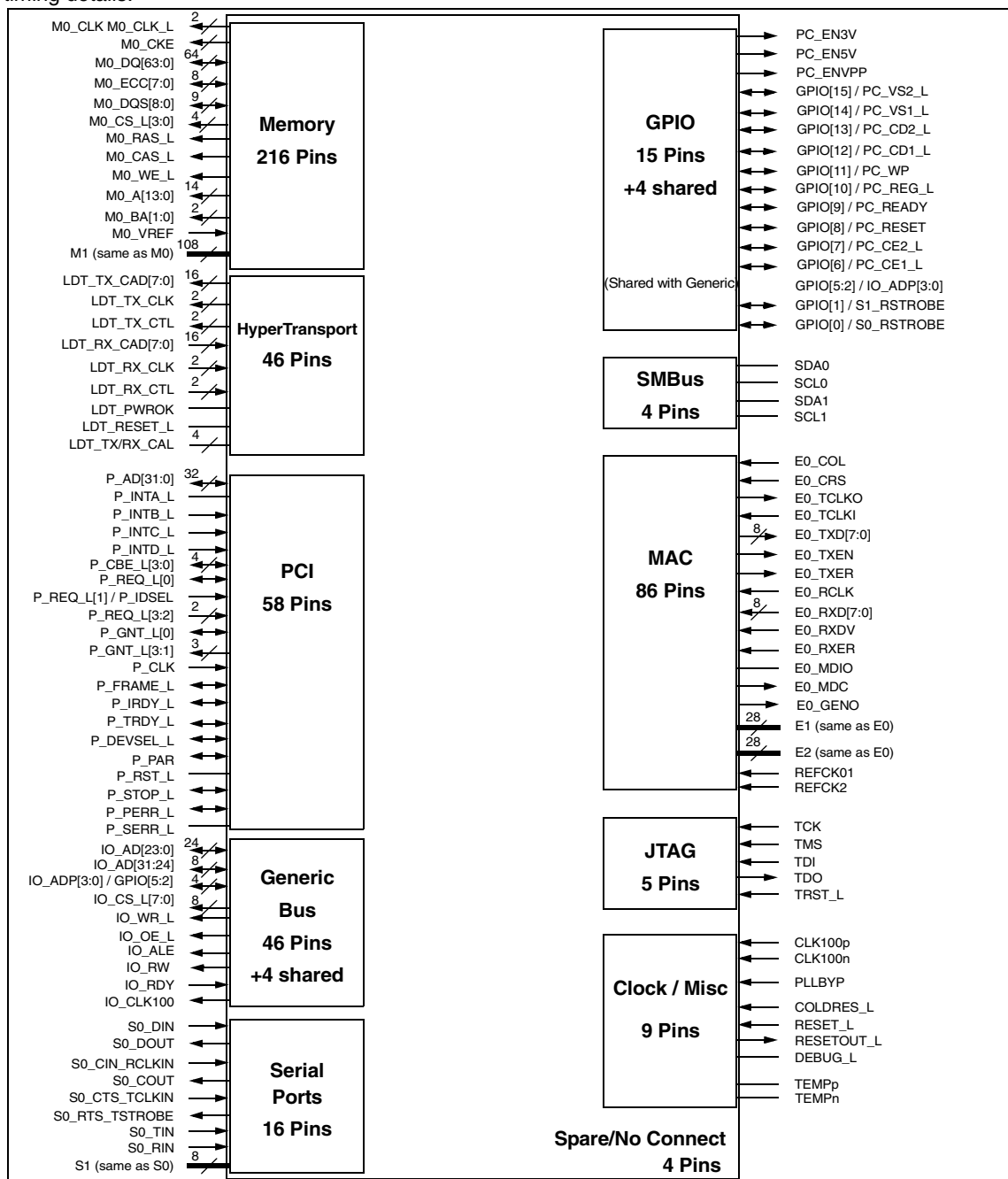


Figure 3: BCM1250 Signals

# BCM1125/H SIGNAL GROUPS

The signal pins of the BCM1125 and BCM1125H can be divided into functional groups, primarily related to the peripheral that they are attached to. Hardware designers should refer to the BCM1125/H Data Sheet for full pinout and timing details.

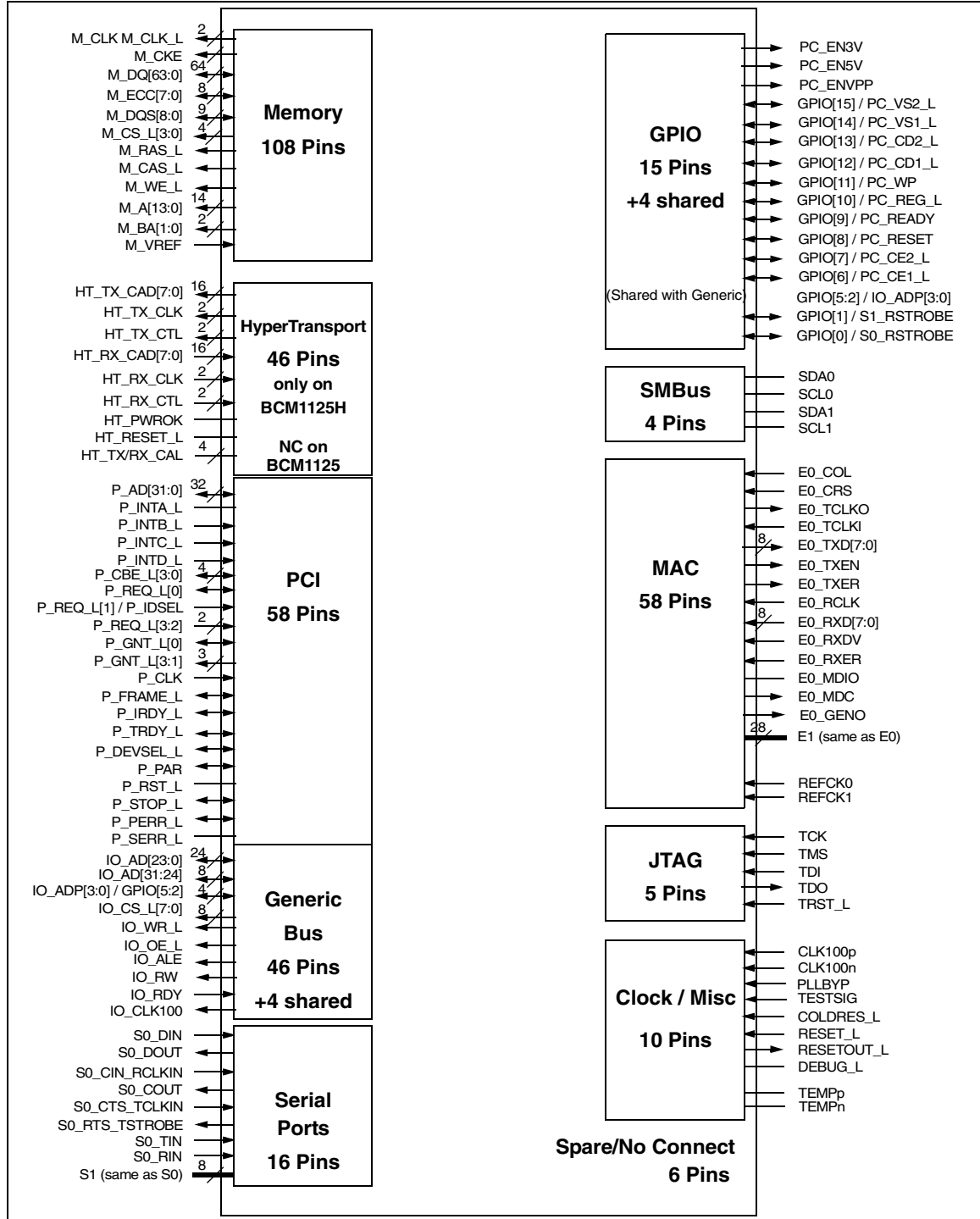


Figure 4: BCM1125/H Signals



## Section 3: System Overview

### INTRODUCTION

A logical block diagram of the BCM1250 and BCM1125/H family is shown in Figure 5. This figure does not exactly match the implementation details, but it gives a more useful model for programmers and system designers to use. The system is based around the ZBbus, a high speed split-transaction multiprocessor bus. It connects the CPU(s), the level 2 cache (L2), the memory controller, two I/O bridges and the System Control and Debug unit (SCD).

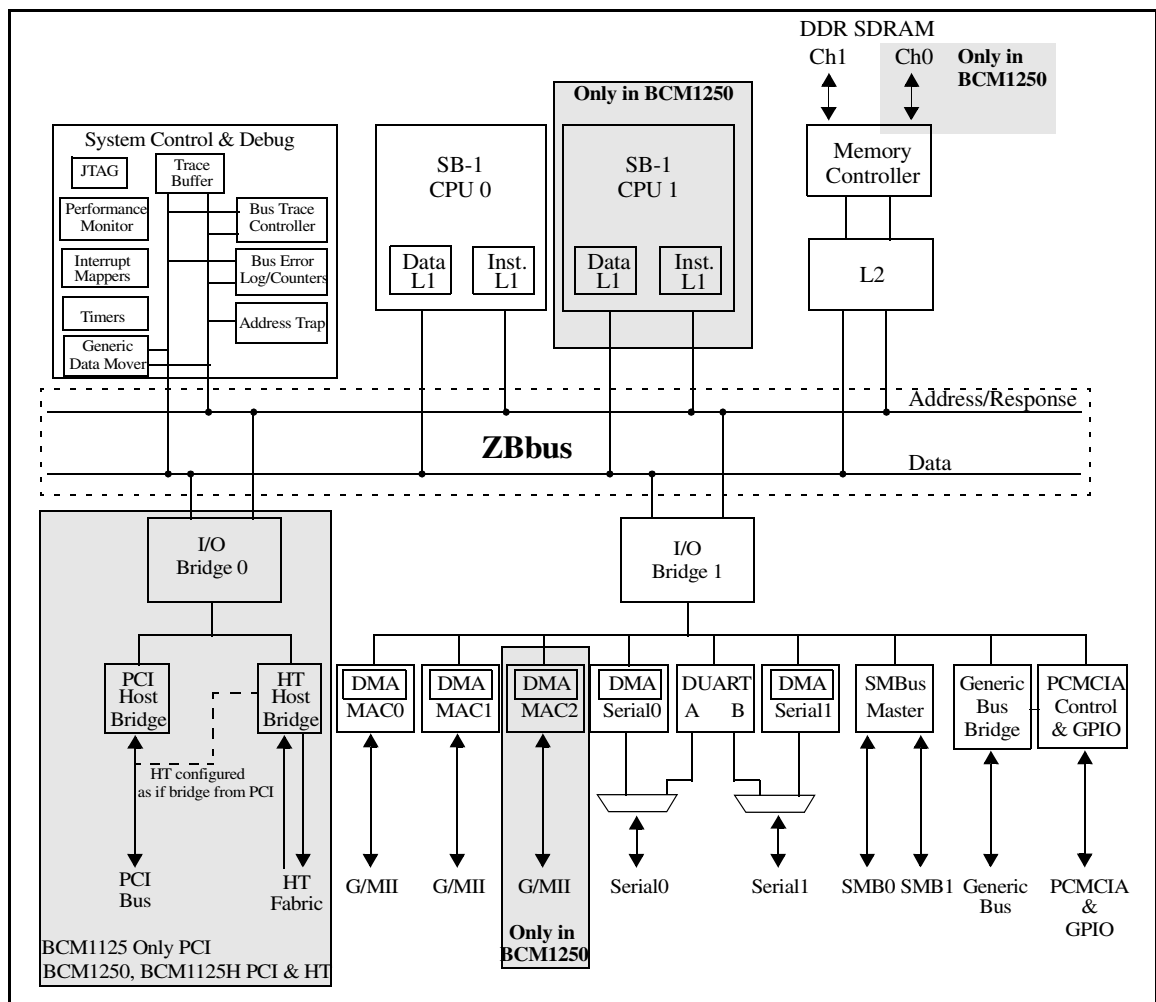


Figure 5: Logical Block Diagram of BCM1250 and BCM1125/H

This section gives an overview of the whole system. Each of the system elements are described in detail in later sections.

The ZBbus is the on-chip multiprocessor system bus:

- Data is 256 bits wide, running at half the CPU frequency. (For an 800 MHz part this gives  $(800/2)*256 = 102.4$  Gbit/s bus bandwidth).
- The address is 40 bits (matching the CPU physical address and the HyperTransport I/O fabric address).
- The separate address and data sections are arbitrated for independently, and operate as a split transaction bus.
- All transactions are tagged, data responses can come back in any order.
- The cache block ownership protocol (MESI) ensures memory coherence is maintained.
- Individual buffer-full indications from each bus target allow selective flow-control by requestors.

The processors are Broadcom SB-1 CPUs implementing the MIPS64 architecture. These are described in detail in the SB-1 User Manual. On the BCM1250 the two CPUs are identical in all respects apart from the processor number that will be read from the Processor Identification register (CP0 register 15). The reset logic in the System Control and Debug unit (SCD) is different for the two processors. Following a system reset only CPU 0 will be brought out of reset, it can then setup the system and release CPU 1 when ready. In the BCM1125/H parts the processor is CPU 0 and will report that it is a uniprocessor in the Processor Identification Register.

The Level 2 cache (L2) is organized slightly differently than L2 caches in other systems. It is shared by the processor(s) and any I/O DMA masters. It is best understood as a cache on the front of the memory (as shown in [Figure 5](#)), rather than by using the traditional model where the L2 is associated with a CPU. All memory accesses are checked in the L2. The bus includes a signal to indicate that the data should be allocated in the L2 cache on an L2 miss, this signal may be used by DMA masters to write data to the L2 so it can quickly be accessed by the CPU (this has to be done in a controlled way to avoid disrupting the normal gains of having a L2 cache).

The I/O bridges isolate the peripherals from the bus, and implement the bus and coherence protocols. They include buffering to allow multiple outstanding I/O requests, and support for DMA masters. The PCI and HyperTransport expansion buses share an I/O bridge, so any peer-to-peer traffic (between PCI and HyperTransport devices) is hidden from the ZBbus.

The part can be run as a big endian system or a little endian system. This is set by an input that is read at reset time. Internally the few data paths that need to will swap byte lanes, however the system is designed to have as few changes as possible. The CPU endian mode bit (the BE bit in the CP0 Config register) will always reflect the endian mode of the system. The interface to the PCI and HyperTransport expansion buses includes byte lane swappers that can be used if required to interface to these devices.

## INTERNAL REGISTERS

There are a large number of internal registers. Their definitions are given in the sections of this manual that describe their use. Section: "Internal Register Addresses by Function" on page 445 has a summary of all the register addresses. Each register is 1 byte, 2 bytes, 4 bytes or 8 bytes. However, even if a smaller size is implemented registers are always allocated in an aligned 8 byte (64 bit) field. Most registers can be read or written using a double-word (8 bytes), word (4 bytes), half-word (2 byte) or single byte access. However, some of the registers can only be written with their full width (in these registers a wider write will work, but a narrower write will leave the register value UNPREDICTABLE). All registers can be read with any size read, if the size requested is larger than the size that is implemented in the register then the extra bits will be filled with UNPREDICTABLE data (unless otherwise documented).

The address given in this manual for all registers is aligned to a double-word (i.e. the final hex digit is a 0 or an 8), reflecting the field size rather than the register size. As is illustrated in Figure 6 the base address may be used for all access widths in a little endian system.

In a big endian system, if the access to the registers is as a double-word the base address should be used, for a word access the address is the base address plus 4, for a half-word access use the base address plus 6 and for a byte access use the base address plus 7. For example, consider a register with 1 valid byte and base address 1234\_0000. Its double-word access address is 1234\_0000; word access address is 1234\_0004; half-word access address is 1234\_0006; and single byte access address is 1234\_0007. This fits with the big endian model for data position in double-word fields.

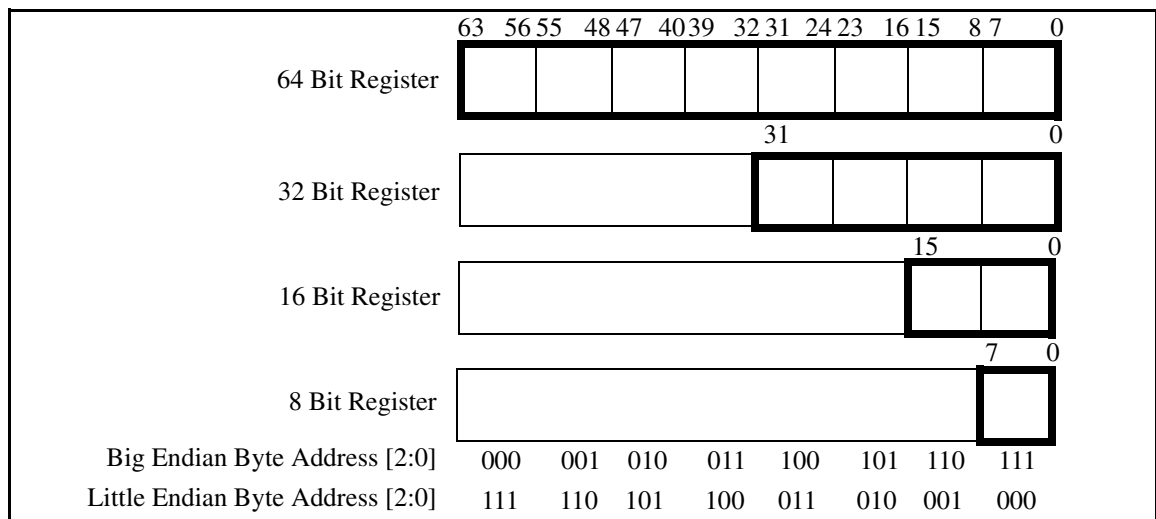


Figure 6: Internal Control and Status Register Alignment

The internal registers should be referenced by the CPUs in uncacheable space, so transactions will never be wider than 8 bytes (and will never span more than a single 8 byte aligned range).

One source for initialization errors happens when a bit in a register is defined to have an UNPREDICTABLE (1'bx) value after reset. It is possible that on all parts used in a small prototype build the bit has the same value, but on a few parts in a larger build it has the other value. Thus if it is not initialized the error may only be discovered late in the testing process.

When a register is marked as cleared by a read it will be cleared by any read to it. Care should therefore be taken to ensure that a compiler or debugger does not split an access to the register into multiple reads. For example if a 64-bit access is split into two 32-bit reads, the first of these reads will get valid data and clear the register and the second will get zeros (compilers will probably do this if they are configured to only use 32-bit registers). Similarly, the value of these registers cannot be read from a byte-by-byte memory dump. Most (but not all) of the registers with read side effects have an alias (with `_debug` appended to the register name) that does not have the side effect for a debugger to use.

## COHERENCE

The system maintains coherence for all memory operations including DMA. This coherence means that a read reference to memory space will return the most up to date version of the data. The L2 cache, processors, memory controller and I/O bridges all cooperate to deliver the correct version of each cache block. The coherency is managed by hardware, other than selecting the correct cachability mode no action is needed by software.

Coherence works at cache line granularity, for each cache block (aligned 32 byte block of memory) there is at all times an owner. The default owner of a block is either the L2 cache or main memory, they work together to service bus requests. Blocks that are being shared between bus agents are owned by the L2 or memory. Any agent that wishes to modify a cache line must become exclusive owner. When it makes the request the current owner of the block will give it up and any shared copies will be invalidated. Although the data transfer may occur at any time following the request, the ownership transfer is deterministic. If an agent receives ownership of a block it is possible that it will lose ownership before it has received the data; in this case it can perform one operation on the block before passing the data to the new owner, this is required to avoid live-lock when two agents are trying to write to the same block. Coherent memory references check L1 and L2 tags and the partial-line merge buffer in the I/O bridges at bus speed. If a block is exclusively owned, a request for that line can only be serviced by the owner, if the block is shared it can reside in more than one location, but will be returned from the L2 or memory (as the default owner). When block ownership is transferred (either from one exclusive owner to another, or from an exclusive owner to the default) the data transfer is snooped by the L2/memory so the new owner gets a clean copy of the line.

A line in the level 1 cache (in the processor) will be in one of six states: Invalid, Shared, Exclusive Clean, Exclusive Dirty, Non-coherent Clean or Non-coherent Dirty. Stores are permitted to all valid lines except for Shared ones, which must be upgraded to Exclusive before a store is possible. Dirty lines will be written back if the line is evicted to make way for a new fill, if evicted due to another device requesting the line (an intervention), or if explicitly written back by software (either a CACHE instruction forcing write-back, or a PREFetch instruction with the nudge hint).

Lines will only be put in the L1 cache Non-coherent if their TLB entry has the cacheable non-coherent cache attribute. These lines will not play a part in the ZBbus coherence protocol, and will need careful management by software. In general this should only be done for accessing memory that is genuinely non-coherent, which is true for any memory attached on the PCI bus or HyperTransport fabric. Memory attached on any of these interfaces will not be cached in the L2 cache. Memory attached on these is outside the coherency domain: they are I/O connections and do not carry coherency messages, so some other device could update the memory without the CPU being informed that its cached copy is now out of date.



Blocks with main memory addresses should be marked coherent in the CPU. If these blocks are not shared there is no difference in performance between marking them cacheable coherent and cacheable non-coherent, but there is a much higher chance of unexpected behavior if the block is marked non-coherent (for example due to false sharing or missing cache flushes before DMA operations). When these blocks are used for DMA operations the system performance will generally be better if the blocks are marked coherent and the coherence protocol is used to fetch modified data from the CPU cache when it is needed by the DMA engine. Having the blocks marked non-coherent and using CACHE operations to flush the cache prior to initiating DMAs will give a lower system performance as well as being prone to programming errors.

Lines in the L2 cache are in one of three states: Invalid, Clean or Dirty. All cacheable accesses to the address range controlled by the memory controller check the L2 cache.

The coherence mechanism will be circumvented by doing uncached or cacheable non-coherent accesses into memory that has previously been cacheable and coherent. If this is done the results are UNPREDICTABLE. If areas of memory are never cacheable and coherent then it is the responsibility of software to track ownership. The coherence mechanism can also be circumvented if software uses the Invalidate CACHE operation on a block that is held exclusive in the L1 cache. If a snoop request is received on the bus at the same time this instruction executes, the snoop will hit on the line in the L1 but the line is invalidated before it can be evicted. The CPU will respond by returning UNPREDICTABLE data marked with a fatal bus error. There is no problem with using the Writeback&Invalidate CACHE operation.

The generic bus section of memory may be mapped cacheable coherent. The I/O bridge will act as default owner in the MESI protocol (it behaves as the L2/Memory does for memory addresses). The CPUs are able to take exclusive ownership of cache blocks from memory on the generic bus, if ownership is transferred the new owner will acquire the block clean and if needed a writeback will be done to the generic bus. This allows the cache attribute for kseg0 to be set to cacheable coherent to cover both main memory and the Boot ROM and allows for RAMs on the generic bus with no special software management.

The PCI/HyperTransport space should not be mapped cacheable coherent. The I/O bridge will respect an ownership assertion by one of the CPUs (so there will not be two replies to a read request) but it will not act as default owner and will not copy back dirty data on ownership changes. Changes to data may therefore be lost leading to UNPREDICTABLE behavior. In error cases the behavior of cacheable coherent accesses through I/O Bridge 0 is UNDEFINED.

The I/O bridges provide the entry point into the coherent domain for any DMA traffic (from the on-chip network interfaces or PCI/HyperTransport master devices). The bridges will check the address being accessed. For any memory address cacheable coherent accesses will be used, and partial line writes will cause a read (exclusive)-modify-write cycle. For any address that is not to memory uncacheable accesses are done and partial line accesses will result in only some of the byte enables being set.



## ORDERING RULES AND DEVICE DRIVERS

The interaction between the ordering rules imposed by the SB-1 CPU, the ZBbus and the peripheral agents simplifies device programming in most situations. In these situations it is the ordering between cacheable coherent memory and uncacheable peripheral registers that is important.

The five important rules for the SB-1 are:

- 1 Cacheable coherent stores are visible in program order.
- 2 Uncached loads and stores will issue to the ZBbus in program order, and will only issue when all earlier cacheable coherent stores are visible.
- 3 The CPU will stall until data returns when it needs the result of an uncacheable load (or a cacheable load that has missed in the cache).
- 4 Uncached stores will not issue to the ZBbus if there are any uncached loads outstanding.
- 5 The SYNC instruction will prevent other instructions from issuing until all outstanding cacheable or uncacheable loads are satisfied, all cacheable stores are visible, all uncacheable stores have been sent on the ZBbus, and the write buffer has drained.

The internal I/O bridges and peripheral devices have a simple rule:

- 6 In all queues transactions will remain in the order of ZBbus Address phase.

The PCI and HyperTransport expansion busses follow the PCI ordering rules (Appendix E of the PCI Specification revision 2.2). Of particular importance is the rule:

- 7 Posted Writes can pass Delayed (Non-Posted) Reads.

Rule (1) allows the programming style where a processor writes some data and then sets a flag to indicate it is done. Another processor polling the flag will always see the new value of the data if it sees the new value of the flag. Rule (2) allows the "flag" to be a peripheral register: memory data (such as DMA descriptors or data buffers in cacheable coherent space) can be updated and then a control register (in uncached space) written (for example to start the DMA) without the need for any special intermediate operations.

Note that there is no rule in the other direction; a cacheable coherent store may be visible before an uncacheable load or store that precedes it in program order. If this ordering is required it can be enforced by either using a SYNC instruction (rule 5) or performing and using the result of an uncacheable load (by rule 2 and 3). This is important if the processor is using the PCI producer-consumer model and is setting a flag in memory to indicate that it has performed a write to a device. When this model is used the uncacheable write to PCI or HyperTransport space must be flushed from the CPU before the flag is written in cacheable space. The I/O Bridge will prevent the read-data-return from the polling request from passing an uncacheable write, so a SYNC will ensure the uncached write is queued in the bridge before the flag changes.

Rule (6) allows there to be multiple outstanding uncached operations. For internal peripherals and the generic bus all requests to the same peripheral will be seen in program order. Note that there are no such guarantees between different peripherals (which may be on different ZBbus agents, equivalent to being on different PCI busses), but this order only matters if there are back-channels between the devices (which will break any ordering rules).



Rule (7) is a potential problem, since writes could pass earlier reads to the same device (and thus the read could see the state after the write completes). However, rule (4) ensures that this situation never arises for code running on the SB-1. Other than blocking this write following read problem, rule (4) does allow multiple outstanding uncacheable accesses which will reach the peripherals in order. A series of loads performed to a FIFO will give the expected results, and by having multiple loads in flight the FIFO can be driven at full speed.

Cacheable non-coherent requests have the same timing as cacheable coherent requests as far as the CPU L1 data cache is concerned. They will be written to the data cache in program order, and the order with respect to uncached operations will be as described above. However, since these requests are outside the coherence domain it is unknown when writes will be visible to the rest of the system (i.e. flushed to L2 or memory and not hidden behind a different cached non-coherent copy of the block).

Locks can be implemented using the load linked (ll) and store conditional (sc) instructions. These have additional rules:

- 8 Load linked will not issue speculatively, it is held at the issue point until all preceding instructions have graduated.
- 9 Store Conditional includes a partial SYNC operation. No instructions will be issued from the time the store conditional is issued until it graduates.

In most cases no extra SYNC instructions are needed when acquiring a lock. The load linked is used to check if the lock is free and the store conditional can be used to claim it. If the load indicates that the lock is in use or the store conditional fails the code should spin waiting for the lock (or the process can be blocked, depending on the situation). If the load indicates the lock is free and the store conditional succeeds then the lock has been acquired.

There is a potential problem when freeing a lock. A normal store is used to release the lock. However, a SYNC may be required before the store to force completion of cacheable loads before the lock is released. This will only be the case if loads are performed while the lock is held, but the use of the data is delayed until the lock has been released. Consider the case where two loads are done and one hits in the L1 cache but the other misses in the L1 cache and is not used until after the lock is released. The CPU will not stall and (if there is no SYNC) can therefore execute the write to release the lock. There is a (very small) chance that the other CPU can claim the lock (by getting the line exclusive from the first CPU and having the load linked/store conditional succeed) and modify the data before the load from the first CPU gets the data. Thus the first CPU will not see the data that was present when it held the lock, which is probably an error. This can be solved using the SYNC, or ensuring there is a use of the load before the lock is released (rule 3).

## CPU SPECULATIVE EXECUTION

The SB-1 CPU can speculatively execute cacheable loads (i.e. a read is issued to the ZBbus to get the data before the load is guaranteed to graduate), this only happens if the load is canceled because of an exception on an earlier instruction (branches are always resolved before the load is committed). This is normally not a problem since cacheable reads do not have side effects. However, the MIPS kernel address map includes areas where the cacheability attribute is supplied by the address rather than the TLB (kseg0 and xkphys). If a register contains one of these addresses that forces the cacheable attribute then a speculative load may be made to a peripheral address. To protect against this the internal peripherals check the cacheability of reads, and will prevent cacheable reads from having side effects. The generic bus has a mechanism to block cacheable reads which can be enabled for external peripherals that have read side effects (but should normally be disabled for memories). The only space where speculation can cause a problem is therefore the xkphys aliases for PCI and HyperTransport peripherals.

In the absence of program error (i.e. the register just contains the wrong value) there are two reasonable instruction sequences that could cause this behavior if at the start of the sequence the register happens to hold a kseg0 or xkphys cacheable alias of a peripheral. In both cases an exception is used to fix the address for the load.

```
; at this point r1 contains junk
syscall 73      ; returns address in r1
ld      r2, 0(r1)

; This code intends to check r1 before the load
tle     r1, r2  ; trap if r1 is out of range
```

These two sequences can be fixed by adding two superscalar no-ops (`ssnop`) instructions before the loads to ensure the load is not speculatively executed before the exception triggers.

A similar situation arises if the `ld` is replaced with an indirect jump (`jr r1`), which can put a junk value into the jump register cache and cause a subsequent speculative instruction fetch to issue a cacheable read from a peripheral address. Again inserting two `ssnop` instructions will prevent the problem.

The CPU (`break`) instruction has a special issue rule, so does not suffer from this problem. It is safe for a debugger to use (`break`) to insert breakpoints in the normal way.

## ERROR CONDITIONS

The system makes every effort to ensure that processing is never done based on erroneous data. Error conditions are signalled in several ways, as well as being passed in a flag with the data. All the critical structures where data corruption could occur are protected with ECC (allowing for correction of single bit errors) or parity (in structures that never contain the only copy of the data). As a result of an error condition being detected on data destined for the CPU, it will receive an exception (as required by the MIPS architecture) and will be signalled interrupts from the source that detected the problem or the Bus Watcher (see [Section: "Bus Watcher" on page 64](#)) or from both source and Bus Watcher. If the error is detected on data destined for a DMA engine, the DMA channel will stop and raise an interrupt to the CPU and the Bus Watcher will report the error. (Note that reading the Bus Watcher status register will clear it and enable logging of future errors, so it should be the last register read when the Bus Watcher information is dumped.)

## CACHE ERROR EXCEPTIONS

The cache error exception is raised on the CPU both by error conditions detected in the internal caches and by error returns that are flagged on the bus as having data errors (but valid addresses). This follows the MIPS Architecture use of the exception. The practical upshot of this is that code diagnosing the exception should examine the state of external error reporting registers in addition to the ones in the CPU. Note that when data is returned to the data cache marked with an error code it will be written to the cache with an uncorrectable ECC error (after ECC is calculated the bottom two bits of each double-word are inverted to force the error) to ensure that it is not used on this processor and that the error is preserved if the line is evicted or snooped out of the cache by another CPU or DMA engine.

The error registers within the CPU are in the CP0 set and are described in the "Error Reporting Registers" part of Section 9 of the SB-1 User Manual. The error control register **ErrCtl** (register 26, select 0) indicates which cache suffered the error (or was being filled if it is an error signalled over the bus). It also flags recoverable errors from the data cache, these have been corrected and the exception can immediately return unless it needs to gather error statistics. Depending on which cache saw the error either **CacheErr-I** (register 27, select 0) and **EPC** or the **CacheErr-D** (register 27, select 1) and **CacheErr-DPA** (register 27, select 3) contain additional information. The external registers are in the Bus Watcher, the memory error counter (**bus\_mem\_io\_errors**) and L2 error counter (**bus\_l2\_errors**) indicate how many errors have happened and the **bus\_err\_status** register will indicate the type of error and participants involved. Because instructions may be issuing and probing the cache at the same time an external error is found on a fill it is possible that an internal error is raised before the external one can be signalled, therefore it is normally useful for debugging if the error reporting code always dumps the Bus Watcher errors even if it appears to be an internal problem.

## BUS ERROR EXCEPTIONS

The bus error exception is raised on the CPU by external errors that are not reporting data corruption. The main cause is when an access is detected to a non-existent address. Cacheable accesses that result in a bus error will be put in the cache with uncorrectable ECC errors (as described for the cache error exception). As with cache errors, both CPU and Bus Watcher registers are useful.

The error registers within the CPU are in the CP0 set and are described in the “Error Reporting Registers” part of Section 9 of the SB-1 User Manual. The **BusErr-DPA** (register 26, select 1) contains the address of the bus error (or first detected bus error) and the error control register **ErrCtl** (register 26, select 0) indicates if multiple bus errors have been seen. In the Bus Watcher the bus error counter (**bus\_mem\_io\_errors**) indicate how many errors happened and the **bus\_err\_status** register will indicate the type of error and participants involved. If the bus error is generated from the generic bus (address in the range 00\_1009\_0000 - 00\_3FFF\_FFFF responder id=Br1) then the **io\_interrupt\_status** register contains more information on the cause of the bus error. Again, error reporting code can usefully dump all these registers.

There are a couple of interesting corner cases for bus errors. The first occurs when code is executed out of some physical address and at a later time that physical address is no longer valid (this is only likely to happen for unmapped addresses such as kseg0 and xkphys, since the software would presumably have removed any TLB entries that point to invalid physical addresses). For example the generic bus chip select or memory controller address parameters may have been changed. If any branches were executed while the address was valid the CPU will store the branch target information in its branch prediction structures. At a later time it may predict that a branch destination is at the now invalid address and issue a speculative instruction fetch. In this case the CPU will *not* take a bus error exception (because it will discover the speculation was incorrect), but an interrupt may come in from the peripheral indicating a bad access was made and the Bus Watcher will record and report that the error passed over the bus (the transaction ID recorded by the Bus Watcher will help indicate this has happened, see [Table 51: “Decode of some TIDs for system revision PERIPH\\_REV3,”](#) on page 81 and [Section: “Overview of the ZBbus Protocol”](#) on page 20).

The second case arises because the bus error exception is imprecise. It can be reported on any instruction from the load that causes the bus error until a few instructions after the use of the register that was the target of the load. In most cases the bus error is a programming error and causes the process to be terminated or the system to enter error recovery. However, in systems with devices that can be hot-swapped the first indication of device removal may be a time out or target abort during an access to the device. In this case the imprecise reporting of the bus error may make it hard to determine that the exception handler should signal the device driver to clean up its state and resume operation (`eret`). One approach to solving this is to have a `sync` instruction after each load to the hot-swappable device which will force the bus error to be reported on the `sync` instruction. Alternatively, if two `ssnop` instructions are put after the instruction that uses the target of the load then the bus error will be signalled on one of the instructions between the load and the second `ssnop`.

---

## CPU TO CPU COMMUNICATION (BCM1250 ONLY)

The two CPUs in a BCM1250 can communicate and share information in several ways. The MIPS Load Linked (ll and lld) and store conditional (sc and scd) instructions can be used to implement atomic operations. When there is genuine sharing, the L1 cache to L1 cache latency is 28-36 CPU cycles.

Inter-processor interrupts are possible in several ways using a per-CPU mailbox register. This is described in more detail in [Section: "Mailbox Registers" on page 46](#). The generating CPU will do a write across the ZBbus to the mailbox, which directly raises the receiving CPUs interrupt line. The total latency for this is 20-30 CPU cycles plus any context switch time needed. 7-12 of these cycles are taken between the receiving CPUs interrupt being raised and its fetching from the exception vector. The interrupt is attached as an exception to a convenient instruction: if the pipeline is running there will immediately be an instruction to use so the shorter time will apply, the longer time is taken when the pipeline is stalled and the instruction that will carry the exception has to be generated.

## EXTERNAL INTERRUPTS

External interrupts from the PCI inputs are synchronized into the internal clock and pass through the interrupt mapper (see [Section: "Interrupts" on page 47](#)) before raising the CPU interrupt input. Including the 7-12 cycles in the CPU, there is a latency of 17-22 CPU cycles plus about 2ns from the external pin being asserted to the CPU fetching the first instruction from the exception vector. (Note that this is a guideline only, these times are not specified or tested and may vary between parts.)

External interrupts from the GPIO pins are synchronized and filtered to remove glitches before being passed through the interrupt mapper and to the CPU. When configured for the 60ns glitch filter there is a total latency of 20-25 CPU cycles plus about 112ns from the external pin being asserted and the CPU fetching the first instruction of the exception vector. (Note that this is a guideline only, these times are not specified or tested and may vary between parts.)

## OVERVIEW OF THE ZBBUS PROTOCOL

This section gives an overview of the ZBbus protocol. It contains the details needed to do system and performance debugging using the bus trace unit, but is not a full specification for the protocol. The bus is entirely internal to the part, and it connects the main blocks of the system. Each block that directly connects to the ZBbus is called an agent. [Figure 5](#) shows the agents and [Table 1](#) lists their four bit bus ID number. The bus runs at half the CPU clock speed and can transfer a new request and 256 data bits every cycle.

**Table 1: ZBbus Agent IDs**

<i>Agent</i>	<i>ID</i>	<i>Description</i>
CPU0	0	SB-1 CPU 0.
CPU1	1	SB-1 CPU 1. (Only in BCM1250)
IOB0	2	I/O Bridge 0, connects PCI and HyperTransport interfaces to the ZBbus.
IOB1	3	I/O Bridge 1, connects the MACs and slow speed peripheral interfaces to the ZBbus.
SCD	4	System Control and Debug Unit.
	5	Reserved
L2C	6	L2 Cache.
MC	7	Memory Controller.

The bus is split into an address section and a data section. These are arbitrated for separately and run independently. For a given transaction, use of the data bus always follows use of the address bus, but there is no other ordering imposed. Many transactions can be in progress at a given time. In theory each agent could have 64 outstanding operations on the bus, In practice, the agents are limited by their internal transaction tracking buffers to fewer than 64. [Table 2](#) lists the signals in each section of the bus.

**Table 2: ZBbus Signals**

<b>Address/Control Section</b>	
A_AD[39:5]	Address of cache block moved by the request. The byte enables complete the address.
A_BYEN[31:0]	Byte enables. A_BYEN[31] corresponds to D_DA[255:248], down to D_BYEN[0] which corresponds to D_DA[7:0]. (See <a href="#">Table 5</a> )
A_ID[9:0]	Transaction ID: [9:6] are requester agent ID (see <a href="#">Table 1</a> ). [5:0] are unique for that requester.
A_CMD[2:0]	Command (see <a href="#">Table 3</a> ).
A_L1CA[1:0]	L1 (base) cache attribute (see <a href="#">Table 4</a> ).
A_L2CA	Request that the L2 cache allocate on miss.
R_SHD[5:0]	Response indicating block is shared. Each agent drives the bit corresponding to its ID.
R_EXC[5:0]	Response indicating block is exclusive. Each agent drives the bit corresponding to its ID.
R_L2HIT	Response indicating block is in the L2 cache. This signal is only used following a read command, but is also valid for most writes. This signal may be incorrect following writes to the L2 registers, or after an uncorrectable tag ECC error.





**Table 2: ZBbus Signals (Cont.)**

Data Section	
D_DA[255:0]	Data. There is valid data only on the byte lanes that had byte enables set (on A_BYEN) in the A-phase. The data on the other byte lanes is UNPREDICTABLE.
D_ID[9:0]	Transaction ID (copied from associated A_ID).
D_CODE[2:0]	Data status/error code (see <a href="#">Table 6</a> ).
D_RSP[3:0]	Agent ID of the agent driving the data bus (see <a href="#">Table 1</a> ).
D_MOD	Set to indicate the data is modified (dirty).

Each transaction is marked with a Transaction ID (TID), a 10 bit number constructed by the requester. The top four bits of the TID are the requester's agent number (given in [Table 1](#)), and the lower six bits are chosen by the requester to be unique for all transactions it has in progress (See also [Section: "Magic Decoder Ring For Using The Trace Buffer" on page 81](#)).

The bus supports coherent and non-coherent transactions. The bulk of the protocol is used to track ownership of coherent blocks and implement the standard MESI coherent states:

- **M** - modified. This block is dirty (has been updated, and is different from memory). Modified is a subset of exclusive, an agent must be exclusive owner before changing the data.
- **E** - exclusive. The agent that has this data is the only agent that has it. This agent is permitted to modify the block.
- **S** - shared. The line is present in multiple agents, but is clean (memory or the L2 cache has the most current update). No agent may modify the block.
- **I** - invalid. The block is not present in a particular agent.

A transaction on the bus involves three phases and two arbitrations.

- 1 The requesting agent arbitrates for the address bus. The bus will be granted in a fair manner.
- 2 The address phase (A-phase) starts the transaction.
- 3 The response phase (R-phase) gathers the coherency state for the transaction and thus determines which agent is responsible for providing the data. Ownership transfers at the end of the R-phase.
- 4 The agent providing the data arbitrates for the data bus.
- 5 The data phase (D-phase) transfers the associated data and ends the transaction.

Normally an agent only arbitrates for the bus when it is prepared to make a transaction. However, occasionally agents (particularly the CPUs) will speculatively request the bus and then discover the transaction has been canceled. In this case the agent will still use the bus cycle, but issues a NOP command. An A-phase NOP will also be caused if the previous cycle on the bus was a transaction for the same address or if the destination agent blocks transactions as the source agent is granted the bus.

## ARBITRATION

The ZBbus arbitration uses a fair history based system. Requests are granted based on the last time the agent was granted the bus. If all agents request every cycle (this never happens because the bus bandwidth is much higher than the I/O device bandwidths) this will become a simple round robin scheme. In practice most agents request the bus ahead of needing it thus hiding the arbitration cycle.

## ADDRESS PHASE

In the address phase the requesting agent puts the TID, address, byte-enables, command and cache attributes onto the address bus. All agents will examine the request and respond by setting their R\_SHD and R\_EXC bits in the R-phase.

The address has cache block granularity, and is therefore bits [39:5] of the physical address being accessed. Thirty-two byte enables indicate which bytes within the block are to be transferred.

The bus commands are in [Table 3](#). Uncached and non-coherent accesses use the same commands, but will be responded to by the agent that owns the address. The default owner of a block is the L2 or memory controller for all addresses in the memory address space, the generic bus is the default owner for addresses in the generic bus range.

**Table 3: ZBbus Commands**

A_CMD[2:0]	Command	Action	Data Bus
READ_SHD 000	Read Shared	The block is provided by the current owner and becomes shared. If the current owner is L2/memory and no other agent has a shared copy then the acquiring agent may take exclusive ownership (with no other action).	Data will be supplied by the exclusive owner if one is identified in the response phase, otherwise the default owner will supply it.
READ_EXC 001	Read Exclusive	The block is provided by the current owner and becomes exclusively owned by the acquiring agent. All other copies of the block are invalidated.	Data will be supplied by the exclusive owner if one is identified in the response phase, otherwise the default owner will supply it.
WRITE 010	Write	The block is written back to L2/memory.	Data will be supplied by the requester.
WRITE_INV 011	Write Invalidate	The requesting agent acquires ownership of the block and invalidates all copies (even if they are dirty), it then releases ownership and writes the block back to L2/memory. This is an optimization that avoids the need for a READ_EXC and transfer of data that will be discarded, it is used when the requesting agent is overwriting a complete block with new data (for example when a packet is DMAed from the network).	Data will be supplied by the requester.
INVALIDATE 100	Invalidate Block	The requesting agent acquires ownership of the block and all copies are invalidated (even if they are dirty). It is used when an agent upgrades a shared line to exclusive (in which case none of the copies will be dirty), or if the agent needs to become exclusive owner and guarantees to overwrite the complete line.	There is no data phase for this command.
101	Reserved		
110	Reserved		
NOP 111	No Operation	This command is used when the agent arbitrated for the bus and was granted, but is unable to complete the transaction. This will happen occasionally if the access is blocked by a buffer becoming full between the time the agent requested the bus and the address phase. It can also happen as a result of an exception in the requesting agent.	There is no data phase for this command.



Two fields describe the cache attributes of a transfer. The first, shown in [Table 4](#), describes the base (Level 1 cache) attribute. This must be set consistently by all agents using a block or all accesses to the block become UNPREDICTABLE.

**Table 4: ZBbus Level 1 Cache Attributes**

A_L1CA[1:0]	Attribute
00	Cacheable non-coherent. The block will be cached but will not play a part in the coherence protocol. Any agent with a copy of the block is free to write it. Software must manage the coherence.
01	Cacheable coherent. The block will be cached and uses the full coherence protocol. During usual operation all blocks with main memory addresses should use this attribute.
10	Uncacheable. The data will not be cached, and will not play a part in the coherence protocol. (Although not required, this attribute is normally used for uncacheable data that consists of 8 bytes or less.)
11	Uncacheable. The data will not be cached and will not play a part in the coherence protocol. (Although not required, this attribute is normally used for uncacheable data that has been merged in a buffer and may therefore be any number of bytes.)

The second attribute is only used for cacheable transactions that have an address in the memory controller range. The A\_L2CA (Level 2 Cache Allocate) signal indicates that the block should be allocated in the L2 cache if it misses. Following the D-phase of a transaction with this attribute the data will always be in the L2 cache. Note that this attribute only controls the behavior on a cache miss: all cacheable accesses to the memory controller range are checked in the L2 cache, reads will always be serviced by the L2 if they hit and writes will always update the L2 if they hit.

## RESPONSE PHASE

The response phase determines which agent will supply the data for a read command and is the point of ownership transfer for a block. Following the R-phase the new owner is responsible for the block (even if it does not yet have the data).

Each agent has R\_EXC and R\_SHD status signals. It asserts the R\_EXC signal to indicate it has exclusive ownership of the block and that it will provide read data. It asserts the R\_SHD signal to indicate it has a shared copy of the block. The memory controller and L2 cache use these to determine if they must supply or accept the data. The requesting agent will examine the status after doing a Read (shared) command, if no other agent has a shared copy of the data then the requesting agent is permitted (but not required) to take exclusive ownership of the block as if it had issued a Read Exclusive command.

An agent can assert both R\_EXC and R\_SHD to indicate an error. This is done if there is a parity or ECC error in the tags and the agent is unable to determine if it has ownership. If an exclusive owner can be identified then it supplies the block as normal. If there is no other exclusive owner then the memory controller will respond with UNPREDICTABLE data marked with a Fatal Error.

The L2 cache will assert the R\_L2HIT to indicate to the memory controller that it will act as the default owner for the block. The default owner supplies data for a read if there is no exclusive owner, and captures the data on a write or ownership transfer.



Table 6: ZBbus Data Status Codes (Cont.)

<i>D_CODE[2:0]</i>	<i>Status of Data on D_DA</i>	<i>Sources of Error</i>
100	Bus error. A CPU will take a bus error exception if it receives this error.	I/O bridge0: PCI parity error, master or target abort. HyperTransport NxA or error return. I/O bridge1: Generic bus error (no chip select for the address, time-out during an access or I/O bus parity error). Memory Controller: No chip select decoded for the address. SCD: The SCD will return this error with UNPREDICTABLE data if the bus watcher detects an illegal address.
101	Fatal bus error. The ownership of the block is unclear. This can be caused by a CPU tag parity error, or because software used a CACHEOP to invalidate an exclusive line and the CPU had committed to supply the line in the window while the operation executed. A CPU will take a bus error exception if it receives this error.	Memory Controller: The memory controller returns UNPREDICTABLE data with this error code if an agent asserts both R_SHD and R_EXC. I/O bridge1: This error code is returned with UNPREDICTABLE data and the io_coh_err bit will be set if some agent asserts both R_SHD and R_EXC during a coherent generic bus access.
110	Uncorrectable ECC error in the tag. A CPU will take a cache error exception if it receives this error.	CPU: Tag parity error. L2 Cache: Tag ECC error.
111	Uncorrectable ECC error in the data (L2 cache or memory controller). A CPU will take a cache error exception if it receives this error.	CPU: Data cache had uncorrectable data ECC error. Memory Controller: Data ECC error. L2 Cache: Data ECC error I/O bridge: Write of a Read-modify-write operation that received an uncorrectable data error during the read.

If an error is signalled during a data transfer the system takes care to maintain the error condition. If bad data is sent to the L2 cache or memory controller an uncorrectable ECC pattern is written with the data, so that subsequent reads will also see an error and bad data will not be used in processing. This can lead to a cascade of errors during the period between the initial detection of an error and software recovery code running (the SCD error log will record the first problem).

The D\_MOD signal is asserted if the data being transferred is dirty (i.e. differs from L2/memory). In this case the L2 cache or memory controller will take a copy of the data, so that the new owner receives it clean.

The memory controller and L2 cache work together to supply data for any of the memory regions of the address space. Figure 7 shows the decision process that is used on reads and writes to the memory region to determine which agent supplies the data and which accepts write data. It also shows which of the memory or L2 will write back data when dirty data is supplied by an exclusive owner.

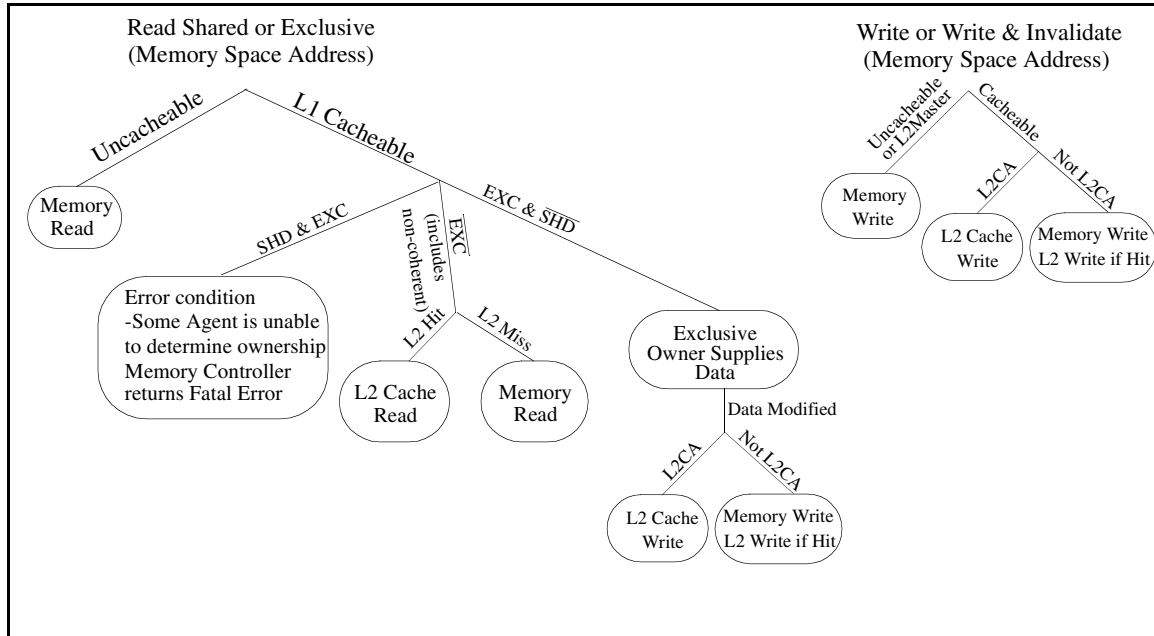


Figure 7: Decision Tree for Memory Space Address Accesses

## RESET

The COLDRES\_L input is used for power-on reset of the device. It must be held low until the power supplies are within the operating range and the reference clock is stable. The cold reset will read the PLL multiplier bits and start the PLL, an internally generated delay allows the PLL to lock before the system is started. At the end of the cold reset delay reset time configuration information is read from the generic bus IO\_AD lines (see below). The RESET\_L input is used to cause a warm reset of the device, this resets all the internal logic but does not restart the PLL, read the configuration information or wait the cold reset delay (RESET\_L need not be asserted at power-on since the cold reset delay has higher precedence). The end of the cold reset sequence or the deassertion of RESET\_L signal starts the internal reset sequence to establish the internal state, the SCD will then release the system and CPU0. The device will source the reset signal for the board (RESETOUT\_L), the PCI bus (P\_RST\_L) and the HyperTransport fabric (LDT\_RESET\_L and LDT\_PWROK if the link needs a cold reset). These are all driven during the internal reset period and other than P\_RST\_L can be asserted separately under software control.

Reset can also be initiated by software or time-out of the watchdog timers. These can be a soft reset which restarts the part but is not signalled externally or a system reset which will assert RESETOUT\_L. The software initiated system reset will re-sample the configuration information (except for the PLL ratio), the watchdog system reset will not.



Table 7 summarizes the reset options.

**Table 7: Operation of Different Reset Sources**

	<i>COLDRES_L</i> <i>pin asserted</i>	<i>Software</i> <i>System Reset</i> <i>system_cfg[60]</i>	<i>Watchdog</i> <i>2nd time-out</i> <i>Type=system</i>	<i>RESET_L</i> <i>pin</i> <i>asserted</i>	<i>Software</i> <i>Soft Reset</i> <i>system_cfg[58]</i>	<i>Watchdog</i> <i>2nd Time-</i> <i>out</i> <i>Type=sbssoft</i>
Sample pll_div bits Restart PLL Clear system_cfg[63]	Y	-	-	-	-	-
Sample configuration bits (other than pll_div)	Y	Y	-	-	-	-
Assert RESETOUT_L	Y	Y	Y	Y	-	-
Reset ZBbus	Y	Y	Y	Y	Y	Y
Reset Agents	Y	Y	Y	Y	Y	Y

During reset pull-up and pull-down resistors on the generic Address/Data bus are used to set static system options. These pins have weak internal pull-up or pull-down resistors (as described in the Hardware Data Sheet), external resistors are required to set the other state. These options change the behavior of pins that need to be active at startup time to ensure proper operation.

**Table 8: Static Configuration Options**

<i>IO_AD</i> <i>Bit</i>	<i>Name</i>	<i>Pulled Up to 3.3V</i>	<i>Pulled Down</i>	<i>Section</i>
0	Reserved	Reserved	Normal Operation.	N/A
1	clk100_src	The internal 100MHz clock and IO_CLK100 come directly from the CLK100_p reference. The IO_CLK100 will have a duty cycle only a little worse than the reference clock.	The internal 100MHz clock and IO_CLK100 are generated by dividing down the output of the PLL that feeds the CPU clock. The IO_CLK100 will match the internal clocking and will run at a few MHz (typically less than 10MHz) during COLDRES_L.	<a href="#">Section: "Clocks" on page 31</a>
2	ldt_minrstcnt	Broadcom Use Only. Enable HyperTransport reset test mode.	Normal Operation.	N/A
3	ldt_bypass_pll	Broadcom Use Only. Bypass the HyperTransport PLL.	Normal Operation.	N/A
4	pci_test_mode	Broadcom Use Only. Enable PCI test mode.	Normal Operation.	N/A
5	iob0_div	IOB0 clock is CPU clock/3 Use with slow CPU clocks.	IOB0 clock is CPU clock/4 Use with fast CPU clocks.	<a href="#">Section: "Clocks" on page 31</a>
6	iob1_div	IOB1 clock is CPU clock/2 Use with slow CPU clocks.	IOB1 clock is CPU clock/3 Use with fast CPU clocks.	<a href="#">Section: "Clocks" on page 31</a>
11:7	pll_div	These bits are used to set the PLL clock ratios		<a href="#">Section: "Clocks" on page 31</a>

**Table 8: Static Configuration Options (Cont.)**

<b>IO_AD Bit</b>	<b>Name</b>	<b>Pulled Up to 3.3V</b>	<b>Pulled Down</b>	<b>Section</b>
12	ser0_enable	Serial port 0 is synchronous. This sets the reset configuration, software may change it.	Serial port 0 is UART A.	Section: “Synchronous Mode” on page 336
13	ser0_rstb_en	GPIO[0] is driven as S0_RSTROBE by serial port 0.	GPIO[0] is a GPIO pin.	Section: “Synchronous Mode” on page 336
14	ser1_enable	Serial port 1 is synchronous. This sets the reset configuration, software may change it.	Serial port 1 is UART B.	Section: “Synchronous Mode” on page 336
15	ser1_rstb_en	GPIO[1] is driven as S1_RSTROBE by serial port 1.	GPIO[1] is a GPIO pin.	Section: “Synchronous Mode” on page 336
16	pcmcia_enable	PCMCIA controller enabled. GPIO[15:6] are used by the PCMCIA logic. IO_CS_L[6] is PCMCIA select.	PCMCIA controller disabled. GPIO[15:6] are GPIO pins. IO_CS_L[6] is a general chip select.	Section: “Introduction” on page 421
17	boot_mode[0]	The IO_CS_L[0] region of the generic bus space that is used for the boot ROM is configured for non-multiplexed operation with 8 data bits and 24 address bits.	The IO_CS_L[0] region of the generic bus space that is used for the boot ROM is configured for multiplexed operation with 32 data bits and 32 address/enable bits.	Section: “Configuring a Chip Select Region” on page 362
		The SMBus EEPROM boot is configured for large (> 16 kbit) EEPROMs using the eeprom read word protocol.	The SMBus EEPROM boot is configured for small (<= 16 kbit) EEPROMs using the read word protocol.	Section: “Booting Using an SMBus EEPROM” on page 412
18	boot_mode[1]	The boot address will access an EEPROM on the SMBus.	The boot address will access a ROM on the generic bus.	Section: “Boot ROM Support” on page 372. Section: “Booting Using an SMBus EEPROM” on page 412.
19	pci_host	The PCI interface is run as the host bridge.	The PCI interface is run as a regular device.	Section: “Introduction” on page 190.
20	pci_arbiter	The PCI interface uses the internal arbiter. This must not be set if bit [19] is pulled down.	The PCI interface uses an external arbiter.	Section: “PCI Arbiter” on page 222
21	southOnLDT	The South Bridge is on the HyperTransport fabric or on a bus bridged from the HyperTransport fabric.  If there is no South Bridge this bit may be set either way.	The South Bridge is on the PCI bus.	Section: “The SouthBridge, VGA and Subtractive Decode” on page 195.
22	big_endian	The system is Big Endian.	The system is Little Endian.	Section: “Introduction” on page 9.
23	genclk_en	Enables output of the generic bus clock on the IO_CLK100 pin.	Set the IO_CLK100 pin to a high impedance state.	Section: “Generic Bus Timing” on page 364





**Table 8: Static Configuration Options (Cont.)**

<b>IO_AD Bit</b>	<b>Name</b>	<b>Pulled Up to 3.3V</b>	<b>Pulled Down</b>	<b>Section</b>
24	ldt_test_en	Broadcom Use Only. Enable HyperTransport test mode.	Normal Operation.	N/A
25	gen_parity_en	Parity is enabled on the generic bus. GPIO[5:2] are used by the Generic Bus controller.	Parity is not used on the generic bus GPIO[5:2] are GPIO pins.	<a href="#">Section: “Generic Bus Parity” on page 363.</a>
31:26	config	Sampled at reset time and available in the SCD configuration register. These bits can be interpreted by software for system configuration.		N/A

After a system reset (or power-on reset) CPU 0 and all peripherals are brought out of reset, but on a BCM1250 CPU 1 continues to be held in reset and will be isolated from the system. This allows CPU 0 to perform essential system initialization before releasing CPU 1. Since both CPUs have the same reset vector (virtual FFFF\_FFFF\_BFC0\_0000, physical 00\_1FC0\_0000) the initial code will probably branch based on the CPU number which can be read from the **Prld** CP0 register.

The boot address is in the space that the I/O system uses for the generic expansion bus. At reset time this is configured to allow access to a (slow) boot ROM or flash memory, or can be diverted to SMBus interface 0, which will fetch code from a serial EEPROM. (A debugger may use the EJTAGBOOT option to cause the CPU to fetch the boot code from the JTAG port as described in [Section: “EJTAGBOOT Instruction” on page 425.](#))

Following reset, no coherent accesses should be made until the data cache tags in the CPU(s) and the L2 cache tags have been invalidated. The DMA controllers and requests from the expansion buses can use coherent accesses, so they must not be used until the tags have been cleared. On a single CPU, once the level 1 cache tags have been invalidated cacheable non-coherent accesses may be done to the boot memory space. It is recommended that CPU 0 invalidates its cache tags and the L2 cache tags prior to releasing CPU 1 from reset, CPU 1 should clear its tags and signal CPU 0 that coherent accesses may be used.

It is possible to independently reset the CPUs, either from software or the watchdog timers. This is a potentially hazardous operation since while in reset the CPU is removed from participation in the coherence protocol. When the CPU goes in to reset it may be in the process of supplying data and releasing a exclusive lock on a line, since these operations are aborted some other agent may hang waiting for the data or lock. When the CPU is released from reset it will have stale coherency information in its caches and software initialization will clear all state. The independent reset features can be used to implement error recovery when one CPU fails while maintaining availability by allowing the other to continue to operate, but very careful consideration must be given to the possible state of the system if this is done (or it may be optimistically done with the fall-back position of a full reset if the partial restart fails).

A controlled reset of a running CPU can be done in a safe way. First the data cache must be flushed by doing indexed-writeback-invalidate cache ops on all entries in the cache. Secondly, it must be ensured that the data for all evicts has been sent to the bus (rather than just being visible). After these steps, the CPU will not be involved in any coherent operations and can be safely reset. The sequence of operations can be summarised:

```

index wb invalidate over whole Dcache
sync
uncached store to somewhere that does not matter
uncached store to somewhere that does not matter
sync
uncached store to reset bit

```

The first `sync` will ensure all the index-writeback-invalidate operations have completed and the CPU will not hit for any future snoops, but up to two writebacks may still be in transit in the bus interface buffers. The uncached stores will queue behind the writebacks, and the second `sync` ensures they have reached the bus buffers and therefore the writebacks have completed. At this point it is safe for the CPU to write to its reset bit. The data cache state is not affected by reset, so when the CPU comes out of reset the cache (and duplicate tags used by snoops) will still have all entries invalid. If the CPU has been shutdown using this sequence it is therefore safe to release it from reset in a system that is already running with coherent accesses.

The different actions of the resets may be used to identify which reset happened. A sequence similar to the following could be used (this assumes that the device uses a UART on port 0 for the console but the IO\_AD[12] is set for synchronous port).

```

// First check the sw_flag. It is only cleared by coldreset
If system_cfg.sw_flag == 0
    resetType = Coldreset
// Next check for the software initiated full reset, this resamples
// the configuration bits so will set the synchronous port
else if system_cfg.ser0_enable == 1
    resetType = SoftwareSystemReset
// Next check for a soft reset, since the bit is not self-clearing
// it will still be set
else if system_cfg.sb_softres == 1
    resetType = SoftwareSbSoftReset
// No internal way to tell between an external RESET_L
// and a Watchdog timeout
else
    resetType = ResetLOrWatchdog
// later BCM1250 have a flag for watchdog reset
if BCM1250 stepping C0 or later
    if watchdog wd_has_reset bit set
        resetType = Watchdog
    else
        resetType = Reset

// Now set conditions for the next time
// And allow use of the UART, set not-coldreset flag, clear softres
system_cfg.ser0_enable = 0
system_cfg.sw_flag = 1
system_cfg.sb_softres = 0

```

The system\_scratch register could be used as an alternative to using the configuration bit for the serial port. The software would have to put a unique value into the register (or could set a bit that is reserved for this purpose) before writing the system\_reset bit.

## CLOCKS

The device is provided with a 100 MHz clock from which it generates the other internal clocks. The clock is multiplied up with a PLL to provide the CPU clock, which is divided down to provide the internal bus clocks and memory clock. A separate PLL is used to generate the HyperTransport clocks. An internal 100MHz clock is generated either directly from the reference clock or by dividing the CPU clock by the PLL ratio to drive the internal timers, the baud rate generator and for the timing parameters on the generic bus. The internal 100 MHz clock may be driven out on the IO\_CLK100 pin. Using the reference source ensures that for all PLL ratios the IO\_CLK100 will have a duty cycle only a little worse than the reference clock and that the IO\_CLK100 runs at the same 100MHz frequency as the reference clock during cold reset and while the PLL locks. Using the internally generated reference ensures that the IO\_CLK100 tracks the internal clocks (it will run slowly during cold reset even if the reference clock has not started), and if an integer PLL ratio is used (i.e. IO\_AD[7] was sampled low) ensures the duty cycle is no worse than 40:60 regardless of the reference clock. However, if a nonintegral PLL ratio is used only the rising edge of the internally generated IO\_CLK100 is valid, the time the output spends high can vary from cycle to cycle and be as short as 10% of the cycle.

The PCI bus and network interfaces have their own bus clocks that must be externally generated to match the attached components. The synchronous serial port clock either comes from the baud rate generator or from an external source.

Table 9 shows the supported clock ratios for the CPU and HyperTransport interface. Note that the speed grade of the part determines the maximum frequency that is permitted. The ratio for the CPU is set statically using the reset value on IO\_AD[11:7] as described in the previous section. The ratio for the HyperTransport may be set directly (by enabling an extension register in the HyperTransport configuration space), but is normally set indirectly by encoding the value from the HyperTransport Link Frequency register.

**Table 9: Core and HyperTransport Clock Settings**

Code	Ratio	Main PLL (Code from Reset Time IO_AD[11:7])		HyperTransport PLL (Code from HyperTransport Frequency Register)	
		CPU Clock (MHz)	ZBbus Clock (MHz)	HyperTransport Clock (MHz)	HyperTransport Data Rate (Mbps/pair)
00100	2x	200	100	200	400
00101	2.5x	250	125	250	500
00110	3x	300	150	300	600
00111	3.5x	350	175	350	700
01000	4x	400	200	400	800
01001	4.5x	450	225	450	900
01010	5x	500	250	500	1000
01011	5.5x	550	275	550	1100
01100	6x	600	300	600	1200
01101	6.5x	650	325		
01110	7x	700	350		
01111	7.5x	750	375		



**Table 9: Core and HyperTransport Clock Settings (Cont.)**

Code	Ratio	Main PLL (Code from Reset Time IO_AD[11:7])		HyperTransport PLL (Code from HyperTransport Frequency Register)	
		CPU Clock (MHz)	ZBbus Clock (MHz)	HyperTransport Clock (MHz)	HyperTransport Data Rate (Mbps/pair)
10000	8x	800	400		
10001	8.5x	850	425		
10010	9x	900	450		
10011	9.5x	950	475		
10100	10x	1000	500		
10101	10.5x	1050	525		
10110	11x	1100	550		

The clocks for the I/O bridges that connect the peripherals to the ZBbus (see [Figure 5 on page 9](#)) are synchronously divided down from the CPU clock. The divide ratio needs to be set as part of the reset time configuration (see [Section: “Reset” on page 26](#)) based on the CPU frequency. The I/O Bridge clock does not affect the speed of the peripheral interfaces directly, but it does affect the bandwidth (and latency) between the peripheral and the ZBbus. Increasing the bridge clock speed will increase the bandwidth, but will use more power. Conversely the speed can be decreased to save power if the full bandwidth is not needed. I/O Bridge 0 is designed for operation with the clock about 200 MHz, it should be set in the range 166-266 MHz (check the data sheet for the maximum frequency for a given speed grade). I/O Bridge 1 is designed for operation with the clock about 266 MHz, it should be set in the range 233-333 MHz (check the data sheet for the maximum frequency for a given speed grade). The maximum frequency for the bridge clocks is given in the Data Sheet as Fbr0 and Fbr1 in the Clock, Reset and Test timing parameters.



The memory clock is synchronously divided down from the CPU clock. The divide ratio is programmed into the memory controller configuration registers. Figure 8 gives an overview of the internal clock generation.

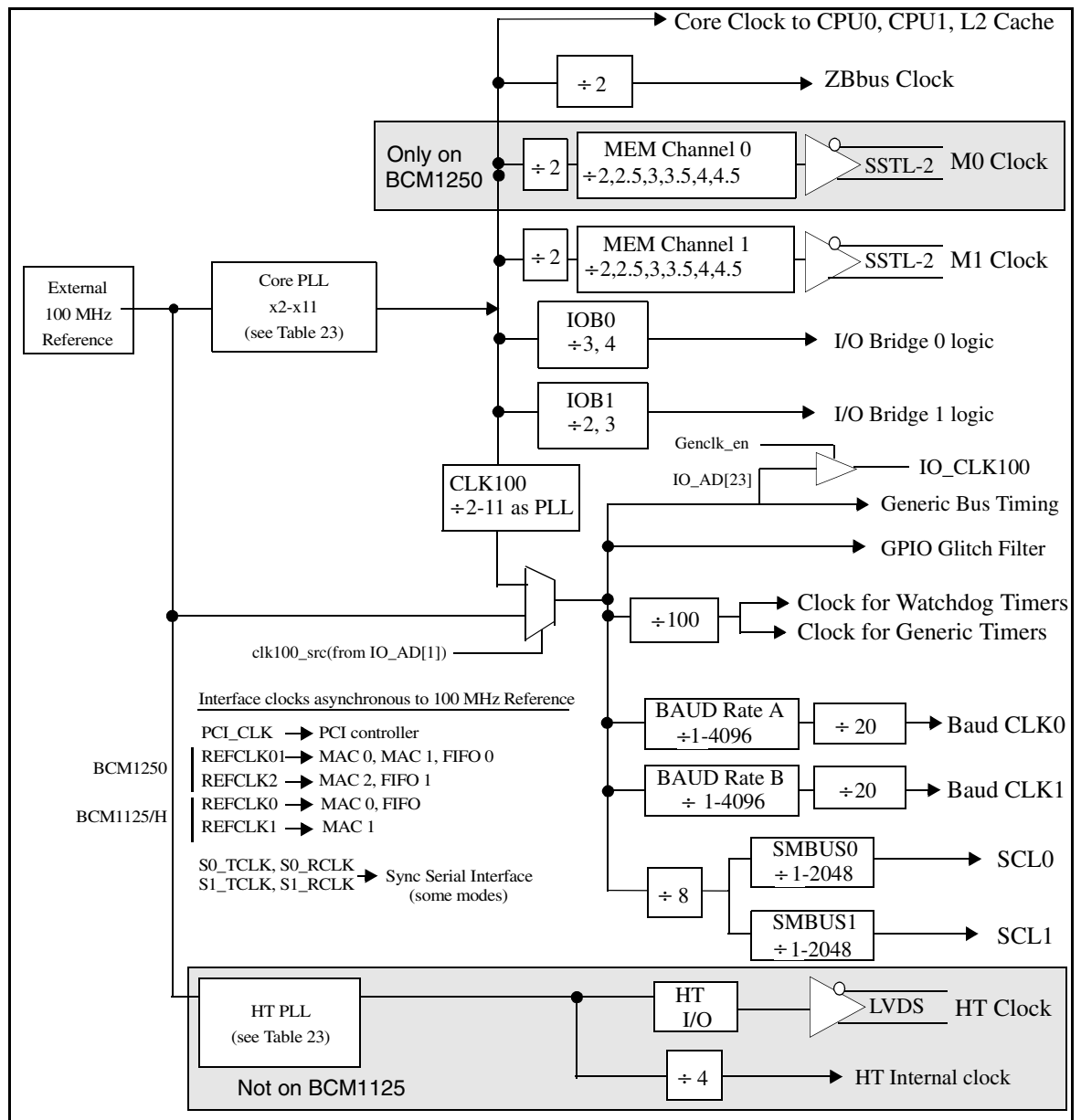


Figure 8: Clock Distribution Overview

## MEMORY MAP

The memory map is designed to be usable in systems that only have 32 bit addressing and to provide expansion for systems that can support the full 64 bit virtual address and 40 bit physical address of the SB-1 CPU. There are some additional restrictions that the MIPS architecture imposes:

- The reset vector of the CPU is to physical address 00\_1FC0\_0000.
- The exception vectors of the CPU are at physical address 00\_0000\_0000. (They may be offset from the base by a multiple of 64K using the SB-1 MultiProcessor Vector extension in the CPU config register).
- The first 512 MB of memory is addressable uncached (and unmapped) in KSEG1 and is therefore a good place for fixed address peripherals (they can be accessed without using any TLB entries).

In addition devices on the PCI need to be able to DMA into all of the physical memory (within the 32 bit PCI address range) and be able to enable or disable endian swapping on each transaction (see [Section: "Endian Policies" on page 201](#) for a full discussion of PCI and HyperTransport endian policies). An overview of the memory map is given in [Table 10](#) and [Figure 9](#) and a more detailed view is in [Table 11 on page 36](#).

**Table 10: Overview of BCM1250 Physical Address Map**

Base	Top	Owner
00_0000_0000	00_0FFF_FFFF	Memory controller.
00_1000_0000	00_1005_FFFF	System control and debug.
00_1006_0000	00_3FFF_FFFF	I/O system.
00_4000_0000	00_5FFF_FFFF	HyperTransport/PCI memory mapped I/O (32 bit addressing range) Match byte lane endian policy.
00_6000_0000	00_7FFF_FFFF	HyperTransport/PCI memory mapped I/O (32 bit addressing range) Match bit lane endian policy.
00_8000_0000	00_9FFF_FFFF	Memory controller.
00_A000_0000	00_BFFF_FFFF	Reserved
00_C000_0000	00_CFFF_FFFF	Memory controller.
00_D000_0000	00_D7FF_FFFF	L2 controller test.
00_D800_0000	00_D92F_FFFF	HyperTransport special operations Match byte lane endian policy. Not on BCM1125.
00_DC00_0000	00_DDFF_FFFF	HyperTransport/PCI I/O space Match byte lane endian policy.
00_DE00_0000	00_DFFF_FFFF	HyperTransport/PCI configuration space Match byte lane endian policy.
00_E000_0000	00_F7FF_FFFF	Reserved
00_F800_0000	00_F92F_FFFF	HyperTransport special operations Match bit lane endian policy. Not on BCM1125.
00_FC00_0000	00_FDFF_FFFF	HyperTransport/PCI I/O space Match bit lane endian policy.
00_FE00_0000	00_FFFF_FFFF	HyperTransport/PCI configuration space Match bit lane endian policy.
01_0000_0000	7F_FFFF_FFFF	Memory controller expansion.
80_0000_0000	F7_FFFF_FFFF	HyperTransport expansion (40 bit addressing range). Not on BCM1125.
F8_0000_0000	F8_FFFF_FFFF	PCI bus full access (match byte lane endian policy).
F9_0000_0000	F9_FFFF_FFFF	PCI bus full access (match bit lane endian policy).
FA_0000_0000	FC_FFFF_FFFF	Reserved
FD_0000_0000	FF_FFFF_FFFF	Reserved (Special HyperTransport range).



The memory controller supports up to 1 GB of memory in a system that is restricted to 32-bit physical addresses, and up to 4 GB ( 2 GB on BCM1125/H) using 512 Mb technology DRAMs (up to 8 GB when 1Gb technology SDRAMs are available, and with an option to double the size at the cost of speed with an external decoder) on systems with a full 40-bit address. The address map includes 512 MB for mapping PCI and HyperTransport memory mapped peripherals that use 32-bit addressing, and an alias for this space that will byte swap accesses when the system is in big endian mode.

Accesses to Reserved or Unused regions will result in UNPREDICTABLE behavior. Writes will be discarded.

FF_FFFF_FFFF	Reserved
FA_0000_0000	PCI Full Access
F8_0000_0000	HT Devices Not on BCM1125
80_0000_0000	SDRAM Expansion
01_0000_0000	Maps to 00_D800_0000 - 00_DFFF_FFFF With "Match Bit" Endian Policy
00_F800_0000	Reserved
00_E000_0000	PCI/HT Config
00_DE00_0000	PCI/HT I/O Space
00_DC00_0000	HT/PCI Special
00_D800_0000	L2 Direct Access
00_D000_0000	Fourth SDRAM Region
00_C000_0000	Reserved
00_A000_0000	Third SDRAM Region
00_9000_0000	Second SDRAM Region
00_8000_0000	PCI/LDT Memory Space Match Bit Lane Endian Policy
00_6000_0000	PCI/LDT Memory Space Match Byte Lane Endian Policy
00_4000_0000	
00_2000_0000	Boot ROM (Default for IO_CS0)
00_1FC0_0000	Generic Bus Devices
00_1009_0000	Internal Devices
00_1006_0000	System Control and Debug
00_1000_0000	First SDRAM Region
00_0000_0000	

Figure 9: Memory Map



**Table 11: Address Map Details**

Base	Top	Size	Owner	Use
00_0000_0000	00_0FFF_FFFF	256 MB	MC	Base DRAM.
00_1000_0000	00_1001_FFFF	2*64 KB	SCD	Reserved/Debug JTAG serviced addresses.
00_1002_0000	00_1002_0FFF	4 KB	SCD	Reset config, CPU 0 interrupt mapper, timers, addr trap, trace, bus log and counters.
00_1002_1000	00_1002_1FFF	4 KB	SCD	SCD CPU 0 Mailbox alias.
00_1002_2000	00_1002_2FFF	4 KB	SCD	CPU 1 interrupt mapper. (Reserved on BCM1125/H)
00_1002_3000	00_1002_3FFF	4 KB	SCD	CPU 1 Mailbox alias. (Reserved on BCM1125/H)
00_1002_4000	00_1002_FFFF	48 KB	SCD	Reserved
00_1003_0000	00_1003_FFFF	64 KB	SCD	ZBbus cycle count.
00_1004_0000	00_1004_FFFF	64 KB	SCD/L2	L2 registers.
00_1005_0000	00_1005_FFFF	64 KB	SCD/MC	Memory controller registers.
00_1006_0000	00_1006_00FF	0.25 KB	I/O	SMBus, GPIO.
00_1006_0100	00_1006_01FF	0.25 KB	I/O	Duart ch A.
00_1006_0200	00_1006_02FF	0.25 KB	I/O	Duart ch B.
00_1006_0300	00_1006_03FF	0.25 KB	I/O	Duart status and control.
00_1006_0400	00_1006_07FF	1 KB	I/O	Sync serial, dma and HDLC ch 0.
00_1006_0800	00_1006_0BFF	1 KB	I/O	Sync serial, dma and HDLC ch 1.
00_1006_0C00	00_1006_0FFF	1 KB	I/O	Unused
00_1006_1000	00_1006_17FF	2 KB	I/O	Generic Bus config.
00_1006_1800	00_1006_1FFF	2 KB	I/O	Generic Bus status and log, PCMCIA config and status.
00_1006_2000	00_1006_2FFF	4 KB	I/O	Reserved
00_1006_3000	00_1006_3FFF	4 KB	I/O	Unused
00_1006_4000	00_1006_4FFF	4 KB	I/O	MAC 0.
00_1006_5000	00_1006_5FFF	4 KB	I/O	MAC 1.
00_1006_6000	00_1006_6FFF	4 KB	I/O	MAC 2. (On BCM1125/H: alias of MAC 0; do not use!)
00_1006_7000	00_1006_FFFF	36 KB	I/O	Unused
00_1007_0000	00_1008_FFFF	2*64 KB	I/O	Reserved
00_1009_0000	00_3FFF_FFFF	767 MB	I/O	Generic/boot interface.
00_4000_0000	00_5FFF_FFFF	512 MB	PCI/HT	Memory mapped I/O space. Match byte lane endian policy.
00_6000_0000	00_7FFF_FFFF	512 MB	PCI/HT	Memory space mapped I/O space. Match bit lane endian policy.
00_8000_0000	00_8FFF_FFFF	256 MB	MC	Second DRAM bank.
00_9000_0000	00_9FFF_FFFF	256 MB	MC	Third DRAM bank.
00_A000_0000	00_BFFF_FFFF	512 MB	XX	Unused
00_C000_0000	00_CFFF_FFFF	256 MB	MC	Fourth DRAM bank.
00_D000_0000	00_D7FF_FFFF	128 MB	L2C	L2 special test address range. Match byte lane endian policy.





Table 11: Address Map Details (Cont.)

Base	Top	Size	Owner	Use
00_D800_0000	00_D8FF_FFFF	16 MB	HT	EOI signaling. Match byte lane endian policy. (Reserved on BCM1125)
00_D900_0000	00_D90F_FFFF	1 MB	HT	lack signaling. Match byte lane endian policy. (Reserved on BCM1125)
00_D910_0000	00_D91F_FFFF	1 MB	HT	System management. Match byte lane endian policy. (Reserved on BCM1125)
00_D920_0000	00_D92F_FFFF	1 MB	HT	N/A
00_D930_0000	00_DBFF_FFFF	45 MB	HT	Reserved
00_DC00_0000	00_DDFF_FFFF	32 MB	PCI/HT	I/O space (only 25 bits). Match byte lane endian policy.
00_DE00_0000	00_DEFF_FFFF	16 MB	PCI/HT	Configuration Space. Match byte lane endian policy.
00_DF00_0000	00_DFFF_FFFF	16 MB	PCI/HT	Reserved
00_E000_0000	00_EFFF_FFFF	256 MB	XX	Unused
00_F000_0000	00_F7FF_FFFF	128 MB	XX	Unused
00_F800_0000	00_F8FF_FFFF	16 MB	HT	EOI signaling. Match bit lane endian policy. (Reserved on BCM1125)
00_F900_0000	00_F90F_FFFF	1 MB	HT	lack signaling. Match bit lane endian policy. (Reserved on BCM1125)
00_F910_0000	00_F91F_FFFF	1 MB	HT	System Management Match bit lane endian policy. (Reserved on BCM1125)
00_F920_0000	FD_F92F_FFFF	1 MB	HT	N/A
00_F930_0000	00_FBFF_FFFF	45 MB	HT	Reserved
00_FC00_0000	00_FDFF_FFFF	32 MB	PCI/HT	I/O space (only 25 bits). Match bit lane endian policy.
00_FE00_0000	00_FEFF_FFFF	16 MB	PCI/HT	Configuration Space. Match bit lane endian policy.
00_FF00_0000	00_FFFF_FFFF	16 MB	PCI/HT	Reserved
01_0000_0000	7F_FFFF_FFFF	508 GB	MC	DRAM expansion space.
80_0000_0000	F7_FFFF_FFFF	480 GB	HT	HyperTransport devices. (Reserved on BCM1125)
F8_0000_0000	F8_FFFF_FFFF	4 GB	PCI	PCI full address range. Match byte lane endian policy.
F9_0000_0000	F9_FFFF_FFFF	4 GB	PCI	PCI full address range. Match bit lane endian policy.
FA_0000_0000	FC_FFFF_FFFF	12 GB	PCI	Reserved
FD_0000_0000	FD_F7FF_FFFF	3968 MB	HT	Reserved (HyperTransport reserved space).
FD_F800_0000	FD_F8FF_FFFF	16 MB	HT	Reserved (Interrupt signaling).
FD_F900_0000	FD_F90F_FFFF	1 MB	HT	Reserved (lack signaling).
FD_F910_0000	FD_F91F_FFFF	1 MB	HT	Reserved (System Management).
FD_F920_0000	FD_F92F_FFFF	1 MB	HT	Reserved (PREQ Protocol).
FD_F930_0000	FD_FBFF_FFFF	45 MB	HT	Reserved
FD_FC00_0000	FD_FDFF_FFFF	32 MB	HT	Reserved (PCI I/O).
FD_FE00_0000	FD_FFFF_FFFF	32 MB	HT	Reserved (PCI Configuration).

Table 11: Address Map Details (Cont.)

---

Base	Top	Size	Owner	Use
FE_0000_0000	FF_FFFF_FFFF	8 GB	HT	Reserved

---



---

This Page is left blank for notes



This Page is left blank for notes



## Section 4: System Control and Debug Unit

### INTRODUCTION

The System Control and Debug unit (SCD) includes all the system control functions, interrupt mappers, system level performance monitoring, and debugging functions.

### SYSTEM CONTROL

The system controller is used to bring the part out of reset. It holds the reset-time configuration options and contains the extended JTAG interface that allows external control and monitoring of the system.

The JTAG interface is described in detail in [Section: "TAP Controller" on page 421](#). The basic scan chain has been extended to allow an external debugger access to a selection of internal state both in the CPUs and in the system. The JTAG interface can access system debug and performance monitoring features such as the performance counters and the ZBbus trace logic. No software support on the target system is required to run this debugging interface.

System Reset is controlled by this section of the SCD. The external COLDRES\_L pin must be asserted at system power-up and can be asserted later to cause a cold reset of the system. The RESET\_L pin can be used to cause a warm reset of the system. The SCD generates all the required internal reset pulses and timing and drives the external reset output RESETOUT\_L.

There are two registers associated with the system controller. One is the **system\_revision** register which identifies the part and gives the chip revision number. The second is the system configuration register **system\_cfg** which reports the states of the reset time configuration options and allows resetting of various sections of the system. The **system\_cfg** register is accessible both from the system and from the JTAG port.

A CPU can cause a system reset by setting the system\_reset bit of the **system\_cfg** register. This will behave the same as the COLDRES\_L pin of the part being asserted (except the PLL is not restarted), and is the standard way for software to cause a full restart. The restart will clear the system\_reset bit. The sb\_softres bit is identical except the **system\_cfg** register bits are not restored to their defaults (note that this bit does not self clear). Software may use the other bits of the register to reset an individual CPU or signal an external reset. To prevent lockup, the CPU 0 reset bit automatically clears. JTAG access to the register may also manipulate the reset bits, in this case they do not automatically clear so the JTAG probe has full control of the resets. The JTAG probe can also reset individual ZBbus agents. While reset is asserted the agent is isolated from the system, in this state any access to the agent will hang.

The lower bits of the **system\_cfg** register reflect the state that was latched from the generic bus IO\_AD pins at reset time. These are described in [Section: "Reset" on page 26](#). Some of these are used to configure device pin options, others are free for interpretation by system software (for example the board revision could be read in this way).

Many bits of the **system\_cfg** port are only writable from the JTAG port. They provide access to test features and are reserved for use by Broadcom. They must be left at zero for normal operation of the part.

The uniprocessor bits and soft-reset bit may be used on the BCM1250 to disable one of the processors. The clock to the disabled processor will be stopped following reset and it will enter a low power state (lower power than being left in reset). The processor will be enabled on the next full reset (RESET\_L pin asserted, system\_reset bit written with a one or a watchdog double timeout). Following a full reset only CPU0 will be running and CPU1 will be held in reset. If a uniprocessor system is desired, the CPU0 reset sequence should check for a multiprocessor environment by reading the multiprocessor bit in the **Prid** (CPU CP0 Processor Id register) or the uniprocessor bits in the **system\_cfg** register. If the system is running as a multiprocessor the unicpu0 bit should be set in the **system\_cfg** register, and the sb\_softres bit written with a one to cause the system to reboot as a uniprocessor.

The **system\_scratch** register is a read/write register that is not used by the hardware. It is free for software use. For example it may be used by firmware to store a pointer to bootstrap information for use by the operating system, or it may be used to pass (small) messages between the processors before memory is configured. The value of the register is UNPREDICTABLE following cold reset, the value is preserved over other resets. The sw\_flag bit in the **system\_cfg** register may be used to indicate the validity of the scratch value, since the flag is cleared on a cold reset and preserved over other resets. The sw\_flag may also be used by the system to distinguish between cold resets and others. Software can check the bit after reset: if the bit is zero then the reset was a cold reset and the bit should be set, if the bit is already set then the reset was some other type (also see [Section: "Reset" on page 26](#)).

**Table 12: System Identification and Revision Register**

system_revision - 00_1002_0000 READ ONLY			
Bits	Name	Default	Description
7:0	reserved	8'hff	Reserved, reads as 8'hff.
15:8	revision	xx	Revision of the part. See <a href="#">Table 13</a> .
31:16	Part	xx	Part type. BCM1250 16'h1250 16'h1150 (configured as uniprocessor) 16'h1125 (configured as uniprocessor, half L2 aka 1125Y) BCM1125 16'h1123 BCM1125H 16'h1124  Bits 27:24 Indicate the number of processors. Bits 23:20 Indicate the L2 cache size (1-128KB, 2-256KB, 5-512KB, 0-1024K). Bits 19:16 Indicate the peripheral set (0,2,5 - BCM1250 peripherals, 3 - BCM1125 peripherals, 4 - BCM1125H peripherals).
63:32	wid	xx	Wafer ID.

**Table 13: Part Revisions**

Part	Stepping	Peripherals	Revision	Nickname	Comment
BCM12500	An	PERIPH_REV1	8'h01 or 8'h02	"Pass1"	Prototype only parts, refer to earlier manual.
BCM1250	An	PERIPH_REV2	8'h03 - 8'h0b	"Pass2"	Initial production BCM1250 parts.
BCM1250	Bn	PERIPH_REV2	8'h10 or 8'h11	"Pass2.2"	
BCM1250	Cn	PERIPH_REV3	8'h20	"Pass3"	BCM1250 with peripherals upgraded
BCM1125/H	An	PERIPH_REV3	8'h20 or 8'h21	"Pass1"	Initial production BCM1125/H parts
BCM1125/H	Bn	PERIPH_REV3	8'h30	"Pass2"	

**Table 14: Manufacturing Information Register**

<b>system_manuf - 00_1003_8000 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
63:0	mid	xx	Manufacturing ID. Broadcom Use Only.

**Table 15: System Configuration Register**

<b>system_cfg - 00_1002_0008</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
0	reserved	ext	Read Only, reflects the strap resistor on IO_AD[0].
1	clk100_src	ext	Read Only, reflects the strap resistor on IO_AD[1] that selects the source for the internal 100MHz clock and IO_CLK100 (if enabled).
2	ldt_minrstcnt	ext	Read Only, reflects the strap resistor on IO_AD[2]. Broadcom Use Only. This must be zero for normal operation.
3	ldt_bypass_pll	ext	Read Only, reflects the strap resistor on IO_AD[3]. Broadcom Use Only. This must be zero for normal operation.
4	pci_test_mode	ext	Read Only, reflects the strap resistor on IO_AD[4]. Broadcom Use Only. This must be zero for normal operation.
5	job0_div	ext	Read Only, reflects the strap resistor on IO_AD[5] that controls the clock divider for I/O Bridge 0. 0: IOB0 runs at CPU clock/4, for use with fast CPU clocks. 1: IOB0 runs at CPU clock/3, for use with slow CPU clocks.
6	job1_div	ext	Read Only, reflects the strap resistor on IO_AD[6] that controls the clock divider for I/O Bridge 1. 0: IOB1 runs at CPU clock/3, for use with fast CPU clocks. 1: IOB1 runs at CPU clock/2, for use with slow CPU clocks.
11:7	pll_div	ext	Read Only, reflects the strap resistors on generic IO_AD[11:7] that select the PLL Divide ratio.
12	ser0_enable	ext	Read/Write. The default reflects the strap resistor on generic IO_AD[12] but this bit can be written by software to change the setting. 0: Serial interface 0 is in asynchronous (uart) mode. 1: Serial interface 0 is in synchronous mode. If this bit is changed by software it must also re-initialize the interface.
13	ser0_rstb_en	ext	Read Only, reflects the strap resistor on generic IO_AD[13] that allocates GPIO[0] pin to the synchronous serial interface.
14	ser1_enable	ext	Read/Write. The default reflects the strap resistor on generic IO_AD[14] but this bit can be written by software to change the setting. 0: Serial interface 1 is in asynchronous (uart) mode. 1: Serial interface 1 is in synchronous mode. If this bit is changed by software it must also re-initialize the interface.
15	ser1_rstb_en	ext	Read Only, reflects the strap resistor on generic IO_AD[15] that allocates GPIO[1] pin to the synchronous serial interface.
16	pcmcia_enable	ext	Read Only, reflects the strap resistor on generic IO_AD[16] that configured the PCMCIA mode.
18:17	boot_mode	ext	Reflects the strap resistor on generic IO_AD[18:17] that configured the boot mode. 00: 32 bit generic bus ROM (multiplexed) 01: 8 bit generic bus ROM (non-multiplexed) 10: SMBus EEPROM <= 16 kbit (read word protocol) 11: SMBus EEPROM > 16kbit (eeprom read word protocol). Bit 17 is Read Only. Bit 18 can be changed by software to clear the SMBus boot mode and allow general use of the SMBus 0 interface.



**Table 15: System Configuration Register (Cont.)**

system_cfg - 00_1002_0008			
Bits	Name	Default	Description
19	pci_host	ext	Read Only, reflects the strap resistor on generic IO_AD[19], that configures the PCI interface to be host or device mode.
20	pci_arbiter	ext	Read Only, reflects the strap resistor on generic IO_AD[20], that configures the PCI interface to use an internal or external arbiter. (If the PCI is set in device mode the resistor must be set for an external arbiter)
21	southOnLDT	ext	Read Only, reflects the strap resistor on generic IO_AD[21], that configures the southbridge to be on the HyperTransport fabric or PCI bus.
22	big_endian	ext	Read Only, reflects the strap resistor on generic IO_AD[22], that configures the system to be big or little endian.
23	genclk_en	ext	Read Only, reflects the strap resistor on generic IO_AD[23], that enables output of the generic bus clock on IO_CLK100. If this bit is zero then the IO_CLK100 will be held in a high impedance state.
24	ldt_test_en	ext	Read Only, reflects the strap resistor on IO_AD[24]. Broadcom Use Only. This must be zero for normal operation.
25	gen_parity_en	ext	Read Only, reflects the strap resistor on generic IO_AD[25] that configured the generic bus parity.
31:26	config	ext	Read Only, reflects the strap resistor on generic IO_AD[31:26]. These configuration bits are available for interpretation by software.
32	clkstop	1'b0	Writable via JTAG only. Broadcom use only.
33	clkstep	1'b0	Writable via JTAG only. Broadcom use only.
41:34	clkcount	8'b0	Writable via JTAG only. Broadcom use only.
42	pllbyypass	1'b0	Writable via JTAG only. Broadcom use only.
44:43	pll_iref	2'b0	Writable via JTAG only. Broadcom use only.
46:45	pll_vco	2'b0	Writable via JTAG only. Broadcom use only.
48:47	pll_vreg	2'b0	Writable via JTAG only. Broadcom use only.
49	mem_reset	1'b0	Writable via JTAG only. When set the memory controller is held in reset.
50	l2c_reset	1'b0	Writable via JTAG only. When set the level 2 cache is held in reset.
51	io_reset_0	1'b0	Writable via JTAG only. When set the I/O bridge to the PCI and HyperTransport fabric is held in reset.
52	io_reset_1	1'b0	Writable via JTAG only. When set the I/O bridge to the slow speed devices and generic bus is held in reset.
53	scd_reset	1'b0	Writable via JTAG only. When set the SCD is held in reset.
54	cpu_reset_0	1'b0	Always reads as zero. When written with a 1 a standard length reset pulse is delivered to CPU 0. (A device reset will also cause a standard length reset pulse to CPU 0).
55	cpu_reset_1	1'b1	When set CPU 1 will be held in reset. This bit is set on a device reset, causing the processor to remain in reset until released under software control.
56	unicpu0	1'b0	Set to indicate uniprocessor using physical processor 0. (This bit will always be set on the BCM1125/H.)
57	unicpu1	1'b0	Set to indicate uniprocessor using physical processor 1. (BCM1250 only)
58	sb_softres	1'b0	When a write changes this bit from a 0 to a 1 a soft reset will be performed. This will reset of whole chip except for this register. Note that once it is set the bit must be cleared before writing a 1 will again cause a soft reset.
59	ext_reset	1'b0	When set the RESETOUT_L pin will be asserted.



**Table 15: System Configuration Register (Cont.)**

<b>system_cfg - 00_1002_0008</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
60	system_reset	1'b0	When written with a 1 a full system reset will be performed (thus setting this bit back to zero).
61	misr_mode	1'b0	Broadcom Use Only.
62	scd_misr_reset	1'b0	Broadcom Use Only.
63	sw_flag	1'bx	This read/write bit is cleared by a cold reset. Its value is preserved on any other reset. It may be used by software to detect the reset type, or for any other use.

**Table 16: Scratch Register**

<b>system_scratch - 00_1002_0C10</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
63:0	value	64'hx	This register is available for any use by software. When the chip is powered up its value is UNPREDICTABLE. Its value is preserved over reset.

## MAILBOX REGISTERS

Each CPU has a mailbox register that can be used to signal events or small messages. The register may be accessed by the other CPU (on a BCM1250) or a bus master peripheral. If the part is a device on the PCI bus (i.e. not the host bridge) then the host on the PCI can use the mailbox to signal the CPU.

A mailbox register is 64 bits wide and has four interrupt lines, for each group of 16 bits the corresponding interrupt is raised whenever any bit is set. To set a bit in the mailbox a 1 must be written to the appropriate bit position in the **mbox\_set** location. To clear a bit in the mailbox a 1 must be written to the appropriate bit position in the **mbox\_clear** location. Thus there is no need for an atomic read-modify-write. Typically only the CPU which owns the register would write the clear register and the signalling agent would write the set register. However, there is no protection in the hardware to enforce this (any agent that can access the **mbox\_set** and **mbox\_clear** locations can change the mailbox state). The interrupt outputs from a particular mailbox are only routed to the interrupt mapper of one CPU, thus CPU0 cannot be interrupted by the setting of bits in **mailbox\_1** and CPU1 cannot be interrupted by the setting of bits in **mailbox\_0**.

Interrupts from the HyperTransport interface can set bits in the mailbox, allowing peripherals to signal events. This is setup by the way the peripheral interrupts are configured by the system software.

The system software determines the use of the mailbox register and the meaning associated with each of the bits. Other than the mapping from bit positions to interrupt lines the hardware imposes no restrictions.

The simplest way to use the mailbox is as sixty-four event channels grouped into four (or fewer) priority levels. The signalling agent writes to set the event bit and the signaled CPU can clear it when it has responded. One or more of the four interrupts will be raised whenever there are outstanding events in the mailbox. This is likely to be the best method to use if high speed I/O interrupts will be feeding the mailbox.

At the other extreme, each of the quarters of the mailbox could be considered a channel for passing 16 bit messages. An example use is in inter-processor procedure calls. The source of the call will marshal arguments into a shared memory buffer, then make the call by writing the buffer index to the mailbox set location. This raises an interrupt on the service processor, which will read the buffer index from the mailbox and write the mailbox clear location. Since the shared memory buffer is coherent the service processor will always see the correct data, and the ordering imposed by the write-interrupt cleanly transfers buffer ownership. When the procedure is completed the results could be passed back using the reverse mechanism.

In addition to the normal address, the **mbox\_set** and (read only) **mailbox** locations have aliases that are the only registers in a 4k page. This page is accessible from the PCI interface (through BAR2 for the CPU 0 mailbox and BAR3 for the CPU1 mailbox). Thus PCI devices can only set bits in the mailbox (to signal the CPU) or read the current mailbox status (to see if a signal was cleared), but can never clear the mailbox. The CPU (or any other agent in the part) can access all the mailbox registers and can therefore both set and clear bits.

**Table 17: Mailbox Registers**

<b>mailbox_cpu_0 - 00_1002_00C0, alias_mailbox_cpu_0 - 00_1002_1xx0 (Read Only)</b> <b>mailbox_cpu_1 - 00_1002_20C0, alias_mailbox_cpu_1 - 00_1002_3xx0 (Read Only)</b>							
63	48	47	32	31	16	15	0
mbox_int_0 if any bit set		mbox_int_1 if any bit set		mbox_int_2 if any bit set		mbox_int_3 if any bit set	
<b>mbox_set_cpu_0 - 00_1002_00C8, alias_mbox_set_cpu_0 - 00_1002_1xx8 (Write Only)</b> <b>mbox_set_cpu_1 - 00_1002_20C8, alias_mbox_set_cpu_1 - 00_1002_3xx8 (Write Only)</b>							
When this location is written, any bits that are set will cause the corresponding bit to be set in the mailbox register.							
<b>mbox_clr_cpu_0 - 00_1002_00D0 (Write Only)</b> <b>mbox_clr_cpu_1 - 00_1002_20D0 (Write Only)</b>							
When this location is written, any bits that are set will cause the corresponding bit to be cleared in the mailbox register.							



## INTERRUPTS

The chip has a large number of interrupt sources. These come from both internal peripherals and external signals. Each CPU has its own interrupt mapper which allows the destination of the interrupt from each source to be set to one of the six architected hardware interrupts, the non-maskable interrupt (NMI) or the Debug Interrupt (DINT). The mapper block includes the CPU mailbox registers described in the previous section.

Most interrupt sources are common to each interrupt mapper. The only sources that are specific to a CPU are directed interrupts from the HyperTransport (as described below, the interrupt message includes which CPU it should be delivered to) and the per-CPU mailbox interrupts.

The basic operation of the interrupt mapper is described in this section. Interrupts from the HyperTransport fabric extend the basic model as described in the next section. [Section: “The Full Interrupt Mapper” on page 50](#) has a diagram of the complete mapper, including system, mailbox and HyperTransport sources and describes all the associated registers.

The interrupt mapper receives the level sensitive interrupts from all sections of the part. Each source has an associated mask bit and a 3 bit map register. If the source is interrupting and not masked out it is driven to one of the CPU interrupt lines according to the mapping. [Table 18](#) shows the mapping, and the corresponding IP bit in the Cause register.

**Table 18: Interrupt Mappings**

Mapping	Interrupt	CP0 Cause Register
000	I0	IP[2] bit 10.
001	I1	IP[3] bit 11.
010	I2	IP[4] bit 12.
011	I3	IP[5] bit 13.
100	I4	IP[6] bit 14.
101	I5	IP[7] bit 15.
110	NMI	Reset/NMI vector, flagged in status register not cause register.
111	DINT	CPU debug interrupt, flagged in debug register.

Each source maps to one CPU interrupt line. There is no limit on how many sources may map to a single CPU line. The I5 interrupt line is also used by interrupts generated internal to the CPU, these will be merged with any external requests. The MIPS architecture uses software based interrupt dispatch, so the software can assign priorities to the six regular interrupt lines as it sees fit. A typical system would merge most of the system sources on a few of the interrupt lines, and assign one source per line for sources that need rapid dispatch.

To assist software dispatch there are status registers associated with each of the six CPU interrupt lines, DINT and the NMI. These indicate which sources are interrupting, pass the mask and are routed to this interrupt line. In addition there is a global status register that shows which sources are interrupting regardless of their mask setting or routing.

The debug interrupt (DINT) to the CPU may be raised by the interrupt mapper. It will also be raised if a trace trigger is marked to send DINT (see [Section: "Trigger Sequences" on page 73](#)) or if the break control bit is set in the EJTAG control register (see [Section: "EJTAG Control Register" on page 439](#)).

There is a diagnostic register built into the interrupt mapper that can be used to force interrupts to be asserted into the mask and map logic.

Since there is a full copy of the mapper for each CPU, system interrupts may be masked and mapped differently on the two processors. In many cases a particular source would always be assigned to a particular processor, delivering it to both requires software to coordinate servicing the interrupt.

## HYPERTRANSPORT INTERRUPTS

The HyperTransport bus protocol includes passing interrupt messages across the HyperTransport fabric. These messages include identification of the interrupt vector (as described below there are some non-vectorized interrupts, carrying meanings specific to the x86 architecture, but these are thought of as having an implicit vector number in the interrupt mapper). The MIPS architecture does not directly support hardware interrupt vectoring, so the vector number is used by the hardware to decide which input to the interrupt mapper will be triggered and software can do any required vectoring.

The interrupt packet also includes a destination. This is either a physical processor number or "a system specific logical mapping". The BCM1250 CPUs (and their related interrupt controllers) are physically numbered 0 and 1. The BCM1125H CPU (and interrupt controller) is physical number 0. The logical mapping used is the example one from the HyperTransport specification, a processor is selected as the target by setting the bit position corresponding to its physical processor number. This scheme allows for up to eight processors. Use of the logical mapping allows broadcast and multicast interrupts (having only two CPUs, these amount to the same thing on the BCM1250). The mapper supports the 8 bit destination from the HyperTransport Specification rev 1.0 and earlier, and will ignore the additional 24 destination bits added by the HyperTransport Specification rev 1.01.

The HyperTransport interrupt messages must maintain ordering with transactions that come across the HyperTransport link ahead of them. They therefore follow the same flow as other HyperTransport transactions and are delivered to the SCD across the ZBbus as writes to the **interrupt\_idt\_set** register. Writes to this register are decoded and used to set bits in either the **mailbox** register or the **interrupt\_idt** register. The CPU can write the **interrupt\_idt\_set** register during testing to simulate the arrival of an interrupt message. The data written is a 64-bit double-word containing the data from the HyperTransport Interrupt message as described in [Table 19](#).

**Table 19: Interrupt Message Format for Writes to interrupt\_idt\_set Register**

Data Bits	Description
2:0	HyperTransport Interrupt Message Type: 000: Fixed 001: Arbitrated (Low Priority) 010: Non-vectored: SMI 011: Non-vectored: NMI 100: Non-vectored: INIT 101: Non-vectored: Startup 110: Non-vectored: External Interrupt 111: Reserved
3	HyperTransport Interrupt Trigger Mode 0=Edge, 1=Level (non-vectored interrupts must be edge).
4	HyperTransport Interrupt Destination Mode 0=Physical, 1=Logical.
12:5	HyperTransport Interrupt Destination.
20:13	HyperTransport Interrupt Vector.

The HyperTransport defines three sorts of interrupts:

**Non-vectored interrupts** carry a type field rather than source vector information. They are used to carry interrupt messages that are traditionally associated with pins on the x86 CPU. The five types are SMI (system management interrupt), NMI (non maskable interrupt), Init (initialize processor, on x86 resets integer registers but does not affect FP or caches), Startup (used to start application processors in x86 multiprocessor systems) and Ext. Int. (external interrupt, used to signal an interrupt from a PIC style PC interrupt controller). These are statically mapped to source vectors h32 - h36 before being reported to the interrupt mapper of the CPU(s) to which they are directed.

**Fixed interrupts** are the standard form. They include the destination information and the source vector and are delivered to all interrupt mappers to which they are directed.

**Arbitrated (Low Priority) interrupts** behave the same as fixed interrupts. However if multiple destinations are indicated they only get delivered to one of them. The HyperTransport specification requires that the destination selected is either the lowest priority or the CPU that is currently servicing an interrupt from the same source. Since MIPS architecture CPUs give no external indication of which interrupt they are currently servicing the mapper implements this by delivering the interrupt to the lowest numbered CPU to which it is directed.

All HyperTransport interrupts are thus presented to the interrupt mapper as an 8 bit vector number. The top two bits of the vector number are used to determine how the low 6 bits are used:

**Table 20: Delivery of HyperTransport Interrupts**

Vector[7:6]	Delivery of Vector[5:0]
00	The bit corresponding to Vector[5:0] in the interrupt_idt register is set, raising that interrupt line to the mapper.
01	The bit corresponding to Vector[5:0] in the mailbox register is set, raising one of the mailbox interrupts to the mapper.
10	Interrupt will be discarded.
11	Interrupt will be discarded.

HyperTransport interrupt messages specify if the source is edge or level sensitive. Edge sensitive interrupts do not need to be acknowledged, and it is source dependent whether the CPU has to take action before further interrupts can be signalled. Level interrupts require the CPU to send an EOI message to acknowledge the servicing of the request. The interrupt source device will wait until an EOI is received before issuing any further interrupts. In the MIPS architecture the CPU does not produce a signal to indicate EOI. Software must do a write to the EOI space at the end of the interrupt service routine (see [Section: “HyperTransport End Of Interrupt \(EOI\) Signaling Space” on page 198](#)). Since the HyperTransport specification requires that edge and level interrupts are not reported with the same source vector ID, software will know from the vector number that an EOI is needed. The access to the EOI space must be a write, so the CPU is not stalled while the request is issued. (Reads to EOI space have UNDEFINED results.)

Interrupts from a PC style PIC interrupt controller are signaled using the External Interrupt message, which has no additional source information associated with it. The PIC will provide an 8-bit source vector in response to an interrupt acknowledge (IACK) cycle. On the BCM1250 and BCM1125H software is responsible for running the IACK cycle by performing a byte read within the reserved IACK range as described in [Section: “Legacy Interrupt Acknowledge \(IACK\) Space” on page 199](#). A read to this address range gets routed to the southbridge. If the southbridge is a native HyperTransport device, then this command packet will be routed directly to the southbridge. If the southbridge is connected to a PCI device bridged from the HyperTransport then the command packet will be routed to the intervening HyperTransport-PCI bridge. (If the southbridge is on the PCI interface (set by the southOnLDT configuration bit being clear) the IACK access will be run as a PCI IACK cycle, but in that case the interrupt would not have come in as a HyperTransport External Interrupt message.)

## THE FULL INTERRUPT MAPPER

The full interrupt controller is illustrated in [Figure 10 on page 51](#). This shows the mailbox and HyperTransport sources and all the mapper registers.

The system sources and the mailbox interrupts come into the **interrupt\_source\_status** register. They are combined with the HyperTransport interrupts and the data in the **interrupt\_diag** register. This is a read/write register that the CPU can use to activate interrupt request lines. It is primarily used to allow testing of the interrupt routing by simulating device interrupts.

The **interrupt\_idt** register bits are set by the HyperTransport interrupt controller as described in the previous section. The register is read by the CPU to determine the HyperTransport interrupt source and cleared by a write to the **idt\_interrupt\_clear** register.

The **interrupt\_mask** register is used to block interrupts. Masking is done after the three sources (system, HyperTransport and diagnostic) have been combined, if interrupt lines are shared either all sources are masked off or none. All unmasked interrupts are passed to the interrupt mapper. This has a 3 bit Map register for each of the sources which maps the source to one of the CPU interrupt lines (regular INT[5:0], NMI or DINT). Each source can be mapped to only one CPU line, but each CPU line can have any number of sources.

The regular interrupt signals to the CPU and DINT are level sensitive and will continue to be asserted as long as there are interrupting sources. The NMI is edge triggered and will be asserted for a single cycle whenever the mapper output asserts (i.e. after an NMI all interrupting sources must be clear for at least a cycle before a subsequent NMI will be generated).

The **interrupt\_trace** register is used to select which interrupts to this CPU are sent to the trace unit to be used as part of a trigger condition. The bits in this register provide an accept mask. Any interrupt that has its corresponding bit set in this register will cause the trigger.

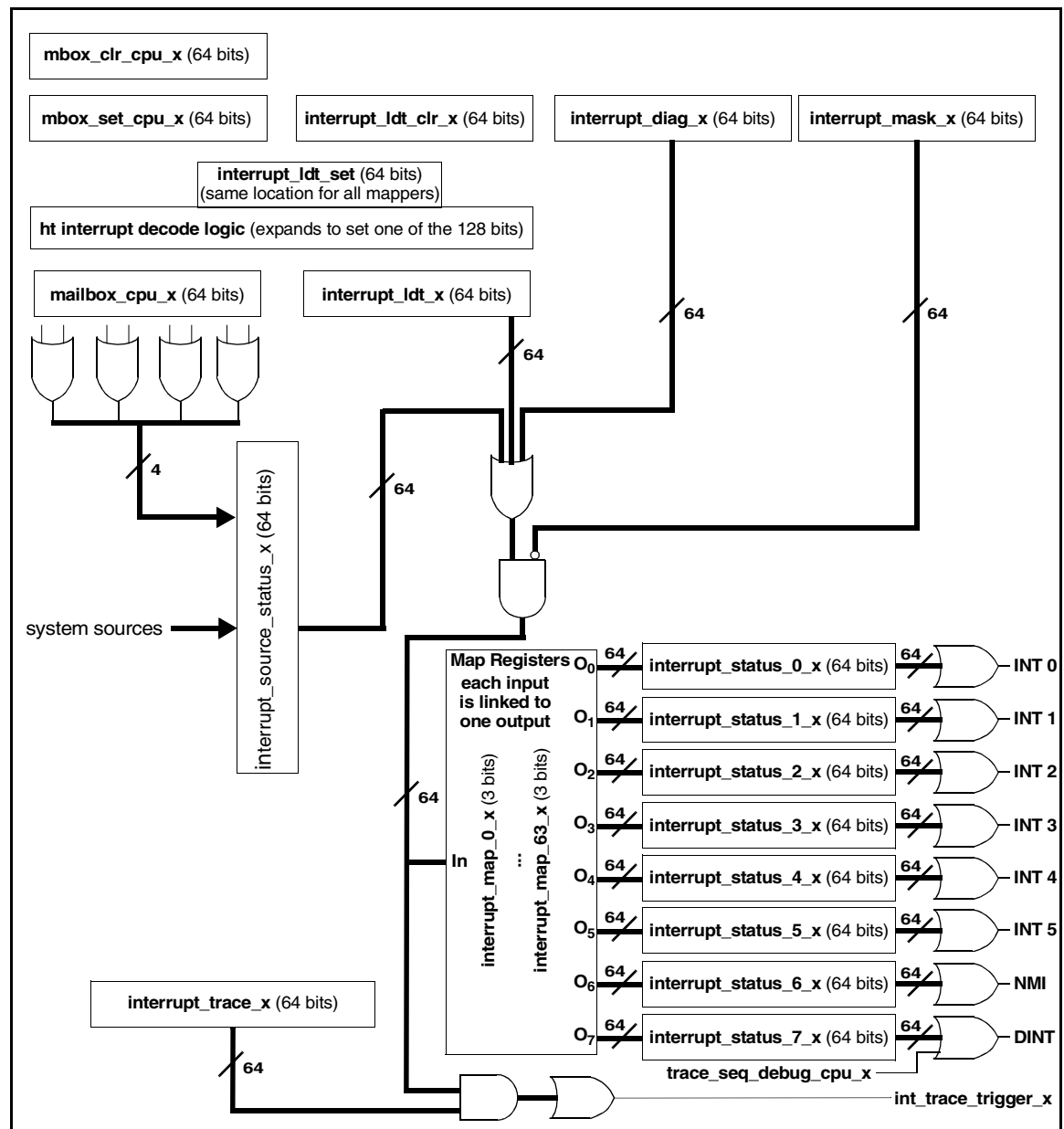


Figure 10: Per-CPU Interrupt Mapper (replicated for each CPU; x = 0 or 1)

The main interrupt registers are all 64 bits wide:

**Table 21: Interrupt Registers**

Name	Description
<b>interrupt_diag</b> _0 - 00_1002_0010 _1 - 00_1002_2010	Setting a bit in this register raises the corresponding interrupt. Software must clear the bit to remove the interrupt. This register is intended for diagnostics only.
<b>interrupt_ldt_set</b> 00_1002_0048	This write only register is written with a HyperTransport interrupt message and will decode it into setting a single bit in either the interrupt_ldt or mailbox register, and thus an interrupt will be raised.
<b>interrupt_ldt</b> _0 - 00_1002_0018 _1 - 00_1002_2018	The HyperTransport interrupt controller will set the bit corresponding to the interrupt vector number for vectors in the range 0-63. The CPU can read this register to find which HyperTransport interrupts have been posted.
<b>interrupt_ldt_clr</b> _0 - 00_1002_0020 _1 - 00_1002_2020	This is not a register. Writing a one to a bit in this location clears the corresponding bit in the interrupt_ldt register.
<b>interrupt_mask</b> _0 - 00_1002_0028 _1 - 00_1002_2028	Setting a bit in this register masks the corresponding interrupt. All bits in this register are set at reset, masking out all interrupts.
<b>interrupt_status_n</b> n=0-7 _0 - 00_1002_0100..138 _1 - 00_1002_2100..138	Read Only. There is a status register for each of the outputs from the mapper. A bit will be set only if the corresponding source is interrupting, is not masked and is mapped to this output.
<b>interrupt_source_status</b> _0 - 00_1002_0040 _1 - 00_1002_2040	Read Only. A bit will be set if the corresponding source is interrupting regardless of the state of the interrupt mask.
<b>interrupt_trace</b> _0 - 00_1002_0038 _1 - 00_1002_2038	Setting a bit in this register allows the interrupt to trigger the trace unit.
<b>interrupt_map0</b> ... <b>interrupt_map63</b> _0 - 00_1002_0200..3F8 _1 - 00_1002_2200..3F8	There are 64 interrupt mapper registers. Each is 3 bits wide and encodes the mapping from the interrupt number given in <a href="#">Table 22</a> onto the CPU interrupt lines as described in <a href="#">Table 18</a> .

**Table 22: Interrupt Sources**

Number	Name	Description	Method to clear
0	watchdog_timer_int_0	Watchdog timer 0 timeout	Interrupt is raised on first timeout of the timer (the second will reset the machine). It is cleared by writing to the timer configuration register. See <a href="#">Section: "Watchdog Timers" on page 57</a> .
1	watchdog_timer_int_1	Watchdog timer 1 timeout	Interrupt is raised on first timeout of the timer (the second will reset the machine). It is cleared by writing to the timer configuration register. See <a href="#">Section: "Watchdog Timers" on page 57</a> .
2	timer_int_0	General timer 0	Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See <a href="#">Section: "General Timers" on page 58</a> .
3	timer_int_1	General timer 1	Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See <a href="#">Section: "General Timers" on page 58</a> .
4	timer_int_2	General timer 2	Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See <a href="#">Section: "General Timers" on page 58</a> .



Table 22: Interrupt Sources (Cont.)

Number	Name	Description	Method to clear
5	timer_int_3	General timer 3	Interrupts when timer reaches zero, cleared by a write to the timer configuration register. See <a href="#">Section: "General Timers" on page 58.</a>
6	smb_int_0	SMBus 0 interrupt	Interrupts when transfer completes or an error is signalled by SMBus unit 0. Cleared by reading status register or writing the error bit. See <a href="#">Section: "Programming Model" on page 409.</a>
7	smb_int_1	SMBus 1 interrupt	Interrupts when transfer completes or an error is signalled by SMBus unit 1. Cleared by reading status register or writing the error bit. See <a href="#">Section: "Programming Model" on page 409.</a>
8	uart_int_0	UART channel A interrupt	Interrupts from UART channel A are merged into this one signal, cleared as defined in <a href="#">Section: "Interrupts" on page 324.</a>
9	uart_int_1	UART channel B interrupt	Interrupts from UART channel B are merged into this one signal, cleared as defined in <a href="#">Section: "Interrupts" on page 324.</a>
10	ser_int_0	Synch serial 0 interrupt	DMA status and errors from the synchronous serial interface are combined into this interrupt, cleared as defined in <a href="#">Section: "Synchronous Serial Interrupts" on page 351.</a>
11	ser_int_1	Synch serial 1 interrupt	DMA status and errors from the synchronous serial interface are combined into this interrupt, cleared as defined in <a href="#">Section: "Synchronous Serial Interrupts" on page 351.</a>
12	pcmcia_int	PCMCIA controller interrupt	There are three sources for interrupt from the PCMCIA controller, they are cleared by reading the <b>pcmcia_status</b> register as described in <a href="#">Section: "Using The PCMCIA Card" on page 389.</a>
13	addr_trap_int	Address trap interrupt	Raised by any of the address traps counting from 1 to 0. Cleared by reading the <b>addr_trap_reg</b> . See <a href="#">Section: "Address Trapping" on page 67.</a>
14	perf_cnt_int	System performance counter interrupt	Raised by any of the system performance counters reaching its maximum count. Cleared by clearing the counters.
15	trace_freeze_int	Trace buffer frozen interrupt	This interrupt is raised when the trace buffer is frozen, and cleared when the buffer is reset. See <a href="#">Section: "Trigger Sequences" on page 73.</a>
16	bad_ecc_int	Uncorrectable ECC error or Bus Error	This interrupt is raised by the bus watcher when a ZBbus data transfer is marked as having an uncorrectable ECC error or some other form of fatal error. This indicates an uncorrectable ECC from either the L2 cache or memory, or a fatal error from one of the other agents. It is cleared by a read of the <b>bus_err_status</b> register.
17	cor_ecc_int	Correctable ECC error	This interrupt is raised by the bus watcher when a ZBbus data transfer is marked as having a corrected ECC error. This indicates an corrected ECC from either the L2 cache or memory, or a non-fatal error from one of the other agents. It is cleared by a read of the <b>bus_err_status</b> register.
18	io_bus_int	Generic bus error: illegal address, timeout waiting ACK, parity error	Raised by an error condition in the generic bus logic. The access that caused the error is logged (see the Generic Bus description). The interrupt is cleared by reading the <b>io_int_status</b> register.

**Table 22: Interrupt Sources (Cont.)**

Number	Name	Description	Method to clear
19	mac_0_int	MAC 0 interrupt	Raised by an error in the MAC or when any of the four DMA channels need service. Cleared by reading the <b>mac_status_0</b> register or servicing the DMA interrupt.
20	mac_1_int	MAC 1 interrupt	Raised by an error in the MAC or when any of the four DMA channels need service. Cleared by reading the <b>mac_status_1</b> register or servicing the DMA interrupt.
21	mac_2_int	MAC 2 interrupt	Raised by an error in the MAC or when any of the four DMA channels need service. Cleared by reading the <b>mac_status_2</b> register or servicing the DMA interrupt.
22	dm_ch_0_int	Data Mover channel 0 interrupt	Raised at the end of data transfer in channel 0 if a data mover descriptor has the interrupt bit set. Cleared by reading the <b>dm_dscr_base_0</b> register.
23	dm_ch_1_int	Data Mover channel 1 interrupt	Raised at the end of data transfer in channel 1 if a data mover descriptor has the interrupt bit set. Cleared by reading the <b>dm_dscr_base_1</b> register.
24	dm_ch_2_int	Data Mover channel 2 interrupt	Raised at the end of data transfer in channel 2 if a data mover descriptor has the interrupt bit set. Cleared by reading the <b>dm_dscr_base_2</b> register.
25	dm_ch_3_int	Data Mover channel 3 interrupt	Raised at the end of data transfer in channel 3 if a data mover descriptor has the interrupt bit set. Cleared by reading the <b>dm_dscr_base_3</b> register.
26	mbox_int_0	Mailbox bits [63:48] non zero	Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.
27	mbox_int_1	Mailbox bits [47:32] non zero	Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.
28	mbox_int_2	Mailbox bits [31:16] non zero	Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.
29	mbox_int_3	Mailbox bits [15:0] non zero	Interrupts when another CPU or the HyperTransport interrupt controller sets any bit, cleared when the CPU clears all the bits.
30	cycle_cp0_int	ZBbus Cycle Count Match 0	Interrupts when the ZBbus cycle count matches compare register 0. Cleared by any write to <b>zbbus_cycle_cp0</b> .
31	cycle_cp1_int	ZBbus Cycle Count Match 1	Interrupts when the ZBbus cycle count matches compare register 1. Cleared by any write to <b>zbbus_cycle_cp1</b> .
32	gpio_int_0	GPIO pin 0 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
33	gpio_int_1	GPIO pin 1 interrupt	
34	gpio_int_2	GPIO pin 2 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
35	gpio_int_3	GPIO pin 3 interrupt	



Table 22: Interrupt Sources (Cont.)

Number	Name	Description	Method to clear
36	gpio_int_4	GPIO pin 4 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
37	gpio_int_5	GPIO pin 5 interrupt	
38	gpio_int_6	GPIO pin 6 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
39	gpio_int_7	GPIO pin 7 interrupt	
40	gpio_int_8	GPIO pin 8 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
41	gpio_int_9	GPIO pin 9 interrupt	
42	gpio_int_10	GPIO pin 10 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
43	gpio_int_11	GPIO pin 11 interrupt	
44	gpio_int_12	GPIO pin 12 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
45	gpio_int_13	GPIO pin 13 interrupt	
46	gpio_int_14	GPIO pin 14 interrupt	Interrupts when external source raises an interrupt. If level sensitive, the external source must clear the interrupt. If edge triggered the <b>gpio_clr_edge</b> register must be written to clear the interrupt. Generation of these two interrupts may be disabled in the <b>gpio_int_type</b> register to free these vectors for use by HyperTransport interrupts.
47	gpio_int_15	GPIO pin 15 interrupt	
48	ldt_fatal_int	HyperTransport fatal error interrupt	This bit is set when a fatal error is detected by the HyperTransport interface. The HyperTransport Error Control register is used to classify the HyperTransport errors. Software must clear all fatal error status bits to clear the interrupt.
49	ldt_nonfatal_int	HyperTransport nonfatal error interrupt	This bit is set when a non fatal error is detected by the HyperTransport interface. The HyperTransport Error Control register is used to classify the HyperTransport errors. Software must clear all nonfatal error status bits to clear the interrupt.
50	ldt_smi	HyperTransport signaled SMI	This bit is reserved in the system interrupt register, and comes only from the <b>ldt_interrupt</b> register. This bit is set when an SMI interrupt packet directed to this CPU had been received from the HyperTransport bus. This bit is cleared using the interrupt clear register. HyperTransport interrupts are directed, so the source for this bit differs for each interrupt controller.

**Table 22: Interrupt Sources (Cont.)**

Number	Name	Description	Method to clear
51	ldt_nmi	HyperTransport signaled NMI	This bit is reserved in the system interrupt register, and comes only from the <b>ldt_interrupt</b> register. This bit is set when an NMI interrupt packet directed to this CPU had been received from the HyperTransport bus. This bit is cleared using the interrupt clear register. HyperTransport interrupts are directed, so the source for this bit differs for each interrupt controller.
52	ldt_init	HyperTransport signaled INIT	This bit is reserved in the system interrupt register, and comes only from the <b>ldt_interrupt</b> register. This bit is set when an INIT interrupt packet directed to this CPU had been received from the HyperTransport bus. This bit is cleared using the interrupt clear register. HyperTransport interrupts are directed, so the source for this bit differs for each interrupt controller.
53	ldt_startup	HyperTransport signaled STARTUP	This bit is reserved in the system interrupt register, and comes only from the <b>ldt_interrupt</b> register. This bit is set when a STARTUP interrupt packet directed to this CPU had been received from the HyperTransport bus. This bit is cleared using the interrupt clear register. HyperTransport interrupts are directed, so the source for this bit differs for each interrupt controller.
54	ldt_ext_int	HyperTransport signaled ExtInt	This bit is reserved in the system interrupt register, and comes only from the <b>ldt_interrupt</b> register. This bit is set when an ExtInt interrupt packet directed to this CPU had been received from the HyperTransport bus. This bit is cleared using the interrupt clear register. HyperTransport interrupts are directed, so the source for this bit differs for each interrupt controller.
55	pci_error_int	PCI error interrupt	This bit is set when an error is seen on the PCI bus. It can be caused by PCI parity errors, target-abort, master-abort, timeout transfer abort or a serious system error signalled on the SERR pin. The cause can be read from the PCI interface configuration space (bus=0, dev=0, function=0) Status register (bits 8, 11, 12, 13 and 14), and the PCI Additional Status and Control register. Software must clear all the error status bits to clear the interrupt.
56	pci_inta_int	PCI interrupt A	This bit is set when an interrupt is signalled on the PCI inta line.
57	pci_intb_int	PCI interrupt B	This bit is set when an interrupt is signalled on the PCI intb line.
58	pci_intc_int	PCI interrupt C	This bit is set when an interrupt is signalled on the PCI intc line.
59	pci_intd_int	PCI interrupt D	This bit is set when an interrupt is signalled on the PCI intd line.
60	spare_int	Spare interrupt	There is no system interrupt matching this bit. It can be used for vectored HyperTransport interrupts.
61	mac_0_ch1_int	MAC 0 Channel 1 interrupt	Raised when channel 1 transmit or receive DMA engine needs service. Cleared by any read to <b>mac_status1_0</b> .
62	mac_1_ch1_int	MAC 1 Channel 1 interrupt	Raised when channel 1 transmit or receive DMA engine needs service. Cleared by any read to <b>mac_status1_1</b> .
63	mac_2_ch1_int	MAC 2 Channel 1 interrupt	Raised when channel 1 transmit or receive DMA engine needs service. Cleared by any read to <b>mac_status1_2</b> .

## TIMERS

There are two types of timers on the part: watchdog and general. Watchdog timers can be used to detect the liveness of the system and will cause a full system reset if software does not prove its correct operation by regularly resetting the timer. General timers are available for any use and can be set to generate a regular interrupt or act as one shot timers. Both sorts of timer count in 1 us steps and have a maximum timeout of over 8 s.

The timers all have a similar programming interface. Each timer module consists of an initial count register, a down counter which is loaded with the initial count and then decrements, and a configuration register. The initial count register should only be written while the timer is disabled. Whenever the timer enable bit in the control register is written with a 1 the down counter is loaded with the initial count and begins decrementing. When the counter reaches zero an interrupt is raised.

### WATCHDOG TIMERS

The watchdog timers have a 23 bit counter and are clocked by a 1 MHz clock derived from the 100 MHz reference clock. This allows a maximum count of 8388608, giving the maximum timeout of 8.3888608 s.

A watchdog timer is normally not permitted to timeout. System software should regularly re-enable the timer by writing a 1 to the enable bit in the configuration register. This will restore the count to the initial count and force the counter to restart decrementing from there. If the timer is permitted to reach zero then on the next clock tick an interrupt will be raised, the counter will reload from the initial count and decrementing will continue. This interrupt warns the system software that it has not been re-enabling the timer. The interrupt will be cleared by any write to the configuration register. If the timer times out for a second time (i.e. before the interrupt has been cleared) the default behavior of the watchdog is to signal the system controller to perform a full reset of the part (and any external devices that take reset from the RESETOUT\_L, LDT\_RESET\_L/HT\_RESET\_L or PCI P\_RST\_L signals). Once the watchdog has signalled a reset it will be disabled.

There are two watchdog timers on the part. They are identical other than using different registers (distinguished by appending register names with `_0` and `_1`) and different interrupt lines. On the BCM1250 this allows the liveness of each processor to be separately monitored. This will be of benefit when the two CPUs are running loosely coupled (for example when one runs a real-time task and the other does general system control). The interrupt mapper can be used to route a timeout interrupt from either watchdog to either or both processors (this allows each of them to monitor the other).

The watchdog can be configured to do a more selective reset when it times out for the second time. It can send an `sb_softres` (as described in [Section: "System Control" on page 41](#)), reset just one CPU, or (on the BCM1250) reset both CPUs but no peripherals. Care must be taken if this is done, the reset software will need to probe for devices that are hung or in inconsistent states. The behavior should only be changed from the default in high availability systems or in some cases where the two CPUs are running distinct operating systems.

Care must be taken in systems that boot from an SMBus EEPROM, none of the selective resets set the SMBus channel to boot operation if it has been returned to normal operation. An additional problem with resetting only a CPU is that it will immediately be removed from the coherence machinery. It is possible that the CPU had just been detected as exclusive owner of a cache block and was in the process of providing it when reset. Since the data will never be supplied the ZBbus transaction will never be terminated and the requester will hang.



## GENERAL TIMERS

The part has four general timers. Each has a 23 bit counter and is clocked by a 1 MHz clock derived from the 100 MHz reference clock. This allows a maximum count of 8388608, giving the maximum one-shot timeout of 8.388608 s, and a maximum periodic interrupt every 8.388608 s. The timers can be either set to count down from the initial count to zero and stop, or repeatedly count down. A timer should always be disabled before its operating mode is changed.

When the timer is enabled in the one-shot mode the counter is loaded from the initial count register and will decrement to zero. When the count reaches 0, the next clock tick causes an interrupt to be generated and the timer is disabled. The count will remain at 0 until the timer is re-enabled by software. The interrupt is cleared by any write to the configuration register, regardless of whether the timer is kept in the disabled state or enabled. If the timer is re-enabled before its counter reaches 0, its counter is reloaded with the initial count and will restart decrementing.

In the second mode, the counter is loaded from the initial count and will decrement to zero. When the counter reaches zero the next clock tick cause an interrupt to be generated and the counter to be reloaded with the initial count and start counting down again. When the counter is in this mode it will not be reloaded if it is re-enabled (but it will if it is disabled and then enabled again). The interrupt is cleared by any write to the configuration register, if regular interrupts are required this write should re-enable the counter. There is no detection of missed interrupts, once the interrupt has asserted it will only be raised again after it has been cleared and then the counter reaches zero.

## TIMER SPECIAL CASES

If the initial count of any timer (regardless of mode) is set to zero and the timer is enabled then an interrupt will be signalled every clock tick. It is UNPREDICTABLE if writing the configuration register will cause the interrupt to deassert. It is UNPREDICTABLE how many ticks a watchdog timer will take to assert reset.

If the initial count register is updated or the mode is changed between one-shot and continuous while the counter is enabled the behavior is UNPREDICTABLE.

## ZBBUS CYCLE COUNT AND COMPARE

There is an additional read only counter that is initialized to zero and increments every ZBbus cycle. Even in the fastest parts it will take more than 1000 years for the counter to wrap. It can therefore be used to provide a sequence number that is unique and monotonically increasing through the runtime of the system. This is useful for software that needs to assign an order to events. The counter may also be used as a high resolution clock for timing. The read only **zbbus\_cycle\_count** register is placed at the start of a 64KB memory range, if additional registers are added in this range they will be Read Only with no side effects. This range can therefore be safely mapped into user address space. There are two comparison registers which will raise an interrupt when the count matches the comparison value. The interrupt will remain asserted until the compare register is written. Note that the interrupt is only raised on equality, so writing the compare register with a value lower than the current count will ensure the interrupt is never raised. Since cycle 0 is guaranteed to have passed by the time the CPU executes the first instruction, writing the compare register with zero is always a safe way to clear the interrupt and disable it.

## TIMER REGISTERS

**Table 23: Watchdog Timer Initial Count Registers**

watchdog_timer_init_cnt_0 - 00_1002_0050 watchdog_timer_init_cnt_1 - 00_1002_0150			
Bits	Name	Default	Description
22:0	watchdog_timer_init_cnt	23'bx	Watchdog timer initial count register. Sets the watchdog timeout time in microseconds. This register should only be written when the timer is disabled. If written when the timer is enabled the results are UNPREDICTABLE.
63:23	reserved	41'h0	Reserved

**Table 24: Watchdog Timer Current Count Registers**

watchdog_timer_cnt_0 - 00_1002_0058 watchdog_timer_cnt_1 - 00_1002_0158 <b>READ ONLY</b>			
Bits	Name	Default	Description
22:0	watchdog_timer_cnt	23'bx	Watchdog timer current count register. When the counter is enabled the count decrements every microsecond.
63:23	reserved	41'h0	Reserved

**Table 25: Watchdog Timer Configuration Registers**

watchdog_timer_cfg_0 - 00_1002_0060 watchdog_timer_cfg_1 - 00_1002_0160 <b>Write clears interrupt</b>			
Bits	Name	Default	Description
0	watchdog_timer_enable	1'b0	When this bit is written with a 0 the timer will be disabled. When this bit is written with a 1 regardless of its previous state the timer will be loaded from the initial count and start decrementing.
1	reserved	1'b0	Reserved
4:2	reset_type	3'b0	This field sets the type of reset that is generated when the watchdog times out for the second time. In most situations the (default) full system reset is the correct behavior, since this will clear all state. xx0: Full system reset (Default). 001: SB Soft Reset (full reset retaining <b>system_cfg</b> register value). 011: Reset only CPU 0. 101: Reset only CPU 1. 111: Reset only CPU 0 and CPU 1 (no peripherals).
5	wd_has_reset	1'bx	This bit is reserved prior to PERIPH_REV3. On PERIPH_REV3 and later it indicates if the watchdog timer has caused a reset. Following power up the value of this bit is UNPREDICTABLE. Whenever this register is written this bit becomes zero. If the watchdog causes a reset this bit is set.
7:6	reserved	2'b0	Reserved
63:8	notimp	56'bx	Not Implemented.



**Table 26: General Timer Initial Count Registers**

general_timer_init_cnt_0 - 00_1002_0070 general_timer_init_cnt_1 - 00_1002_0078 general_timer_init_cnt_2 - 00_1002_0170 general_timer_init_cnt_3 - 00_1002_0178			
Bits	Name	Default	Description
22:0	timer_init_cnt	23'bx	Timer initial count register. This sets the maximum value of the counter, and will be one less than the timeout in microseconds for one-shot mode, and one less than the period in microseconds in continuous mode. This register should only be written when the timer is disabled. If written when the timer is enabled the results are UNPREDICTABLE.
63:23	reserved	41'h0	Reserved

**Table 27: General Timer Current Count Registers**

general_timer_cnt_0 - 00_1002_0080 general_timer_cnt_1 - 00_1002_0088 general_timer_cnt_2 - 00_1002_0180 general_timer_cnt_3 - 00_1002_0188 READ ONLY			
Bits	Name	Default	Description
22:0	timer_cnt	23'bx	Timer current count register. When the timer is running the count decrements every microsecond.
63:23	reserved	41'h0	Reserved

**Table 28: General Timer Configuration Registers**

general_timer_cfg_0 - 00_1002_0090 general_timer_cfg_1 - 00_1002_0098 general_timer_cfg_2 - 00_1002_0190 general_timer_cfg_3 - 00_1002_0198 Write clears interrupt			
Bits	Name	Default	Description
0	general_timer_cfg_enable	1'b0	When high, enable the general purpose timer. If the timer is disabled when this bit is written with a 1 the timer will be loaded from the initial count register and will start decrementing. If the timer is enabled when this bit is written with a 1 the timer continues to decrement, but if the general_timer_cfg_mode bit is 0 the counter will also be reloaded with the initial count. If this bit is written with a 0 the timer will stop decrementing.
1	general_timer_cfg_mode	1'b0	When low, the general purpose timer stops at 0. When high, the general purpose timer runs continuously.
7:2	reserved	6'h0	Reserved
63:8	notimp	56'bx	Not Implemented.



**Table 29: ZBbus Count Register**

<b>zbbus_cycle_count - 00_1003_0000</b>			
<b>READ ONLY</b>			
<b>Note 00_1003_0008 - 00_1003_FFFF is safe for user space access</b>			
Bits	Name	Default	Description
63:0	count	64'h0	Count of ZBbus cycles since reset. Writes will be ignored.

**Table 30: ZBbus Count Compare Registers**

<b>zbbus_cycle_cp0 - 00_1002_0C00</b>			
<b>zbbus_cycle_cp1 - 00_1002_0C08</b>			
Bits	Name	Default	Description
63:0	compare	64'hFFFF_FFFF _FFFF_FFFF	Value is compared to zbbus_cycle_count and the interrupt corresponding to the register is raised when the count is equal to the value. The interrupt is cleared by any write to the compare register.

## SYSTEM PERFORMANCE COUNTERS

There are four system performance counters. They count system events and interrupt when saturated. All four counters are cleared and/or enabled simultaneously. When one counter saturates, all counters are frozen. Thus, the readings across the four counters cover the same interval. The counters are writable; if they are written while active the count will continue from the new value (note that the time from the CPU issuing a store to the counter getting the new value is UNPREDICTABLE).

The counters are all controlled by the **perf\_cnt\_cfg** register. This is used to configure the count source for each counter, to enable or disable all the counters and to clear all the counters. The counter clear bit is write only, the counters will all be cleared to zero at the end of the cycle that this bit is written with a 1. If both the enable and clear bits are written with 1 in the same write, the counters are cleared and will start counting the cycle after the write completes.

If any counter reaches its maximum count the counters will all be disabled (the enable bit in the **perf\_cnt\_cfg** register is cleared) and the **perf\_cnt\_int** interrupt will be raised. The interrupt is removed and all the counters are cleared when the clear bit in the **perf\_cnt\_cfg** register is written with a 1.

Since it takes a cycle to disable the counters there is one additional cycle sampled after one (or more) of the counts reaches **FF\_FFFF\_FFFF** during which the counter may overflow (if its source is still active). There is an additional bit in the performance counters that gets set when they overflow, thus the counter (or counters) that disabled monitoring could have the value **FF\_FFFF\_FFFF** or **100\_0000\_0000** when the interrupt is seen.

**Table 31: System Performance Counter Configuration Registers**

<b>perf_cnt_cfg - 00_1002_04C0</b>			
Bits	Name	Default	Description
7:0	Counter 0 source	8'b0	Sets the source for counter 0.
15:8	Counter 1 source	8'b0	Sets the source for counter 1.
23:16	Counter 2 source	8'b0	Sets the source for counter 2.

**Table 31: System Performance Counter Configuration Registers (Cont.)**

perf_cnt_cfg - 00_1002_04C0			
Bits	Name	Default	Description
31:24	Counter 3 source	8'b0	Sets the source for counter 3.
32	clear	1'b0	If this bit is set in a write to the register the counters will be reset, the bit self-clears and always reads as zero.
33	enable	1'b0	Set to enable the counters, clear to stop them. Cleared automatically by any counter reaching its maximum count.
63:34	reserved	30'b0	Reserved

**Table 32: System Performance Counters**

perf_cnt_0 - 00_1002_04D0 perf_cnt_1 - 00_1002_04D8 perf_cnt_2 - 00_1002_04E0 perf_cnt_3 - 00_1002_04E8			
Bits	Name	Default	Description
39:0	Count	40'bx	Performance counter.
40	oflow	1'b0	This bit is set if the counter overflows.
63:41	reserved	23'b0	Reserved

**Table 33: System Performance Counter Sources**

Value	Condition Counted
00	No count.
01	Bus cycles are counted. The counter is incremented each ZBbus clock cycle (half the CPU clock).
02	Bus cycles used. The counter is incremented every cycle the address portion of ZBbus is used.
03	Address bus cycles waiting grant. The counter is incremented every cycle some agent loses arbitration for the address bus (i.e. there are two or more requesters in arbitration).
04	Address bus cycles arbitrated but not used. The counter is incremented every time the address bus is granted but the agent drives a NOP command.
05	Address match 0 count - counts each bus address cycle that matches the address trap comparator. The occurrence counter (that is used to determine if the address trap should interrupt) is ignored, the performance counter will increase every cycle there is a hit in the trap.
06	Address match 1 count. (See comments for Address match 0.)
07	Address match 2 count. (See comments for Address match 0.)
08	Address match 3 count. (See comments for Address match 0.)
09	Data bus cycles waiting grant. The counter is incremented every cycle some agent loses arbitration for the data bus (i.e. there are two or more requesters in arbitration).
0A	Data bus cycles arbitrated but not used. The counter is incremented every time the data bus is granted but the agent drives a NOP command.
0B	CPU0 EDEN. The counter is incremented every time CPU 0 asserts its External Debug Event Notification (EDEN) signal. This can be asserted by software, or watchpoint register hits.
0C	CPU1 EDEN. The counter is incremented every time CPU 1 asserts its External Debug Event Notification (EDEN) signal. This can be asserted by software, or watchpoint register hits.



**Table 33: System Performance Counter Sources (Cont.)**

Value	Condition Counted
0D	Interrupt request cycles. The counter is incremented every bus cycle the int_trace_trigger_x output of the interrupt mapper is asserted (see <a href="#">Figure 10 on page 51</a> ). This records the time taken from the interrupt assertion to software clearing the interrupt condition. If a single interrupt source is selected and the count is read at the start of the interrupt service routine and cleared when the interrupt is cleared, software will be able to track the interrupt service time. Counters 0 and 1 count based on int_trace_trigger_0, counters 2 and 3 count based on int_trace_trigger_1.
0E	Interrupt request count. The counter is incremented every time the int_trace_trigger_x output of the interrupt mapper changes from not asserted to asserted (see <a href="#">Figure 10 on page 51</a> ). This records the number of interrupts, and can be used with the interrupt request cycles count to determine the average number of cycles between an interrupt being raised and cleared. Counters 0 and 1 count based on int_trace_trigger_0, counters 2 and 3 count based on int_trace_trigger_1.
0F	Reserved
1n	Agent n Request count - counts number of cycles that agent n requested the bus.
2n	Agent n Grant delay count - counts the total sum of cycles between agent n requesting the bus, and agent n receiving grant on the bus.
30	Memory channel 0 read accesses. Counts the number of reads to memory.
31	Memory channel 0 write accesses. Counts the number of writes to memory.
32	Memory channel 0 turnarounds. Counts the number of times the controller changes from read-to-write or write-to-read.
33	Memory channel 0 page hits. Counts the number of page hits in the memory controller.
34	Memory channel 0 controller wait cycles. Counts the number of cycles the controller is waiting for precharge and chip select overhead.
35	Memory channel 1 read accesses. Counts the number of reads to memory.
36	Memory channel 1 write accesses. Counts the number of writes to memory.
37	Memory channel 1 turnarounds. Counts the number of times the controller changes from read-to-write or write-to-read.
38	Memory channel 1 page hits. Counts the number of page hits in the memory controller.
39	Memory channel 1 controller wait cycles. Counts the number of cycles the controller is waiting for precharge and chip select overhead.
3A	L2 read requests. Counts the number of read requests to the L2 cache.
3B	L2 read misses. Counts the number of read requests that missed the L2 cache.
3C	L2 write requests. Counts the number of write requests to the L2 cache.
3D	L2 write misses. Counts the number of write requests that missed the L2 cache.
3E	L2 evicts. Counts the number of lines evicted from the L2 cache.
3F-FF	Reserved

## BUS WATCHER

The Bus Watcher monitors ZBbus data transfers, detecting error reports. Corrected ECC errors from memory and the L2 cache are counted. Uncorrectable ECC errors are counted and logged. The bus watcher will raise an interrupt when it detects an error.

When a data bus transaction is made with an error code the bus watcher logs the transaction (in the **bus\_err\_data** registers and the **bus\_err\_status** register), and increments the appropriate counter. If the error was an unrecoverable one (uncorrectable ECC error or bus error or fatal error) the log is frozen until the **bus\_err\_status** register is read. (The unrecoverable errors are Dcode=4-7).

The bus watcher contains the following counters:

**Table 34: Bus Watcher Counters**

Name	Use
bus_l2_cor_d_ecc	Counts Correctable L2 cache data ECC errors. The cor_ecc_int interrupt is raised every time this counter increments.
bus_l2_bad_d_ecc	Counts Uncorrectable L2 cache data ECC errors. The bad_ecc_int interrupt is raised every time this counter increments.
bus_l2_cor_t_ecc	Counts Correctable L2 cache tag ECC errors. The cor_ecc_int interrupt is raised every time this counter increments.
bus_l2_bad_t_ecc	Counts Uncorrectable L2 cache tag ECC errors. The bad_ecc_int interrupt is raised every time this counter increments.
bus_mem_cor_d_ecc	Counts Correctable memory data ECC errors. The cor_ecc_int interrupt is raised every time this counter increments.
bus_mem_bad_d_ecc	Counts Uncorrectable memory data ECC errors. The bad_ecc_int interrupt is raised every time this counter increments.
bus_error	Counts fatal errors and bus errors. The bad_ecc_int interrupt is raised every time this counter increments.

Each counter is 8 bits, and saturates at 8'hff. Software may write any value into the counters, a counter is cleared by writing zero to it. The counters are collected into two 32-bit registers, **bus\_l2\_errors** for the L2 cache counts and **bus\_mem\_io\_errors** for the other counts. These registers should only be written with 32-bit or 64-bit writes, using smaller will result in the value of some of the counters becoming UNPREDICTABLE.

In addition to the type of error the **bus\_err\_status** register records information about the transaction that encountered the error. The upper four bits of the transaction id can be used to determine which of the ZBbus agents initiated the transaction that resulted in an error, see [Table 1 on page 20](#) for the number of each agent. In some cases the lower six bits of the transaction id can be decoded to get more information about the request, see [Table 51 on page 81](#). The responder id gives the number of the agent that reported the error. The data that was returned with the error code is also captured, but in many cases it will be of UNPREDICTABLE value.

Note that when a bus error or fatal error is returned to a CPU it will take a bus error exception and when an uncorrectable ECC error is returned it will take a cache error exception. These exceptions are taken with a higher priority than the interrupt from the bus watcher, or any interrupt raised by the source of the error. DMA engines that receive any of these errors will stop and report the error in some way. If any of these errors are returned to the PCI or HyperTransport interfaces they will be converted into the appropriate error returned on the interface and flags will be set in the CSRs to indicate this has been done. Thus an error is likely to be reported several times, ensuring that processing is never done based on corrupted data. The correctable



errors do not raise exceptions or cause the DMA engines to stop, so the Bus Watcher interrupt is likely to be the only way they are reported.

**Table 35: Bus Watcher Error Status Register**

<b>bus_err_status - 00_1002_0880</b>			
<b>READ ONLY, Read will clear all fields and the cor_ecc_int and bad_ecc_int interrupts</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
7:0	reserved	8'b0	Reserved
17:8	tid	10'b0	Transfer ID that had error code. Top 4 bits indicate the initiator of the transfer, see <a href="#">Table 1 on page 20</a> for the mapping. Low 6 bits may give more details on the what within the agent caused the transaction, see <a href="#">Table 51 on page 81</a> .
21:18	rid	4'b0	Responder ID identifies the agent that reported the error. See <a href="#">Table 1 on page 20</a> .
24:22	dcode	3'b0	Data transfer error code. See <a href="#">Table 6 on page 24</a> .
29:25	reserved	5'b0	Reserved
30	mult_errors	1'b0	Set to indicate multiple errors. The first fatal one, or the most recent non-fatal one is logged (all were counted).
31	reserved	1'b1	Reserved. Reads as 1.
63:32	notimp	32'bx	Not Implemented.

**Table 36: Bus Watcher Error Status Debug Register**

<b>bus_err_status_debug - 00_1002_08D0 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
63:0	value	64'hx	This register contains the same information as the <b>bus_err_status</b> register, but reads do not have any side effects.

**Table 37: Bus Watcher Error Data Registers**

<b>bus_err_data_0 - 00_1002_08A0</b>			
<b>bus_err_data_1 - 00_1002_08A8</b>			
<b>bus_err_data_2 - 00_1002_08B0</b>			
<b>bus_err_data_3 - 00_1002_08B8</b>			
<b>READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
63:0	data	64'hx	Data from error transaction. The register number is address bits [4:3]. Register <b>_0</b> contains ZBbus bits 255:192 Register <b>_1</b> contains ZBbus bits 191:128 Register <b>_2</b> contains ZBbus bits 127:64 Register <b>_3</b> contains ZBbus bits 63:0

**Table 38: Bus Watcher L2 ECC Counter Register**

<b>bus_l2_errors - 00_1002_08C0</b> Should only be written with 32-bit or 64-bit write			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
7:0	l2_cor_d_ecc	8'b0	Count of correctable L2 cache data errors, saturates at 8'hff, write the register with zero to clear the count.
15:8	l2_bad_d_ecc	8'b0	Count of uncorrectable L2 cache data errors, saturates at 8'hff, write the register with zero to clear the count.
23:16	l2_cor_t_ecc	8'b0	Count of correctable L2 tag errors, saturates at 8'hff, write the register with zero to clear the count.
31:24	l2_bad_t_ecc	8'b0	Count of uncorrectable L2 tag errors, saturates at 8'hff, write the register with zero to clear the count.
63:32	notimp	32'bx	Not Implemented.

**Table 39: Bus Watcher Memory and I/O Error Counter Register**

<b>bus_mem_io_errors - 00_1002_08C8</b> Should only be written with 32-bit or 64-bit write			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
7:0	mem_cor_d_ecc	8'b0	Count of correctable memory data errors, saturates at 8'hff, write the register with zero to clear the count.
15:8	mem_bad_d_ecc	8'b0	Count of uncorrectable memory data errors, saturates at 8'hff, write the register with zero to clear the count.
23:16	bus_error	8'b0	Count of bus errors and fatal bus errors, saturates at 8'hff, write the register with zero to clear the count.
31:24	reserved	8'b0	Reserved
63:32	notimp	32'bx	Not Implemented.



## ADDRESS TRAPPING

There are four address traps in the SCD. An interrupt is raised by any access that falls within the address range specified by a trap. Hits to a trap can also be used to trigger the performance counters and trace unit. The address traps are mainly used as a debugging aid, but they can also be used to detect (but not prevent) illegal accesses in the event of a system failure.

Each address trap has the same interface consisting of a set of registers. The register names described should have **\_0**, **\_1**, **\_2**, or **\_3** appended to indicate which trap is being used.

A 40 bit upper address register **addr\_trap\_up** sets the top physical address in the trapping range. An address must be less than or equal to this value to be part of the range. A 40 bit lower address register **addr\_trap\_down** sets the base physical address in the trapping range. An address must be greater than or equal to this value to be part of the range.

The fields in the **addr\_trap\_cfg** register are used to make the trap more selective than just based on an address range. A two bit **access\_type** field selects if the trap is active for read, write or all accesses. The **source\_id** field contains a ZBbus source id that is compared at the same time as the address. Flags determine if the trap is hit when the access is inside or outside the address range, and if the source id must match or differ.

Each trap has a 3 bit occurrence counter in the **addr\_trap\_cfg** register that counts down every time the trap condition is hit. Once the occurrence counter is zero it will stick at zero and not continue to decrement.

A simple trap ignores the source information. It is setup by programming the upper and lower address registers to set the range, then writing the configuration register with the access type to trap on, and a greater than 0 occurrence count. Any access of the selected type with an address greater than or equal to the lower register and less than or equal to the upper register matches the range. Thus to trap on a specific address the upper and lower addresses are set to be the same. When an access matches and the occurrence counter is greater than zero, the counter will decrement.

Provided the address trap interrupt is clear, any address trap match will log the address that matched in the **addr\_trap\_reg** and the trap number in the **addr\_trap\_index**. The first time an occurrence counter from any trap decrements from 1 to 0 the address trap interrupt is raised, both signalling the event and preventing the log registers from being overwritten. Further trap matches will continue to decrement their counters, but will otherwise be ignored until the interrupt is cleared by a read of the **addr\_trap\_reg**. The CPU should read the **addr\_trap\_index** before the **addr\_trap\_reg** to ensure a consistent result.

More complex traps can be made by inverting the output of the range match (the trap is only hit by addresses outside the range) and considering the source of the access in addition to the address range. The source id that is provided on the ZBbus by the initiator of the transaction (as the transaction ID bits A\_ID[9:6]) is used in the comparison. There are four forms of complex trap:

- 1 Address match, source match. This is used to detect a particular agent accessing a particular memory range.
- 2 Address differs, source match. This will detect an access by a device outside of the range of addresses that it is expected to use.
- 3 Address match, source differs. If a region of memory is reserved for use by a single agent (for example CPU 1) this will detect an access to the region by any other agent. (Note that this provides detection only, not prevention).
- 4 Address differs, source differs. If all of the memory can be used by one agent (for example the master CPU), but all other agents are confined to a particular region, this will detect illegal accesses by other agents. It could be used if only the master CPU were given access to I/O addresses and other agents are confined to memory.

The complex trap can also use the cacheability attributes to determine if the request matches the trap. The trap can be set to trigger if the access is Uncached, Cacheable Coherent or Cacheable Non-coherent. It can also only trigger for requests that do not match the expected cacheability. This can aid debugging by detecting accesses to areas of the address space that are made with the wrong attribute. The complex traps use the occurrence counter in the same way as simple traps.

When an access hits the trap (ignoring the value of the occurrence counter) a signal is provided to the trace trigger logic (described in [Section: "Trigger Events" on page 70](#)) and to the performance counter selector ([Section: "System Performance Counters" on page 61](#)).

**Table 40: Address Trap Trigger Index Register**

addr_trap_index - 00_1002_00B0 READ ONLY			
Bits	Name	Default	Description
3:0	addr_trap_index	4'b0	Trap that triggered the interrupt.
7:4	reserved	4'h0	Reserved

**Table 41: Address Trap Trigger Debug Register**

addr_trap_reg_debug - 00_1002_0460 READ ONLY			
Bits	Name	Default	Description
63:0	value	64'hx	This register contains the same information as the <b>addr_trap_reg</b> register, but reads do not have any side effects.

**Table 42: Address Trap Trigger Address Register**

addr_trap_reg - 00_1002_00B8 READ ONLY, Read clears interrupt			
Bits	Name	Default	Description
4:0	addr_trap	5'h0	Always reads as zero.
39:5	addr_trap	35'h0	Address that triggered the interrupt.
63:40	reserved	24'h0	Reserved

**Table 43: Address Trap Range Top Address Registers**

addr_trap_up_0 - 00_1002_0400 addr_trap_up_1 - 00_1002_0408 addr_trap_up_2 - 00_1002_0410 addr_trap_up_3 - 00_1002_0418			
Bits	Name	Default	Description
39:0	addr_trap_up	40'hx	Top address in range to compare.
63:40	reserved	24'h0	Reserved



**Table 44: Address Trap Range Base Address Registers**

Bits	Name	Default	Description
<b>addr_trap_down_0</b> - 00_1002_0420 <b>addr_trap_down_1</b> - 00_1002_0428 <b>addr_trap_down_2</b> - 00_1002_0430 <b>addr_trap_down_3</b> - 00_1002_0438			
39:0	addr_trap_down	40'hx	Base address in range to compare.
63:40	reserved	24'h0	Reserved

**Table 45: Address Trap Configuration Registers**

Bits	Name	Default	Description
<b>addr_trap_cfg_0</b> - 00_1002_0440 <b>addr_trap_cfg_1</b> - 00_1002_0448 <b>addr_trap_cfg_2</b> - 00_1002_0450 <b>addr_trap_cfg_3</b> - 00_1002_0458			
2:0	addr_trap_cnt	3'b0	Address trap count. Decrements on address range matches, sticking at 0. The 1 to 0 transition will trigger an interrupt. Note that this count is not applied when an address trap hit is used to trigger a trace event.
3	addr_trap_access_type	1'b0	When low, indicates read trap. When high, indicates write trap.
4	all_accesses	1'b0	Ignore the state of bit 3, and trap on any matching access.
5	addr_inv	1'b0	These bits enable complex matches by allowing the source agent of the transaction to be compared, and inverting the sense of both address and source comparisons.
6	use_source	1'b0	Useful values are:
7	source_inv	1'b0	x00: Trap for all sources if address is in the range. x01: Trap for all sources if address is not in the range. 010: Trap if top 4 bits of tid matches source and address is in the range. 011: Trap if top 4 bits of tid matches source and address is outside the range. 110: Trap if top 4 bits of tid does not match source and address is in the range. 111: Trap if top 4 bits of tid does not match source and address is outside the range.
11:8	source_id	4'b0	Agent ID to use in source comparison.
13:12	cattr	2'b00	These bits enable complex matches by allowing the cacheability attribute of the transaction to be compared, and having the trap match if the access has or does not have the expected attribute.
14	cattr_inv	1'b0	Useful values are: x00: Ignore the cacheability attribute. 001: Trap if Uncacheable. 010: Trap if Cacheable Non-coherent. 011: Trap if Cacheable coherent. 101: Trap if cacheable (not Uncacheable). 110: Trap if not Cacheable Non-coherent. 111: Trap if not Cacheable Coherent. The cacheability is decoded from the A_L1CA signal: 2'b00: Cacheable Non-coherent. 2'b01: Cacheable Coherent. 2'b1x: Uncacheable.
63:15	notimp	49'bx	Not Implemented.

## TRACE UNIT

The trace unit allows ZBbus activity to be non-intrusively traced. There is a 12KB buffer into which a trace of the address or data activity on the bus can be written. The trace may be read out either by a CPU or through the JTAG interface. Triggers and filters allow control of the activity that is traced.

The trace buffer contains 256 entries, each of 384 bits. Each entry can hold either three address/control bundles (each 128 bits) or one full bundle containing address/control information (128 bits) and a databus snapshot (256 bits). When a full bundle is captured it will always start a new buffer entry, marking any unused address/control slots in the previous entry empty.

The trace buffer is reset either by software or using the JTAG interface. A trigger is used to start collection of a trace. Once collection is enabled, filters are applied to every bus cycle which select not sampling, taking an address sample or taking a complete sample that cycle (the filter can also select that every cycle is recorded). The buffer pointer will advance modulo 256, thus the trace RAM is used as a circular buffer leaving the latest 256 entries available. A trigger can stop collection, the trace buffer pointer maintains its value and will continue filling the buffer when a start trigger is again encountered. A trigger can also freeze collection. Following this, collection cannot be restarted until the trace buffer is reset. An option can be set to freeze collection the first time the buffer fills, this turns the circular buffer into a linear buffer.

The same mechanism is used to construct both triggers and filters, they only differ in the flags that are set in the control register. There are two components: events and sequences. A trigger event is signalled when the cycle on the ZBbus (or some other system state) matches a condition. Up to four events are concatenated to make a sequence. The sequence also includes an action, which is one of the trace buffer commands (start, stop or freeze) and an indication if the trace buffer should record information from the ZBbus cycle that is in progress as the sequence completes. When all the events in the sequence have been signalled sequentially the sequence is complete and the action associated with the sequence will take place.

## TRIGGER EVENTS

A trigger event is specified as a conditional match on the ZBbus address phase signals, or a match on ZBbus data phase signals, or the external debug pin being asserted, or a CPU interrupt being raised. There are eight trigger events (event0 - event7), all identical.

The address phase match consists of three parts, all of which must be true for the trigger to happen:

- 1 True if  
the req\_id\_match bit is set and the transaction id source on the bus matches the req\_id field.  
Always true if the req\_id\_match bit is clear.
- 2 True if  
the addr\_match0 bit is set and the address on the bus matches the address trap0 range,  
or the addr\_match1 bit is set and the address on the bus matches the address trap1 range,  
or the addr\_match2 bit is set and the address on the bus matches the address trap2 range,  
or the addr\_match3 bit is set and the address on the bus matches the address trap3 range.  
Always true if none of the addr\_match bits are set.  
The complex (source and cache attribute) address trap match is used to determine a hit, but the trigger does not use the count in the address trap.
- 3 True if  
the read bit is set and the bus command is READ\_SHD or READ\_EXC,  
or the write bit is set and the command is WRITE, WRITE and INVALIDATE, or INVALIDATE.  
Note that having both read and write bits clear disables the address section of the trigger.

The event is triggered by the data phase if the `data_id_match` bit is set and the top four bits of the transaction ID on the data bus (which identify the agent that initiated the transfer) match the `data_id` field, or the `resp_id_match` bit is set and the responder ID (which provided the data) matches the `resp_id` field. However, if both `data_id_match` and `resp_id_match` bits are set then **both** IDs have to match for the event to trigger.

A CPU interrupt can trigger the event. Each interrupt mapper contains an interrupt trace mask register. Bits are set in this register to indicate that when the corresponding interrupt is raised the event should be triggered. The trace event will trigger when the `int_trace_trigger` output (see [Figure 10 on page 51](#)) of either interrupt mapper is asserted. This can be used to start tracing when the CPU gets an interrupt from the device being studied. The trigger is edge sensitive and only happens on the change from neither `int_trace_trigger` outputs being asserted to at least one being asserted. The trigger will not happen again until after the trace outputs from both mappers have cleared.

An external device can trigger an event by pulling the `DEBUG_L` pin low. `DEBUG_L` is a bidirectional open collector signal and must have an external pull-up to 3.3V. The SCD can signal the completion of a trigger event sequence by pulling the signal low for 10 ZBbus cycles (see [Section: "Trigger Sequences" on page 73](#)). The state of the `DEBUG_L` signal is ignored during the 10 cycles that the SCD is pulling it low and for 5 ZBbus cycles after the SCD stops driving, so an event will not be triggered by the SCD signalling a sequence completion. The trigger is edge sensitive, it will happen once when `DEBUG_L` changes from high to low and will not happen again until `DEBUG_L` has returned high.

Expressing the event trigger in pseudo-code:

```

if ((
    // Address section
    (
        ((trace_event_N[15:12] == zb_req_id) | ~trace_event_N[4]) &
        (
            (trace_event_N[0] & addr_match0) |
            (trace_event_N[1] & addr_match1) |
            (trace_event_N[2] & addr_match2) |
            (trace_event_N[3] & addr_match3) |
            (trace_event_N[3:0] == 4'h0)
        ) &
        (
            (trace_event_N[11] & read) |
            (trace_event_N[10] & write)
        )
    ) |
    (
        // Data section
        ((trace_event_N[23:20] == zb_data_id) & (trace_event_N[6:5] == 2'b01)) |
        ((trace_event_N[19:16] == zb_resp_id) & (trace_event_N[6:5] == 2'b10)) |
        ((trace_event_N[23:20] == zb_data_id) &
         (trace_event_N[19:16] == zb_resp_id) & (trace_event_N[6:5] == 2'b11))
    ) |
    // Interrupt
    (trace_event_N[7] & edge_detect (interrupt)) |
    // External pin
    (trace_event_N[9] & edge_detect (~debug_pin))
)
then trigger event N

```

Each trigger event has a counter associated with it. If the counter is greater than zero when the trigger condition is true then the counter decrements. If the counter is zero then the event is signaled to the trigger sequencers, and the counter will be reloaded with its initial value.

The trace event register is shown in [Table 46](#). There are eight such registers `trace_event_0` to `trace_event_7`.

**Table 46: Trace Event Register**

Bits	Field	Default	Description
<p><code>trace_event_0</code> - 00_1002_0A20  <code>trace_event_1</code> - 00_1002_0A28  <code>trace_event_2</code> - 00_1002_0A30  <code>trace_event_3</code> - 00_1002_0A38  <code>trace_event_4</code> - 00_1002_0A60  <code>trace_event_5</code> - 00_1002_0A68  <code>trace_event_6</code> - 00_1002_0A70  <code>trace_event_7</code> - 00_1002_0A78</p>			
3:0	Addr_match[3:0]	4'b0	Each of these bits corresponds to an address trap register (see <a href="#">Section: "Address Trapping" on page 67</a> ). If none of these bits are set then the address on the bus is not considered as part of the trigger. If any of these bits are set then the address is considered part of the trigger, and the event only triggers if the address falls within the range specified by the corresponding trap. Both simple and complex traps may be used. The event will trigger whenever there is a hit in the trap, the occurrence counter in the trap is ignored.
4	req_id_match	1'b0	When set the address portion of the event only occurs if the requester ID of the transaction matches the req_id field of this register.
5	data_id_match	1'b0	When set the data portion of the event only occurs if the data ID of the transaction matches the data_id field of this register.
6	resp_id_match	1'b0	When set the data portion of the event only occurs if the responder ID of the transaction matches the resp_id of this register.
7	interrupt	1'b0	When set the event occurs when an interrupt that is enabled for tracing occurs. The int_trace_trigger outputs from the two interrupt mappers (see <a href="#">Figure 10 on page 51</a> ) are ORed together to signal the event. (No record is made of which interrupt triggered the event). The event triggers once on the assertion of the OR of the two interrupt mapper outputs. Before the event can trigger again the outputs of both mappers must be deasserted.
8	reserved	1'b0	Reserved
9	debug_pin	1'b0	When set the event occurs if the external debug pin is pulled low. The event will trigger once on the falling edge of the DEBUG_L pin. DEBUG_L must return to its deasserted (high) state before the event will trigger again.
10	write	1'b0	When set the event only occurs if the transaction is a write.
11	read	1'b0	When set the event only occurs if the transaction is a read.
15:12	req_id[3:0]	4'b0	Requester ID used for matching.
19:16	resp_id[3:0]	4'b0	Responder ID used for matching.
23:20	data_id[3:0]	4'b0	Data ID used for matching.
31:24	count[7:0]	8'b0	If the count is not 0 then the event does not take place. Instead the counter is decremented. If the counter is 0 then the event occurs and the count is restored to its initial value. Writing this register sets the initial value of the counter, reading will return the current value.



## TRIGGER SEQUENCES

A trigger sequence is a set of up to four conditions that must be satisfied in sequence and a trace function that will be executed when the sequence is complete. The conditions are combinations of trigger events. There are eight trigger sequencers. Sequencers 0-3 have events 0-3 as their sources, sequencers 4-7 have events 4-7 as their sources.

The following chart shows the combinations for sequencers 0-3 (for sequencers 4-7 add 4 to the event numbers):

0	-	event0						
1	-	event1						
2	-	event2						
3	-	event3						
4	-	event0		event1				
5	-	event1		event2				
6	-	event2		event3				
7	-	event0		event1		event2		
8	-	event0		event1		event2		event3
9	-	event0	&	event1				
A	-	event0	&	event1	&	event2		
B	-	event0	&	event1	&	event2	&	event3
C	-	(event0 & event1)		event2				
D	-	(event0 & event1)		(event2 & event3)				
E	-	(event0 & event1)		event2		event3		
F	-	ignored						

The codes from these events are combined to make the sequence. For example the sequence 0123 requires that event0 happens, then event1, then event2, following this when event3 happens the function associated with the sequencer will be executed. Sequences of less than four events have the ignored code (F) in their lower bits, hence a sequence that will execute its function every time event 1 or event 2 triggers is expressed as 5FFF.

The functions that can be performed when a sequence is complete are:

- 1 Start** collecting samples. This enables collection of samples into the trace buffer, unless collection has been frozen. The sample that triggers the start will be recorded if the sequence is marked with one of the filter bits described below.
- 2 Stop** collecting samples. This suspends collection of samples into the trace buffer until a start function is performed. The sample that triggers the stop will be recorded if the sequence is marked with one of the filter bits.
- 3 Freeze** the trace buffer. This stops collection of samples into the trace buffer until the buffer is reset by software or via the JTAG port. The sample that triggers the freeze will be recorded if the sequence is marked with one of the filter bits. An interrupt is raised when the buffer is frozen.

In addition there are three filter flags that are only used if collection is in progress:



**A Sample** causes an address/control bundle to be recorded in the trace buffer. These samples are packed three per line of the trace memory, so a maximum of 768 samples can be in the buffer. Note that if the final trigger of the sequence is a databus trigger then the A-phase signals captured may not be valid (the latches within the trace unit will only capture address information when an agent is driving the bus, this data is held in the latches until the next time the bus is driven).

**D Sample** causes an address/control bundle and the 32 bytes of data from the bus to be recorded in the trace buffer. A D sample always starts a new line in the trace buffer (if the previous line was only partially full from A samples the empty spaces are marked unused). So a maximum of 256 D samples can be taken. Since a D sample is a superset of an A sample, if both filter bits are set a D sample is taken. When a D sample is taken the Aphase signals may not be valid.

**Clear Use** causes the counter that determines the buffer usage to be cleared. Thus following the current sample 255 more entries will be filled before the buffer full condition is raised. This flag is normally used with the **Start** function when the start trigger may repeat and the freeze should only come when the buffer fills after the last start.

The final two flags allow the completion of a sequence to be signalled outside the trace logic (these occur regardless of the state of trace collection).

**Debug Pin** causes the SCD to pull the DEBUG\_L pin low for 10 ZBbus clock cycles.

**Debug CPU** causes the SCD to assert the debug interrupt to both CPUs. The debug interrupt is cleared by a read of the **trace\_cfg** register.

The outputs from all the sequencers are combined. If multiple functions are signalled, **freeze** has the highest priority, then **stop**, then **start**.

Note that when an address based trigger is used, the data bus may not be valid. So, forcing a Dsample from such a trigger could waste trace buffer entries.

There are eight sequence control registers **trace\_sequence\_0** to **trace\_sequence\_7**, as shown in [Table 47](#).

**Table 47: Trace Sequence Control Registers**

Bits	Field	Default	Description
<b>trace_sequence_0</b> - 00_1002_0A40 <b>trace_sequence_1</b> - 00_1002_0A48 <b>trace_sequence_2</b> - 00_1002_0A50 <b>trace_sequence_3</b> - 00_1002_0A58 <b>trace_sequence_4</b> - 00_1002_0A80 <b>trace_sequence_5</b> - 00_1002_0A88 <b>trace_sequence_6</b> - 00_1002_0A90 <b>trace_sequence_7</b> - 00_1002_0A98			
3:0	Event select 4	4'b0	These bits set the event selection that forms the fourth event in the sequence. The bits should all be set (the ignore code) if the sequence is shorter than four events.
7:4	Event select 3	4'b0	These bits set the event selection that forms the third event in the sequence. The bits should all be set (the ignore code) if the sequence is shorter than three events.
11:8	Event select 2	4'b0	These bits set the event selection that forms the second event in the sequence. The bits should all be set (the ignore code) if the sequence is shorter than two events.
15:12	Event select 1	4'b0	These bits set the event selection that forms the first event in the sequence. The bits should all be set (the ignore code) if the sequence should always trigger (used with the filter bits to capture all active bus cycles).

**Table 47: Trace Sequence Control Registers (Cont.)**

Bits	Field	Default	Description
<pre> <b>trace_sequence_0</b> - 00_1002_0A40 <b>trace_sequence_1</b> - 00_1002_0A48 <b>trace_sequence_2</b> - 00_1002_0A50 <b>trace_sequence_3</b> - 00_1002_0A58 <b>trace_sequence_4</b> - 00_1002_0A80 <b>trace_sequence_5</b> - 00_1002_0A88 <b>trace_sequence_6</b> - 00_1002_0A90 <b>trace_sequence_7</b> - 00_1002_0A98 </pre>			
17:16	Function	2'b0	These bits set the function that should be performed when the sequence completes. 00: Nop. No function is performed. 01: Start. The trace collection is started. 10: Stop. Trace collection is stopped. 11: Freeze. The trace buffer is frozen.
18	Asample	1'b0	If this bit is set then an address trace is taken when the sequence completes, if Dsample is set this bit is ignored, unless all the select fields are 4'hF. If the select fields are all 4'hF and Asample is set then an address trace is taken every time the address bus is granted (i.e. NOP cycles are included).
19	Dsample	1'b0	If this bit is set then an address and a data trace is taken when the sequence completes. If the select fields are all 4'hF and Dsample is set then an address and a data trace is taken every time the data bus is granted (i.e. NOP cycles are included).
20	DebugPin	1'b0	If this bit is set then the external debug pin is pulled low when the sequence completes.
21	DebugCPU	1'b0	If this bit is set then debug interrupts will be sent to both CPUs when the sequence completes.
22	ClearUse	1'b0	If this bit is set then the buffer use counter is cleared, so the buffer full condition will not be raised until this buffer entry and 255 others have been used.
23	Alld_a	1'b0	If the select fields are all 4'hF and this bit is set then an address/control sample is taken whenever the data bus is used.
24	All_a	1'b0	If the select fields are all 4'hF and this bit is set then an address/control sample is taken on every bus cycle.

The special event selection sequence 16'hFFFF written in one of the trace sequence registers will cause all bus cycles to be selected. There are several possible actions depending on which of bits 18,19,23 and 24 are set:

- Asample (bit 18): An address/control sample is taken every time the address bus is used.
- Dsample (bit 19): An address/control and data sample is taken every time the data bus is used.
- Alld\_a (bit 23): An address/control sample is taken every time the data bus is used.
- Asample and Alld\_a (bits 18 and 23): An address/control sample is taken every time either address bus or data bus is used.
- Asample and Dsample (bits 18 and 19): An address/control sample is taken every time the bus is used, with data also being captured if the data bus is used.
- All\_a (bit 24): An address/control sample is taken on every bus cycle.

Using these sequences a trace of the entire active bus can be taken. To avoid the count of cycles between samples overflowing, the forceCnt bit can be set in the **trace\_cfg** register to force an address/control sample to be taken just before overflow. The sample collected when a section of the bus is not used is UNPREDICTABLE. The Avalid and Dvalid bits in the address/control bundle are used to indicate which bits in the bundle are valid. The All\_a flag can be used in profiling to monitor the blocking and interrupt signals on every cycle.



## USING THE TRACE BUFFER

The trace buffer starts off empty and tracing is disabled. The trigger events and sequencers are configured by software or over the JTAG debug interface. Then the system is run and the trace is recorded in the buffer. When the buffer freezes the trace can be read out, again either by software on the BCM1250 or BCM1125/H or via the JTAG link.

Control of the trace features is done through the **trace\_cfg** register. The JTAG interface or debug software can write this register to trigger the start, stop or freeze functions, and also to reset and enable the trace buffer. The trace unit can also be set to execute the **Freeze** function and/or the **Debug Pin** function when the buffer fills (these functions are described in [Section: "Trigger Sequences" on page 73](#)).

A 64 bit register is provided for reading out the collected trace. The buffer should be frozen before samples are read out, if it is not then the results of the read are UNPREDICTABLE. The startread bit in the **trace\_cfg** register is written with a one to start the read process. Then each read of the **trace\_read** location (00\_1002\_0A08) will provide the next 64 bits from the trace memory. Software (either on the target, or external debug agent) must post-process the trace to extract the information. (Broadcom provides code to do this.) The buffer can only be read once after samples have been collected.

The trace control register is shown in [Table 48](#).

**Table 48: Trace Control Register**

trace_cfg - 00_1002_0A00			
Bits	Field	Default	Description
0	reset	1'b0	When this bit is written with a 1, the trace buffer pointers will be reset ready for collection to begin. This bit always reads as zero.
1	startread	1'b0	When this bit is written with a 1, the trace buffer read pointer and latch are prepared for reading. This bit always reads as zero. Note that the buffer can only be read once after a collection, if this bit is written with a 1 for a second time the data read back will be UNPREDICTABLE.
2	start	1'b0	When this bit is written with a 1, the start function is triggered, and collection will begin if the buffer is not frozen. When read this bit will be set if collection is in progress.
3	stop	1'b0	When this bit is written with a 1, the stop function is triggered, and collection will be suspended until a start if the buffer is not frozen. When read the bit will be set if collection is stopped and not frozen.
4	freeze	1'b0	When this bit is written with a 1, the trace buffer is frozen. When read this bit will be set if the buffer is frozen. The buffer must be reset to allow further collection.
5	freezeFull	1'b0	If this bit is set the buffer will freeze when it becomes full. In order to prevent overflow the buffer will freeze as soon as there is any data in the final entry, if it freezes with space for two more Address/Control bundles (but no room for a Data sample) the trcFull bit will not be set.
6	debugFull	1'b0	If this bit is set the DEBUG_L pin will be asserted when the buffer becomes full. The DEBUG_L pin is deasserted again when the first read of the trace buffer begins or the trace is reset.
7	trcFull	1'b0	Read Only. This bit is set when the trace buffer is full (if this is set 256 entries can be read from the buffer)
8	forceCnt	1'b0	If this bit is set and collection is in progress then an address/control sample is forced to be collected before the COUNT field overflows regardless of the state of the filters. This ensures an exact count between events (at the expense of some additional entries used in the buffer).
9	reserved	2'b0	Reserved



**Table 48: Trace Control Register (Cont.)**

trace_cfg - 00_1002_0A00			
Bits	Field	Default	Description
17:10	trcAddr	8'b0	Current trace buffer address. If the trcFull bit is not set this number plus 1 entries can be read from the buffer.
63:18	notimp	46'bx	Not Implemented.

All entries in the trace buffer contain an address/control bundle. The bundle includes the response phase signals that match with the address phase signals that are captured. The data phase signals are captured at the same time as the A phase signals, and have no relationship with the transaction that the A-phase and R-phase are part of. The address/control bundle is 128 bits organized as shown in [Table 49](#).

**Table 49: Trace Buffer Address/Control Bundle**

Bits	Field	Description
2:0	D_CODE	Data code (this has no relationship to the A and R phase signals). 000: NOP, data invalid (gets counted as arbitrated but not used). 001: Data valid. 010: Data valid, tag ECC corrected (L2). 011: Data valid, data ECC corrected (Mem, L2). 100: Bus error (IO bridge). 101: Fatal bus error (cache block ownership unclear). 110: Uncorrectable tag ECC error (L2). 111: Uncorrectable data ECC error (CPU, L2, Mem).
3	D_MOD	D-phase indication that the data is dirty (this has no relationship to the A and R phase signals).
7:4	D_RSP[3:0]	Data phase responder ID (this has no relationship to the A and R phase signals).
17:8	D_ID[9:0]	Data phase transfer ID (this has no relationship to the A and R phase signals). [9:6] requester ID. [5:0] unique number within requester.
18	R_L2HIT	The L2 hit during the response phase of the A-phase transaction.
24:19	R_EXC[5:0]	Response phase exclusive bits that match the A-phase transaction.
30:25	R_SHD[5:0]	Response phase shared bits that match the A-phase transaction.
31	A_L2CA	L2 cacheability bit. 0: Do not allocate on miss. 1: Allocate on miss.
36:32	reserved	Always reads zeros.
37	int_trace_1	Records the int_trace_trigger_1 output from the CPU 1 interrupt mapper (see Figure 8)
38	int_trace_0	Records the int_trace_trigger_0 output from the CPU 0 interrupt mapper (see Figure 8)
39	Dvalid	Set if the data bus had a valid cycle when the sample was captured (indicates the D_ signals are valid)
40	Avalid	Set if the address bus had a valid cycle when the sample was captured (indicates the A_ and R_ signals are valid).
51:41	BLOCK[10:0]	ZBbus blocker signals.
59:52	A_BYT	Encoded byte enables. Indicates valid bytes within doublewords. BYT[n=7:0] =A_BE[n]   A_BE[n+8]   A_BE[n+16]   A_BE[n+24]

**Table 49: Trace Buffer Address/Control Bundle (Cont.)**

Bits	Field	Description
63:60	A_DW	Encoded byte enables. Indicates any doublewords with valid bytes. $DW[n=3:0] = A\_BE[8*n + 0]   A\_BE[8*n + 1]   A\_BE[8*n + 2]   A\_BE[8*n + 3]   A\_BE[8*n + 4]   A\_BE[8*n + 5]   A\_BE[8*n + 6]   A\_BE[8*n + 7]$ If only one doubleword is set then the BYT field indicates the valid bytes, if more than one doubleword is set then knowledge of the transaction will be needed. Other than uncached accelerated writes, all CPU transactions will be within a single doubleword or of the full cacheline.
65:64	A_L1CA	L1 cacheability bits. 00: Cacheable non-coherent. 01: Cacheable coherent. 10: Uncacheable. 11: Uncacheable (accelerated, may have merged writes).
68:66	A_CMD	Address phase command bits 000: READ_SHD. 001: READ_EXC. 010: WRITE. 011: WRITE and INVALIDATE. 100: INVALIDATE. 101-110: Reserved 111: No command valid. (NOP)
103:69	A_AD[39:5]	Line address of the transaction on the bus.
113:104	A_ID[9:0]	This is the value of the address ID bits on the bus. A_ID[9:6] - Requester ID. A_ID[5:0] - Unique number within granted requester.
125:114	COUNT	This is the number of bus cycles between the last traced entry and this entry. This is a 12 bit saturating counter. If two samples are captured on consecutive cycles the count will be zero. The count is set to zero when the trace buffer is reset.
126	DTRIG	This bit is set when a sequence completing with Dsample caused this control entry to be traced. The next 256 bits contains the data bits [255:0].
127	ATRIG	This bit is set when a sequence completing with Asample caused this control entry to be traced.

The data trace consists of two 128 bit entries. The first contains bits [255:128] from the databus, the second has bits [127:0].



## READING THE TRACE BUFFER

Each entry in the trace buffer is 384 bits, and falls in to one of two possible formats. Format 1 is used when the entry contains one to three address/control samples, the first bundle (t0) will always come from an address trigger (the ATRIG bit will be 1) and will indicate format 1 by not having an associated data sample (the DTRIG bit will be 0). Note that if the sample was forced by an all-bus-cycle trigger, a data trigger or to avoid the inter-sample interval count overflowing, the Avalid bit may be clear indicating the A-phase information is invalid. The Dvalid bit indicates if there was databus activity when the sample was taken and the D-phase information is valid. Format 2 covers the address and data sample, in this case there may or may not have been an address trigger (ATRIG could be either 0 or 1) but there will always have been a data trigger (DTRIG will be 1). Again the Avalid and Dvalid bits in the control bundle indicate what activity was on the bus and thus which fields of the sample are valid. Software or the JTAG probe must perform six reads to collect the entire sample before decoding it. [Table 50](#) describes the entry formats and the order in which they are read.

**Table 50: Trace Entry Format and Read Order**

Bits	Read Order	Format 1 (Address/Control)	Format 2 (Address/Data)
383:320	1	Address/Control Bundle t2 [127:0]	Data [255:0].
319:256	2	This is the third sample recorded in this entry. It may not be valid.	
255:192	3	Address/Control Bundle t1 [127:0]	
191:128	4	This is the second sample recorded in this entry. It may not be valid.	
127:64	5	Address/Control Bundle t0 [127:0]	Address/Control Bundle t0 [127:0]
63:0	6	This is the first sample recorded in this entry. It will always be valid for a format 1 sample. The DTRIG bit will be 0.	This is the address sample captured with the data. It may not be valid. The DTRIG bit will be 1.

When all the entries have been read from the buffer the ATRIG and DTRIG bits are forced to zeros in any subsequent reads. This condition can be used to terminate reading the buffer, or the trcFull and trcAddr fields of the **trace\_cfg** register can be used to compute the number of entries that should be read. The startread bit in the **trace\_cfg** register must be written with a 1 before the trace can be read out. The buffer can only be read out once, any additional attempts will return UNPREDICTABLE data.



An algorithm for decoding the bundle is:

```
if (b[127]==1)
{
    if (b[126]==1)
    {
        /* Have valid address and valid data sample */

        Format = 2
    }
    else
    {
        Format = 1

        if (b[255] == 0)
        {
            /* Have 1 valid address sample */
        }

        else if (b[383] == 0)
        {
            /* Have 2 valid address samples */
        }

        else
```

The entries are returned with the newest one first and the earliest available read out last. The postprocessing software must take care to reverse the order to show causality.

## MAGIC DECODER RING FOR USING THE TRACE BUFFER

This section includes some notes for using the Trace Buffer when debugging. Nothing in this section is part of the official specification of the device and things (in particular the TID information) could change from revision to revision.

The trace control entries include the transaction ID generated by the source of the transaction. In many cases additional information about the transaction can be deduced from the TID. For chips with system revision indicating PERIPH\_REV3 [Table 51](#) gives some details.

**Table 51: Decode of some TIDs for system revision PERIPH\_REV3**

TID[9:6] Agent ID See <a href="#">Table 1</a>	TID[5:0] Set by the agent	Description
<b>CPU</b>		
000x	11nnnn	Instruction Fetch.
000x	000nnn	Read for read queue entry nnn.
000x	10nnnn	Write or Evict.
<b>Bridge 0</b>		
0010	00pnnn	Read from port p (1=PCI, 0=HyperTransport).
0010	01pnnn	Write from port p (1=PCI, 0=HyperTransport).
0010	10pnnn	Read-modify-write (i.e. partial line write) from port p (1=PCI, 0=HyperTransport).
<b>Bridge 1</b>		
0011	00pnnn	Read from DMA engine p (1=Serial, 0=MAC).
0011	01pnnn	Write from DMA engine p (1=Serial, 0=MAC).
0011	10pnnn	Read-modify-write (i.e. partial line write) from DMA engine p (1=Serial, 0=MAC).
<b>SCD</b>		
0100	cc00aa	Read for buffer aa of channel cc. (cc bits always 00 prior to PERIPH_REV3)
0100	cc01aa	Write of buffer aa of channel cc. (cc bits always 00 prior to PERIPH_REV3)
0100	cc11aa	Read for read-modify-write with buffer aa of channel cc. (cc bits always 00 prior to PERIPH_REV3)
0100	cc1000	Descriptor fetch for channel cc. (cc bits always 00 prior to PERIPH_REV3)
0100	001001	JTAG Probe transaction. (010000 prior to PERIPH_REV3)

**Table 52: Encoded Byte Enables for CPU Transactions**

CPU Operation	DoubleWord A_DW	Byte A_BYT	Comments
Cacheable fill or evict or Instruction Fetch	1111	11111111	Cacheable operations always move a full block, so the byte enables can be ignored.
Uncacheable Instruction Fetch	1100 0011	11111111	Uncacheable instruction fetches always fetch four instructions (either the first or second half of a cache block).
Uncacheable lb/sb	One bit set	One bit set	The byte can be identified from the encoded enables.
Uncacheable lh/sh	One bit set	11000000 00110000 00001100 00000011	The half word can be identified from the encoded enables.
Uncacheable lw/sw	One bit set	11110000 00001111	The word can be identified from the encoded enables.
Uncacheable ld/sd	One bit set	11111111	The double word can be identified from the A_DW code.
Uncacheable lwl/lwr/swl/swr	One bit set	1-4 bits set	The bytes can be identified from the encoded enables (but some combinations will match word/halfword/byte accesses).
Uncacheable ldl/ldr/sdl/sdr	One bit set	1-8 bits set	The bytes can be identified from the encoded enables (but some combinations will match double/word/halfword/byte accesses).
Uncached Accelerated Writes	1-4 bits set	1-8 bits set	If only doubleword stores are involved they can be identified. If word stores are used then information may be lost (reconstruction may be possible from the source code).

## CONNECTIONS TO THE TRACE LOGIC

The trace logic connects to many other parts of the SCD. Figure 11 shows these connections.

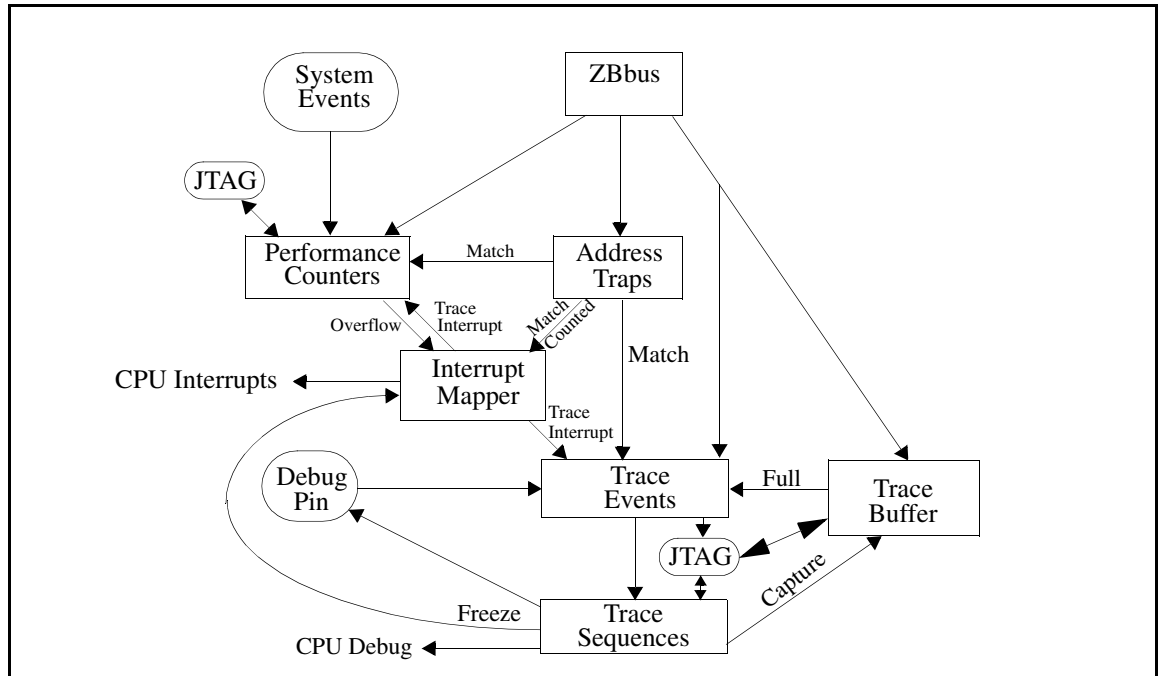


Figure 11: Connections to Trace Logic

## TRACE EXAMPLE 1: ALL CPU0 ACTIVITY

The code running on CPU0 is being monitored, for a particular section of code all address bus activity will be recorded to check the L2 cache behavior.

The code is patched to do a dummy access to an address at the start and end points of the code to trace, and the trace buffer is set up:

```
Addrmatch0 = start trigger
Addrmatch1 = stop trigger

trace0 = reqid=cpu0 & (read|write)
trace1 = reqid=cpu0 & Addrmatch0 & (read|write)
trace2 = reqid=cpu0 & Addrmatch1 & (read|write)
trace_cfg = freezeFull

seq0 = trace1, ignore, ignore, ignore -> Start
seq1 = trace0, ignore, ignore, ignore -> A-sample
seq2 = trace2, ignore, ignore, ignore -> Stop
```

When CPU0 accesses the dummy start address trace1 and seq0 will cause trace collection to be started. Then trace0 and seq1 will cause every address requested by CPU0 to be written to the buffer, until the dummy access that causes trace2 and seq2 to stop collection (the A-sample bit could be set on this sequence to record the dummy access if a separator was required).

When the trace buffer is full, the **trace\_cfg** setting causes it to freeze, and raise an interrupt. CPU0 would service this interrupt and (with as little cache disturbance as possible) read out the trace into a bigger in-memory buffer. At the end of the ISR the trace buffer is reset and started (it must have been collecting for it to fill) and collection continues.

At the end of the test the buffer would be frozen by software and the final samples copied to the memory buffer. The memory buffer would then be post-processed for analysis.



## TRACE EXAMPLE 2: NETWORK PACKET HEADERS

This example uses the trace buffer as a non-obtrusive record of the most recent network packet headers that were received from one of the MACs. It could be used, for example, when there is a bug that is thought to be caused by the processing of some rare sequence of network packets.

The trick is to notice that the DMA controller will always write the receive DMA status back into the DMA descriptor at the end of packet reception. This is used to prime collection of the first two transfers of the next packet.

```

Addr0 = range of receive DMA descriptors for the MAC
Addr1 = range of receive buffers

trace0 = reqid=iobr1 & Addrmatch0 & write (A phase of descr write)
tracel1 = reqid=iobr1 & Addrmatch1 & write (A phase of write)
trace2 = rspid=iobr1 & dataid=iobr1 (write data from iobridge)

seq0 = trace0, ignore, ignore, ignore -> Start
seq1 = tracel1, ignore, ignore, ignore -> A-sample
seq2 = tracel1, trace2, ignore, ignore -> D-sample
seq3 = tracel1, trace2, tracel1, trace2 -> Stop

```

The first address trap is set up to detect the range of addresses that cover the receive DMA descriptor ring. trace0 detects a write to this space from the I/O bridge that connects the MACs. This write is done at the end of a packet reception to put the status flags into the descriptor (the requester ID is needed here because the CPU will also be writing the ring to update the descriptors). seq0 uses this to start the trace collection. When a packet arrives the DMA will write it to memory in 32 byte cache line chunks. The address associated with the write request triggers trace1, and seq1 filters it to be recorded (this may not be needed in this example, and will be fairly costly because it will unalign the trace buffer pointer from the entry boundary).

The trace2 is used to detect a write data transfer from the MAC to the buffer memory. It does this by detecting a transfer on the data bus where the data source (in the resp\_id field) and the initiator of the transaction (in the data\_id field) are the same. This is enough to detect that some write data is in flight from the I/O bridge. The seq2 sequence will first need trace1 to detect the write request to the buffer addresses before looking at trace2 to detect the data transfer part of the transaction. When the sequence is triggered the data sample is taken and written to the trace buffer.

Since the network packet header is in the first 64 bytes of the packet only the first two DMA transfers should be recorded. seq2 will trigger on both, causing data to be written to the trace buffer. seq3 has to see both transfers before it triggers and stops collection.

This process repeats, using the trace memory as a circular buffer, until the Freeze command is given (by software writing the **trace\_cfg** register). At this point the trace buffer can be read out, showing the last 64 packet headers. (Two lines in the buffer are used up for each DMA transfer recorded -- the first has the seq1 address sample and two empty slots, and the second the seq2 address/data sample. Since the first two transfers of each packet are recorded there are 4 trace buffer lines per packet).

### TRACE EXAMPLE 3: PCI DRIVER DEBUGGING

Device drivers for PCI devices quite often rely heavily on macros and library input/output routines. Particularly when porting code from a little endian (e.g. x86) driver to the BCM1250 or BCM1125/H running big endian, it can be hard for the programmer to track down exactly what is being transferred back and forth to the device on the PCI bus. This example uses the trace buffer to record every transfer between the CPU and the PCI bus, and any DMA activity initiated by the PCI device.

```
Addr0 = PCI range

trace0 = reqid=cpu0 & Addrmatch0 & (read|write)
trace1 = reqid=iobr0 & (read|write)
trace2 = rspid=cpu0 & dataid=cpu0
trace3 = rspid=iobr0 & dataid=cpu0

trace4 = dataid=iobr0

seq0 = trace0, ignore, ignore, ignore -> A-sample
seq1 = trace0, trace2 | trace3, ignore, ignore -> D-sample
seq2 = trace1, ignore, ignore, ignore -> A-sample

seq4 = trace4, ignore, ignore, ignore -> D-sample
```

The driver code has writes to the **trace\_cfg** register added to start and stop samples, and to freeze and read out the results.

Any access from the CPU to the PCI range is detected by trace0. If the transaction is a write then trace2 will detect the data transfer; the `resp_id` indicates that the data source is the CPU and the `data_id` indicates that the transaction was initiated by the CPU. (This detects any write data from the CPU, it is used in sequence with trace0 to detect this particular write.) If the transaction is a read then trace3 detects the data returning from the I/O bridge in response to the CPU request (the I/O space read is uncacheable and the CPU only has one outstanding, so in sequence with trace0 this always catches the correct data).

Master mode accesses from the PCI are detected by trace1 for addresses and trace4 for the data. These just use the transaction id to identify the I/O bridge as the initiator of the operation. Note that since trace4 is used the detection sequence had to move to seq4 on the second sequencer block. This will detect DMAs from devices on the HyperTransport fabric too, so this is not a perfect trace. However, since it is being used to debug during the device driver bring-up, it should be relatively easy to prevent other DMA devices from being active.

This Page is left blank for notes



---

This Page is left blank for notes



## Section 5: L2 Cache

### INTRODUCTION

The on-chip second level cache is shared by the processors and any I/O DMA masters. On the BCM1250 the L2 cache is 512 KBytes, on the BCM1125/H it is 256 KBytes. This section describes the normal operation of the L2 cache, a mode where banks of it can be used as an on-chip SRAM and the management interface to it. After reset the management interface must be used to initialize the L2 cache before it can be used, if a cacheable reference is made before the L2 is initialized multiple cache lines may be selected causing an internal bus clash.

### NORMAL OPERATION

The L2 cache is 256 or 512 KBytes, organized into 32 byte cache lines four way set associative. All accesses to the L2 are in full cache blocks. It is a non-inclusive/non-exclusive cache, thus there are no restrictions on which cache blocks can be in the L2. A random replacement policy is used when a victim line must be found. The L2 runs internally at the CPU core speed and is fully pipelined. This allows one access per ZBbus cycle, including all the required housekeeping needed to evict dirty lines.

The L2 cache tags and the data blocks are ECC protected. Single bit errors in either are corrected while an access is in progress (the L2 will internally take time to perform the full recovery, but the system continues running). The ECC cleanup is started by a cache hit on a line with a single bit error. Cleanup will be pre-empted by a write to another way at the same index, so occasionally a single bit error will be flagged twice. Tags with uncorrectable errors result in UNPREDICTABLE data being returned to the requestor with an uncorrectable tag error signalled. Data with uncorrectable errors will be returned to the requestor with an uncorrectable data error signalled. In either error case software recovery is required. The bus watcher in the SCD will log the data associated with the error and raise an interrupt. The L2 will record the tag associated with the error in the **I2\_ecc\_tag** register (this can be read to give the tag from the most recent ECC error). The management interface to the L2 can be used to invalidate the line and clear the ECC bits. Any data that is received by the L2 cache marked with an error is written into the cache with an uncorrectable ECC error, so subsequent reads will also get an error.

The L2 cache is physically one of the ZBbus agents, but architecturally it sits between the system bus and the main memory, and there are dedicated signals between the L2 and memory controller that coordinate them. Every bus access is accompanied with the L1 cache and coherence attributes for the access. In the CPU these are set in the TLB and indicate cacheable-coherent, cacheable-non-coherent, uncacheable, and uncacheable-accelerated (which indicates writes may have been merged). For DMA accesses the L1 attributes are provided from the DMA control registers and should match the attributes the CPUs use for that area of memory. The L2 is coherent with memory for all L1 cacheable accesses (non-cacheable accesses to main memory will bypass the L2 cache, they should never be done to areas of memory that have previously been cacheable). The L2 cache is checked by all L1 cacheable memory accesses put on ZBbus. As described in [Section: "Coherence" on page 12](#), it acts in conjunction with the memory controller to provide any blocks that are not owned exclusively by another bus agent.

Every bus request is accompanied by an L2 cacheability flag, if this is set and the block is not in the L2 cache then it will be allocated. On a write miss the L2 will allocate space for the block and accept the data when it is transmitted. On a read miss the memory controller will supply the data to the requesting agent, the L2 will allocate space for the block at the time the request is made and will take a copy of the data as it passes on the bus. The L2 cacheability flag only affects the behavior on an L2 miss. Regardless of the L2 cacheability flag, if a request that is L1 cacheable hits in the L2 then it will supply the data for a read and receive the data on a write.

The L2 cache behaves the same whatever the source of the request, so DMA masters will read data from the L2 if it is there. More interestingly, DMA writes can be done with the L2 cacheability bit (see A\_L2CA in [Section: "Address Phase" on page 22](#)) set to cause the data to be written to the L2 cache (and on to memory only if it gets evicted). This can be used to reduce the access latency for any data the CPU will need to manipulate. However, this operation can potentially pollute the L2 cache so it must be used with care. The internal DMA controllers have a control bit to enable this behavior, its use by external DMA masters is described in the host bridge section for the buses.

To allow programs further control of the L2 cache, the CPU provides a mechanism to prevent the L2 cacheability flag from being set on accesses that are cacheable in the L1 cache. This is of particular use for reducing L2 pollution with data that is highly likely to be referenced only by one of the CPUs and used for a short time. There are two ways this can be done. Pages can be marked in the TLB with one of the cacheable coherent no L2 attributes (codes 0 and 1), which will cause all blocks in the page to be accessed with the L2-allocate flag clear. Alternatively, the pages can be marked with the usual cacheable coherent attributes (codes 4 and 5) and the PREFetch instruction with a streaming hint can be used to access individual lines around the L2 cache. In the second case since the block is marked for streaming when it is put in to the L1 it is flagged as the first of the 4 lines at that index to be evicted (it will be evicted first regardless of the LRU information).

When the L2 cache is accessed address bits [16:5] ([15:5] on a part with or configured for a 256KB cache and [14:5] for a part with or configured for a 128KB cache) are used to form an index into the cache array, and the four ways are examined at that index. When designing for best performance the potential for bad aliasing must be taken into consideration. For example in an embedded system using 64K pages and a simple memory allocator, it is possible that memory is always allocated in 64K chunks starting at a page boundary. Thus (for a 512KB cache) data in the first cache block of allocated objects will be found at one of 8 locations in the L2 cache (the index will either be zero or have just bit [16] set, and there are 4 ways at each possible index). A programmer unaware of the allocation policy could see a large amount of unexpected cache thrashing, and thus get significantly lower performance than expected. A similar (though less dramatic) effect can be seen if network packet buffers are allocated as 1KB long aligned at a 1KB boundary and the DMA engine is set to cache 64 byte headers in the L2. The packet headers will all have address bits [9:6] as zero, so will be limited to use 1/16th of the cache. If 64 bytes of padding is put between the buffers (or the buffer size is increased to 1088) then all of the cache will be used.

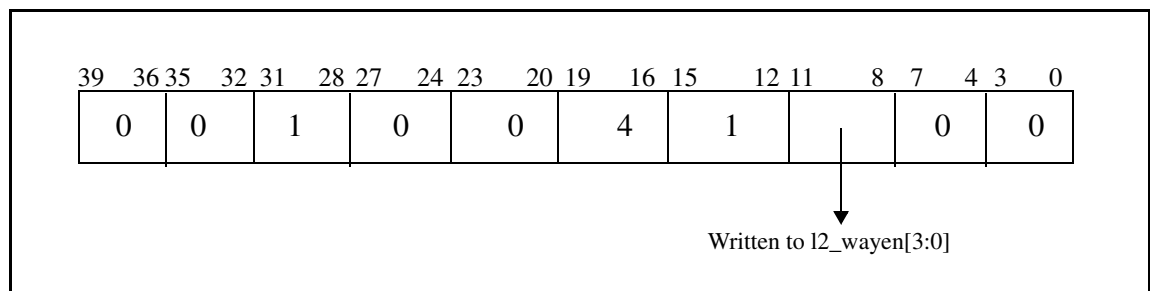
Statistics for the L2 cache are gathered by the debug/performance monitor unit. These are discussed in the [Section: "System Performance Counters" on page 61](#).



## USING THE L2 CACHE AS MEMORY

Some applications require more control over the on-chip memory than is provided by a cache, and would work best with a local RAM. A fast RAM can be provided by removing ways from the L2 cache. Each way that is removed from the cache provides a 128 KByte memory that can be accessed at the same speed as the L2 (or 64KB memory on a 256KB cache).

Ways are removed from cache use (they are removed from the random replacement algorithm) by clearing the bit corresponding to the way number in the `l2_wayen[3:0]` register. This is a control register hidden in the L2 cache that can be updated by doing a write to the `l2_way_enable` space as shown in Figure 12. Whenever any write is made to this address range the data is ignored, and bits [11:8] of the address get written to bits [3:0] of the register that enables the way. This register resets to have all bits set, so all ways of the cache are enabled. To remove way 2 from the cache and enable all the other ways the access address would be to the `l2_way_enable` space with address bit 10 clear and the others set, giving an address of `00_1004_1B00`. Software should not clear all four bits in the `l2_wayen` register, the resulting behavior of the cache is UNPREDICTABLE if there is any L2 cacheable activity in the system (the system will not hang, but data corruption will occur in one way of the cache). On parts with system revision indicating `PERIPH_REV3` and later the register can be read back from the `l2_misc_value` register.



**Figure 12: Level 2 Cache Way Disable Access Address**

The memory removed from the cache must always be accessed as cacheable space. Cacheable transfers are always done as full blocks and the L2 cache always operates on full cache lines. Writes smaller than a cache line (which will be the case for most uncacheable stores) will result in the whole line being written with UNPREDICTABLE data. The ECC logic remains active for the memory. On writes the correct ECC is generated and written. On reads the ECC is checked, correctable errors are fixed and uncorrectable errors are flagged as data errors.

Removing ways from the L2 cache reduces both its size and associativity. This is likely to impact the performance of the processors. A system design trade-off must be made between the control given by having the local fast memory, and the degradation of cache performance. In general purpose systems using all the memory as cache is most likely to be the best solution, in well-characterized embedded applications removal of one way of the L2 cache can improve the predictability of critical loops.

There are two main methods for using ways removed from the L2 cache. The simplest is just to use the memory as a block of on-chip RAM with a fast access time (that initially contains random data). The second method is to load data from main memory into the L2 cache and then lock it in place by preventing it being replaced.

## STANDARD RAM

Ways removed from the L2 cache can be used as a standard RAM block by accessing the memory using the management access address range with the way bits set appropriately and the special control bits set to zero (the management mode is described in [Section: “Cache Management Access” on page 94](#)). This provides a bank of memory that the system will need to initialize and that has no corresponding main memory locations. If consecutive ways are removed from the L2 cache the banks provided will be contiguous, forming a larger memory. [Table 53](#) shows the address range that should be used for each RAM bank. Note that special management accesses can be invoked by using addresses other than those listed, potentially corrupting the data.

**Table 53: Addresses for Memory Banks**

Way	Base	End (for 512KB cache)	End (for 256KB cache)
0	00_D018_0000	00_D019_FFFF	00_D018_FFFF
1	00_D01A_0000	00_D01B_FFFF	00_D01A_FFFF
2	00_D01C_0000	00_D01D_FFFF	00_D01C_FFFF
3	00_D01E_0000	00_D01F_FFFF	00_D01E_FFFF

[Figure 13 on page 95](#) gives details on how these addresses were derived. The special ECC diagnostic mode bits are clear, the valid and dirty bits are set and the way bits match the desired way. Bits [26:23] are ignored by the access (although they get written to the cache tag), so there are 15 aliases to the addresses in [Table 53](#) that will work equally well.

## MEMORY LOCKED IN THE L2 CACHE

The second method for using the L2 cache as a controlled memory is to lock data into it. The L2 is initialized with the data from main memory before the way is removed from the replacement algorithm, then any accesses to those main memory addresses will always access the L2 cache. This scheme is more complicated to set up and care must be taken that the addresses to be locked do not collide in the cache (since only one way is being locked only one address can be used per cache index), however it may be useful for code or data that cannot be easily relocated.

One method to initialize the cache is to ensure there are no copies of the code in L1 or L2 caches (since the initialization would most likely run as the system boots, it may be possible to arrange this by design). Three ways of the L2 cache are then disabled and cacheable reads are used to fetch the code or data into the fourth way. Once the data is in place the three ways are enabled and the fourth way disabled, locking the data in the cache. As with any L2 cache manipulation, care must be taken to ensure that there is no other activity in the system during this process.





## COMMENTS ON USING THE L2 AS MEMORY

This section discusses some of the system design trade-offs associated with using the L2 cache as memory, and some of the associated questions.

In normal operation the L2 cache is 512KB or 256KB and 4 way associative. There is no on chip memory. In general this is the best combination, since the goal of a cache is to automatically keep recently used information in fast on-chip memory and spill older data out to the main memory. However this is statistical, there are no guarantees of behavior because sometimes a full memory cycle will be needed. Turning a way of the cache into a memory bank will give some memory with guaranteed (fast) access time (subject only to bus contention), but the performance of the cache is degraded both because it is smaller and because the associativity has reduced. There is no easy way to tell what the performance impact will be for an embedded system without modeling the details (there are many studies of general computing systems but they are less useful because most uses of the part will have a much higher amount of I/O buffer and descriptor manipulation than the studies typically include). The L2 cache is non-inclusive and non-exclusive (i.e. what is in the L2 and what is in the L1 need not be related) which mitigates the effects some amount. But with a smaller L2 there is very likely to be reduced performance. On the other hand on a BCM1250 a 256KB 2 way associative cache (i.e. with half of the cache set aside as on chip memory) may be sufficient for the control processor if the task is split such that the other CPU is running a real-time-loop. The real time processor can reduce its impact on the L2 cache by keeping its main code loop in its L1 Instruction cache and most of its data either in its L1 Data cache or fetched around the L2 from a large table in memory (using PREFetch for streaming or by mapping the page with the no-L2 allocate cache attribute).

When the L2 cache is used as memory there is a restriction that writes must always be of full 32 byte cache lines. In practice this is not a problem. The space must be mapped cacheable in the CPUs (which is reasonable for data that needs a fast access time), this is not a burden for I/O devices doing DMA since the coherence protocol will ensure the most up to date copy of the data is returned for a DMA read and the latency is about the same regardless of whether the data comes from the L1 or L2 cache. In cacheable space the CPU will only do full 32 byte block reads (to fill the L1 cache line) and full 32 byte block writes (when a line is evicted from the L1 cache), so the full line write requirement is met. The I/O bridges (the point at which any DMA traffic hits the coherent domain) recognize the L2-as-memory addresses as coherent memory space so on a read will use the coherence protocol to get the latest data, and if a partial block write (i.e. <32 bytes or not cache aligned) is made the I/O bridge will do a read (exclusive)-modify-write of the full line to coherently merge in the new data. Therefore the full line write requirement is met in this case too. The Data Mover has the same restriction as the CPU: the space must be marked cacheable to ensure full line writes.

Another option that should be considered rather than using the L2 cache as memory is to make use of the L2 as a cache in which I/O devices can cache data. The ZBbus L2CA flag causes an L2 cache allocate on a miss. This flag accompanies every bus command (see [Section: "Address Phase" on page 22](#)). The on-chip DMA engines can be configured to assert the flag to cache descriptors and/or packet headers. HyperTransport devices can assert the flag by setting the isochronous bit in a HyperTransport command or (with revision 3 of the interface) system revision PERIPH\_REV3) hitting in an address range defined by the IsocBAR and IsocMask. PCI devices can assert it through the mapping in the BAR0 translation registers. Using this rather than partitioning the cache will allow the cache to behave as a cache and store recently accessed data. This will work well in situations where the CPUs are expected to be manipulating packet headers and descriptors shortly before (on transmit) or after (on receive) the I/O device. The drawback of this method is that occasionally the data cached by the I/O device will cause some CPU cached data to be evicted and force the CPU to re-fetch it from memory (this is always true of the other CPU as well), and in overload, newly arrived packets will displace older ones in the cache, so as the processors try to catchup, they will have to re-fetch from memory (this can be easily prevented by detecting overload and flipping the bit in the DMA controllers to disable the L2CA flag).

## REDUCED CACHE SIZE

The BCM1250 can be used to develop code for lower end members of the SiByte processor family. To allow performance tuning, the L2 cache size can be reduced to match the other parts, otherwise any performance numbers may be skewed by the large L2.

A 4 bit register **I2\_cache\_disable** is used for this. It is similar to the **I2\_way\_enable** register in that it is hidden and written from address bits. The register is accessed by writing dummy data to address `00_10042r00`, bits [11:8] (denoted *r*) are the control. On parts with system revision indicating PERIPH\_REV3 and later the register can be read back from the **I2\_misc\_value** register.

- To restore the full 512K L2 cache *r* should be 4'h0.
- For a 256K 4 way associative L2 cache *r* should be 4'h1 or 4'h2.
- For a 128K 4 way associative L2 cache *r* should be 4'h5 or 4'h6 or 4'h9 or 4'hA.
- Other values of *r* will give UNDEFINED results.

## CACHE MANAGEMENT ACCESS

In addition to regular accesses to the L2 cache there is a management mode. This is used to invalidate the cache when the system is reset, and is used during recovery from uncorrectable ECC errors. It can also be used to force dirty lines to be flushed from the L2 cache, however this should never be necessary in normal operation. There are two memory mapped registers associated with the management mode.

Performing cache operations using the management mode requires software to maintain ordering and control of traffic to the L2 cache. Accesses to the L2 cache by the other CPU or DMA engines could cause the cache operation to fail. In normal systems management accesses are only required as part of system startup or to recover from uncorrectable ECC errors, so it should be possible to have this exclusive access.

A portion of the address space (`00_D000_0000 - 00_D7FF_FFFF`) is allocated to L2 cache management operations. Accesses in this range (called "management reads" or "management writes" in this section) will be ignored by the memory controller and will always hit in the cache even when the access made is uncacheable. The usual lower address bits select the cache index, two address bits select the way of the cache. [Figure 13 on page 95](#) and [Table 54 on page 95](#) show how a cache management address is created

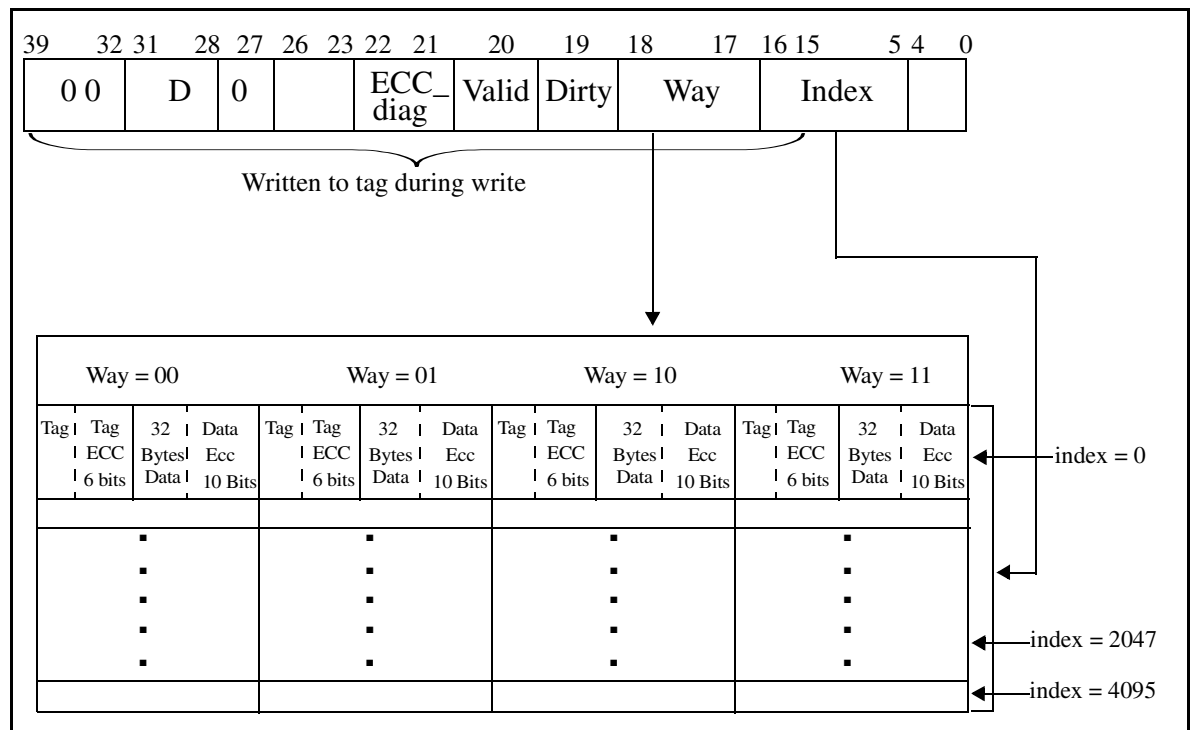


Figure 13: Cache Management Address

Table 54: Management Address

Bits	Use
4:0	Offset (always 0, or treated as such by L2).
16:5	Index. The cache is divided in to sets, each containing four 32 byte lines. These bits select which set is indexed. The BCM1250 has all 4096 sets, the BCM1125/H has only 2048 so bit 16 is not part of the index. Bits 16:15 are also written to the tag (they are used as tag bits in 256 KB and 128 KB cache configurations).
18:17	Way. These bits select which of the four lines in a set are accessed. During management writes these address bits are written to the tag.
19	Dirty. The dirty bit indicates that the line held in the L2 cache contains more recent data than the block in main memory. If this bit is set the block must be written back to memory before the line can be used for another block. During management writes this address bit is written to the dirty bit and the tag.
20	Valid. The valid bit indicates that the line contains valid data. During management writes this address bit is written to the valid bit and tag.
22:21	ECC_diag[1:0]. If this field is nonzero during a management mode access an ECC diagnostic access is performed. During management writes these address bits are written to the tag.
26:23	During management writes these address bits are written to the tag.
39:27	b0000000011010 (00_D000_0000 - 00_D7FF_FFFF) this address range selects management access. During management writes these address bits are written to the tag.

Management accesses may be done with cacheable or non-cacheable operations. However, the L2 cache will always operate on a full 32 byte cache line. On reads there is not a problem since the destination will filter the data (data is always carried on its natural byte lane on the ZBbus), but writes that are smaller than a cache block will cause random values to be written to the other bytes in the line. Uncacheable writes will normally be smaller than a full cache block (the exception being uncached-accelerated writes that have merged to a full block in the CPU write buffer), so they cannot be used to set particular values in the cache. However, uncached writes may be used for initializing the cache (where the data is unimportant, since the goal is to clear the valid bit for all the lines in the cache) or clearing data ECC errors by overwriting the bad data. This works because the other bytes on the bus during an uncached write will always be driven with valid (although UNPREDICTABLE) data.

The management accesses will still signal hit or miss to the performance counters based on the comparison between the management access address and the cache tags.

Following any management access to the cache that is to an enabled way, the replacement algorithm state is set such that the way of the cache that was accessed is used for the victim (if one is needed) on the next cache access (note this is the next access **not** the next miss). Therefore a line can be flushed from the cache by doing a management read of the line to be flushed followed by a regular cacheable read of a block that uses the same index and is not currently in the L2. The management access selects the way as the victim for the subsequent fill. To avoid the line being flushed ending up in the L1 cache, the management access is likely to be uncacheable, however the regular access must be cacheable. The ordering must be maintained by software (an ALU operation can be performed on the data returned by the uncached management access to ensure it has completed before the cacheable load is issued). If multiple lines are to be flushed care should be taken that the dummy cacheable fetch has completed before the next uncacheable management access (i.e. an ALU operation should be performed on the dummy data or a SYNC used).

Management mode accesses flow through the same paths as normal accesses. If a write is done to a line followed by a read of the same address, the data used to satisfy the read may be bypassed from the cache input queue. This increases the performance in normal operation, but is a problem when the cache RAM is being tested. There must be more than six ZBbus cycles between the write data (i.e. the ZBbus D-phase for the write transaction) and the read request (i.e. the ZBbus A-phase for the read transaction) arriving at the cache to ensure the read is from the RAM array. This can be arranged by doing two reads from the same location. Alternatively, a sequence of writes to other addresses can be done before the read.

## STANDARD MANAGEMENT MODE ACCESSES (BOTH ECC\_DIAG ADDRESS BITS ZERO)

Standard management mode accesses are any made to the management address range with the ECC\_diag address bits clear.

Management reads will return the line from the addressed index and way. The normal ECC checking and correction is performed. Data with correctable ECC errors will be returned with the errors corrected, flagged as ECC corrected. Data with uncorrectable errors will be flagged with the uncorrectable data ECC error code. Similarly, if the tag has a correctable ECC error it will be corrected and the data returned with the tag ECC corrected flag, and if the tag has an uncorrectable ECC error data is returned with the uncorrectable tag error code.

As the management read is done the tag associated with the entry is transferred to the **I2\_read\_tag** register. This register will contain the tag after ECC correction and the raw tag ECC bits. The register can be read by the CPU with an uncacheable read. If cacheable reads are used for the management read that gets the data care is required since the CPU implements run-under-miss and there is no ordering guaranteed between cacheable and uncacheable reads. The ordering can be forced by the CPU performing a SYNC or an ALU operation on the data read from the cache management access (e.g. add zero) before reading the tag register. There is no ordering problem if the management access is done using an uncacheable read because the CPU will maintain the order between the management access and the **I2\_read\_tag** register, but uncached access is less efficient if all the data in the line needs to be examined.

Management writes will write the data to the addressed index and way. The data is written with correct ECC bits. The previous data in the addressed cache line will be lost, evicts are never done. If the previous data had an ECC error this will not be reported. A new cache tag (with correct ECC) is written, tagging the line with the address used in the write (the bits put in the tag are shown in [Figure 13 on page 95](#)). The valid and dirty bits of the tag are copied from bits 20 and 19 of the address.

The L2 cache can be invalidated by doing standard management writes to each index and way of the cache using an address that causes the tag to be written with the valid bit clear. In addition to invalidating the cache this will cause correct ECC to be written to all the cache blocks and tags, so there will not be spurious ECC errors on the first use of a cache line. This must be done during system startup before the cache can be used.

ECC error recovery code should not directly read the erroneous data from the cache. If the data contains an uncorrectable error it will still be signalled as such, so the CPU would see it as a cache error. However the bus watcher in the SCD logs the data whenever an uncorrectable error is reported, and the L2 controller logs the address and tag (with the error) in the **I2\_ecc\_tag** register.

## ECC DIAGNOSTIC MANAGEMENT ACCESSES (ECC\_DIAG BITS NONZERO)

Management accesses with the ECC\_diag field nonzero modify the behavior of the cache to allow access to the raw memory bits for testing.

Management reads with the ECC\_diag [0] bit set will return the raw data from the addressed index and way. The data ECC checking and correction is disabled, the raw data from the cache memory array will be returned and will always be marked valid.

As the management read is done the tag associated with the entry is transferred to the **I2\_read\_tag** register. Again the ECC correction is modified. The register will contain the raw tag data and the raw tag ECC bits. The register can be read by the CPU as in the standard management access, with the same care required about read ordering.

The ECC\_diag address field modifies the behavior of management writes to allow data and tag ECC errors (of any number of bits) to be written into the cache, to test both the hardware and software ECC detection and recovery mechanisms.

Management writes with the ECC\_diag address field nonzero are used to check the ECC bits. The combinations allow direct writes from the data into the cache tag, and can write using ECC information from a previous write. [Table 55](#) summarizes the operations.

**Table 55: ECC Diagnostic Operations**

<b>ecc_diag[1:0]</b>	<b>Read</b>	<b>Write</b>
2'b00	Normal	Normal
2'b01	Raw Access ECC disabled	Write data to the cache, write the tag from the data bits. Generate tag ECC bits. The tag address bits are written from ZBbus bits [23:0]. The tag valid bit is written from ZBbus bit [24]. The tag dirty bit is written from ZBbus bit [25]. Note that these are from the double-word at offset h18.
2'b10	Reserved	Write current data with the ECC bits from the previous cache write.
2'b11	Reserved	Write data to the cache, write the tag from the data bits (as in code 2'b01). Use tag ECC bits from the previous write.

Data ECC errors can be stimulated with two writes. The first write (which is a normal write or standard management write) sets the index and way for the test along with the data ECC bits to be used for the test. The second write (an ECC diagnostic write of type 2'b01) replaces only the actual data. If the data of the second write differs by a single bit from the data in the first write a subsequent standard read will get the data from the first write flagged with a corrected ECC error. If the second write had two bits difference the subsequent read will get an uncorrectable ECC error.

Tag ECC errors can also be stimulated with two writes. The first write writes the tag bits with correct ECC to be used for the test (using ECC diagnostic write of type 2'b10). The second write changes the tag bits but uses the ECC bits from the first write (using an ECC diagnostic write of type 2'b11). If the tag generated by the first write differs by a single bit from the tag in the second, a subsequent standard management read will be flagged with a corrected tag ECC error and the **I2\_read\_tag** register will hold the reconstructed tag for the second write. If the tag generated by the first write had two bits difference the subsequent read will get an uncorrectable tag ECC error.



## CACHE CONFIGURATION REGISTER

There is a 4 bit register **I2\_misc\_config** that can be used to modify the L2 behavior. It is similar to the **I2\_way\_enable** register in that it is hidden and written from address bits. In most systems this register should not be modified. The register is accessed by writing dummy data to address 00\_10043r00. Bits 8 and 9 are reserved and should be written as zero. If bit 10 is set the L2 cache will use the low priority memory controller blocker rather than the high priority one. If bit 11 is set the automatic ECC cleanup is disabled. On parts with system revision indicating PERIPH\_REV3 and later the register can be read back from the **I2\_misc\_value** register.

## EXAMPLE STARTUP CODE TO CLEAR THE L2 CACHE

The following code will work on the BCM1250 and BCM1125/H using 4096 sets. It could be optimized for the BCM1125/H to only clear 2048 sets.

```

# Save the old status register, and set the KX bit.
# This allows 64 bit addressing through xkphys.

        mfc0    t2,C0_SR
        or      t1,t2,M_SR_KX
        mtc0    t1,C0_SR

# Start the index at the base of the cache management
# area, but leave the address bit for "Valid" zero.
# Note that the management tags are at 00_D000_0000,
# which cannot be accessed through kseg1,
# so generate a 64-bit xkphys uncacheable address to get to it.
# The PHYS_TO_XKPHYS mapping adds 9000_0000_0000_0000 to the address

        dli     t0,PHYS_TO_XKPHYS(A_L2C_MGMT_TAG_BASE)

# Loop through each entry (4096) and each way (4)

        li      t1,L2C_ENTRIES_PER_WAY*L2C_NUM_WAYS

# Write a zero to the cache management register at each
# address. Align and do four per loop to run faster since the code
# is running in uncached space.

1:      .align 4
        sd      zero,0(t0)
        sd      zero,CACHE_LINE_SIZE(t0)
        sd      zero,2*CACHE_LINE_SIZE(t0)
        sd      zero,3*CACHE_LINE_SIZE(t0)
        daddu   t0,(4*CACHE_LINE_SIZE) # size of a cache line
        subu    t1,4
        bne     t1,0,1b

#
# Restore old KX bit setting
#

        mtc0    t2,C0_SR

        j       ra                # return to caller

```

## REGISTERS

**Table 56: Level 2 Cache Tag Register**

Bits	I2_read_tag - 00_1004_0018 I2_ecc_tag - 00_1004_0038 READ ONLY
4:0	0
14:5	Index
15	Tag bit 39 (holds index bit 15 in a 256KB or 512KB cache and is a tag bit in a 128KB cache)
16	Tag bit 40 (holds index bit 16 in a 512KB cache and is a tag bit in a 256KB or 128KB cache)
38:17	Tag bits
39	0
45:40	Tag ECC (raw).
47:46	Way
48	Dirty
49	Valid
59:50	Data ECC (raw)
63:60	Reserved

The L2 tag registers are UNPREDICTABLE until the first event that causes them to be written. The performance of reads to the registers will be low, while a read is in progress hardware will delay accesses to the L2 Cache registers and any registers in the SCD.

**Table 57: Level 2 Cache Settings Register**

Bits	I2_misc_value - 00_1004_0058 READ ONLY Sytem Revision PERIPH_REV3 and later only
3:0	Cache quadrant information [t,b,r,l] (Broadcom Use Only)
7:4	I2_cache_disable register value
8	Reads back enable for use of low priority memory blocker from value written as bit [10] in the I2_misc_config register address
9	Reads back disable of ECC cleanup from value written as bit [11] in the I2_misc_config register address
13:12	Reads back I2_way_enable register
63:14	Reserved





---

This Page is left blank for notes



This Page is left blank for notes



**Broadcom Corporation**

## Section 6: DRAM

### INTRODUCTION

The part incorporates a DDR SDRAM controller that works closely with the level 2 cache to provide a high performance memory system. The BCM1250 memory controller includes two channels each providing a 64 bit data path with 8 bit ECC. The BCM1125/H parts have a memory controller with a single channel providing a 64 bit data path with 8 bit ECC. Each channel can directly support up to two standard two physical bank JEDEC 184 pin DDR DIMMs (for a total of four DIMMs on a BCM1250) running at 133MHz clock (266MHz data rate, sometimes called PC2100 DIMMs based on PC266A parts) and allows for performance to increase as the DIMMs support higher data rates. In a more controlled electrical environment, with the DRAMs mounted on the main board and traces tightly controlled the memory controller can run up to 200MHz clock giving a 400MHz data rate. The peak memory bandwidth for a single channel using standard (133 MHz clock) DIMMs is 16 Gbit/s and increases up to 50 Gbit/s for a high speed (200 MHz clock) design using both channels.

The memory controller supports three types of memory all of which use SSTL\_2 signalling levels and the conventional multiplexed address bus.

- **Standard DDR SDRAM** which may be mounted on board or on DIMMs.
- **Standard DDR SGRAM** which can be on DIMMs but are designed to be mounted on the main board and run with faster cycle times than standard DDR. The memory controller does not make use of any of the graphics features.
- **Fujitsu/Toshiba/Samsung DDR FCRAM** (Fast Cycle RAM) which is designed to be mounted on the main board. These parts have smaller rows and a much simplified command set allowing them to have both faster cycle times and lower access latency than standard DDR.

Each channel can support up to 1 GByte of memory using 256 Mbit technology parts. As larger DRAMs become available this will increase to 2 GByte with 512 Mbit parts and 4 GByte with 1 Gbit parts (note that gigabit parts that use 14 row address bits are only supported on the BCM1125/H and versions of the BCM1250 supporting PERIPH\_REV3 or later). A special large memory mode allows these sizes to be doubled, but requires the use of an external decoder and may be lower performance.

### A COMMENT ON THE TERM BANK

The term "Bank" is unfortunately used in a couple of different ways when discussing DDR SDRAM memory systems.

The first use of bank is to describe the organization within a memory device, most devices have four internal banks and thus need two bank address bits (BA[1:0]). The number of banks on a device sets the number of pages that it can have open at a given time.

The second use of bank is to describe the physical groups of devices that are enabled by a single chip select. Standard DDR 184 pin DIMMs can have two physical banks and thus need two chip selects. Each physical bank corresponds to one load on the data and DQS signal lines. The number of loads on the address lines for a physical bank depends on the organization of the memory devices. For the 64 bit wide data path used by the memory channel using x8 devices gives 8 address loads per physical bank (9 if ECC is used). Using x16 devices gives 4 (or 5) loads and using x32 devices gives 2 (or 3) loads.

In this chapter the unqualified term bank will only be used to refer to internal banks. Physical banks will always be described as such.

## MEMORY CONTROLLER ARCHITECTURE

The memory controller consists of a ZBbus interface and one or two memory channel interfaces. The channels on the BCM1250 are numbered 0 and 1, the BCM1125/H only has channel 1. The block diagram is shown in [Figure 14 on page 105](#).

The ZBbus interface consists of:

- The request queue (RQQ).
- A data buffer (DBF) for read and write.
- The scheduler for issuing requests to memory (out-of-order).

The Request Queue (RQQ) is a 16 entry shift-register queue. Entries are added in-order as requests come from ZBbus and are issued out-of-order to the memory channels. Any entry in the RQQ can be issued to memory. Entries in the queue can be reserved for use by network DMA, this reduces the latency of reads and improves performance when using slower or highly loaded memory systems.

The Data Buffer (DBF) is also 16 entry and is shared between reads and writes. Each entry in the buffer is a full 256 bit block wide. The buffer has multiple pairs of read and write ports; one pair for ZBbus which can transfer a full block each access, and a pair for each of the memory ports that transfer 64 bit double words.

When a request is received from ZBbus, entries are allocated in both the RQQ and the DBF. Entries are normally added to the RQQ in sequential order. However, checks are first done to detect any write to read or write to write address dependencies.

The RQQ holds the memory access information and initially the DBF is empty. If the request is a write then the request is not ready for submission to the memory until the data has been received from ZBbus and written to the DBF, at which time the RQQ entry becomes valid for the memory scheduler. Both entries are freed when the write request is issued to the SDRAMs. If the request is a read the RQQ entry becomes valid as soon as the L2 cache reports a miss, and the DBF entry is used to stage the data when it returns from memory. In this case the entries are freed when the data is sent on ZBbus.

The memory controller calculates an ECC during writes, and will check it during reads. This provides correction of all single bit errors and detection of double bit errors. An extra 8 bit ECC bus accompanies the 64 bit data bus, matching the standard 72 bit DDR DIMMs. ECC checking can be disabled.

The memory controller will service uncacheable requests from ZBbus. A read-modify-write cycle for the full line is needed for uncacheable writes, merging the new data into the existing block and recalculating the ECC. The additional memory accesses and the poor utilization of the DBF will cause the memory system performance to degrade if many uncacheable accesses are done.

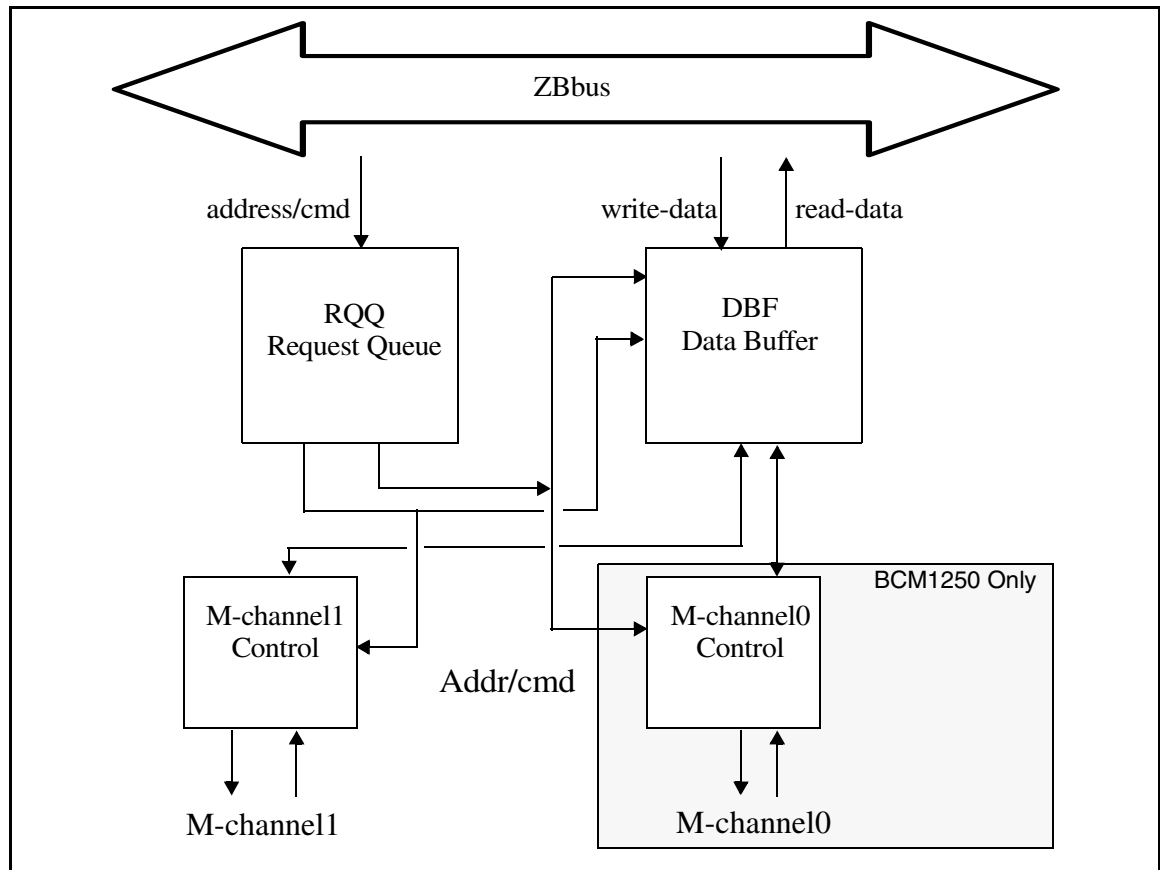


Figure 14: Memory Controller Block Diagram

Entries are selected for issue from the RQQ based on the state of the other entries and the status returned by the memory pipeline controller for each of the channels. The issues can be out-of-order, and use the following rules:

- Any access to configuration registers causes serialization of the queue. These accesses can be in progress for some time (40-50 ZBbus clocks), so should not be done during normal operation.
- Any access with address conflict with an entry already in the queue will be only issued when it reaches the head of the queue. (It can be bypassed by other requests).
- If priority is enabled in the **mc\_config\_1** register, any read requests from I/O bridge 1 are given priority and are issued in-order. If a read request conflicts with an existing write, the write is also given priority.
- If a channel is currently doing a read, all reads to that channel will be issued ahead of any writes.
- If a read has been bypassed enough times to reach its *age\_limit* (see below) it will go ahead of other reads.
- If a channel is currently doing a write, all writes to that channel will be issued ahead of any reads up to a maximum number of consecutive writes that are set in the channel configuration register.
- If the open page policy is being used an entry that hits on an open page goes first.
- If the open page policy is disabled an entry that is to a precharged bank (closed bank) goes first.
- If a channel is currently idle, reads will be issued before writes.
- Otherwise entries are issued in the order they were allocated.

When a channel switches from writes to reads (and back) there is a delay while the data bus lines and timing strobes (DQSs) turnaround from the controller driving to the memory driving (or the reverse). To get the best bandwidth from the channel the controller batches writes and reads to minimize the number of these turnaround delays. If the controller is doing reads then any writes get buffered in the RQQ. Eventually there will either be no new read requests or the RQQ will become full of writes, the controller will then only have writes to service and can drain them. Since all writes are posted (once they have been sent the sender receives no reply) there is no problem in delaying them in this way.

When the controller is performing a write burst any reads will suffer additional latency as they wait in the RQQ for the writes to drain. The simple case is that the read is trapped behind all the writes that have been buffered up during the previous read burst. However, since the writes drain the RQQ additional writes could be inserted into the queue further delaying the read. To minimize the extra latency a read will suffer during a write burst, the controller counts the number of writes done in the burst and will switch to reads if the burst is longer than the `wr_limit` (and there are reads waiting). The write limit is set per-channel in the `mc_config` register. The memory bus utilization can be increased by increasing the limit, the average read latency will be decreased by decreasing the limit.

A similar situation arises for a channel that is just doing reads. Reads to an open page will be issued ahead of reads to a closed page. Once a read is completed the data can be returned and the RQQ entry freed. It is therefore possible for more requests to the open page to be added to the queue, these will also bypass the read to the closed page. To limit the length of time the read to the closed page is blocked, the controller limits the number of reads that may pass it. Every time a read is bypassed its age is incremented, when it reaches the `age_limit` no new entries will be permitted to pass it and it (and any entries ahead of it) will drain from the queue. The `age_limit` is set per-channel in the `mc_config` register. The trade-off is the same as for the `wr_limit`: the memory bus utilization will be increased by increasing the limit, the average read latency will be decreased by decreasing the limit.

Reads from I/O bridge 1 can be given priority to ensure timely servicing. In most systems this should be enabled (which is the default) to avoid transmit buffer underruns at high data rates. The low relative frequency of I/O bridge 1 DMA requests minimizes the impact of this prioritization on other requests, but ensures that the latency sensitive requests are not delayed behind a high frequency request stream (for example from the CPU or Data Mover). Along with this RQQ entries may be reserved for I/O bridge 1 requests. If the number of entries in the queue reaches the limit set in the `job1_qsize` field in the `mc_config_1` register then all agents apart from I/O bridge 1 will back off from accessing the memory until the number of entries in the queue falls below the `job1_qsize` set in `mc_config_0`. Some hysteresis is provided by having the two limits. The number of buffers that should be reserved depends on the number of active DMA channels, their bandwidth, and the amount of other memory activity in the system. For a relatively high load on the system a reasonable starting point is to reserve 5 buffers. Note that if zeros are used for the `job1_qsize` fields the memory controller performance will be extremely poor.

Each memory channel interface consists of:

- The memory pipeline control queue and scheduler (MCQ).
- Memory data FIFO registers (MFIFO).
- Configuration registers (CRREG).

The MCQ keeps track of the activities of the memory cycle, open rows and banks. It supplies this information to the issue logic for the request queue and the memory scheduler.

## MEMORY ACCESS SEQUENCING

Certain applications need more control over the memory access pattern than is available using the normal operations in the memory controller, for example when the memory is being used for transaction logging. The controller was designed to optimize the normal case, but there are several methods that software can use to force the sequence.

The ultimate control is to set the `force_seq` bit in the `mc_config` register. This forces the controller to send requests to the SDRAMs in the same order their address phase happens on the ZBbus. This prevents most of the memory optimizations and is likely to be low performance.

In many cases a memory barrier operation is sufficient. This can be provided by an access to the memory controller configuration registers. An access to one of the channel registers will force all reads/writes ahead of it to complete before any behind it. In the normal case a CPU doing a write of A to address 1 followed by a write of B to address 2 provides no guarantee that A gets written into the SDRAM before B. However, if the CPU performs a write of A to address 1 followed by a write of 0 to the `mc_test_ecc` register and finally the write of B to address 2 then A is always written to memory before B. All these operations are posted, so the CPU has no way of knowing when they complete.

If the CPU needs to both place a memory barrier and be informed of the completion then a read may be done to one of the controller configuration registers. If the CPU performs a write of A to address 1 followed by a read of the `mc_test_ecc` register and finally a write of B to address 2 then A is always written to memory before B and when the CPU sees the result of the read it knows that A will have been written to the SDRAM (but B may still be in the queue). A sync instruction or use of the read result can be used to stall the CPU until the read completes.

## CLOCK RATIOS AND CLOCKING SCHEME

The memory clock range is from 133 MHz - 200 MHz produced by dividing down the ZBbus clock, which is half the CPU core frequency. The ratio of the clocks can be set from 1:2 to 1:4.5 in steps of 0.5, by programming the channel configuration register. The channels may run at different speeds.

*Table 58: Clock Speed*

CPU Clock	ZBbus Clock	Memory Clock 4.5:1	Memory Clock 4:1	Memory Clock 3.5:1	Memory Clock 3:1	Memory Clock 2.5:1	Memory Clock 2:1
1200	600	133.3	150.0	171.4	200.0	240.0	300.0
1150	575	127.8	143.8	164.3	191.7	230.0	287.5
1100	550	122.2	137.5	157.1	183.3	220.0	275.0
1050	525	116.7	131.3	150.0	175.0	210.0	262.5
1000	500	111.1	125.0	142.9	166.7	200.0	250.0
950	475	105.6	118.8	135.7	158.3	190.0	237.5
900	450	100.0	112.5	128.6	150.0	180.0	225.0
850	425	94.4	106.3	121.4	141.7	170.0	212.5
800	400	88.9	100.0	114.3	133.3	160.0	200.0
750	375	83.3	93.8	107.1	125.0	150.0	187.5
700	350	77.8	87.5	100.0	116.7	140.0	175.0
650	325	72.2	81.3	92.9	108.3	130.0	162.5

**Table 58: Clock Speed (Cont.)**

CPU Clock	ZBbus Clock	Memory Clock 4.5:1	Memory Clock 4:1	Memory Clock 3.5:1	Memory Clock 3:1	Memory Clock 2.5:1	Memory Clock 2:1
600	300	66.7	75.0	85.7	100.0	120.0	150.0
550	275	61.1	68.8	78.6	91.7	110.0	137.5
500	250	55.6	62.5	71.4	83.3	100.0	125.0
450	225	50.0	56.3	64.3	75.0	90.0	112.5
400	200	44.4	50.0	57.1	66.7	80.0	100.0

Table 58 shows possible memory clock frequencies for different divide ratios and CPU clock speed. Table 59 shows the percentage difference for the closest frequency below some common DIMM frequencies for each of the CPU clock speeds (entries are left blank if the closest frequency is more than 10% under the required one)

**Table 59: Percent Deltas from Popular DIMM Frequencies**

CPU Clock	ZBbus Clock	133 MHz DIMM Clock	143 MHz DIMM Clock	166 MHz DIMM Clock	183 MHz DIMM Clock	200 MHz DIMM Clock
1200	600	0.0	-6.8	-10.0	-6.5	0.0
1150	575	-4.1	-	-1.4	-	-4.2
1100	550	-8.3	-3.8	-5.7	0.0	-8.3
1050	525	-1.5	-8.2	-10.0	-4.5	-
1000	500	-6.2	-0.1	0.0	-9.1	0.0
950	475	-	-5.1	-5.0	-	-5.0
900	450	-3.5	-10.1	-10.0	-1.8	-10.0
850	425	-8.9	-0.9	-	-7.3	-
800	400	0.0	-6.8	-4.0	-	0.0
750	375	-6.2	-	-10.0	-	-6.3
700	350	-	-2.1	-	-4.5	-
650	325	-2.5	-9.1	-2.5	-	-
600	300	-10.0	-	-10.0	-	-
550	275	-	-3.8	-	-	-
500	250	-6.2	-	-	-	-
450	225	-	-	-	-	-
400	200	-	-	-	-	-





## MEMORY CONFIGURATIONS

Before the first SDRAM access starts, the memory configuration registers must be initialized, and the correct startup sequence issued to the memory chips. This is normally done by the BootROM code. The Broadcom CFE firmware includes memory controller initialization which uses the Serial Presence Detect (SPD) EEPROM information to support a wide range of memory types. This code provides the best starting point for memory initialization.

### MAPPING

The physical address space is mapped to memory address space. The physical address map has four 256 Mbyte regions and one expanded 508 Gbyte region of the address space allocated to the memory controller. The memory controller will map the four 256 Mbyte regions into a single contiguous 1 Gbyte region of memory address space for use by the memory channels. Memory address space is only used in the configuration of the memory controller.

The constraints on the physical memory map of the part (see [Section: "Memory Map" on page 34](#)) caused the main memory address ranges in the low 4GB of the address space to be allocated in non-contiguous blocks. However, the memory controller configurations are more flexible if it sees them as a single contiguous block. For example, consider a system where the actual memory consists of one single physical bank (i.e. only one chip select) 512 MB DIMM. Since there is only one chip select this must be configured to have a single address range. But the physical address map has one 256 MB memory block starting at zero (which contains the CPU exception vectors and must be present) and the next 256 MB block starts at 00\_8000\_0000. If the memory controller used physical addresses there would not be a way to configure the address range to use the whole DIMM. However, in memory address space the two memory blocks can be made contiguous and the DIMM can be configured.

**Table 60: Mapping Physical Address To Memory Controller Address**

Physical Address Space (used to access the memory)	Memory Address Space (used to configure the controller)	Size
00_0000_0000 - 00_0FFF_FFFF	00_0000_0000 - 00_0FFF_FFFF	256 MB
00_8000_0000 - 00_8FFF_FFFF	00_1000_0000 - 00_1FFF_FFFF	256 MB
00_9000_0000 - 00_9FFF_FFFF	00_2000_0000 - 00_2FFF_FFFF	256 MB
00_C000_0000 - 00_CFFF_FFFF	00_3000_0000 - 00_3FFF_FFFF	256 MB
01_0000_0000 - 7F_FFFF_FFFF	01_0000_0000 - 7F_FFFF_FFFF	508 GB

The mapping for the four physical address ranges that have zero for the upper eight bits is configured in the channel configuration register. They can be mapped into the first four 256 MB blocks of the memory address space. The default mapping is shown in [Table 60 on page 109](#). There is normally no need to change from the defaults.

### CHANNEL SELECT

On the BCM1250 the two channels may be interleaved or they may be independent. If they are interleaved then they must match in all parameters. When they are interleaved a single address bit is used to select between them. When the channels are independent the start and end address set for the chip select generation will also select between the two channels.

## CHIP SELECT

A memory controller channel has four active low Chip Selects:CS[3:0]. The address range that they become active for is configurable. On the BCM1250 if the two channels are interleaved then the chip selects for each channel are programmed to an identical address range, which is twice the size of the memory attached to one of the channels. Within a channel there are three methods for generating the chip selects: interleaved-CS, msb-CS and mixed-CS.

- msb-CS** divides the memory space into contiguous or non-contiguous sub-spaces based on address ranges. The start and end address for each select line is individually programmable. For example CS[3] covers the top portion of memory space and CS[0] covers the bottom portion. The memory size covered by each CS can be different.
- interleaved-CS** decodes the chip select from two non-MSB memory address bits, for example the lower column bits or the lower row bits. This increases the possibilities for having active banks. In this configuration the channel must be assigned a contiguous memory address range and each chip select covers the same memory size.
- mixed\_CS** allows interleaving between a pair of chip selects (either CS[0] and CS[1], or CS[1] and CS[2], or CS[2] and CS[3]) and allows the others to be set using their start and end registers. This mode allows devices to be interleaved when only two chip selects are in use.

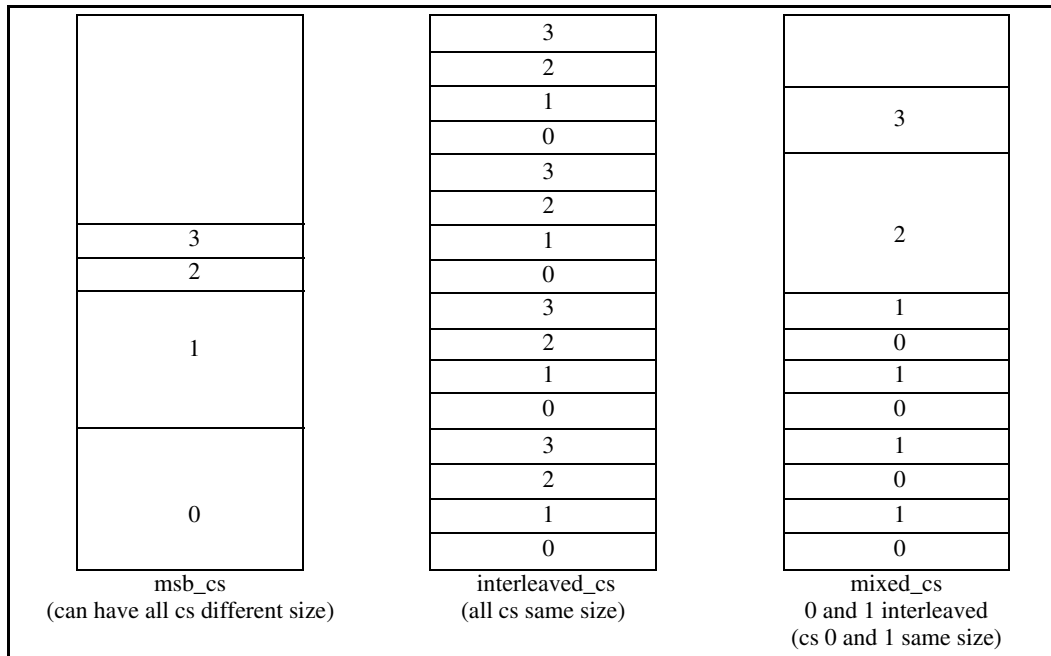


Figure 15: Chip Select Options

Figure 15 on page 110 shows some chip select options. On the left msb\_cs is used to separate the four chip selects into different regions. The diagram in the center shows a full interleaved\_cs, which requires all the chip select regions are the same size. On the right mixed\_cs has the SDRAMs on chip select 0 and 1 interleaved and keeps the SDRAMs on chip select 2 and 3 separate. For the memory covered by chip selects 0 and 1 there can be twice as many active banks (although there maybe a chip-to-chip bubble on some accesses).

The address range covered by each of the chip selects is set in the **mc\_cs\_start** and **mc\_cs\_end** registers. The start address register holds address bits [39:24] that when extended with zeros in bits [23:0] give the lowest address in the range. The end address register holds address bits [39:24] that when extended with zeros in bits [23:0] give the first address above the range. The comparison is done after addresses have been translated from physical addresses to memory addresses (see [Section: "Mapping" on page 109](#)).

If two chip select regions are interleaved (*mixed\_cs* mode) their start address registers must be set to the start of the common range, and their end address registers are set to the end address of the common range plus 1 (i.e. they are programmed identically, to a range twice the size of one of them). The address bit that selects between the two chip selects is configured by writing a single one in that bit position of the **mc\_cs\_interleave** register.

If all four chip select regions are interleaved (*interleaved\_cs* mode) their start address registers must be set to the start of the common range, and their end address registers are set to the end address of the common range plus 1 (i.e. they are programmed identically, to a range four times the size of one of them). The address bits that selects between the two chip selects is configured by writing two ones in adjacent bit positions of the **mc\_cs\_interleave** register.

If a chip select is not used its start and end address should both be set to zero.

If the two channels are interleaved then the chip select region sizes will be doubled and must be identical for both channels.

If a request is received by the memory controller that does not match in any of the chip select regions no SDRAMs will be selected. On writes the data will be discarded. On reads the memory controller will terminate the cycle by reading UNPREDICTABLE data. The controller can be configured to return either a bus error or a valid data flag with the data. In normal operation the bus error would be used to signal that a bad memory address was used. During system testing and sizing it may be useful to have the controller always return valid data. The controller will not hang during an access with no chip select asserted.

### EXAMPLE CHANNEL AND CHIP SELECT CONFIGURATIONS

Configuring the memory map for channels and chip selects is usually straightforward, but there are a few special cases. In this section some example configurations are presented. The values that are set for chip select start and end addresses are shown in the figures, these represent bits [39:24] of the address.

The simplest memory system uses a single chip select on a single channel. Using 256Mb memory technology a memory system could be built using 4 16Mx16 memory chips (5 if ECC is required) to give a total of 128MB of memory.

Figure 16 shows how this could be configured. The standard mapping from physical to memory addressing is used, although only the First SDRAM region is needed for 128MB from physical address 00\_0000\_0000 to 00\_07FF\_FFFF. This maps to the same address in memory space. The controller chip selects are all disabled apart from channel 0 chip select 0 which is set with start and end to cover this range. Any reads to the space that is not configured will result in a bus error or UNPREDICTABLE data being returned depending on the setting of the berr\_disable bit in the mc\_config register.

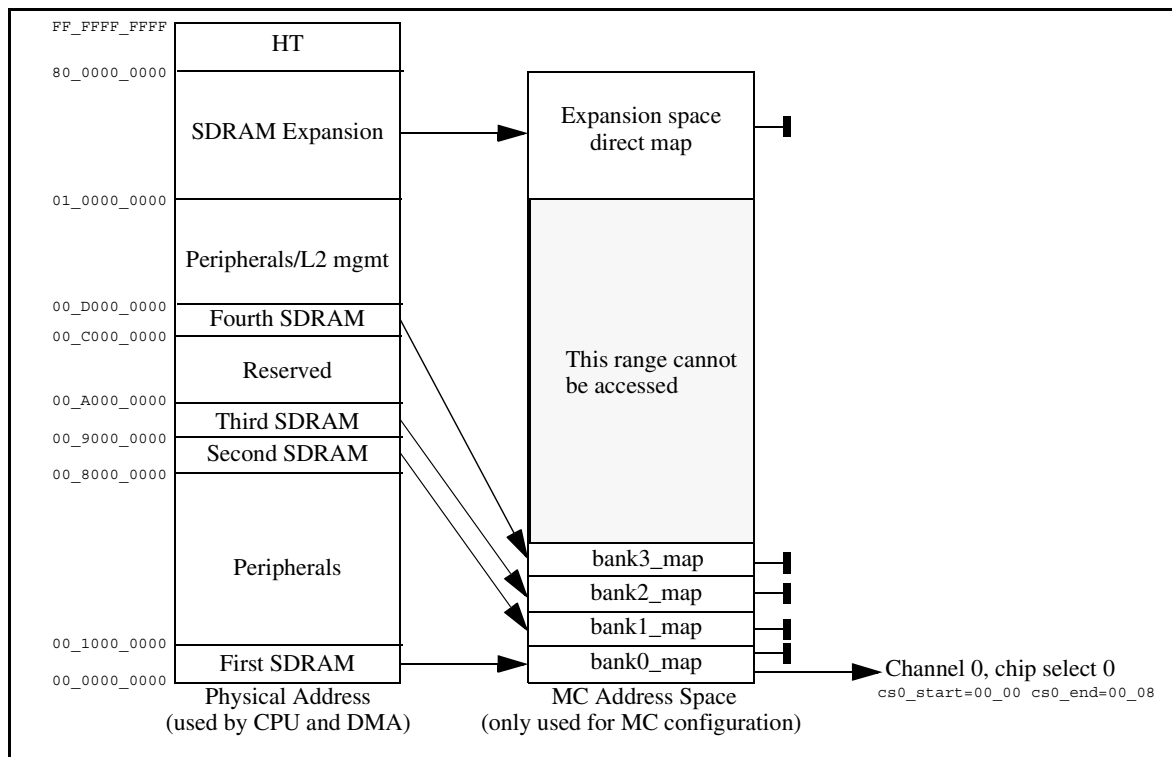


Figure 16: Example Single Channel 128MB

Using the 512Mb memory technology and using 8 (or 9 with ECC) 64Mx8 allows each chip select to have 512MB of memory. Two physical banks will therefore allow a 1GB memory system to be built. There are three interesting configurations (two on the BCM1125/H). In these advantage is taken of the memory address space collapsing the configuration range into a contiguous space. In the physical address space the CPU and DMA engines use the 1GB is split up into four 256MB chunks, but in the actual memory there are just two 512MB physical banks. The translation into memory address space pulls the four physical address regions into a single 1GB chunk. In the simplest configuration this can be split in two and allocated to chip selects.

Figure 17 shows the first configuration. This configuration has chip select 0 for the low half and chip select 1 for the upper half and the start and end addresses of their range are set to reflect this.

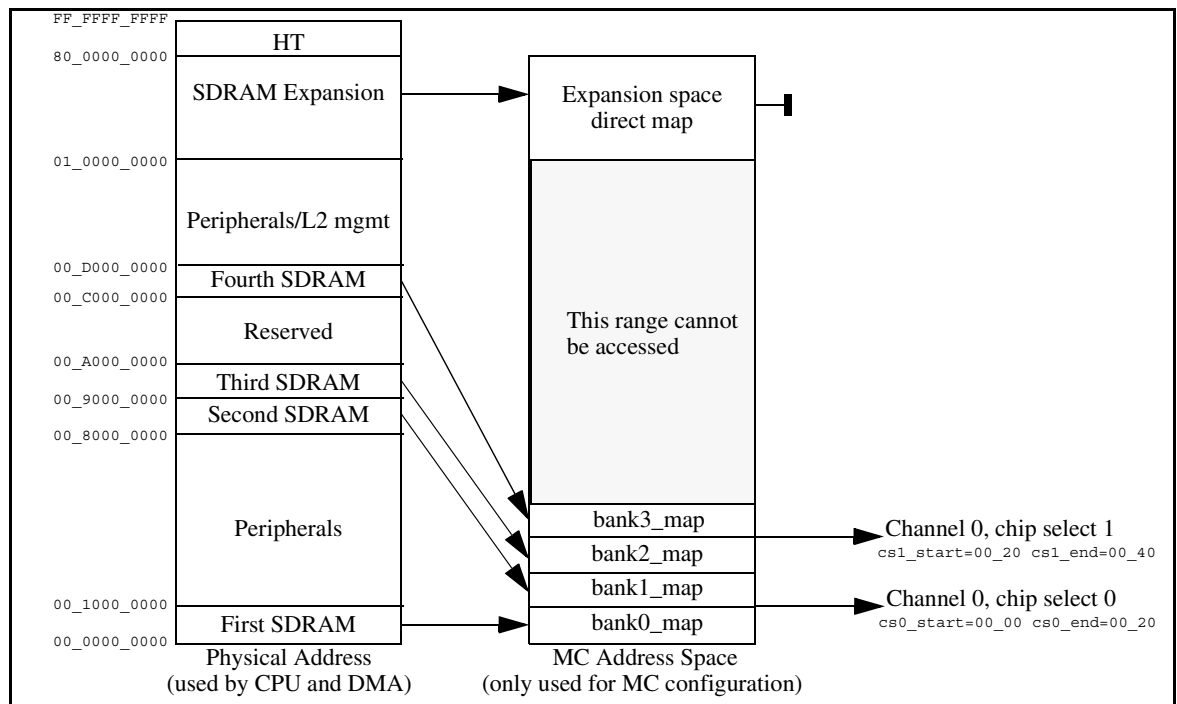


Figure 17: Example 1GB with two chip selects on one channel

Figure 18 is an alternative that uses chip select interleaving to (on average) increase the number of active banks. Notice the chip select ranges are now the same for both selects.

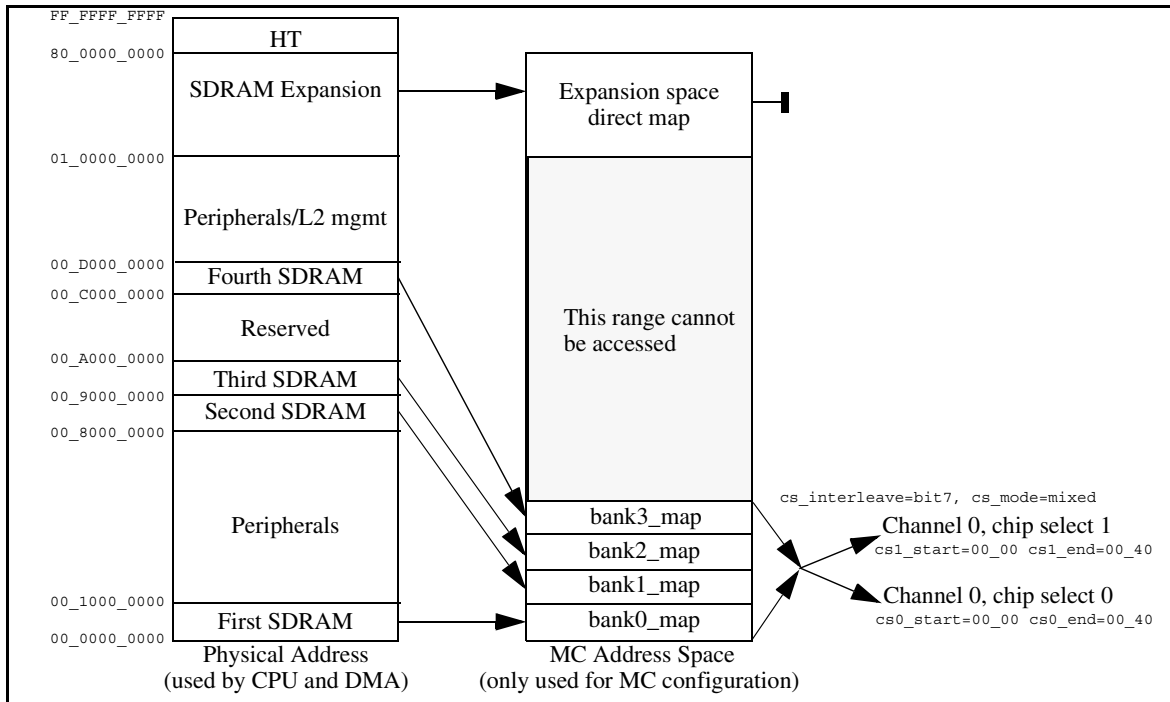
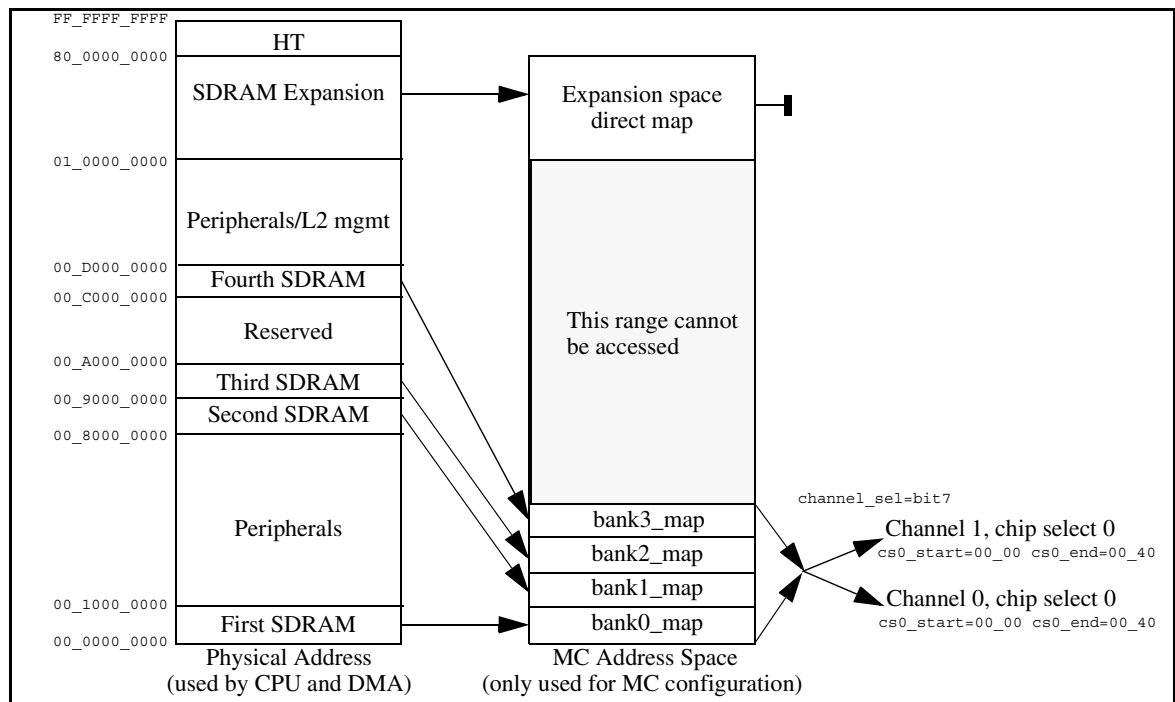


Figure 18: Example 1GB with two chip selects interleaved on one channel

On the BCM1250 there is a higher performance possibility. Rather than putting both physical banks on the same channel one can be put on each channel. This doubles the number of data bits in use (since the controller can run both channels in parallel), allowing twice the peak bandwidth.

Figure 19 shows this, with the interleave on a relatively low address bit. Notice that again the chip select ranges are the same for both selects, but in this case they are for different channels.



**Figure 19: Example 1GB with two chip selects interleaved across both channels**

These examples have benefited from the memory address space map having a contiguous 1GB range. Additional memory could be added in the expansion space. However, there are some configurations that require a 2GB chip select region. For example if the configurations used in Figure 17 - Figure 19 were upgraded to use 1Gb memory technology (rather than 512 Mb) the size will double. The same effect will be seen if two additional physical banks are added and 4-way chip select interleave used in the single channel case or channel and chip select interleaving used in the dual channel case. Systems using the Big Memory mode (see Section: "Larger Memory Systems" on page 124) will also face this issue even when using lower density memory parts.

The 2GB problem can be solved with a little help from software. The hardware is configured in a way that creates an alias of the low 1GB of memory which software should ensure is never used (for example by the virtual-physical address translation in the TLB). Rather than creating a 2GB chip select range, the chip selects are programmed for a 6GB range and address bits [32:31] are not selected for use as a row, column or bank address. Thus the real 2GB range appears 3 times in the range given to the chip selects.

In Figure 20 it can be seen that 3GB of the aliases are hidden in the memory address space that cannot be accessed from the physical space. The only alias that needs to be considered is the 01\_0000\_0000 - 01\_3FFF\_FFFF alias of 00\_0000\_0000 - 00\_3FFF\_FFFF. A particular memory location must be accessed through only one of these ranges or the coherency algorithm will break down. The CPU exception vectors are all at the bottom of memory so in most systems the 00\_0000\_0000 - 00\_3FFF\_FFFF space will be used. Some systems may choose to use 00\_0000\_0000 - 00\_1FFF\_FFFF (accessible through kseg0) and 01\_2000\_0000 - 01\_3FFF\_FFFF to keep the upper range contiguous with the second 1GB -- however this only allows DMA access to the low 256MB from the PCI (or from 32-bit addressing peripherals bridged from the HT interface) since the DMA and CPU accesses must use the same physical range. (PCI accesses translated by the BAR0 map could be used.)

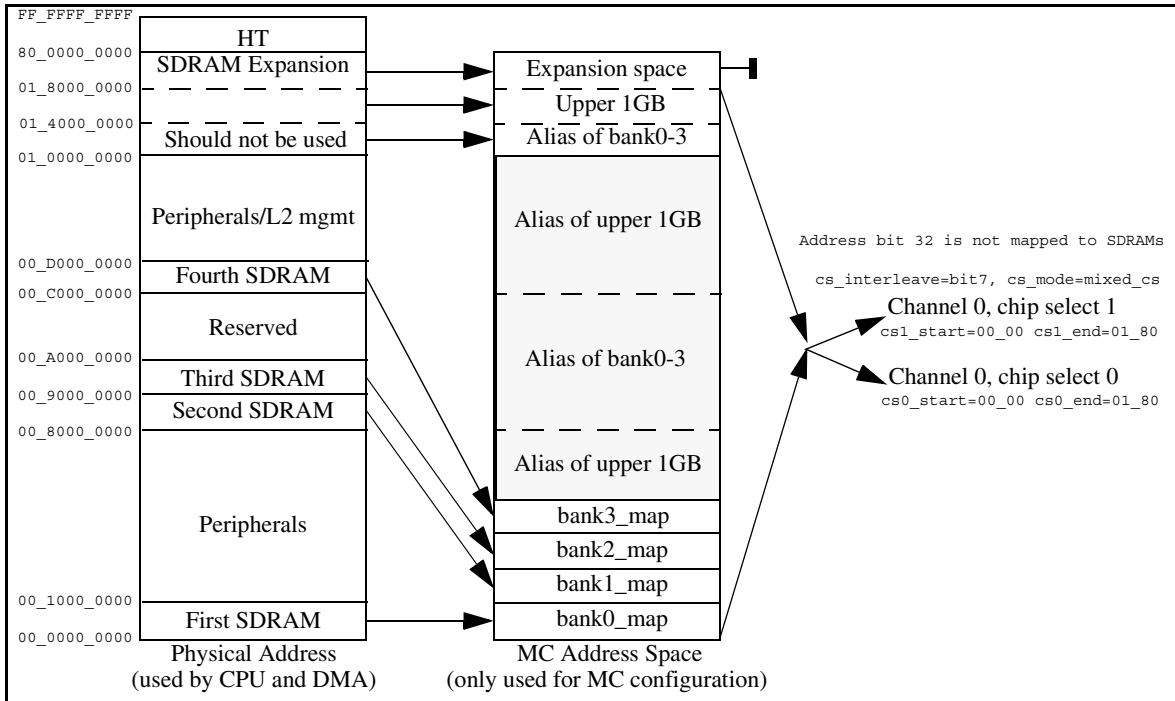


Figure 20: Example 2GB with two chip selects interleaved on one channel



## ROW, COLUMN AND BANK CONFIGURATION

The memory channel can be structured in many different ways with many different types of SDRAMs on different types of DIMMs and different DIMM counts per memory channel. In order to support such different configurations, the assignment of the address bits for row, column, and internal bank selection are fully software configurable along with the SDRAM timing parameters and operating modes.

Each memory channel is 8 bytes (64 bits) wide with ECC (8 bits), so the three lowest physical address bits play no part in the memory addressing. The memory controller always accesses memory with a burst of four 64-bit double-words, to access a full cache line. Therefore bits [4:3] of the physical address are always used as the lowest two column address bits. (The SDRAMs count the burst internally, so the memory controller only ever drives 00 on these two bits.)

**Table 61: Address Bits Used by a Memory Channel**

DRAM Address	Selected From	Comment
Row Address[13:0] Row Address[14:0] (FCRAM)	Contiguous block of bits from Address[34:10]	Regular SDRAMs have 13 row address bits (or 14 on BCM1125/H and BCM1250 PERIPH_REV3 or later). FCRAMS use WE_L and CAS_L as extra row bits.
Column Address[1:0]	Fixed to use Address[4:3]	Using 256 Mb technology parts only maximum of 10 column bits are used. For larger devices the JEDEC standard requires that the column address is presented on A[12:11,9:0] retaining the use of A10 as the auto pre-charge bit.
Column Address[12:11, 9:0]	One or two contiguous blocks of bits from Address[20:7] and Address[6:5]	
Bank Address[2:0]	Contiguous block of bits from Address[36:5]	Regular SDRAMs have only two bank address bits, so only two bits should be set. If three bits are set for bank address the top address bit becomes BA[2] and is not used for row or column. The top address bit is: A[12] for SDRAM when the ram_with_A13 bit is clear A[13] for SDRAM when the ram_with_A13 bit is set A[14] for FCRAM
Chip Selects (4)	Address range decode or CS Interleave selection	Only one chip select will be active (low) during an access.

Table 61 shows the maximum number of address bits that can be used for the row, column and bank parts of the SDRAM address. Each of the memory channels can theoretically address 4GB of memory (for a total of 8GB attached on a BCM1250). This comes from the number of RAS (13 or 14), CAS (12 or 11), bank (2), chip select (2) and bytes in dword(3) = 32 address bits per channel. (The gigabit SDRAMs are mostly configured with 14 row and 11 column address bits, to support these the extra A13 pin was added in PERIPH\_REV3 of the BCM1250 and on the BCM1125/H.) However, there are only four chip selects (hence the 2 bit equivalent shown above) per channel, and four bit wide devices are not supported. The maximum size is therefore achieved using four physical banks of 8 (or 9 with ECC) eight bit wide devices. Using 256Mb technology parts this limits the total size to 1GB per channel or 2GB total. Using 512Mb parts this doubles and will double again when the 1Gb DDR parts come out (reaching the theoretical maximum of 8GB).

The memory controller can use fast DDR parts (200 MHz clock). To work at this speed the signal trace lengths between the controller and the memory devices must be carefully controlled, maintaining tight tolerances and avoiding long stubs. Using a carefully chosen termination scheme DIMMs may be run at this speed.

Section: “Larger Memory Systems” on page 124 describes a mode of the memory controller that allows external generation of eight chip selects rather than four. This mode does not increase the theoretical maximum size of the memory, but it does double the maximum achievable size for the 256 Mb and 512 Mb parts. However, the address loading may require the use of registered DIMMs, the data loading will limit the maximum clock speed and there are limits on the supported memory configurations because the controller acts as if only two chip selects are in use.

The controller is programmed per chip select to determine the address bits used to generate the bank, row and column address bits sent to the SDRAM. The row and column address size depends on the devices used. The controller is programmed with bit masks that are applied to the memory address to extract the bank, row and column address. Bits are set in the mask to indicate the corresponding address bit should be used. The bank address is normally two bits corresponding to the BA[1:0] pins on the controller. If three bits are set in the bank address mask the controller will use the topmost address bit (A[12] for regular SDRAM, A[14]/CS[3] for FCRAM) as an extra bank bit and will work with 8 bank parts. With two exceptions described below, the set bits in the masks must be contiguous, so the right-most bit sets the lowest bit used and the number of set bits should match the number of address bits needed by the device.

Example masks are shown in Table 62. The bank is selected from the lowest bits, the column next and the row from the upper bits. Note that the column masks need not have bits [4:3] set, these address bits are always used as the low column bits.

**Table 62: Example for 128 MByte CS Region with 4K Rows, 1K Columns**

	Row Address Bits [26:15], Column Address Bits [14:7,4:3], Bank Address Bits [6:5]
<b>Row</b>	00000000_00000111_11111111_10000000_00000000
<b>Column</b>	00000000_00000000_00000000_01111111_10000000
<b>Bank</b>	00000000_00000000_00000000_00000000_01100000

One problem with this example is that the internal bank is switched every cache line, so streaming data that transfers in 64 byte blocks (for example HyperTransport reads or DMAs to the MACs) will not make good use of open pages in the SDRAM. The interleaving for banks (or chip selects or channels) is likely to work better with a larger block. The two exceptions to the rule that the masks must be contiguous allow for this.

The column mask may be split so that bit 5 or bits [6:5] may be set in addition to the contiguous bits. This allows for the interleave to be on 64 or 128 byte chunks. Changing the example to use 64 byte blocks gives the masks shown in Table 63, and using 128 byte blocks gives the masks in Table 64. This example only shows interleaving the internal banks of a single device, if there were chip select or channel interleaving there would be more than two zeros separating the set bits in the column mask.

**Table 63: Example for 128 MByte CS Region with 4K Rows, 1K Columns, 64 Byte Interleave**

	Row Address Bits [26:15], Column Address Bits [14:8,5,4:3], Bank Address Bits [7:6]
<b>Row</b>	00000000_00000111_11111111_10000000_00000000
<b>Column</b>	00000000_00000000_00000000_01111111_00100000
<b>Bank</b>	00000000_00000000_00000000_00000000_11000000



**Table 64: Example for 128 MByte CS Region with 4K Rows, 1K Columns, 128 Byte Interleave**

	Row Address Bits [26:15], Column Address Bits [14:9,6:5,4:3], Bank Address Bits [8:7]
<b>Row</b>	00000000_00000111_11111111_10000000_00000000
<b>Column</b>	00000000_00000000_00000000_01111110_01100000
<b>Bank</b>	00000000_00000000_00000000_00000001_10000000

The previous examples show interleaving between the banks within a device. It is also possible to use interleave between chip selects on a channel. This is done by setting an appropriate mode in the **mc\_config** register. If there are two chip selects available on a channel (i.e. there are two physical banks of memory of the same size) then the **mixed\_cs** mode can be used to allow them to be interleaved. When this mode is set the **mc\_cs\_interleave** register can have a single bit set to program which bit of the address is used to switch between chip selects. [Table 65](#) gives an example of this, the same 128 byte interleave is done between banks on the same device, but the upper row and column bits have been moved to allow for switching between the devices on the channel using bit [9].

**Table 65: Example for 256 MByte Region with 4K Rows, 1K Columns, two CS, and 128 Byte Interleave**

	Row Address Bits [27:16], Column Address Bits [15:10,6:5,4:3], Bank Address Bits [8:7], mixed_cs Selected by Bit [9]
<b>Row</b>	00000000_00001111_11111111_00000000_00000000
<b>Column</b>	00000000_00000000_00000000_11111100_01100000
<b>Bank</b>	00000000_00000000_00000000_00000001_10000000
<b>CS Interleave</b>	00000000_00000000_00000000_00000010_00000000

If there are four physical banks of memory then the **interleave\_cs** mode can be used to allow a full interleave between the physical banks. This is shown in [Table 66](#).

**Table 66: Example for 512 MByte Region with 4K Rows, 1K Columns, four CS, and 128 Byte Interleave**

	Row Address Bits [27:17], Column Address Bits [16:11,6:5,4:3], Bank Address Bits [8:7], Interleaved_cs Selected by Bit [10:9]
<b>Row</b>	00000000_00011111_11111110_00000000_00000000
<b>Column</b>	00000000_00000000_00000001_11111000_01100000
<b>Bank</b>	00000000_00000000_00000000_00000001_10000000
<b>CS Interleave</b>	00000000_00000000_00000000_00000110_00000000

Combining the bank, row and column masks with the chip select selection allows for more complicated mixing on a single channel. Table 67 shows a configuration using 128 MByte single chip select DIMM on chip select 0, and a 128 MByte dual chip select DIMM (also called a two bank DIMM using the term bank to refer to the physical banks) on chip select 1 and 2. Mixed-cs mode is used to allow the first DIMM to be standalone and the other to be interleaved. In this case the switch between the two physical banks is made based on address bit 15, so the full column addressed page is contiguous in the address space.

**Table 67: Example for 128 MByte + 64 MB + 64 MB Mixed\_CS Mode**

<b>CS0</b>	<b>Row Bits [26:15], Column Bits [14:9,6:5,4:3], Bank Bits [8:7]</b>
<b>Row</b>	00000000_00000111_11111111_10000000_00000000
<b>Column</b>	00000000_00000000_00000000_01111110_01100000
<b>Bank</b>	00000000_00000000_00000000_00000001_10000000
<b>CS1,2</b>	<b>Row Bits [26:16], CS Interleave Bit [15], Column Bits [14:9,6:5,4:3], Bank Bits [8:7]</b>
<b>CS Interleave</b>	00000000_00000000_00000000_10000000_00000000
<b>Row</b>	00000000_00000111_11111111_00000000_00000000
<b>Column</b>	00000000_00000000_00000000_01111110_01100000
<b>Bank</b>	00000000_00000000_00000000_00000001_10000000

### CHOOSING INTERLEAVE PARAMETERS

The best method of interleaving the memory is highly system dependant, so there is no single rule about how the memory controller should be configured. However, there are some general guidelines that give a starting point. The general goal is to maximize the number of banks (internal and physical) of the memory that are in use, which in turn leads to more overlap of transactions and less unused time on the channels. There are different costs involved in switching between banks: if the driver of the databus switches from one device to another (a change in physical banks) there must be an additional turnaround cycle which is not needed for a bank to bank switch in the same device (a change in internal banks). There are several measurable parameters: memory bandwidth and average access latency are normally the most important, but maximum access latency may also be important. Some things (e.g. a faster memory clock) will improve all of them, but many of the options improve one at the expense of others. In particular reductions in average memory latency are often at the expense of increasing the maximum latency.

In most cases the system will use more than one physical bank of memory. These can be put on different chip selects on the same channel or on the different channels. From a performance standpoint using both memory channels is almost always the correct choice. The peak memory bandwidth increases because the two channels have physically separate data buses and operate independently of one another. In some cases using both channels may not be possible on the BCM1250 (for example if the system has a single DIMM slot then a two physical-bank DIMM is necessarily on a single channel), but both channels should be used if possible (to continue the example: a second DIMM slot should go on the other channel in preference to using the remaining two chip selects of the first channel). On the BCM1125/H, two DIMM slots with two physical-bank DIMMs should be used if possible and full chip select interleaving should be enabled.



The simple configuration with two channels each with one physical bank of memory provides a good illustration of the trade-offs involved. The most straightforward way to assign the channels is to have one cover address 0 to N-1 and the other address N to 2N-1. However, most of the time there is a reasonable degree of locality (for example, it is likely that the program and its data will reside entirely in the first N locations of memory) so all activity is going to a single channel -- the effective memory bandwidth has been halved and the access latency will increase because of contention for the channel. In some systems this may be desirable, the second channel will have a lower and more deterministic access latency since it is so lightly loaded. But this gain has come at a very large cost to the system performance as a whole. At the other extreme the channels could be interleaved every cache line. This seems a good choice since the distribution of accesses is likely to be equal across even and odd cache lines, so both channels will be equally used. But since a contiguous access (e.g. a packet streaming in or out of the system) will flip back and forth between the two channels there is less likelihood that good use can be made of open pages in the memory. A good compromise is to interleave every four cache lines.

Note that the argument of the previous paragraph will also apply to packet buffers. Dedicating one channel to packet buffers and one to the program and associated data will quite often result in the packet buffer channel (which has CPU accesses as well as inbound and outbound DMA) being very heavily used and the other channel (which will only be used on accesses that miss in both the L1 and L2 caches) being under-used. In this case the network traffic is limited to half the memory bandwidth and will incur latency associated with using a busy channel. The system performance will be improved by interleaving the channels and thus removing the hot spot and allowing the bandwidth to be shared more evenly.

A good general starting point applies the principle from the previous examples: keep a few cache lines contiguous to allow for page mode accesses, then use low bits to interleave across the two channels, the internal banks within a device, and the physical banks (chip selects) on a channel. Using the low bits make it likely that even over short periods of time there is a reasonably even distribution of accesses across the regions. The address is therefore broken up as:

RRRR...R	CCCC...C	NN	BB	P	CC	cc000
7	6	5	4	3	2	1

This format can be used to set the mask bits.

- 1 The bottom 3 bits are ignored and should be set to zero. The next two bits (*cc*) are also ignored, but are always used as column bits, so they must be considered when the total number of column bits in the device is checked.
- 2 The next two bits are used for column interleave. For 32-byte blocks (and no column interleave), do not use any column bits here. For 64-byte blocks, use one column bit here, and for 128 byte blocks use two column bits here. These bits will be set in the **mc\_csN\_col** registers.
- 3 As discussed above, the next bit is a good one to use for interleaving between the two channels. This bit number is assigned in the **mc\_config** register.
- 4 The next bits are used to select the bank within a memory device. Most devices have four banks, so two bits will be set in the **mc\_csN\_ba** registers.
- 5 When interleaving across physical banks on a channel via chip-selects one or two bits must be set in the **mc\_cs\_interleave** register. If there are only two physical banks then *mixed\_cs* mode will be used and one bit set, if there are four physical banks then *interleaved\_cs* mode is used and both interleave bits are needed.
- 6 The remaining column bits are set in the **mc\_csN\_col** registers. These must be contiguous and the number of bits set will be the number of column address bits needed minus the number set in steps (1) and (2).
- 7 The remaining address bits form the row address in the **mc\_csN\_row** registers. The number of bits should match the number of row address bits needed by the memory device.

This will give a good starting point and should give acceptable results in most systems. Modifications can then be made as a result of performance monitoring.

## PAGE POLICY

The memory controller supports page mode access to the DDR SDRAMs, and provides several different policies for controlling page accesses. The FCRAMs do not support page mode (they always auto-precharge following CAS).

The Closed Page policy is the simplest. Pages are always closed by using CAS cycles with AutoPrecharge. If the memory uses FCRAMs the controller must be configured to use this policy.

The CAS time check policy will check the request queue just before the CAS cycle is issued to a channel. If there are requests in the queue that hit on the page then it is left open (and the same test will be performed when the new request is serviced) otherwise it is closed by issuing the AutoPrecharge command along with the CAS. If a sequence of accesses are done to the same page there is a good chance the page will be open when required.

The Hint Based policy is the same as the CAS time check, except that it accepts a hint with each memory request. The hint is part of the command on the ZBbus, and will be asserted by the DMA engines, PCI and HyperTransport interfaces when doing block moves. If the hint is set on a request then even if there are no more requests the page is left open following its CAS time check. If the hint is clear for the request then the behavior matches the CAS time check policy. Pages that are left open by hints will be closed by subsequent accesses that have the hint clear and fail the CAS time check, or will be closed using an explicit Precharge instruction if a page conflict occurs to the same bank.

The Open Page policy leaves the page open until a page conflict occurs to the same bank or an AutoRefresh is done. In the case of a page conflict an explicit Precharge command is performed to close the open page, delaying the RAS cycle to open the new page.

The page policy is set per chip select, allowing different areas of memory to have different policies. The optimal policy can therefore be selected based on the data access patterns. A good starting point is to use the CAS time check policy with hints enabled.

## SUPPORTED DRAMS AND DIMMS

### DDR SDRAMS

The memory controller supports standard JEDEC DDR SDRAMs in x8 and x16 configurations, and also the x32 DDR SGRAMs. In addition the FCRAMs are supported in both x8 and x16 configurations. The x32 SGRAM parts only use one DQS signal to cover all 32 data bits, the DQS from the DRAM should be connected to the corresponding four DQS pins on the controller (M\_DQS[7:4] for the upper word and M\_DQS[3:0] for the lower word); the controller will only drive on one of the lines but needs to receive on all of them. X4 configuration parts are not supported since the interface has insufficient DQS signals.

**Table 68: Supported SDRAMs**

DDR SDRAMs	Technology	Size (row x col)	Size (row x col)
JEDEC std (4 bank)	64 Mb	8Mx8 (4k x 512)	4Mx16 (4k x 256)
	128 Mb	16Mx8 (4k x 1k)	8Mx16 (4k x 512)
	256 Mb	32Mx8 (8k x 1k)	16Mx16 (8k x 512)
Non-std (8 bank)	64 Mb	2Mx32	
	128 Mb	4Mx32	
	256 Mb	8Mx32	
FCRAM (4 bank)	256 Mb	32Mx8(32k x 256)	16Mx16(32k x 128)

If the memory channel is configured for SGRAMs the A10 and A8 signals may need to be swapped. The memory controller follows the DDR standard and puts the AutoPrecharge flag on the A10 address bit, but many SGRAMs need it on A8. (In parts with system revision indicating PERIPH\_REV3 or greater the pre\_on\_A8 bit can be set in the **mc\_drammode** register to cause the controller to put the AutoPrecharge flag on A8 instead of A10.)

Non-standard 8 bank DDR SDRAMs use A[12] as the third bank address bit, 8 bank FCRAMS use A[14] as the third bank address bit. This is enabled if three bits are set in the bank address selection register. For regular SDRAMs if the ram\_with\_A13 bit is set in the **mc\_drammode** register the BA[2] is brought out on the A13 pin and A12 reverts to being a row address.

### DDR FCRAMs

Memory channels can be configured to use FCRAMs. These require two extra row address bits and use a function (FN) flag to distinguish between a read and write access. The JEDEC defined footprint for standard DDR and FCRAM parts are identical with the exception of pins 21, 22, and 23. On a DDR SDRAM these pins are WE#, CAS#, and RAS#, respectively. On the FCRAM, they become A14, A13, and FN (effectively WE#). The BCM1250 or BCM1125/H uses the same mapping when a channel is configured for FCRAM use, so:

- Mn\_WE\_L becomes Mn\_A[14] for FCRAM.
- Mn\_CAS\_L becomes Mn\_A[13] for FCRAM.
- Mn\_RAS\_L becomes Mn\_FN for FCRAM.
- Mn\_CKE is called Mn\_PD\_L but is the same.

This allows the standard footprint to be used and either type of memory populated. The FCRAM parts generally have a CAS to write data latency that is one less than the CAS latency, this is supported by setting the tCwD parameter in the SDRAM timing register.

The FCRAM includes a write mask or variable write (VW) function. The memory controller always does a full burst write and will set the bits accordingly (A[14:11] = 4'b1010 during LAL command).

### DIMMs

All standard DDR SDRAM DIMMs of 64+8bit data path are supported, except those that use x4 parts (which replace DM pins with extra DQS pins). Both registered and unbuffered DIMMs can be used (for registered DIMMs the timing parameters must be adjusted to take account of the additional cycle for the latch on the address and command lines). The controller provides four chip selects per channel, allowing two DIMM slots that use dual physical bank modules (i.e. need two chip selects) or four DIMMs with single physical bank modules (each using a single chip select). Software should read the DIMM configuration information and program the memory controller accordingly.

On a single channel it is not possible to mix registered and unregistered DIMMs.

## LARGER MEMORY SYSTEMS

The memory controller can support larger memory systems by externally decoding 8 chip selects per channel. This may limit the maximum speed of the memory and reduces the number of available configurations. The increased loading requires use of registered DIMMs in the large memory system. This mode is enabled by setting the `large_memory` bit in the DRAM type field of the `mc_drammode` register. Application Note 1250-AN600-R "BCM1250 Big Memory System" describes the design and implementation of a board using the large memory extension.

The extension works by configuring the memory controller to only use two chip selects, and driving two additional address bits on the other two chip select outputs. Externally each of the chip selects is used to enable a 2-to-4 decoder to generate the actual chip selects for the physical memory banks (the delay through the decoder must be taken in to account when timing analysis is done, but since the chip select signals will be registered on the DIMM this should not be a critical timing parameter). Internally the memory controller accounts for the switch of chips when the decoded chip select changes, but it only keeps track of the standard four open pages for the two internal chip selects (potentially degrading performance by not making use of open pages).

The external connections are:

- M\_CS\_[0] and M\_CS\_[1] are the two chip select lines. They should be connected to the active low enable input of the two halves of a dual 2-to-4 decoder.
- M\_CS\_[3:2] provide two additional address bits to qualify the chip selects. They should be connected to the address inputs of the 2-to-4 decoder
- The (active low) outputs of the 2-to-4 decoder provide 8 chip selects for DRAMs.

The internal configuration should be:

- The full address space should be covered using the `mc_cs0` and `mc_cs1` address selection registers. The `mc_cs2_start`, `mc_cs2_end`, `mc_cs3_start` and `mc_cs3_end` registers must all be set to zero.
- Two additional row address bits (corresponding to `row_addr[14:13]`) should be set in the `mc_cs0_row` and `mc_cs1_row` registers. These address bits will be output on the M\_CS\_[3:2] pins to qualify the chip select. Note that these bits must come from the memory address bits [34:28].
- The `large_memory` bit in the `mc_drammode` register should be set.



- Initialization of the DRAMs must be done before the large\_memory bit is set. It must be done for each of the real chip selects by stepping through the eight patterns of the chip select field in the **mc\_drammode** register. (Note that setting a bit in the chip select field will assert the active low external signal, so to cover all eight external chip selects {cs3,cs2} should count through the four values with {cs1,cs0} set to both 2'b01 and 2'b10.)
- If the large memory mode is used with parts that use fourteen row address bits (i.e. that need A13) then the ram\_with\_A13 bit must be set in the **mc\_drammode** register, this causes bits [15:14] from the row address mask to be passed to CS[3:2] rather than bits [14:13].
- The page policy should be set to always close page or cas-time-check. Memory accesses will have UNPREDICTABLE results if other page policies are used.
- The refresh rate should be set in the usual way based on the requirement of the SDRAM parts used. The controller will issue refresh to two (of the eight) chip selects at a time (rather than one of four).

## ECC

The memory controller supports an 8 bit ECC calculated over the 64 bit data. Single bit errors will be detected and automatically corrected as the data passes through the controller. Double bit errors are detected. In either case the data is flagged appropriately when it is sent on ZBbus, correctable errors are counted by the SCD; uncorrectable errors will be counted and report a data error at their destination.

Single bit ECC errors will be corrected during the read and the corrected data will be written back into the memory. They will continue to occupy the RQQ entry until the data has been written. Thus a memory location should not return multiple single bit ECC errors. Double bit errors are left as such, so repeated reads of the data will continue to give the error.

ECC checking can be disabled either to support non-ECC DIMMs or during memory testing.

The memory controller can be set to write bad ECC to the memory in order to check the error handling logic and code. There are two registers **mc\_test\_data** which is 64 bits and **mc\_test\_ecc** which is 8 bits. If either of these are non-zero then during memory writes any bits that are set in these two registers will cause the corresponding data or ECC bit to be inverted during a memory write. This will force an error into the memory. If the location is subsequently read an ECC error should be reported.

## SDRAM TIMING

The timing for DDR SDRAM accesses is set per channel, so all parts on a single channel have common timing. The memory pipeline control queue and scheduler is driven from the timing information programmed for the channel and will compute its schedule to match the rules. The base timing is set by the memory clock programmed in the **mc\_clock\_cfg** register, this is divided down from the ZBbus clock as outlined in [Section: "Clock Ratios and Clocking Scheme" on page 107](#).

The **mc\_timing1** and **mc\_timing2** registers are used to configure the number of memory clock cycles required between critical events. These can be obtained from the data sheet for the SDRAM that is being used (or the common JEDEC timing specification for standard parts). Most of the fields in these registers are four bits so can be set to a value between 0 and 15, however the acceptable range for the parameters is smaller than this in many cases. Programming the fields outside their acceptable ranges has UNDEFINED results.

## SDRAM REFRESH

The memory controller uses the SDRAM auto-refresh command to refresh the memory during normal operation. The refresh interval for a channel is set (in memory clocks) in the **mc\_clock\_cfg** register. In parts where the system\_revision indicates PERIPH\_REV3 or later this register also has fields to completely disable refresh (useful only during debugging, to avoid the scope being triggered by refresh cycles) or disable refresh for particular (i.e. unused) chip selects.

Software can cause Auto Refresh commands to be sent to the SDRAM as described in [Section: “SDRAM Initialization and Commands” on page 126](#). This has no effect on the refresh interval counter.

## SDRAM INITIALIZATION AND COMMANDS

The DDR SDRAMs accept a number of commands that are not used as part of regular operation. These can be issued under software control by writing the **mc\_dramcmd** register with the command to be performed and the chip selects that should be asserted when the command is executed. The memory controller will insert a synchronization point before the command is executed so that all buffers are empty. In some cases the commands require additional data encoded in the address lines. The **mc\_drammode** register holds the data to be encoded (on address lines A[11:0] for standard DDR parts and A[14:0] for FCRAMs).

**Table 69: Commands that can be Issued Through mc\_dramcmd Register**

Name	Command	Description
EMRS	Write Extended Mode Register	Write the value in the mc_drammode register to the Extended Mode Register in the selected parts.
MRS	Write Mode Register	Write the value in the mc_drammode register to the Mode Register in the selected parts.
PRE	Precharge all banks	Issue a precharge command to the SDRAMs with address bit 10 high to cause all banks to be precharged. If the pre_on_8 bit is set then this command will run the precharge command with address bit 8 high.
AR	Auto Refresh	Issue an Auto Refresh command to the SDRAMs.
SREF_set	Enter Self Refresh mode	Set the SDRAM into Self Refresh mode by issuing a Self Refresh command and deasserting CKE then stopping the clock.
SREF_clr	Exit Self Refresh mode	Remove the SDRAM from Self Refresh mode by starting the clock, asserting CKE and issuing NOP commands to the part. Software must ensure that the memory is not accessed for the required time after the part is removed from Self Refresh mode (normally 200 memory clocks).
PDN_set	Enter Power Down mode	Set the SDRAM into Power Down mode by issuing a NOP command and deasserting CKE.
PDN_clr	Exit Power Down mode	Remove the SDRAM from Power Down mode by asserting CKE and issuing NOP commands to the part. Software must ensure that the memory is not accessed for the required time after the part is removed from Power Down.



When the part is reset the SDRAM channels are put in power-down mode with the clock stopped and CKE deasserted. Because CKE is deasserted all devices on the memory channel will place their outputs in a high impedance state, this will prevent multiple bus drivers as the channel starts up. To initialize a channel software must first start the clock by programming the **mc\_clock\_cfg**, then wait for 200us for the clock to stabilize before giving the PDN\_clr command to assert CKE. Next the DDR SDRAMs must be initialized (this also needs to be done when exiting power down mode if the DLL has been disabled). The initialization sequence is specified in the SDRAM data sheet, however the JEDEC recommended sequence should be a superset of all manufacturer requirements. It is:

- 1 Precharge all banks (PRE).
- 2 Enable the DLL and set the appropriate output drive strength in the extended mode register (EMRS).
- 3 Reset the DLL in the mode register (MRS).  
From this point no access other than the subsequent initialization steps (4 through 7) may be done for 200 memory cycles.
- 4 Wait for 2 cycles (tMRD).
- 5 Precharge all banks (PRE).
- 6 Issue two auto refresh cycles (AR, AR). (The JEDEC spec allows these to be done before the Precharge in step 5)
- 7 Configure the mode register for Normal Operation, appropriate CAS latency, Sequential burst and burst length of 4. (MRS)
- 8 Once 200 memory cycles have passed since the DLL was reset, the SDRAM is ready for normal operation.

The reset initialization sequence is different for FCRAM from regular DDR parts. The memory controller will not release the enable (CKE) when the PDN\_clr command is used to start the clock, but waits for a PRE command to be issued before it asserts CKE and automatically runs the address sequencing initialization sequence for all chip selects. The sequence of commands needed is therefor:

- 1 Set DRAM type to FCRAM. This MUST be done first. This will start the memory clock on Mn\_CLK.
- 2 Wait for minimum of 200us. This is required by the FCRAM which must have a stable clock for a minimum time before being enabled.
- 3 Issue PDN\_clr command. This asserts CKE and sets the DESL command with CS\_L=1 BA[1:0]=00 A[8]=0 A[7]=1. This covers steps 1-6 of the standard FCRAM initialization sequence.
- 4 Issue PRE command. This will run the sequence starting from CS\_L=1 BA[1:0]=00 A[8]=0 A[7]=1; CS\_L=0 BA[1:0]=11, A[14:8]=1111111 A[7]=0 held for two clocks; CS\_L=1 rest same for 4 clocks; CS\_L=1 and change BA[1:0] and A[14:7], hold for 4 clocks. This covers steps 6-9 of the standard FCRAM initialization sequence.
- 5 Set the Mode Register and issue EMRS command.
- 6 Set the Mode Register and issue MRS command.
- 7 Issue two auto-refresh commands (AR, AR).
- 8 Wait for 200 or 300 memory cycles (see FCRAM data sheet).
- 9 Do a dummy write to each bank of each chip select.
- 10 The device is now ready for normal operation.

## I/O CONTROL

The memory controller supports a wide range of SDRAM speeds and configurations. In particular the signal loading and propagation delays will be very different in a system that is fully populated with DIMMs than it is in a system with point-to-point connections between the part and SDRAMs soldered to the main board. The termination schemes may differ too. The timing budget and electrical characteristics must be calculated for each board the BCM1250 or BCM1125/H is used on. In most cases the correct values for the I/O control registers will be calculated during board design and verified during bring up.

The electrical characteristics of the memory controller outputs can be adjusted to match the environment it is in. The drive strength (and thus slew rate for a particular loading) of the clock, address/command and data output drivers can be set in the channel configuration register (**mc\_clock\_cfg**). There is one bit of coarse control which changes the driver configuration between SSTL\_2 Class 1 (7.6mA min. drive) and SSTL\_2 Class 2 (15.2mA min. drive). In addition there are three slew bits that control the slew rate of the driver from fast (3'b111) to slow (3'b000). In most cases the correct setting will be the fast slew rate, but the slower rates may work better for systems with lower loading and when non-standard termination is used.

**IMPORTANT! The rest of this description explains the DLLs for the BCM1250 and BCM1125/H where the system\_revision indicates PERIPH\_REV2 or later. The address DLL was organized differently on the early-access prototype pass-1 parts (marked BCM12500), please see an earlier revision of this User Manual for description of those parts.**

An internal master DLL is used to control slave DLLs to allow adjustment of input and output timing. Each DLL has a fixed **Offset**, a linear delay adjustment (6 bit unsigned number **M**) and a proportional delay adjustment (4 bit unsigned number **M**). The total DLL delay is given by:

$$Delay = Offset + (N * dll\_step) * (1 + ((M-8) * 5\%))$$

This is the ideal case, in practice there is a part dependent error to the 5%, and it rolls off to more like 4% at the low end. Table 70 gives the typical multiplier **mult** that results from the  $(1 + ((M-8) * 5\%))$  term. The master DLL is fixed to have **M=8** and derives **Nmaster** from the memory clock period **Tmclk** such that:

$$Delay = Offset + (Nmaster * dll\_step) = Tmclk/4$$

The **Offset** is typically 500ps, and the resolution of **dll\_step** is sufficient to cover memory clock frequencies from 95-200MHz. At lower frequencies the delay may saturate, however at these speeds the cycle time is so long that the proportional adjustment provided by the DLL is not required.

**Table 70: Adjustment Percentages and Multiplier for Values of DLL M**

M	M-8	Adjustment% Simulation Result for (M-8)*5%	Multiplier Simulation Result for 1+((M-8)*5%)
4'b0000	-8	-35%	0.65
4'b0001	-7	-31%	0.69
4'b0010	-6	-27%	0.73
4'b0011	-5	-23%	0.77
4'b0100	-4	-19%	0.81
4'b0101	-3	-15%	0.85
4'b0110	-2	-10%	0.90
4'b0111	-1	-5%	0.95



**Table 70: Adjustment Percentages and Multiplier for Values of DLL M (Cont.)**

<b>M</b>	<b>M-8</b>	<b>Adjustment% Simulation Result for (M-8)*5%</b>	<b>Multiplier Simulation Result for 1+((M-8)*5%)</b>
4'b1000	0	center	1.0
4'b1001	1	5%	1.05
4'b1010	2	10%	1.10
4'b1011	3	15%	1.15
4'b1100	4	20%	1.20
4'b1101	5	25%	1.25
4'b1110	6	30%	1.30
4'b1111	7	35%	1.35

The slave DLLs are adjusted to have their **N** set to **Nmaster** and **M** set to the four bit dqj\_skew, dqo\_skew or addr\_skew field from the **mc\_clock\_cfg** register. This value **M** can be converted to a multiplier **mult** according to [Table 70](#). The slave DLL delay is given by:

$$\text{SlaveDelay} = \text{Offset} + (\text{Nmaster} * \text{dll\_step}) * \text{mult}$$

Substituting for **Nmaster**:

$$\text{SlaveDelay} = \text{Offset} + (((\text{Tmclk}/4 - \text{Offset})/\text{dll\_step}) * \text{dll\_step}) * \text{mult}$$

$$\text{SlaveDelay} = \text{Offset} * (1 - \text{mult}) + (\text{Tmclk}/4) * \text{mult}$$

The DLLs are used in three places: to shift the received DQS into the center of the data valid window, to shift the relationship between the output clock and signals, and to shift the DQS driven into the center of the data window.

The first is the most straightforward. During a Read the DRAM will drive the data at the same time as it changes DQS. Since the board traces between the DRAM and the controller are length matched (across a group of 8 data bits and one DQS for regular DDR parts, and a group of 32 data bits and DQS for SGRAM) the data and DQS transitions are still aligned when they are received at the memory controller. The controller delays the DQS using a DLL controlled by dqj\_skew. The data is valid for half of a memory cycle, so the quarter cycle base delay (with **M** = 4'b1000 thus **mult** = 1) will position the delayed DQS at the center of theoretical data window. The DLL can be adjusted about 8.75% of the memory cycle (or 35% of the quarter cycle) either side of this point.

The second use of the DLL is to position the address and data signals relative to the memory clock. This is done by driving the data/address/control from a fixed internal clock and using the DLL to move the external Mn\_CLK with respect to the internal one. Since it is the clock moving rather than the data the direction of a change can seem backwards! [Figure 21](#) shows the external view with timings relative to the clock, this is the view needed when designing the memory system.

The position of the clock relative to the point at which the address and control signals change is set by the addr\_skew parameter. [Figure 21](#) shows the range as t0 to t1. To get the address change earliest in the external clock cycle (shown by t0) the clock needs to be delayed by its maximum, so the addr\_skew (which sets the DLL **M** parameter from [Table 70](#)) needs to be 4'b1111. The other extreme is shown by t1, which has the addr\_skew set to 4'b0000. The range gives the delay from the rising Mn\_CLK edge to the address transition point of approximately 16% to 34% of the memory clock cycle time.

The drive position of the data relative to clock ( $t_4$  and  $t_5$ ) is controlled by the `addr_skew` parameter in the same way as the address, except the data changes on both edges of the clock. The data transition point is thus offset from both edges of the `Mn_CLK` by about 16% to 34% of the memory clock cycle time.

The third use of a DLL is to position the DQS strobe relative to the data during a write to the memory. The DQS transition needs to be shifted to the center of the data valid window. This is done by shifting the DQS transition from the data transition using a DLL controlled by the `dqo_skew`. The minimum delay between the data and DQS ( $t_6$  in Figure 21) is about 16% of the cycle time when `dqo_skew` is 4'b0000, the maximum (shown as  $t_7$ ) is about 34% of the cycle time with `dqo_skew` set to 4'b1111.

The second and third uses of the DLL interact to set the position of the DQS output relative to the clock. The difference between the `addr_skew` and `dqo_skew` controls this delay (if they are set to the same value the strobe will line up with the clock). The early DQS shown by  $t_2$  results from the `addr_skew` being set to `M=4'b1111` and `dqo_skew` to `M=4'b0000`, the difference in the delays will make the DQS precede the clock by  $(35 - (-35)) = 70\%$  of a quarter cycle or 17.5% of the cycle time. The late DQS shown by  $t_3$  is the reverse situation and will have the DQS 17.5% of the cycle time after the clock.

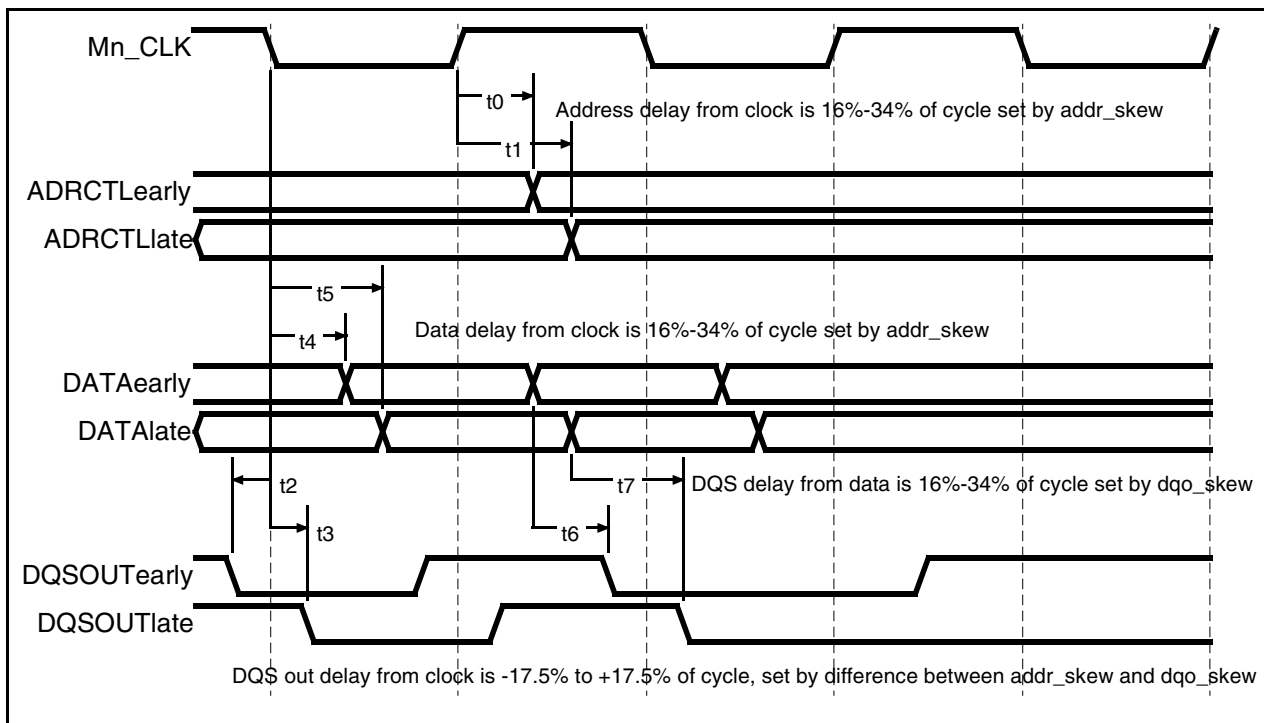


Figure 21: Timing Relationships Set by DLLs



## TIMING PARAMETER GUIDELINES

The memory controller timing parameters must be configured to match the devices and board delays in the system. This section gives some guidelines for the parameters.

During a read from memory the controller will use the DQS strobe supplied with each byte of the data to latch the four data-beats. The data is subsequently read out and passed to the DBF data buffer. Since there will be skew across the nine DQS lines, the different bytes are captured from the memory at different times, but they are eventually read out of the latch as a single value. There are three factors that influence the timing. (1) The controller must not start looking for DQS until the SDRAM drives it to a valid low as part of the preamble. (2) The controller must start looking for the DQS before the earliest DQS returns. (3) The data must not be read from the latch before the byte with the latest DQS has been written and allowed to settle. Taken together these constraints fix the window in which the first DQS of each byte lane must be received at the controller. The  $t_{CrD}$ ,  $t_{CrDh}$  and  $t_{FIFO}$  parameters in the `mc_timing1` register describe this window.

As a starting point the  $t_{CrD}$  is the integer part of the CAS latency, the  $t_{CrDh}$  is set for half-cycle CAS latency and the  $t_{FIFO}$  is set to 1 for normal delay in reading out from the data latch. The full story is shown in [Figure 22](#) which shows the window for the first DQS relative to clock for different settings of the three parameters. The count  $n$  represents the number of cycles since CAS, so for a CAS latency  $n$  device the ideal DQS would be at the rising edge that starts cycle  $n$ .

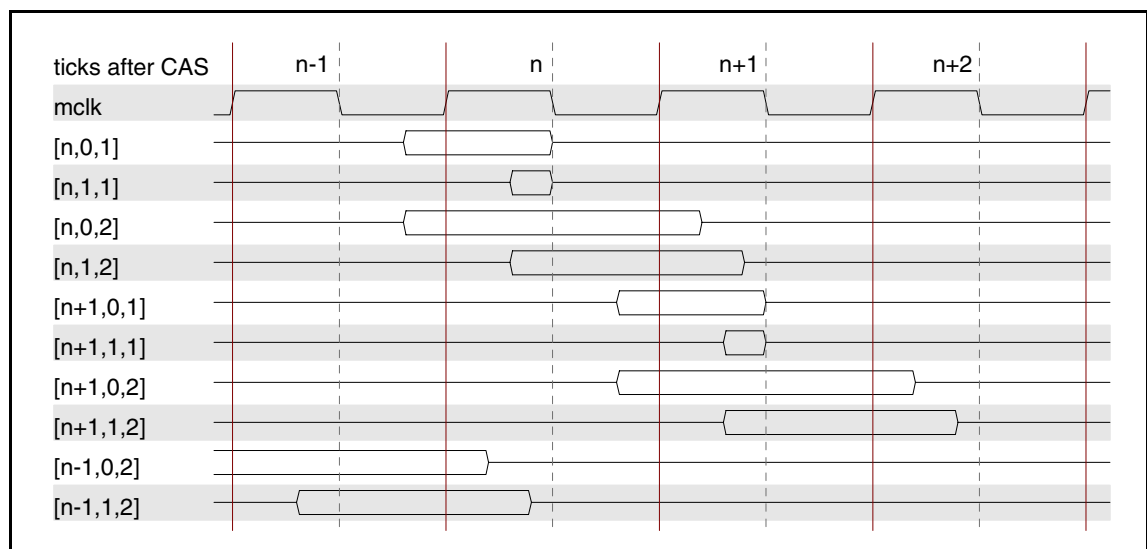


Figure 22: Nominal Windows at 133MHz for First Edge of DQS for Various Settings of [ $t_{CrD}$ ,  $t_{CrDh}$ ,  $t_{FIFO}$ ]

This figure shows where in time the first DQS edges must occur for various settings of the parameters [tCrD, tCrDh, tFIFO]. The following table describes the opening and closing of the windows for different parameter settings:

**Table 71: First DQS Window Opening and Closing (Typical)**

<b>Setting</b>	<b>Window opening</b>	<b>Window closing</b>
[n, 0, 1]	1.5ns before rising edge	3.5ns before rising edge of cycle n+1
[n, 1, 1]	1.5ns before falling edge	3.5ns before rising edge of cycle n+1
[n, 0, 2]	1.5ns before rising edge	2.0ns before falling edge of cycle n+1
[n, 1, 2]	1.5ns before falling edge	2.5ns before falling edge of cycle n+1 plus ¼ cycle

Note that both the MCLK and the DQS are observed at the pins of the controller. When a memory system is analyzed the loop delay must be considered: the clock is driven from the controller and takes time to propagate to the SDRAM, the SDRAM will drive DQS in some window around when it receives the clock edge and DQS will take time to propagate back. In a heavily loaded channel (or with lower drive strength) there may also be a contribution to the delay from the rise/fall time of the clock and DQS. The DQS signals are all being driven by different SDRAM chips and may have different propagation delays (since the board trace length difference between the 8 data bits and their DQS is more tightly controlled than the difference across the DQS bundles). In systems with DIMMs there may be a non-negligible difference in response between a device on the first DIMM and a device on the last DIMM of the channel, increasing the uncertainty in DQS arrival time.

The positions of the windows are affected by the settings of the DLLs described in the previous section. They are shown for the center DLL position ( $M=4'b1000$ ). Increasing the value of `addr_skew` will move the windows to the left (this is not true on the initial pass1 prototypes with the earlier address DLL arrangement, in those parts increasing `dqo_skew` will move the window to the left). Increasing the value of `dqi_skew` will move the windows to the left (this is true on all parts).

The positions of the windows are also affected by the settings of the `clock_class` and `clock_drive` parameters in the memory clock configuration register. The windows in the figure are shown for `clock_class=1` and `clock_drive=7` (i.e. maximum drive strength SSTL\_2 Class II). Values other than these will move the windows to the left.

If the memory system is running slower than 95MHz, it is possible that the DLLs in the memory controller will saturate (i.e., reach their maximum possible delay). When this happens, the windows will move to the right.

A few of the settings are not useful in practice. There is no reason to use the setting [n,1,1] since it is a subset of the [n,0,1], [n, 0, 2] and [n, 1, 2] settings and has a narrower window that all nine DQS signals must hit. The setting for [n-1,0,2] is also not particularly useful. It is only shown for completeness.

The setting for [n-1,1,2] is probably only useful to systems running with such a slow memory clock that the DLL's have saturated. This setting is misleading the controller about the actual CAS latency of the SDRAM. The controller will therefore expect the SDRAM to release the bus earlier, which must be compensated for by always setting `r2wldle` to 1 to restore the cycle in a read to write turnaround.





As an example, consider a memory system with parts soldered to the mainboard using CAS latency 2.5 parts. The simple view would suggest [2, 1, 1] as the starting point. However referring to the figure, we note that with this setting at 133MHz, the window has closed before the earliest point where we might expect DQS to return. As a general rule, systems using CAS latency 2.5 should use either [2,0,2] or [2,1,2]. Which of these settings is better depends on the memory system round trip delay, the DLL settings, and the setting for memory clock drive strength.

Continuing the example, suppose we run the memory at CAS latency 2. In this case the expected setting, [2,0,1], is generally the correct setting. But now suppose that for some reason the memory clock needs to be run slow (below 95MHz). All of the windows in the figure will move right. And in the lightly loaded system the DQS arrival will be relatively early. The width of the [2,0,1] window will continue to work for some time because it can move for a reasonable distance until the opening point reaches the early DQS (which the DDR spec would allow to be before the clock edge). At some frequency the first DQS will not be correctly used to strobe the data and the data received later in the burst will overwrite the earlier (the memory controller reads into the low numbered bits of the ZBbus first, so when this happens the data in the low memory address end of a cache line will be UNPREDICTABLE and the data at the high address will have the value expected from the low address). This is where the memory controller needs to be programmed with a smaller CAS latency (and r2wldle set to 1) to use the right shifted [1,1,2] window.

Most systems will likely want to use one of the following settings:

- [n,0,1] for whole cycle CAS latencies and relatively short board delays.
- [n,0,2] for whole cycle CAS latencies with moderately long board delays.
- [n,1,2] for half-cycle CAS latencies. (Due to a performance bug systems with half-cycle CAS latencies using BCM1250 pass 1 prototype parts should use [n,0,2] if possible).

There are three additional parameters r2wldle, w2rldle, and r2rldle that address system timing concerns, specifically bus conflicts caused by the time delay between signals driven by the controller and signals driven by the SDRAMs.

The r2wldle parameter sets the turnaround time from the memory driving the databus (on a read) to the controller driving the databus (for a subsequent write). It is independent of whether the accesses are to the same physical bank or different physical banks. Assuming a whole cycle CAS latency for the SDRAM, a setting of 0 for the r2wldle parameter nominally provides 0.75 cycles of non-overlap time for the DQ lines. But at the controller, this non-overlap time is reduced by the clock delay from the controller to the SDRAM and the DQ delay from the SDRAM to the controller. Further, if the SDRAM CAS latency is set to some half-cycle value, the non-overlap time is reduced by another 0.5 cycle. So systems using SDRAMs with a half-cycle CAS latency value almost certainly need to set the r2wldle parameter to a 1, increasing the non-overlap time by one cycle. Systems using whole-cycle CAS latency values may need to set this parameter to a 1 if the round-trip delays between the controller and the SDRAMs are significant compared to the clock period.

The w2rldle parameter sets the turnaround time for the controller driving the databus to the memory driving the databus when physical banks are changed. It should always be set to 1.

The r2rldle parameter sets the turnaround time between a read from one physical bank to a read from a different physical bank.

A value of 0 for the r2rldle parameter nominally provides 1 cycle of non-overlap for the DQ lines at the physical bank of SDRAMs being read first. At the physical bank that is the target of the second read, this nominal value is reduced by the DQ delay between the two physical banks. The DQS non-overlap time is nominally 0, but since both physical banks are driving the DQS lines low during the turnaround, there is nominally 0.5 cycle of margin before a conflict occurs on the DQS lines at the first physical bank. Once again, at the second physical bank of SDRAMs, this 0.5 cycle of nominal no conflict time is reduced by the DQS delay between the two physical banks. Therefore it is unlikely that a system will require a non-zero value of the r2rldle parameter.

To summarize, most systems using half-cycle CAS latency will set r2wldle to a 1, w2rldle to a 1, and r2rldle to a 0. Systems using SDRAMs with a whole-cycle CAS latency will set r2rldle to 1 and the other parameters to 0. Systems configuring an n-1 value for tCrD will need to set r2wldle to 1.

## PERFORMANCE MONITORING FEATURES

The memory controller provides signals to the performance counters in the SCD (see [Section: “System Performance Counters” on page 61](#)).

[Add discussion.](#)

## ZBBUS MONITORING

The functionality described in this section in previous releases of the BCM1250 manual was for the prototype chips only and is no longer available. The trace buffer in the SCD (See [“Trace Unit” on page 70](#)) should be used to monitor the ZBbus.

## CONFIGURATION REGISTERS

All memory controller configuration registers are 64 bit wide and they must be written using double-word accesses. If writes smaller than 64 bits are done to a control register its contents become UNPREDICTABLE.

**On the BCM1125/H, all registers of memory channel 0, except mc\_config\_0 are Not Implemented.**

**Table 72: Memory Channel Configuration Register on BCM1250**

mc_config_0 - 00_1005_1100 mc_config_1 - 00_1005_2100			
Bits	Name	Default	Description
7:0	reserved	8'b0	Reserved
15:8	channel_sel	8'h0	If this field is zero the two channels are not interleaved. If this field is non zero it specifies the address bit that selects between the two channels, and must have a value in the range 5-35. If the channels are interleaved the other configuration parameters must match.
19:16	bank0_map	4'h0	Value of physical address bits [31:28] that should map to 0 (1st 256MB block)
23:20	bank1_map	4'h8	Value of physical address bits [31:28] that should map to 1 (2nd 256MB block)
27:24	bank2_map	4'h9	Value of physical address bits [31:28] that should map to 2 (3rd 256MB block)
31:28	bank3_map	4'hC	Value of physical address bits [31:28] that should map to 3 (4th 256MB block)
39:32	probe_mode	8'b0	Reserved, Broadcom Use Only. Setting these bits to any value other than zero will result in UNDEFINED behavior and can cause the ECC lines to be continually driven regardless of the direction of the data transfer.
43:40	iob1_qsize	4'ha	These fields are used to set the range of queue entries reserved for use by I/O Bridge 1. The two channel configuration registers set a range to give hysteresis to the blocking. <b>mc_config_0:</b> Queue size to start blocking agents other than IOB1. <b>mc_config_1:</b> Queue size to stop blocking agents other than IOB1.
47:44	age_limit	4'h8	Maximum number of younger reads that can pass a read in the request queue, before the read is serviced. See <a href="#">Section: "Memory Controller Architecture" on page 104.</a>
51:48	wr_limit	4'h5	Maximum number of writes in a burst to memory before reads will be serviced. See <a href="#">Section: "Memory Controller Architecture" on page 104.</a>
52	iob1_priority	1'b1	<b>mc_config_1:</b> If this bit is set reads from I/O bridge 1 will be given high priority (to reduce their latency). If clear all requests have the same priority. <b>mc_config_0:</b> Reserved.
55:53	reserved	3'b0	Reserved
59:56	cs_mode	4'h0	Chip Selection mode 0000: msb-CS mode: CS[3:0] are determined by the corresponding CS start/end+1 address registers. 1111: Interleaved-CS mode: the values of start/end+1 addresses of CS[3:0] are all same in this mode, CS[3:0] are decoded by the two bits in the interleaved CS position register. 1100: Mixed-CS mode: CS[1] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS3 and CS2, determined by one bit of the interleaved CS position register. The values of the CS[3:2] start/end+1 addresses are the same. 0110: Mixed-CS mode: CS[3] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS2 and CS1, determined by one bit of the interleaved CS position register. The values of the CS[2:1] start/end+1 addresses are the same. 0011: Mixed-CS mode: CS[3] and CS[2] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS1 and CS0, determined by one bit of the interleaved CS position register. The values of the CS[1:0] start/end+1 addresses are the same.

**Table 72: Memory Channel Configuration Register on BCM1250 (Cont.)**

mc_config_0 - 00_1005_1100 mc_config_1 - 00_1005_2100			
Bits	Name	Default	Description
60	ecc_disable	1'b0	ECC disable. If this bit is set ECC checking will not be performed and data will never be reported to have an error.
61	berr_disable	1'b0	Bus error disable. This bit sets the behavior when a read is done to an address that does not match any chip select range. 0: Generate a bus error data return 1: Generate a valid data return containing UNPREDICTABLE data. On the BCM1250 if this bit is set in either channel, bus errors will be disabled for all reads.
62	force_seq	1'b0	Force Sequential. If this bit is set requests will be issued to memory in the same order their A-phase happens on the ZBbus.
63	debug	1'b0	This bit enables debug functions in the controller. It should be clear during normal operation. The action if the bit is set is different for the two channels: Channel 0: Reserved Channel 1: Broadcom Use Only ( was Enable ZBbus monitoring mode).

**Table 73: Memory Channel Configuration Register on BCM1125/H**

mc_config_0 - 00_1005_1100 mc_config_1 - 00_1005_2100			
Bits	Name	Default	Description
15:0	reserved	16'b0	Reserved
19:16	bank0_map	4'h0	<b>mc_config_1</b> : Value of physical address bits [31:28] that should map to 0 (1st 256MB block) <b>mc_config_0</b> : Reserved.
23:20	bank1_map	4'h8	<b>mc_config_1</b> : Value of physical address bits [31:28] that should map to 1 (2nd 256MB block) <b>mc_config_0</b> : Reserved.
27:24	bank2_map	4'h9	<b>mc_config_1</b> : Value of physical address bits [31:28] that should map to 2 (3rd 256MB block) <b>mc_config_0</b> : Reserved.
31:28	bank3_map	4'hC	<b>mc_config_1</b> : Value of physical address bits [31:28] that should map to 3 (4th 256MB block) <b>mc_config_0</b> : Reserved.
39:32	probe_mode	8'b0	<b>mc_config_1</b> : Reserved, Broadcom Use Only. Setting these bits to any value other than zero will result in UNDEFINED behavior and can cause the ECC lines to be continually driven regardless of the direction of the data transfer. <b>mc_config_0</b> : Reserved.
43:40	ioB1_qsize	4'ha	These fields are used to set the range of queue entries reserved for use by I/O Bridge 1. The two channel configuration registers set a range to give hysteresis to the blocking. <b>mc_config_0</b> : Queue size to start blocking agents other than IOB1. <b>mc_config_1</b> : Queue size to stop blocking agents other than IOB1.
47:44	age_limit	4'h8	<b>mc_config_1</b> : Maximum number of younger reads that can pass a read in the request queue, before the read is serviced. See <a href="#">Section: "Memory Controller Architecture" on page 104.</a> <b>mc_config_0</b> : Reserved.
51:48	wr_limit	4'h5	<b>mc_config_1</b> : Maximum number of writes in a burst to memory before reads will be serviced. See <a href="#">Section: "Memory Controller Architecture" on page 104.</a> <b>mc_config_0</b> : Reserved.



Table 73: Memory Channel Configuration Register on BCM1125/H (Cont.)

mc_config_0 - 00_1005_1100 mc_config_1 - 00_1005_2100			
Bits	Name	Default	Description
52	iob1_priority	1'b1	<b>mc_config_1</b> : If this bit is set reads from I/O bridge 1 will be given high priority (to reduce their latency). If clear all requests have the same priority. <b>mc_config_0</b> : Reserved.
55:53	reserved	3'b0	Reserved
59:56	cs_mode	4'h0	<b>mc_config_1</b> : Chip Selection mode 0000: msb-CS mode: CS[3:0] are determined by the corresponding CS start/end+1 address registers. 1111: Interleaved-CS mode: the values of start/end+1 addresses of CS[3:0] are all same in this mode, CS[3:0] are decoded by the two bits in the interleaved CS position register. 1100: Mixed-CS mode: CS[1] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS3 and CS2, determined by one bit of the interleaved CS position register. The values of the CS[3:2] start/end+1 addresses are the same. 0110: Mixed-CS mode: CS[3] and CS[0] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS2 and CS1, determined by one bit of the interleaved CS position register. The values of the CS[2:1] start/end+1 addresses are the same. 0011: Mixed-CS mode: CS[3] and CS[2] are not interleaved and determined by the corresponding start/end+1 addresses. The interleaving occurs between CS1 and CS0, determined by one bit of the interleaved CS position register. The values of the CS[1:0] start/end+1 addresses are the same. <b>mc_config_0</b> : Reserved.
60	ecc_disable	1'b0	<b>mc_config_1</b> : ECC disable. If this bit is set ECC checking will not be performed and data will never be reported to have an error. <b>mc_config_0</b> : Reserved.
61	berr_disable	1'b0	<b>mc_config_1</b> : Bus error disable. This bit sets the behavior when a read is done to an address that does not match any chip select range. 0: Generate a bus error data return 1: Generate a valid data return containing UNPREDICTABLE data. <b>mc_config_0</b> : Reserved.
62	force_seq	1'b0	<b>mc_config_1</b> : Force Sequential. If this bit is set requests will be issued to memory in the same order their A-phase happens on the ZBbus. <b>mc_config_0</b> : Reserved.
63	reserved	1'b0	Reserved

**Table 74: Memory Clock Configuration Register**

mc_clock_cfg_0 - 00_1005_1500 mc_clock_cfg_1 - 00_1005_2500			
Bits	Name	Default	Description
3:0	clk_ratio	4'h4	ZBclk/mclk ratio Mclk when ZBclk=400 MHz 0100 2x 200 0101 2.5x 160 0110 3x 133 0111 3.5x 114 1000 4x 100 1001 4.5x 88 Others Reserved
7:4	cs_absence	4'h0	On parts with system_revision ≥ PERIPH_REV3 setting these bits will disable the refresh cycle for the corresponding chip select. On earlier revisions all chip selects will always be refreshed.
15:8	ref_rate	8'hC4	Refresh Rate: (value+1) x 16 MCLK examples are: 8'h62 99(x16) MCLK cycles at 100MHz (15.84us). 8'h81 130(x16) MCLK cycles at 133MHz (15.64us). 8'hc4 197(x16) MCLK cycles at 200MHz (15.76us). Note that the controller will refresh one chip select at a time (to save the power spike of all the SDRAMs refreshing simultaneously) so refresh cycles will be seen on the channel four times as often as this field implies (in large memory mode two chip selects are refreshed at a time).
18:16	clock_drive	3'b111	This sets the drive strength (and therefore slew rate) of the output drivers for the clocks. 0 gives the weakest drive (slowest slew rate) and 7 the hardest. See <a href="#">Section: "I/O Control" on page 128</a> .
19	clock_class	1'b1	This bit should be clear to have the clock drivers configured for SSTL_2 Class 1 operation, and set for SSTL_2 class 2.
22:20	data_drive	3'b0	This sets the drive strength (and therefore slew rate) of the output drivers for the data lines. 0 gives the weakest drive (slowest slew rate) and 7 the hardest. See <a href="#">Section: "I/O Control" on page 128</a> .
23	data_class	1'b0	This bit should be clear to have the data drivers configured for SSTL_2 Class 1 operation, and set for SSTL_2 class 2.
26:24	addr_drive	3'b0	This sets the drive strength (and therefore slew rate) of the output drivers for the address and control lines. 0 gives the weakest drive (slowest slew rate) and 7 the hardest. See <a href="#">Section: "I/O Control" on page 128</a> .
27	addr_class	1'b0	This bit should be clear to have the address and control drivers configured for SSTL_2 Class 1 operation, and set for SSTL_2 class 2.
29:28	reserved	2'b0	Reserved
30	ref_disable	1'b0	On BCM1250 prior to PERIPH_REV3: Reserved On other parts: If this bit is set refresh cycles are disabled. This can be useful during debugging, but must not be set in normal operation.
31	dll_bypass	1'b0	Set this bit to bypass the DLLs. This should not be set in normal operation.
35:32	dqi_skew	4'b1000	DQS to data input skew control. See <a href="#">Section: "I/O Control" on page 128</a> .
39:36	reserved	4'b0	Reserved
43:40	dqo_skew	4'b1000	DQS to data output skew control. See <a href="#">Section: "I/O Control" on page 128</a> .
47:44	reserved	4'b0	Reserved
51:48	addr_skew	4'b1000	Address/control output delay from rising memory clock. See <a href="#">Section: "I/O Control" on page 128</a> .
55:52	reserved	4'b0	Reserved



**Table 74: Memory Clock Configuration Register (Cont.)**

<b>mc_clock_cfg_0 - 00_1005_1500</b> <b>mc_clock_cfg_1 - 00_1005_2500</b>			
Bits	Name	Default	Description
61:56	dll_default	6'b010000	This is the value to use in place of the DLL output when DLL bypass is enabled.
63:62	reserved	2'b0	Reserved

**Table 75: DRAM Command Register**

<b>mc_dramcmd_0 - 00_1005_1120</b> <b>mc_dramcmd_1 - 00_1005_2120</b> <b>WRITE ONLY</b>		
Bits	Name	Description
3:0	command	Command to issue to the SDRAM 0000: EMRS Write the extended mode register (EMR) (BA[1:0] = 01) 0001: MRS Write the mode register (MR) (BA[1:0] = 00) 0010: PREcharge all banks 0011: Auto Refresh (AR) 0100: Set self-refresh 0101: Clear self-refresh 0110: Set power-down 0111: Clear power-down
4	cs0	If this bit is set, cs0 is asserted when the command is issued.
5	cs1	If this bit is set, cs1 is asserted when the command is issued.
6	cs2	If this bit is set, cs2 is asserted when the command is issued.
7	cs3	If this bit is set, cs3 is asserted when the command is issued.
63:8	reserved	Reserved

**Table 76: DRAM Mode Register**

<b>mc_drammode_0 - 00_1005_1140</b> <b>mc_drammode_1 - 00_1005_2140</b>			
Bits	Name	Default	Description
12:0	emode	15'b0	This value is written to the Extended MODE register via the address lines when the EMRS command is issued to the <b>mc_dramcmd</b> register.
14:13			These bits provide the additional two bits to send on address bits 14:13 during an EMRS command to FCRAMs. They are ignored for standard DDR SDRAMs.
15	reserved	1'b0	Reserved
28:16	mode	15'h22	This value is written to the MODE register via the address lines when the MRS command is issued to the <b>mc_dramcmd</b> register. The mode register must always be programmed for sequential burst order and a burst length of 4. If not memory accesses will be UNPREDICTABLE.
30:29			These bits provide the additional two bits to send on address bits 14:13 during an MRS command to FCRAMs. They are ignored for standard DDR SDRAMs.
31	reserved	1'b0	Reserved

**Table 76: DRAM Mode Register (Cont.)**

mc_drammode_0 - 00_1005_1140 mc_drammode_1 - 00_1005_2140				
Bits	Name	Default	Description	
34:32	dram_type	3'b0	DRAM type: 000: JEDEC DDR (one DQS for every 8 DQ bits) 001: FCRAM 010: DDR SGRAM (one DQS for every 32 DQ bits) Others Reserved	
35	large_mem	1'b0	If this bit is set large memory support (using an external chip select decoder) is enabled. See Section: "Larger Memory Systems" on page 124.	
36	pre_on_A8	1'b0	If this bit is set the AutoPrecharge flag will be output on the A8 address bit rather than A10. (Only supported on parts with system revision PERIPH_REV3 or greater).	
37	ram_with_A13	1'b0	This bit should be set when the large_mem bit is set if the row address uses A13. It causes the CS[3:2] to be driven from bits [15:14] of the row address mask. It also needs to be set to cause BA[2] to be output on A13 rather than A12 for non-standard 8 bank DRAMs. (Only supported on parts with system revision PERIPH_REV3 or greater).	
63:38	reserved	26'b0	Reserved	

**Table 77: SDRAM Timing Register**

mc_timing1_0 - 00_1005_1160 mc_timing1_1 - 00_1005_2160				
Bits	Name	Range	Default	Description
3:0	tRCD	1-6	4'h3	RAS to CAS delay. This should be set to 1 for FCRAMs.
6:4	tCrD	1-6	3'h2	Cycles from CAS(read) to data (the CAS latency).
7	tCrDh	0-1	1'b0	If this bit is set the data capture is delayed by a half cycle from the integer part of the CAS latency set in tCrD. See the discussion in "Timing Parameter Guidelines" on page 131.
11:8	tCwD	1-2 or 1-4	4'h1	Delay from CAS to Write data. On parts with system revision PERIPH_REV3 or greater the range is extended to 1-4.
15:12	reserved	0	4'h0	Reserved
19:16	tRP	1-4	4'h4	Cycles required from PRECHARGE to RAS.
23:20	tRRD	1-4	4'h2	Cycles required from RAS to RAS for a different bank.
27:24	tRCw-1	1-15	4'ha	One less than cycles required from RAS(write) to next RAS of the same bank.
31:28	tRCr-1	1-15	4'h9	One less than cycles required from RAS(read) to next RAS of the same bank.
35:32	reserved	0	4'h0	Reserved
39:36	reserved	0	4'h0	Reserved
43:40	tCwCr	0-15	4'h4	Cycles of separation from write to read. This field is calculated as: $tCwCr = tRCw - (tRCD + tCwD + 2 + tWTR)$ Note that tRCw is one greater than the value set in tRCw-1 field and tWTR is the internal write to read delay for the SDRAM (normally specified as 1 cycle).
47:44	reserved	0	4'h0	Reserved
51:48	reserved	0	4'h0	Reserved
55:52	tRFC	0-15	4'hc	Cycles required from Auto-refresh to Active or Auto-refresh.
59:56	tFIFO	0-1	4'h1	Additional cycles required for data capture FIFO.





**Table 77: SDRAM Timing Register (Cont.)**

mc_timing1_0 - 00_1005_1160 mc_timing1_1 - 00_1005_2160				
Bits	Name	Range	Default	Description
60	w2rldle	0-1	1'b1	Number of idle cycles required for write to read turnaround. This bit must always be set.
61	r2wldle	0-1	1'b1	Number of idle cycles required for read to write turnaround. If this bit is 0 one idle cycle is used. If this bit is 1 two idle cycles are used. Usually set for half cycle CAS latencies and cleared otherwise.
62	r2rldle	0-1	1'b1	Number of idle cycles required for read to read of a different chip turnaround. If this bit is 0 one idle cycle is used. If this bit is 1 two idle cycles are used. This bit is usually 0.
63	reserved	0	1'b0	Reserved

**Table 78: SDRAM Timing Register 2**

mc_timing2_0 - 00_1005_1180 mc_timing2_1 - 00_1005_2180		
Bits	Name	Description
63:0	reserved	Reserved

**Table 79: Chip Select Start Address Register**

mc_cs_start_0 - 00_1005_11A0 mc_cs_start_1 - 00_1005_21A0					
Bits	Name	Default Ch0	Default Ch1	Default Ch1 (BCM1125/H)	Description
15:0	cs0_start	16'h0000	16'h0100	16'h0000	Chip select 0,1,2,3 region start address bits [39:24].
31:16	cs1_start	16'h0010	16'h0000	16'h0010	These address are in the memory address space (after the translation described in <a href="#">Section: "Mapping" on page 109</a> ).
47:32	cs2_start	16'h0020	16'h0000	16'h0020	
63:48	cs3_start	16'h0030	16'h0000	16'h0030	

**Table 80: Chip Select End Address Register**

mc_cs_end_0 - 00_1005_11C0 mc_cs_end_1 - 00_1005_21C0					
Bits	Name	Default Ch0	Default Ch1	Default Ch1 (BCM1125/H)	Description
15:0	cs0_end	16'h0010	16'h0200	16'h0010	Chip Select 0,1,2,3 region end address + 1 bits [39:24].
31:16	cs1_end	16'h0020	16'h0000	16'h0020	These address are in the memory address space (after the translation described in <a href="#">Section: "Mapping" on page 109</a> ).
47:32	cs2_end	16'h0030	16'h0000	16'h0030	
63:48	cs3_end	16'h0040	16'h0000	16'h0040	

**Table 81: Chip Select Interleave Register**

mc_cs_interleave_0 - 00_1005_11E0 mc_cs_interleave_1 - 00_1005_21E0			
Bits	Name	Default	Description
4:0	reserved	5'b0	Reserved
24:5	interleave	20'b0	For mixed_cs mode single set bit should be written to this register in the bit position that should be used to select between the two chip selects in the interleaved portion. For interleaved_cs mode two adjacent bits should be set in this register in the bit positions that should be used to select between the four chip selects.
63:25	reserved	39'b0	Must be zero.

**Table 82: Row Address Bits Select Register**

mc_cs0_row_0 - 00_1005_1200 mc_cs1_row_0 - 00_1005_1260 mc_cs2_row_0 - 00_1005_12C0 mc_cs3_row_0 - 00_1005_1320 mc_cs0_row_1 - 00_1005_2200 mc_cs1_row_1 - 00_1005_2260 mc_cs2_row_1 - 00_1005_22C0 mc_cs3_row_1 - 00_1005_2320			
Bits	Name	Description	
<b>Default: (4k)</b> 64'b00000000_00000000_00000000_00000000_00001111_11111111_10000000_00000000			
9:0	reserved	Reserved	
34:10	select	Address bits are selected for use as the row address by setting the corresponding bits in this register. The number of bits set should match the number of rows of the SDRAM. The set bits must be contiguous.	
63:35	reserved	Must be zero.	

**Table 83: Column Address Bits Select Register**

mc_cs0_col_0 - 00_1005_1220 mc_cs1_col_0 - 00_1005_1280 mc_cs2_col_0 - 00_1005_12E0 mc_cs3_col_0 - 00_1005_1340 mc_cs0_col_1 - 00_1005_2220 mc_cs1_col_1 - 00_1005_2280 mc_cs2_col_1 - 00_1005_22E0 mc_cs3_col_1 - 00_1005_2340			
Bits	Name	Description	
<b>Default: (1k)</b> 64'b00000000_00000000_00000000_00000000_00000000_01111111_10000000			
4:0	reserved	Reserved (The controller always acts as if bits 4:3 are set.)	
20:5	select	Address bits are selected for use as the column address by setting the corresponding bits in this register. The number of bits set should match the number of columns of the SDRAM. The set bits must be contiguous, with two exceptions <ul style="list-style-type: none"> <li>• Bit 5 set and the other bits contiguous (this allows for bank, CS or channel interleave on 64 byte blocks)</li> <li>• Bit 5 and 6 set and the other bits contiguous (this allows for bank, CS or channel interleave on 128 byte blocks)</li> </ul>	



**Table 83: Column Address Bits Select Register (Cont.)**

<b>mc_cs0_col_0</b> - 00_1005_1220 <b>mc_cs1_col_0</b> - 00_1005_1280 <b>mc_cs2_col_0</b> - 00_1005_12E0 <b>mc_cs3_col_0</b> - 00_1005_1340 <b>mc_cs0_col_1</b> - 00_1005_2220 <b>mc_cs1_col_1</b> - 00_1005_2280 <b>mc_cs2_col_1</b> - 00_1005_22E0 <b>mc_cs3_col_1</b> - 00_1005_2340		
Bits	Name	Description
63:21	reserved	Must be zero.

**Table 84: Bank Address Bits Select Register**

<b>mc_cs0_ba_0</b> - 00_1005_1240 <b>mc_cs1_ba_0</b> - 00_1005_12A0 <b>mc_cs2_ba_0</b> - 00_1005_1300 <b>mc_cs3_ba_0</b> - 00_1005_1360 <b>mc_cs0_ba_1</b> - 00_1005_2240 <b>mc_cs1_ba_1</b> - 00_1005_22A0 <b>mc_cs2_ba_1</b> - 00_1005_2300 <b>mc_cs3_ba_1</b> - 00_1005_2360		
Bits	Name	Description
<b>Default:</b> 64'b00000000_00000000_00000000_00000000_00000000_00000000_01100000		
4:0	reserved	Reserved
36:5	select	Address bits are selected for use as the bank address by setting the corresponding bits in this register. The number of bits set should match the number of banks of the SDRAM. The set bits must be countiguous.
63:37	reserved	Must be zero.

**Table 85: Chip Select Attribute Register**

<b>mc_cs_attr_0</b> - 00_1005_1380 <b>mc_cs_attr_1</b> - 00_1005_2380			
Bits	Name	Default	Description
1:0	cs0_page	2'b01	Page policy for Chip Select 0. (See <a href="#">Section: "Page Policy" on page 122.</a> ) 00: Closed Page. AutoPrecharge every CAS. 01: CAS time check, AutoPrecharge unless there is another request in the queue. 10: Hint Based check, as CAS time check but including hint signal. 11: Open Page. The page is always left open.
15:2	reserved	14'h0	Must be zero.
17:16	cs1_page	2'b01	Page policy for Chip Select 1. (See <a href="#">Section: "Page Policy" on page 122.</a> ) 00: Closed Page. AutoPrecharge every CAS. 01: CAS time check, AutoPrecharge unless there is another request in the queue. 10: Hint Based check, as CAS time check but including hint signal. 11: Open Page. The page is always left open.
31:18	reserved	14'h0	Must be zero.
33:32	cs2_page	2'b01	Page policy for Chip Select 2. (See <a href="#">Section: "Page Policy" on page 122.</a> ) 00: Closed Page. AutoPrecharge every CAS. 01: CAS time check, AutoPrecharge unless there is another request in the queue. 10: Hint Based check, as CAS time check but including hint signal. 11: Open Page. The page is always left open.

**Table 85: Chip Select Attribute Register (Cont.)**

mc_cs_attr_0 - 00_1005_1380 mc_cs_attr_1 - 00_1005_2380			
Bits	Name	Default	Description
47:34	reserved	14'h0	Must be zero.
49:48	cs3_page	2'b01	Page policy for Chip Select 3. (See <a href="#">Section: "Page Policy" on page 122.</a> ) 00: Closed Page. AutoPrecharge every CAS. 01: CAS time check, AutoPrecharge unless there is another request in the queue. 10: Hint Based check, as CAS time check but including hint signal. 11: Open Page. The page is always left open.
63:50	reserved	14'h0	Must be zero.

**Table 86: ECC Test Data Register**

mc_test_data_0 - 00_1005_1400 mc_test_data_1 - 00_1005_2400			
Bits	Name	Default	Description
63:0	invert	64'h0	This value is XORed with the data written to memory. Any bits set in this register will cause the corresponding data bit to be inverted compared to the data the ECC bits were calculated for. If only one bit is set a correctable ECC error should result from a read. If two bits are set an uncorrectable ECC error should result from a read.

**Table 87: ECC Test ECC Register**

mc_test_ecc_0 - 00_1005_1420 mc_test_ecc_1 - 00_1005_2420			
Bits	Name	Default	Description
7:0	invert	8'h0	This value is XORed with the ecc code written to memory. Any bits set in this register will cause the corresponding ecc bit to be inverted compared to value that was calculated from the data. If only one bit is set a correctable ECC error should result from a read. If two bits are set an uncorrectable ECC error should result from a read.
63:8	reserved	56'h0	Must be zero.



---

This Page is left blank for notes



---

This Page is left blank for notes



## Section 7: DMA

### DMA CONTROLLERS

There are several DMA controllers. The programming interface is similar for all of them and is described in this section.

Each synchronous serial port has a transmit and a receive DMA channel and each ethernet interface has two transmit and two receive DMA channels. A DMA channel has a group of descriptors which may either be organized as a ring (in which case each descriptor can point to two memory buffers) or a chain (in this case there is only one memory buffer associated with a descriptor and the other pointer is the link pointer to the next descriptor in the chain). Each descriptor contains some control bits that are used to select interface options, and following packet transfer the DMA controller will update some flag bits.

The configuration registers for the ethernet and serial interfaces are the same. The descriptions in this section use generic names. The actual register names for the ethernet interfaces are generated by appending `_mac_N_rx_ch_M` for receive channel *M* (0,1) of ethernet interface *N* (0,1 for the BCM1125/H and 0,1,2 for the BCM1250), and `_mac_N_tx_ch_M` for the transmit channels. The register names for the serial interfaces are generated by appending `_ser_N_rx` and `_ser_N_tx` for serial interface *N* (0,1).

There is an additional generic Data Mover for doing transfers between arbitrary addresses (both memory and I/O can be addressed). This is similar in style to the other DMA controllers, but has a slightly different programming interface since it needs to support both source and destination addresses. The general discussion in the following sections applies to the Data Mover, specific details are given in [Section: "Data Mover" on page 176](#).

### DATA BUFFERS AND DESCRIPTORS

There are two fundamental elements in the DMA system: data buffers and descriptors. Data buffers are sized areas of the physical address space that hold data. Descriptors point to data buffers and hold their parameters. Receive DMA controllers move bytes between an interface and a data buffer, transmit DMA controllers move bytes between a data buffer and an interface, and Data Mover DMA controllers move bytes between two data buffers.

In many systems a data buffer resides in the main memory. However, there is no need for this to be the case, the address generated by the DMA controller is interpreted in the same way that a CPU physical address would be, thus the controller in the ethernet interface could transfer packets directly to a device on the HyperTransport fabric without the data going through memory (for example if incoming packets are encrypted and must be passed through a decryption engine before the CPU can use them).

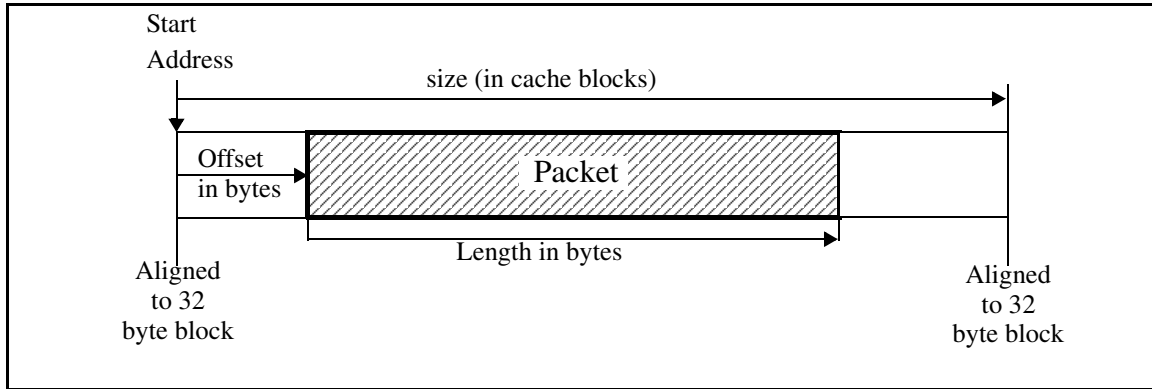


Figure 23: DMA Buffer

Figure 23 shows a data buffer as used by the ethernet and serial interfaces (Data Mover buffers are the same but the alignment restrictions are removed). Three parameters describe a data buffer:

Table 88: Data Buffer Parameters

<b>Start address</b>	The start address of a data buffer is the 40 bit physical address at which the buffer starts. In the ethernet and serial port DMA engines it must be cache block aligned, so the lowest five bits of the address must be zero. The Data Mover buffers can have arbitrary alignment. The Ethernet MAC DMA engines in parts where system_revision indicates PERIPH_REV3, or later, have an additional mode that allows arbitrary alignment.
<b>Size</b>	The size of the data buffer is given in cache blocks, i.e. the buffer must be a multiple of 32 bytes in size. Prior to PERIPH_REV3, if a transmit channel is configured in two block fetch mode (by setting the tbx_en bit in the dma_config0 register) the buffer size must be a multiple of 64 bytes The maximum size buffer is 512 blocks (16 KBytes). The Data Mover does not use the size field, the buffer length matches the length parameter described below.
<b>Owner</b>	The data buffer is either owned by one of the DMA controllers or by the system software. Ownership is not explicit, each DMA controller has a count of the number of descriptors it owns and will use all buffers pointed to from those descriptors. Software needs to keep track of the buffers and must update the count when passing ownership. (When software writes to the counter the value written will be added to the existing count, the software does not need to do a read-modify-write operation).

There are an additional two parameters that describe the data in buffers:

Table 89: Data Parameters

<b>Length</b>	This is the length in bytes of the valid packet data in the buffer. The length may be greater than the current buffer size, in which case the packet spans multiple buffers. If the packet spans multiple descriptors then the length field in the first descriptor is used, and the length given in the subsequent descriptors is ignored. The maximum packet size for interface DMA engines is 16 KB-1, the maximum size for the Data Mover is 1 MB.
<b>Offset</b>	This is the offset in bytes of the start of the data. The offset can be between 0 and 31 bytes. If a packet spans multiple buffers, the offset is only applied to the first buffer. The offset can be used to align the header of the packet more conveniently. For example a packet with a 14 byte ethernet header could be started with an offset of 2 bytes to align the IP header to a 64 bit doubleword boundary. Alternatively the offset may be used to easily add or strip an encapsulation header.



The DMA descriptor contains these parameters for each data buffer. A descriptor consists of two 64 bit double-words, dscr\_a and dscr\_b, that are stored in memory at consecutive locations aligned to a 16 byte boundary. Figure 24 shows a descriptor. It can point to up to two buffers. The offset is only used at the start of a packet and normally applies to buffer A (since a new packet will always start a new descriptor). However, if the offset\_b bit is set the offset is applied to the first B buffer in a packet (no offset is applied on buffer A so it will completely fill before the offset is applied at the start of buffer B). In addition to the fields describing the data the descriptor contains per-packet options that are passed to the interface, and status flags that can be written back into the descriptor by the interface. The options and status for the ethernet descriptors are summarized in Section: "Option and Flag Bits for Ethernet MACs" on page 171, those for the serial interface are summarized in Section: "Control and Flag Bits for Synchronous Serial Interface" on page 174. The DMA engines access the descriptors in cacheable-coherent memory, they must also be mapped this way in the CPU.

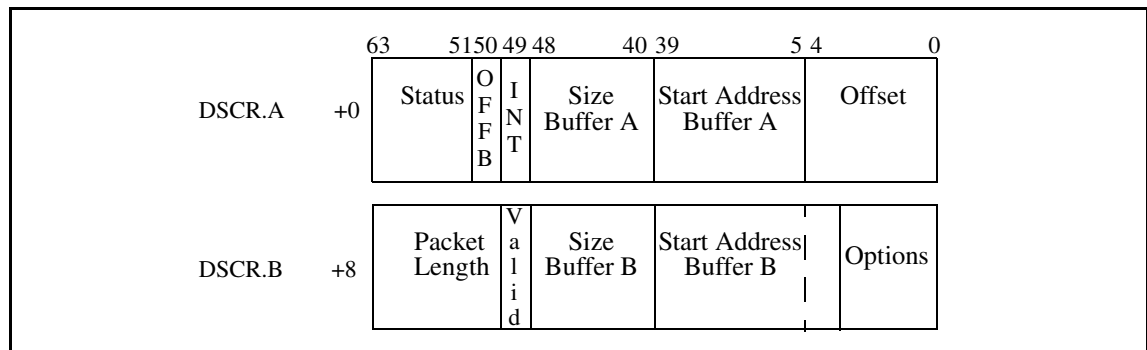


Figure 24: DMA Descriptor

The descriptor also has three control bits. If the INT bit is set then the DMA channel will raise an interrupt when it has completed processing of the descriptor. The second control bit indicates if the buffer B details are valid, if this bit is clear the descriptor only has one associated buffer and the buffer B details are ignored (they are preserved when the status information is written back in to the descriptor, so could be used by software to store state information). The third control bit allows the offset field to be applied to the B buffer rather than the A buffer.

Figure 25 on page 150 shows a packet spanning three buffers. The start of the packet is offset from the start of the first buffer. Following the last byte in the first buffer the packet continues to fill the entire second buffer. Note that the offset is not applied in the continuation buffer. The packet ends in the third buffer, again the offset is not used. The packet length (which is only put in the DMA descriptor associated with the first buffer) is the sum of the length used in the first buffer (i.e. the number of bytes from the offset to the end of the buffer), the full length of the second buffer, and the length used in the third buffer (i.e. the number of bytes from the buffer start to the end of the valid data).

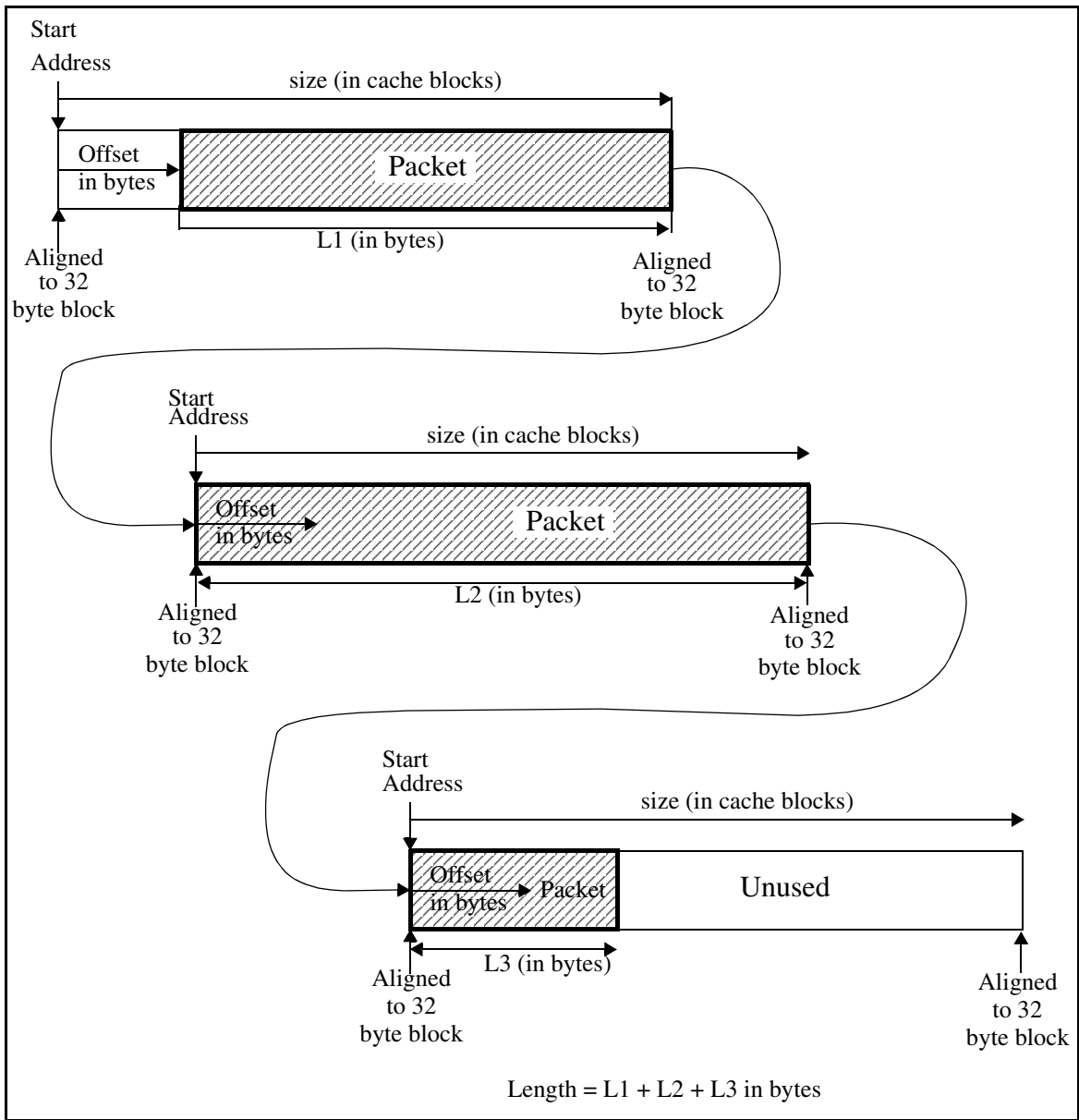


Figure 25: Packet Spanning Three Buffers

## RINGS AND CHAINS

Each DMA engine works from a set of descriptors. These may be organized either as a ring or a chain.

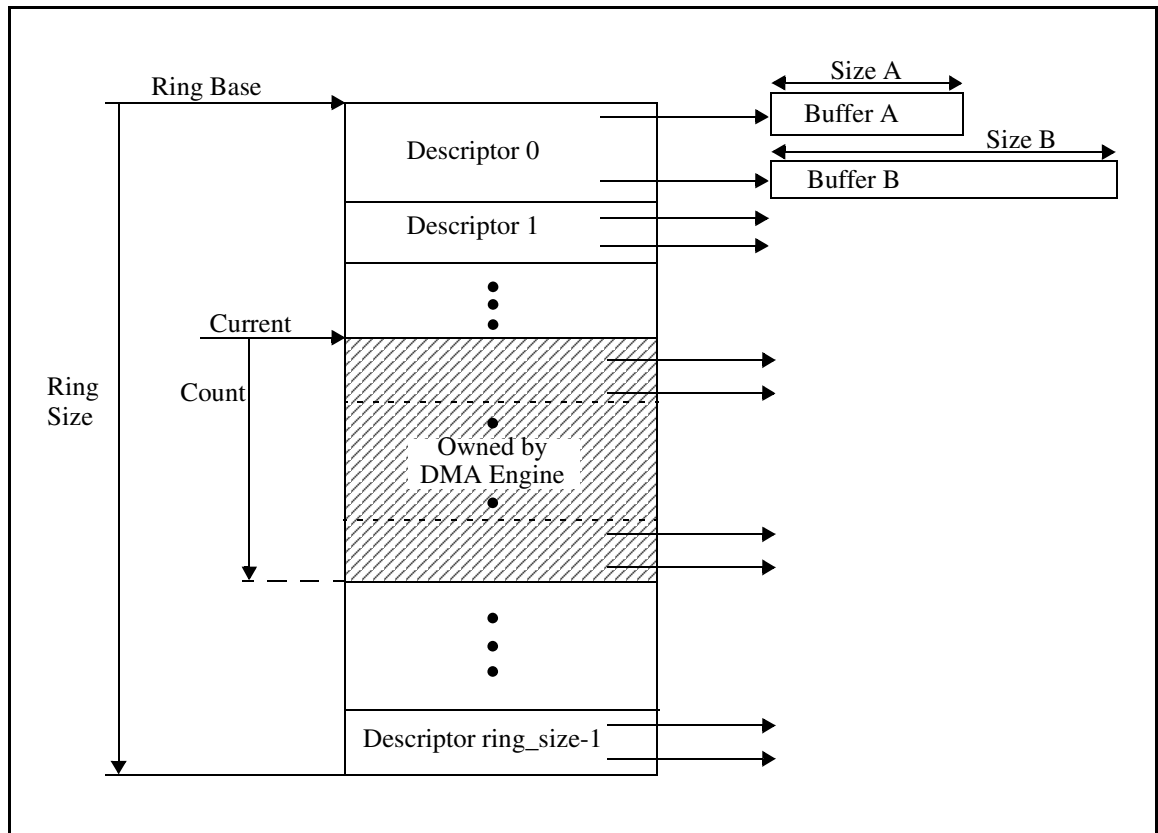


Figure 26: DMA Descriptor Ring

A descriptor ring is shown in Figure 26. Each descriptor can point to two buffers and the next descriptor to use is in memory just after the current descriptor. The number of descriptors in the ring is configured when the ring is established. The DMA engine will detect when it has reached the last descriptor (at the highest memory location) by checking the ring size, its successor is the descriptor at the lowest memory location, which is pointed to by the descriptor base register in the configuration block. The ring size can be from 1 to 65536 descriptors. The DMA controller keeps track of its current position in the ring (initially the base). Software passes ownership of descriptors (and the associated pair of buffers) to the controller by incrementing the count of the number of descriptors owned by the controller. Writing to the count register adds the value written to the current count. The software may write the register at any time, the hardware prevents the potential conflict of the software incrementing as the controller decrements. There is no overflow protection on the count, so software should take care not to let the count pass 65535 descriptors owned by the DMA controller. When the controller has finished with a descriptor it will return ownership by decrementing the count. If the descriptor interrupt bit is set an interrupt will be raised when ownership of the descriptor is returned.

In a receive channel the interface flags the start of a packet when it sends data to the DMA engine. In a transmit channel the length field for one packet allows the DMA engine to directly determine when to expect the start of the next packet. In either case the DMA engine will always advance to a new descriptor when a new packet is started. Thus the first buffer in the pair can be used to point to small packet header buffers and the second buffer can be large and used for packet bodies. (This does not apply for the Data Mover since each transaction has its own descriptor.)

The descriptor associated with the first buffer in a packet is treated specially. It contains per-packet options that are passed to the interface describing any special treatment that this packet should receive. Once a packet has been completely transferred status information is written back into the first descriptor. Received packets accumulate many status bits (described in [Section: "Option and Flag Bits for Ethernet MACs" on page 171](#) and [Section: "Control and Flag Bits for Synchronous Serial Interface" on page 174](#)), and additionally have the packet length written in to the descriptor. Only the first descriptor of the packet is written and the SOP flag bit will always be set. Software can clear the status word in all descriptors used by the receive engine and then use the SOP bit to identify descriptors that start packets. The SOP bit will only be set when the full packet has been received. If the system revision indicates PERIPH\_REV3 or greater then the Ethernet DMA provides the option of overwriting the a\_size field in the SOP descriptor with the number of descriptors that were used by the received packet. If this is enabled (by setting the xtra\_status bit in the **dma\_config1** register) and the standard descriptor format is used then the software will need to keep track of the A buffer size by some method other than reading the descriptor and will need to restore the a\_size field before the descriptor can be used for a subsequent DMA.

For transmitted packets the only change in the status flags is that the SOP bit is cleared, this can be used by software to detect the packet transmission has completed. If software does not make use of this feature the bandwidth used for descriptor management through the I/O bridge (and thus the latency for other accesses) can be reduced by disabling writing back the transmit status.

In the Ethernet DMA engine the SOP bit must always be set for the descriptor associated with the start of a packet, if the engine sees this bit clear in a descriptor that it believes should be the start of a packet then it will report a descriptor error and stop until software fixes the problem. If the SOP bit is set then the options field of the transmit descriptor must be nonzero and is used to enable operations on the data as it is transmitted, if the SOP bit is clear then the options field of the descriptor must be zero.

If a channel is disabled, when it is enabled it will start from the descriptor pointed to by the base address. The count will not be changed. The CPU must either zero the count (by adding the two's-complement of its current value) or reorder the descriptors to put valid ones at the start of the ring or chain. If the system revision indicates PERIPH\_REV3 or greater then an Ethernet transmit DMA channel may be paused by setting the pause\_channel\_en bit in the **dma\_config1** register. This will pause the channel (but not the other transmit channel) at the next packet boundary. Transmission will be resumed when the bit is cleared. This can be used for example by software to stop transmission of a best-effort queue when the link partner indicates it is only able to buffer priority traffic.

The Ethernet interface will normally respond in hardware at the MAC layer to pause frame flow control requests. However, if the system\_revision indicates PERIPH\_REV3 or greater it can also be set to implement the pause at the DMA layer (when this is done the interface may stop more slowly because of any packets buffered in the fifo between the DMA engine and MAC). If this option is enabled (by setting the channel\_base\_fc\_en bit in the **mac\_vlan\_tag** register) then the fc\_pause\_en bit in the **dma\_config1** register determines if the channel is affected by pause frames or not. Thus, provided the link partner is appropriately configured, the interface can be set to pause only one channel and allow the other (high priority) to continue even when the flow control is requested.

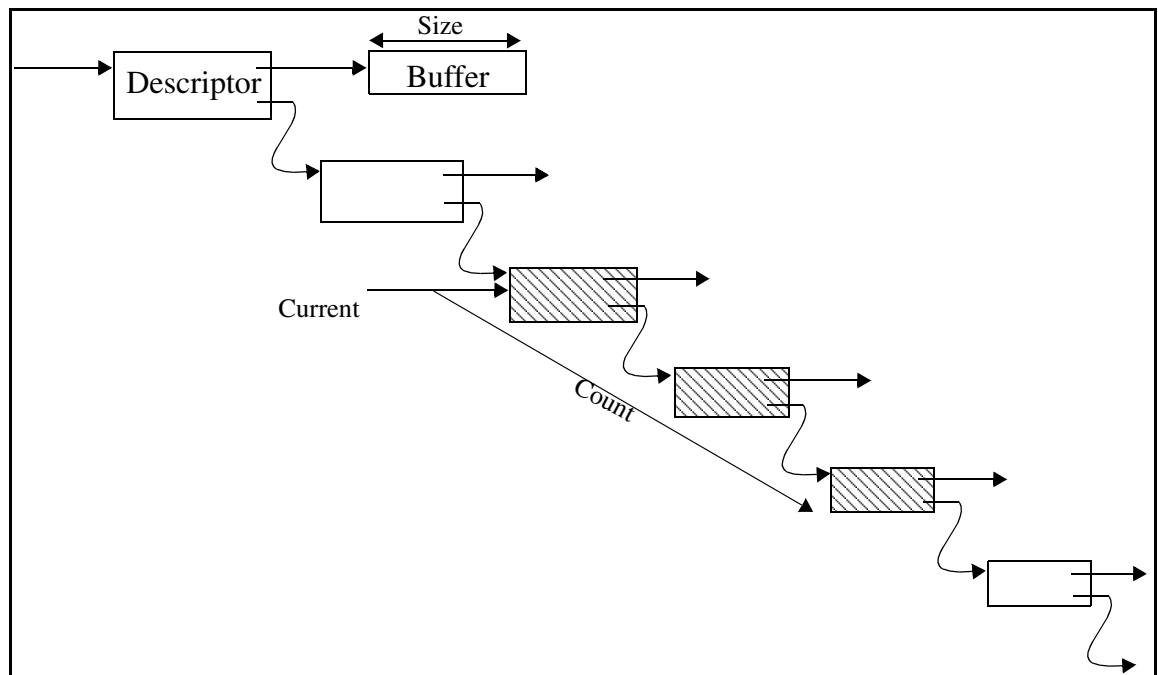


Figure 27: DMA Descriptor Chain

Figure 27 shows a descriptor chain. Each descriptor points to a single buffer, and contains a pointer to the next descriptor. Ownership of descriptors is passed in the same way as ring mode, the count refers to the number of valid descriptors. The last valid descriptor must contain a link pointer to the location where software will place the next descriptor, when the count reaches zero the controller will remember the link value and fetch the new descriptor from that address when the count becomes non-zero.

In both ring and chain modes reading the `dma_cur_dscr_addr` register gives address of the descriptor currently being processed and the current count of descriptors owned by the controller. If the count field in this register is non-zero then the address is of the descriptor that the controller is currently using. If the count is zero then the address is where the controller will fetch the next descriptor from when the count becomes non-zero.

### UNALIGNED BUFFER DESCRIPTOR FORMAT FOR ETHERNET DMA

The alignment restrictions in the standard DMA descriptors may be hard to work with in some existing systems and when a lot of scatter/gather operations are done to form packets for transmission. The DMA engine in the Ethernet/Packet Fifo Interface on parts with system revision indicating PERIPH\_REV3 or greater supports a third descriptor format that allows arbitrary alignment of the start and end of buffers. The descriptors are slightly modified to support this and there are a number of restrictions. Figure 28 on page 154 shows the descriptor format for the standard and unaligned buffer formats (a full definition is in Table 102 and Table 103). The changes and restrictions are:

- A buffer pointers always use the offset field (i.e. are a full 40 bit physical address).
- The size of the A buffer is specified in bytes in bits 21:8 of the dscr\_b (where the buffer B address is in the standard format). Note that the descriptor field must be set to the size of the data plus the low 5 bits of the start address, so the maximum buffer size for arbitrary alignment is 16K-32 bytes.
- The a\_size field is not used and can therefore always be used for the optional extra status information.
- No B Buffers are available.
- Only Ring mode can be used.
- The Packet Length, Status, Interrupt and Options fields work as in the standard mode.

The descriptor format is selected separately for each DMA channel. The unaligned mode is particularly useful for the transmit descriptors when packet manipulations are performed, but the standard mode may be more useful for the receive side where there is more control of the buffer locations and advantage can be taken of A and B buffers.

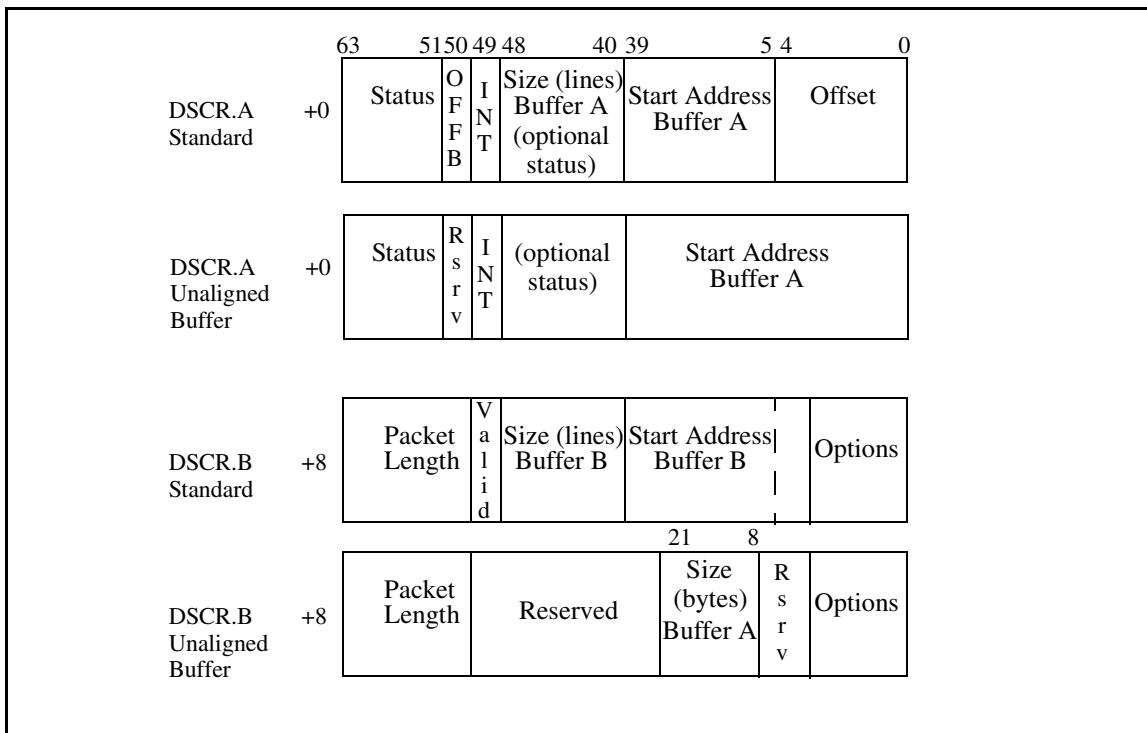


Figure 28: Standard and Unaligned Buffer DMA Descriptors



This format is useful for scatter/gather operations, in particular when transmit packets are being composed and headers inserted or removed. Care must be taken since the format allows for very low performance settings: for example if all buffers are only 3 bytes then the overhead of fetching descriptors will completely dominate the data transfers.

When the unaligned buffer format is used for packet reception and the start and end of a buffer are not aligned to a cache block boundary the DMA engine provides two options. The high performance option is for the interface to always write full cache lines (using the ZBbus WriteInvalidate command) and therefore overwrite the bytes before and after the buffer with UNPREDICTABLE data. The lower performance option is to use a read-modify-write to ensure only the valid bytes are modified, this will have a tendency to back up the read-modify-write queue (since the write must be queued for the read latency) and should therefore be avoided if possible.

## DMA COHERENCE AND CACHE OPTIONS

All DMA transfers are coherent with the CPU L1 caches, the L2 cache and memory. On reads the latest copy of the data will be retrieved from the current owner and on writes any existing copies of the data will be invalidated or updated.

The DMA controllers can drive a L2-cache request attribute with transfers. This will cause the data to be allocated in the L2 cache if the cache misses (if the L2 cache hits it will always be updated regardless of the attribute). The channel configuration sets how many blocks from the start of a packet should be marked for allocation in the L2 cache (this can be set larger than the maximum packet size to have complete packets sent to the cache). The L2 is still being used as a cache, the data still has a memory address and can be evicted from the cache to memory if the L2 replacement algorithm requires it.

The network and serial port interfaces have data buffers that are aligned to a cache block boundary and are an integral number of cache blocks in size, so on incoming transfers they never need to perform read-modify-write operations (the actual packet data may be offset from the start of the buffer and may finish short of the end of the buffer, but the DMA will always be padded out to a full cache line at each end). The writes to memory are always made with the write-invalidate command, which invalidates any copies of the block in L1 caches, and returns ownership of the block to the L2 cache and memory system (the default owner).

In the standard aligned buffer mode, or the unaligned buffer mode with WriteInvalidate, device driver code has to take care not to store any additional data in the cache block that a buffer starts or ends in. Particular care must be taken with buffer link pointers and buffer header information at the start of the buffer. Even if an offset is applied in the descriptor the buffer is still aligned and any bytes before the offset will be overwritten with UNPREDICTABLE values when a packet is received. In the standard mode the end of the buffer is always aligned so there is not a problem, but in the WriteInvalidate unaligned mode if the buffer has an unaligned end the bytes for the rest of the cache block that contain the end of the buffer will be overwritten with UNPREDICTABLE data.

## DMA CONFIGURATIONS

There are many ways that the DMA structure can be configured allowing use with many existing systems. The CPU should always use the cacheable coherent mode to access descriptors and buffers to allow the system to take care of the coherence and avoid the need for software to manage the L1 caches. The DMA engines access descriptors and buffers through the I/O bridge which will always use cacheable coherent accesses to memory addresses and uncacheable (accelerated) accesses for other addresses.

Consideration should be given to the descriptors and their access pattern. In memory space all reads from the controller will be of a full cache block, so if the system were configured to use chain mode descriptors and 32 byte buffers the number of memory accesses fetching descriptors will equal the number of memory accesses for data. The system performance will be lower than expected because half of the bandwidth is consumed by descriptor management. In a new system it is recommended that the descriptors be organized in ring mode and the controller be permitted to prefetch descriptors (`tdx_en` set in the **dma\_config0** register). This reduces the descriptor fetch overhead compared to the number of buffer transfers. In most cases receive descriptors should also be marked for allocation in the L2 cache (`dscr_l2ca` set in the **dma\_config1** register) since they are a shared resource and they are expected to be accessed over a short time period by the DMA controller (twice for the header descriptor which will be updated with the length and status information) and the CPU. When the DMA controller updates a descriptor it must use a read-modify-write operation. If software does not need the SOP bit to be cleared in transmit descriptors, the read-modify-write can be avoided by setting the `no_dscr_updt` bit in the **dma\_config1** register. If this is done, transmit descriptors do not need to be marked for L2 cache allocation by the DMA engine, the CPU will have them cacheable in both L1 and L2 caches so the DMA engine will read them directly from the cache and it never writes them back.

The buffer size can also be increased to reduce the descriptor overhead. While the hardware does support buffers as small as 32 bytes (and these may be ideal as header buffers) the system becomes more efficient as the buffer size is increased. In the transmit channel the `tbx_en` bit should be set in the **dma\_config0** register to allow the controller to prefetch buffer data whenever possible. If this bit is set the controller will mostly fetch pairs of cache blocks back to back and is more likely to make use of an open page in the SDRAM. On the receive side cache block writes are posted as soon as the data is available and this bit has no effect.

The most sensitive interface is the transmit side of the Ethernet (or Packet FIFO in GMII mode). Once transmission of a packet has started there must be no interruptions. This is achieved by using a small data FIFO in the interface and priority in the memory controller. The data FIFO is intended just for speed match buffering. There is a threshold which sets how much of a packet must be in the FIFO for transmission to begin, since the FIFO drains at a constant rate this directly translates into the memory latency that will be covered by the buffer. In general the FIFO will fill quickly (particularly when `tbx_en` allows fetching of pairs of cache blocks), and drain during any high latency memory reads. The worst case is when a new descriptor must be fetched during a packet transmission, since the next block of data cannot be fetched until the descriptor read completes. To allow this to work with a relatively small FIFO the memory controller implements a priority scheme. This is needed because the CPUs and data mover can easily swamp the memory controller (they can access data at much higher bandwidths and frequencies than the I/O DMA engines, so this is an ideal place for priority to protect the low request rate interface from being dominated by a high request rate one). Any reads from the I/O bridge 1 are prioritized over other memory accesses, if they conflict with a write to memory the priority of the write is also raised. In addition some of the memory controller buffers can be reserved for use by the I/O bridge 1 DMA engines. The priority scheme is described in [Section: "Memory Controller Architecture" on page 104](#). If memory is accessed across the HyperTransport there is no priority scheme and the transmit FIFO is likely to underflow during packet transmission (however, because the writes are posted it should be possible to receive into buffers in a memory across the HyperTransport fabric).





## ETHERNET AND SERIAL DMA ENGINES

The Ethernet and serial DMA engines are identical except for using different per-packet options and status flags. Each channel works as described above using either a ring or a chain. The DMA engines have the restriction that neither the descriptors nor data buffers may be at an address serviced by I/O bridge 1, the system behavior is UNDEFINED if this is violated. In practice this restriction just rules out DMA access from the Ethernet or Serial DMA to the generic bus, since the other devices connected through I/O Bridge 1 are all based on control registers and are unreasonable DMA targets. [Figure 5 on page 9](#) shows the devices that are behind I/O Bridge 1.

Both Ethernet and serial DMA controllers are connected to the ZBbus through I/O Bridge 1. In the direction from the ZBbus to the devices there is a separate command queue for read and writes to peripheral registers and response queue for data returning to descriptor or buffer DMA read requests. This ensures that the DMA traffic is not blocked by requests to slow peripherals (such as a slow device on the generic bus). In the direction towards the ZBbus it is important that if a status read reports completion of a command the response does not pass that command, so read and write requests from the DMA engine and responses to register reads are held in a single queue. Generic bus responses do not need this ordering and are given their own return queue.

Each channel has an interrupt associated with it. In the ethernet interface the status for the two receive and two transmit DMA channels are combined into the **mac\_status** register for the interface (see [Table 182 on page 309](#)). The serial interface transmit and receive channel status are combined into the **ser\_status** register for each channel that can be read to determine the interrupt cause (see [Table 240 on page 356](#)). Reading the status will clear all bits in the status register and clear the interrupt.

### DESCRIPTOR COUNT WATERMARKS

The count of descriptors owned by the DMA controller is compared to two watermark registers. The high watermark is set in the **high\_watermark** field of the **dma\_config0** register, and the low watermark is set in the **low\_watermark** field. It is expected that the high watermark is programmed to a larger value than the low watermark. The CPU can opt to be interrupted when the descriptor count falls below either (or both) watermarks.

In a transmit engine setting the low watermark register to 1 and disabling the high watermark interrupt will give an interrupt when all queued transmissions are complete.

In a receive engine the interface can be configured to use automatic flow-control. If this is enabled then when the number of descriptors falls below the low watermark the interface will be signalled to use flow control to apply back pressure, which will only be removed when sufficient descriptors have been supplied to push the count above the high watermark. If the engine is configured to interrupt the CPU when the descriptor count falls below the high watermark, software has a chance to allocate more receive descriptors before flow control is asserted. If automatic flow control is enabled and the high watermark is smaller than the low watermark the behavior of the engine is UNPREDICTABLE. The watermark based flow control will not activate until after buffers have been added to the descriptor queue for the first time, so the link will not be jammed as the interface comes out of reset.

The watermark status, which is used to raise an interrupt, is continually updated. If the number of descriptors falls below a watermark just as the CPU is doing a write to the count register to increase the number of descriptors owned by the DMA engine, the interrupt line may be briefly asserted but all status bits will be clear when the CPU has entered the interrupt service routine and reads the status register. Software should be written to work correctly if these spurious interrupts occur (or only add buffers in the ISR or with the interrupts disabled).

If the receive DMA engine runs out of descriptors during a packet the tail of the packet is lost and the `dscr_err` bit is set in the receive status. In this case the status includes enough information for the software to compute the size of data actually transferred, but in most cases the packet will just be dropped. If the receive engine is out of descriptors when the start of a packet is received the whole packet is dropped and the interface will signal a packet dropped interrupt. In either case packet reception will resume with the start of the first packet received after the count has been written to make more descriptors available. The Ethernet receive DMA channels count the packets that are dropped because the channel is out of descriptors when the start of packet is received, on parts with system revision indicating PERIPH\_REV3 or greater this count can be read from the `dma_oodpktlost` register. Any write to this register (regardless of data) will zero the count.

## COMPLETION INTERRUPTS

The controller has a packet counter and will generate an interrupt after transferring a configurable number of packets. This is of most use in the receive channel. If the interrupt count is set to one then an interrupt will be raised after every packet. Since this can swamp the system with interrupts, the count would typically be set higher and the receive interrupt service routine will be written to accept a batch of packets. In order to avoid imposing a high delay before packets are serviced when they are arriving at a low rate, the interrupt can also be raised by a timer. The timer starts counting when the first packet reception is complete and will increment every 768 CPU clocks (i.e. the counter is clocked at  $\text{CPUclock}/(3 \times 256)$ ). If the interrupt has not been raised because the packet count threshold has been reached it will be forced when the timer has counted to a programmed limit. Reading the interface interrupt status register will clear both the packet counter and resets the timer, both will remain at zero until the next packet transfer is complete. The device driver must be written based on this behavior and should handle (or queue) any interrupts that are marked when the status is read.

The completion interrupts are also available for transmit interfaces. In this case the counter will increment when a packet transmission has completed, and the timer will start running when the first transmission has completed. This could be used to detect the transmitter being unable to send for an unacceptably long period.

The interrupt is cleared by reading the channel's interrupt status register, this disables the timer and zeros the received packet count preparing the system for the next batch of packets.

On parts with system revision indicating PERIPH\_REV3 or greater the current value of the received packet count can be read from the `dma_oodpktlost` register.

## EXPLICIT DESCRIPTOR INTERRUPTS

The descriptors include a flag to cause the controller to raise an interrupt when it has finished using the marked descriptor. The flag can be set on the last descriptor in a batch of packets that are queued for transmission, software will then receive notification when they have all been sent even if more packets have been queued.

## ASIC MODE TRANSFERS

The ethernet and serial interface receive DMA engines have a special "ASIC mode" that enables received packets to be passed through an ASIC on the HyperTransport fabric or PCI bus. This is in addition to just sending the packet by address to an I/O destination. It allows connection of a simple ASIC that is logically inserted into the path from the interface to memory.

The ASIC mode is intended for use in configurations where there is an assist ASIC on the HyperTransport or PCI through which some or all of a packet must be passed. Examples include header classification engines and encryption/decryption devices. Figure 29 shows an example packet flow.

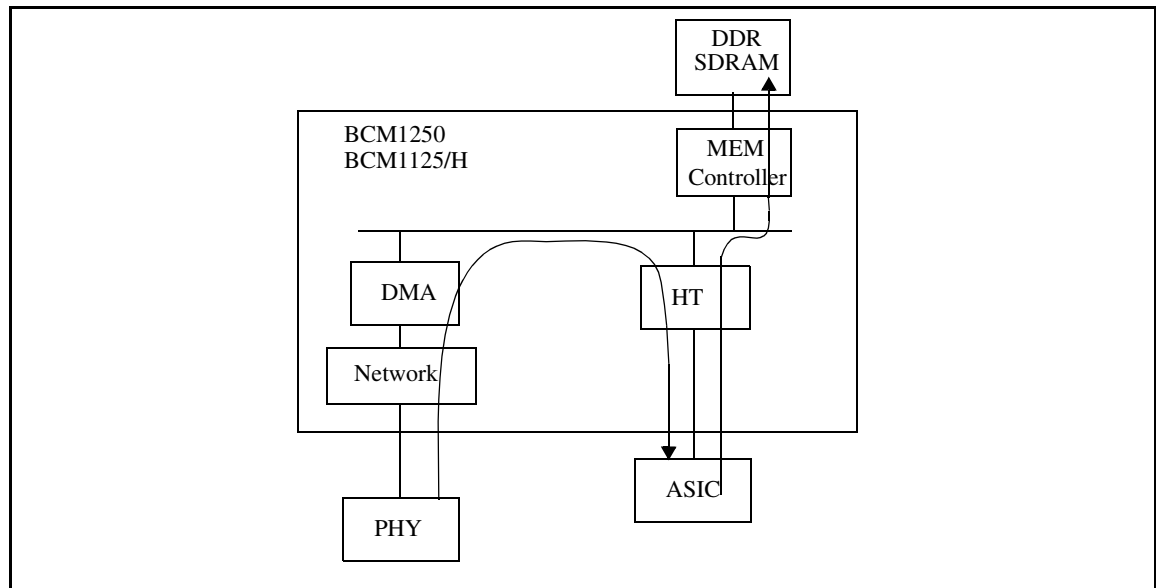
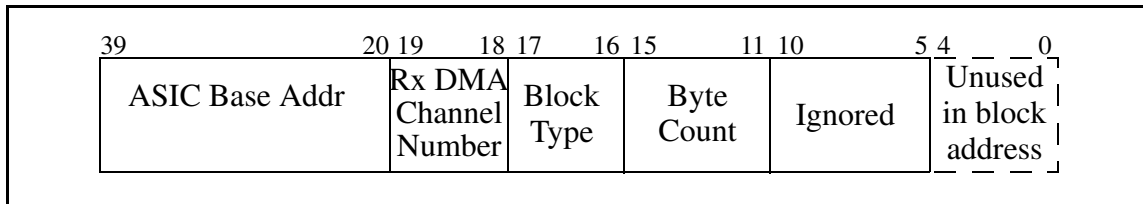


Figure 29: Packet Reception Flow using DMA ASIC Mode

ASIC mode is enabled by setting the `asic_xfr_en` bit in the `dma_config1` register. The normal DMA descriptors are fetched for each packet. Rather than sending the packet header to the first buffer in the descriptor, the packet is directed to the address range set in the `dma_asic_addr` register. The `asicxfr_size` is set to the number of cache lines of the packet that should be sent to the ASIC; if there is an offset specified this is one less than the maximum number of cache lines that should be sent to the ASIC address range (i.e. if the `asicxfr_size` is 0 and `asic_xfr_en` is set then one cache line will be sent to the ASIC). The size sent to the ASIC must match the size set for the first descriptor or behavior of the engine is UNDEFINED. If the packet length is shorter than the size specified by the `asicxfr_size` field then the entire packet is sent to the ASIC, otherwise after the configured size has been sent to the ASIC the DMA controller advances to the second buffer in the descriptor and will transfer the data to memory in the normal way. If the packet spans to subsequent descriptors they will be used to send the data to memory; data is only directed to the ASIC following a start of packet.

The `pre_addr_en` bit enables prepending of the `dscr_a` doubleword of the DMA descriptor to the data sent to the ASIC. If this bit is set then the offset in the DMA descriptor must be set to 8 (or greater than 8) and the `asicxfr_size` must be adjusted as outlined above. The descriptor information can be used by the ASIC to write the processed data back into memory at the address specified in the original DMA descriptor. Since the entire descriptor doubleword is sent the ASIC has access to the size of the buffer, and flags can be passed from the software to the ASIC by setting them up in the status field. (This field is not used by the DMA controller, so can contain any value when it reads the descriptor. However, the descriptor in memory will be updated by the DMA engine on completion of the packet reception, so any value passed from software to the ASIC in these bits will be overwritten). Note that the offset is not sent to the ASIC, these 5 bits are UNPREDICTABLE. The ASIC does not receive the packet status information, except for an error indication described below.

Cache blocks of the packet will be sent to the ASIC tagged by address according to [Figure 30](#) and [Table 90](#).



**Figure 30: ASIC Mode Address Generation**

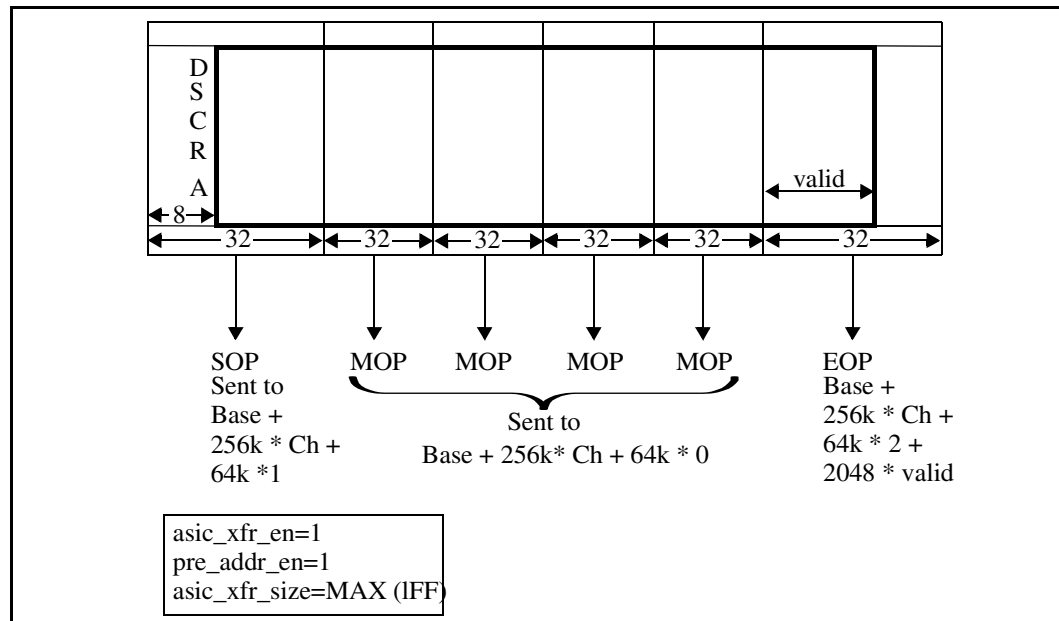
The `dma_asic_addr` specifies the base address for a 1 MB region of memory. The first cache block in a packet (with the descriptor prepended if enabled) will be sent to the Start Of Packet (SOP) address, subsequent blocks are sent to the Middle Of Packet (MOP) address range and the last cache block in the packet will be set to either the End Of Packet (EOP) address or the EOP/Error address. The block sent to the EOP address may contain less than a full cache block of valid data, the byte count indicates the number of valid bytes which will be packed towards the low memory address end of the block sent.

**Table 90: Address Used for ASIC Mode Transfers**

Bits	Name	Description
4:0		These bits are not present, since a full cache line is always transferred.
10:5	ignored	Ignore
15:11	byte_count	In EOP packets this is set to the number of valid bytes in the block. This field is ignored in SOP or MOP blocks. There can be from 1-32 valid bytes, the field is 0 to indicate 32 bytes valid.
17:16	type	Set to the position of the block in the packet. 00: Middle of packet (MOP). 01: Start of packet (SOP), the block includes the prepended descriptor if enabled. 10: End of packet with no errors (EOP), the byte_count field indicates the number of valid bytes in the cache block. 11: End of packet with errors (EOP/Error), the byte_count field indicates the number of valid bytes in the cache block.
19:18	channel	Set to the dma channel number.
39:20	base	Set to the base address from bits [39:20] of the <code>dma_asic_addr</code> register.



Figure 31 shows a full packet being sent using ASIC mode. In this case the `asicxfr_size` is set to its maximum value (which is greater than the maximum packet size) ensuring the complete packet is sent to the ASIC.

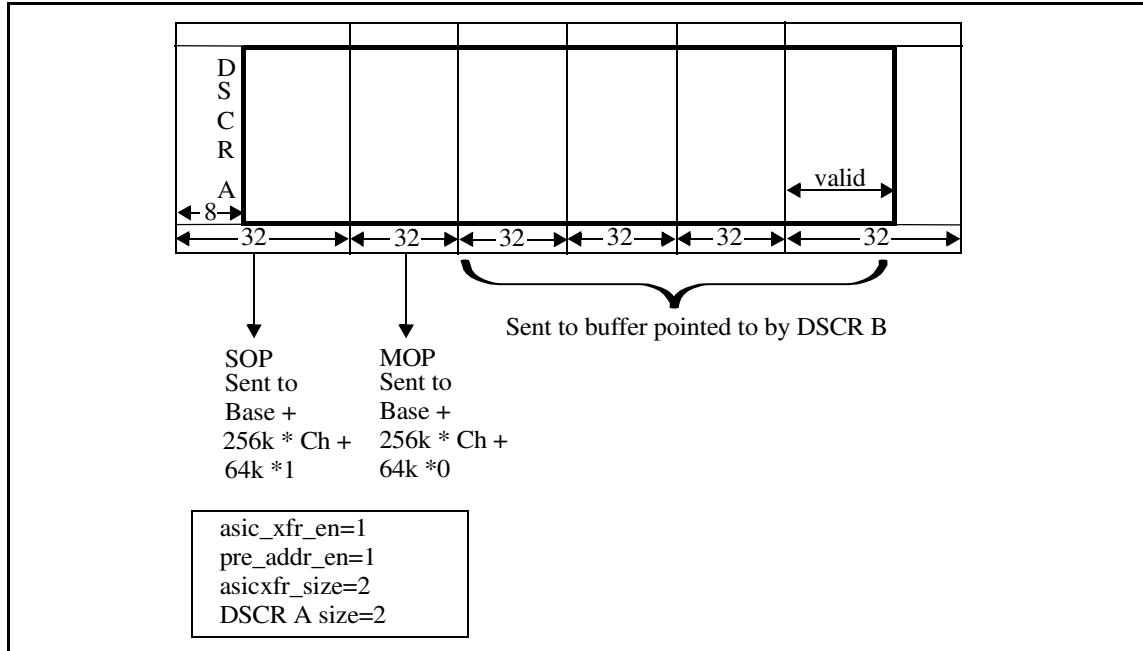


**Figure 31: Sending the Whole Packet in ASIC Mode**

The first cache block is sent to the SOP address, formed by adding the channel number bits and the 01 SOP code bits to the base address. This line includes the prepended first doubleword of the DMA descriptor (`dscr_a`) in the lowest 8 bytes. Subsequent blocks are sent to the MOP address until the last block. The final block is sent to the EOP address formed by adding the channel number bits, the 10 EOP code and the number of valid bytes to the base address. The valid byte count is needed because a full 32 byte block is always sent. Once the complete packet has been sent the DMA controller will write the length and status flags back into the descriptor in the normal way. The next descriptor will be fetched for the next packet (so the B buffer is never used). The ASIC can process the packet, and write data back to memory at the address given in the prepended `dscr_a`.

The ASIC needs a way to inform the CPU that its processing is complete. The DMA controller will generate the end of packet (or completion or watermark) interrupt when it has sent the full packet to the ASIC and has written the status back into the descriptor. There is a latency in the buffering between the interface and the ASIC, the ASIC will take time to process the packet and there will be latency writing the result back to memory, so any interrupts from the DMA controller are likely to be seen by the CPU before the ASIC has completed processing the packet. A simple way for the ASIC to do this is to send a HyperTransport interrupt message after writing the last data. The HyperTransport protocol, and host bridge will ensure that the ordering is maintained and the CPU does not receive the interrupt until the data of the write is visible in the coherency protocol.

Figure 32 illustrates the ASIC mode being used to pass only the header of a packet (for example for classification). The `asicxfr_size` is set to the number of header cache blocks (typically 2, causing 64 bytes of header to be sent to the ASIC with no prepended descriptor or 1 causing 56 bytes of header to be sent with the prepended descriptor). The `dscr_a` size must be set to the same value.



**Figure 32: Sending a Packet Header in ASIC Mode**

The SOP address is still used for the first block in the packet, and MOP for subsequent blocks (if the packet is only two blocks the second will be sent to the EOP address). Once the header size has been sent to the ASIC address the DMA controller advances to buffer B and sends the rest of the packet into memory in the normal way, when it is done it will write the length and flags back into the descriptor. The ASIC can perform its header processing and write results and the header back using the buffer A address. If CPU intervention is required an ASIC on the HyperTransport fabric could opt to write the results and header back with HyperTransport Isochronous writes, which causes the data to be allocated in the L2 cache. If the results are greater than 8 bytes (the offset that was specified to allow the `dscr_a` to be prepended) the actual buffer size may need to be greater than indicated in the descriptor, since the size given in the descriptor must match the size sent to the ASIC. The system software must manage this difference by allocating the larger buffer and adjusting the size in the descriptors.

## ETHERNET AND SERIAL DMA CONTROL REGISTERS

The BCM1250 has MACs 0, 1 and 2.

The BCM1125/H has MACs 0 and 1. Accesses made to the address range allocated to a nonexistent MAC may cause all MACs to exhibit UNPREDICTABLE behavior.

**Table 91: Ethernet and Serial DMA Configuration Register 0**

Bits	Name	Default	Description
0	drop	1'b0	Set to cause all packets to the channel to be dropped. MAC receive channels only, setting this bit in other channels causes UNDEFINED behavior.
2:1	dscr_type	2'b0	This field sets the descriptor format: 00 - Descriptor Ring, Standard Aligned buffer format. 01 - Descriptor Chain, Standard Aligned buffer format. 10 - Descriptor Ring, Unaligned buffer format, WriteInvalidate overwrites a full cache block at start/end of a buffer. 11 - Descriptor Ring, Unaligned buffer format, read-modify-write at start/end of unaligned buffer. Formats 10 and 11 are only supported by the Ethernet DMA engine and only if the system revision indicates PERIPH_REV3 or greater.
3	eop_int_en	1'b0	Set to enable interrupt at end of packet. The interrupt will be generated when the number of packets specified in int_pktcnt have been received, or if the receive interrupt timer has timed out. See <a href="#">Section: "Completion Interrupts" on page 158</a> .
4	hwm_int_en	1'b0	Set to enable an interrupt when the number of descriptors owned by the DMA controller falls below the high watermark.
5	lwm_int_en	1'b0	Set to enable an interrupt when the number of descriptors owned by the DMA controller falls below the low watermark. The receiver will assert flow control when the number of descriptors falls below this watermark.
6	tbx_en	1'b0	Set to cause the transmit DMA engine to fetch two 32 byte blocks at a time (increasing the open page hit rate in the SDRAM). Clear to only fetch one block at a time. The tx_wr_thrs threshold must be set to match the fetch size. MAC transmit channels only, setting this bit in other channels causes UNDEFINED behavior. If system revision indicates a peripheral revision earlier than PERIPH_REV3 then when this bit is set the buffer size should be a multiple of 64 bytes, if the system revision indicates PERIPH_REV3 or greater there is no restriction.
7	tdx_en	1'b0	Set in ring mode to allow the DMA engine to fetch two descriptors at a time. Clear if the engine should use 16 byte fetches to get a single descriptor. If this bit is set and there is only a single descriptor available then the invalid prefetched data is discarded. It is recommended that this bit is set in any high bandwidth DMA channels.
15:8	int_pktcnt	8'b0	This sets the number of packets that must be received before an end of packet interrupt is raised. See <a href="#">Section: "Completion Interrupts" on page 158</a> .
31:16	ringsz	16'b0	When the channel is operating in ring mode this sets the number of descriptors in the ring. It should only be changed when the channel is disabled. If this field is set to 0 the ring will contain 65536 descriptors.

**Table 91: Ethernet and Serial DMA Configuration Register 0 (Cont.)**

47:32	high_watermark	16'b0	This specifies the high watermark for generating an interrupt to the CPU based on the number of descriptors. If the MAC receiver is asserting flow control and the number of descriptors rises above this watermark it will remove the flow control.
63:48	low_watermark	16'b0	This specifies the low watermark for generating an interrupt to the CPU based on the number of descriptors. The MAC receiver will assert flow control when the number of descriptors falls below this watermark.

**Table 92: Ethernet and Serial DMA Configuration Register 1**

Bits	Name	Default	Description																
			<table border="0"> <tr> <td><code>dma_config1_mac_0_rx_ch_0 - 00_1006_4808</code></td> <td><code>dma_config1_mac_0_tx_ch_0 - 00_1006_4C08</code></td> </tr> <tr> <td><code>dma_config1_mac_0_rx_ch_1 - 00_1006_4908</code></td> <td><code>dma_config1_mac_0_tx_ch_1 - 00_1006_4D08</code></td> </tr> <tr> <td><code>dma_config1_mac_1_rx_ch_0 - 00_1006_5808</code></td> <td><code>dma_config1_mac_1_tx_ch_0 - 00_1006_5C08</code></td> </tr> <tr> <td><code>dma_config1_mac_1_rx_ch_1 - 00_1006_5908</code></td> <td><code>dma_config1_mac_1_tx_ch_1 - 00_1006_5D08</code></td> </tr> <tr> <td><code>dma_config1_mac_2_rx_ch_0 - 00_1006_6808</code></td> <td><code>dma_config1_mac_2_tx_ch_0 - 00_1006_6C08</code></td> </tr> <tr> <td><code>dma_config1_mac_2_rx_ch_1 - 00_1006_6908</code></td> <td><code>dma_config1_mac_2_tx_ch_1 - 00_1006_6D08</code></td> </tr> <tr> <td><code>dma_config1_ser_0_rx - 00_1006_0408</code></td> <td><code>dma_config1_ser_0_tx - 00_1006_0488</code></td> </tr> <tr> <td><code>dma_config1_ser_1_rx - 00_1006_0808</code></td> <td><code>dma_config1_ser_1_tx - 00_1006_0888</code></td> </tr> </table>	<code>dma_config1_mac_0_rx_ch_0 - 00_1006_4808</code>	<code>dma_config1_mac_0_tx_ch_0 - 00_1006_4C08</code>	<code>dma_config1_mac_0_rx_ch_1 - 00_1006_4908</code>	<code>dma_config1_mac_0_tx_ch_1 - 00_1006_4D08</code>	<code>dma_config1_mac_1_rx_ch_0 - 00_1006_5808</code>	<code>dma_config1_mac_1_tx_ch_0 - 00_1006_5C08</code>	<code>dma_config1_mac_1_rx_ch_1 - 00_1006_5908</code>	<code>dma_config1_mac_1_tx_ch_1 - 00_1006_5D08</code>	<code>dma_config1_mac_2_rx_ch_0 - 00_1006_6808</code>	<code>dma_config1_mac_2_tx_ch_0 - 00_1006_6C08</code>	<code>dma_config1_mac_2_rx_ch_1 - 00_1006_6908</code>	<code>dma_config1_mac_2_tx_ch_1 - 00_1006_6D08</code>	<code>dma_config1_ser_0_rx - 00_1006_0408</code>	<code>dma_config1_ser_0_tx - 00_1006_0488</code>	<code>dma_config1_ser_1_rx - 00_1006_0808</code>	<code>dma_config1_ser_1_tx - 00_1006_0888</code>
<code>dma_config1_mac_0_rx_ch_0 - 00_1006_4808</code>	<code>dma_config1_mac_0_tx_ch_0 - 00_1006_4C08</code>																		
<code>dma_config1_mac_0_rx_ch_1 - 00_1006_4908</code>	<code>dma_config1_mac_0_tx_ch_1 - 00_1006_4D08</code>																		
<code>dma_config1_mac_1_rx_ch_0 - 00_1006_5808</code>	<code>dma_config1_mac_1_tx_ch_0 - 00_1006_5C08</code>																		
<code>dma_config1_mac_1_rx_ch_1 - 00_1006_5908</code>	<code>dma_config1_mac_1_tx_ch_1 - 00_1006_5D08</code>																		
<code>dma_config1_mac_2_rx_ch_0 - 00_1006_6808</code>	<code>dma_config1_mac_2_tx_ch_0 - 00_1006_6C08</code>																		
<code>dma_config1_mac_2_rx_ch_1 - 00_1006_6908</code>	<code>dma_config1_mac_2_tx_ch_1 - 00_1006_6D08</code>																		
<code>dma_config1_ser_0_rx - 00_1006_0408</code>	<code>dma_config1_ser_0_tx - 00_1006_0488</code>																		
<code>dma_config1_ser_1_rx - 00_1006_0808</code>	<code>dma_config1_ser_1_tx - 00_1006_0888</code>																		
0	hdr_cf_en	1'b0	If this bit is set then the L2_cacheable bit will be asserted during header transfers to cause the L2 cache to be allocated.																
1	asic_xfr_en	1'b0	Set to enable special support for transferring packets to an ASIC on the HyperTransport fabric (or PCI bus).																
2	pre_addr_en	1'b0	Set when asic_xfr_en is set to enable prepending of the DMA descriptor to the packet sent to the ASIC.																
3	flow_ctl_en	1'b0	Set to cause the controller to send a flow control request to the interface when the descriptor count falls below the low watermark, and only remove the request when the count goes above the high watermark. MAC receive channels only, setting this bit in other channels causes UNDEFINED behavior.																
4	no_dscr_updt	1'b0	Set to prevent the descriptor being written with the status at the end of a packet transfer, saving bandwidth through the bridge. (Transmit channels only.)																
5	dscr_l2ca	1'b0	This bit sets the L2 cacheability for descriptors. If it is set the L2CA bit will be set in descriptor requests, causing them to be cached in the L2 cache. If clear the descriptors will not be allocated in the L2 on a miss.																
6	(rx) xtra_status  (tx) cpu_pause_en	1'b0	Ethernet Receive channels only: This bit should only be set for the Ethernet receive engine and only if the system revision indicates PERIPH_REV3 or greater. If this bit is set the a_size field will be overwritten with additional status information in the start of packet descriptor.  Ethernet Transmit channels only: This bit should only be set for the Ethernet transmit engine and only if the system revision indicates PERIPH_REV3 or greater. If this bit is set the channel will pause at the end of the current packet and will resume only when the bit is cleared.																
7	fc_pause_en	1'b0	Ethernet Transmit channels only: This bit should only be set for the Ethernet transmit engine and only if the system revision indicates PERIPH_REV3 or greater. If this bit is set and the interface is set for DMA channel based flow control (ch_base_fc_en bit set in the mac_vlantag register) then this channel will be paused by a flow control frame, if this bit is clear and the interface is set for DMA channel based flow control the channel will not be paused by a flow control frame. If the interface is not configured for DMA channel based flow control this bit is ignored.																
15:8	reserved	8'b0	Reserved																





Table 92: Ethernet and Serial DMA Configuration Register 1 (Cont.)

Bits	Name	Default	Description
20:16	zero	5'b0	These bits must be zero.
29:21	hdr_size	9'b0	This sets the header length in cache blocks. It sets the number of cache blocks for which the L2 cacheable attribute will be attached if <code>hdr_cf_en</code> is asserted. This count is taken from the start of the buffer (the first cache line contains only 32 - offset bytes from the packet).
31:30	reserved	2'b0	Reserved
36:32	zero	5'b0	These bits must be zero.
45:37	asicxfr_size	9'b0	This field sets the size (in cache lines) of the portion of the packet that will be sent to the ASIC if <code>asic_xfr_en</code> is set. Packets shorter than this size will be sent in their entirety using the EOP protocol described in <a href="#">Section: "ASIC Mode Transfers" on page 159</a> to signal the packet end. Care must be taken to adjust the number set here when an offset is used (which must be the case if the <code>pre_addr_en</code> bit is set). The number of cache lines sent to the ASIC will be: Offset zero: The number in this field. (If this field is zero then one cache line is transferred.) Offset nonzero: One greater than the number in this field.
47:46	reserved	2'b0	Reserved
63:48	int_timeout	16'b0	This field sets the timeout for interrupt generation. See <a href="#">Section: "Completion Interrupts" on page 158</a> . If this field is zero then timing of the assertion of the <code>eop_timer</code> interrupt flag is UNPREDICTABLE.

**Table 93: Ethernet and Serial DMA Descriptor Base Address Register**

dma_dscr_base_mac_0_rx_ch_0 - 00_1006_4810    dma_dscr_base_mac_0_tx_ch_0 - 00_1006_4C10 dma_dscr_base_mac_0_rx_ch_1 - 00_1006_4910    dma_dscr_base_mac_0_tx_ch_1 - 00_1006_4D10 dma_dscr_base_mac_1_rx_ch_0 - 00_1006_5810    dma_dscr_base_mac_1_tx_ch_0 - 00_1006_5C10 dma_dscr_base_mac_1_rx_ch_1 - 00_1006_5910    dma_dscr_base_mac_1_tx_ch_1 - 00_1006_5D10 dma_dscr_base_mac_2_rx_ch_0 - 00_1006_6810    dma_dscr_base_mac_2_tx_ch_0 - 00_1006_6C10 dma_dscr_base_mac_2_rx_ch_1 - 00_1006_6910    dma_dscr_base_mac_2_tx_ch_1 - 00_1006_6D10 dma_dscr_base_ser_0_rx - 00_1006_0410    dma_dscr_base_ser_0_tx - 00_1006_0490 dma_dscr_base_ser_1_rx - 00_1006_0810    dma_dscr_base_ser_1_tx - 00_1006_0890			
Bits	Name	Default	Description
3:0	zero	4'b0	These bits must be zero.
39:4	base	36'b0	This is the base address of the descriptor ring, or the pointer to the first descriptor in a chain. This register must only be changed when the channel is disabled. When a channel is disabled and enabled again it will start fetching descriptors from this address.
64:40	reserved	24'b0	Reserved

**Table 94: ASIC Mode Base Address**

dma_asic_addr_mac_0 - 00_1006_4418 dma_asic_addr_mac_1 - 00_1006_5418 dma_asic_addr_mac_2 - 00_1006_6418 dma_asic_addr_ser_0 - 00_1006_0598 dma_asic_addr_ser_1 - 00_1006_0998			
Bits	Name	Default	Description
19:0	zero	20'b0	These bits must be zero.
39:20	base	36'b0	This is the base of the ASIC address space.
64:40	reserved	24'b0	Reserved

**Table 95: Descriptor Count Register**

dma_dscr_cnt_mac_0_rx_ch_0 - 00_1006_4818    dma_dscr_cnt_mac_0_tx_ch_0 - 00_1006_4C18 dma_dscr_cnt_mac_0_rx_ch_1 - 00_1006_4918    dma_dscr_cnt_mac_0_tx_ch_1 - 00_1006_4D18 dma_dscr_cnt_mac_1_rx_ch_0 - 00_1006_5818    dma_dscr_cnt_mac_1_tx_ch_0 - 00_1006_5C18 dma_dscr_cnt_mac_1_rx_ch_1 - 00_1006_5918    dma_dscr_cnt_mac_1_tx_ch_1 - 00_1006_5D18 dma_dscr_cnt_mac_2_rx_ch_0 - 00_1006_6818    dma_dscr_cnt_mac_2_tx_ch_0 - 00_1006_6C18 dma_dscr_cnt_mac_2_rx_ch_1 - 00_1006_6918    dma_dscr_cnt_mac_2_tx_ch_1 - 00_1006_6D18 dma_dscr_cnt_ser_0_rx - 00_1006_0418    dma_dscr_cnt_ser_0_tx - 00_1006_0498 dma_dscr_cnt_ser_1_rx - 00_1006_0818    dma_dscr_cnt_ser_1_tx - 00_1006_0898			
Bits	Name	Default	Description
15:0	count	16'b0	This is the number of descriptors owned by the DMA engine. Reads will return the number of descriptors owned (Count in <a href="#">Figure 26 on page 151</a> and <a href="#">Figure 27 on page 153</a> ). Data that is written to this register will be added to the count (assigning that number of additional descriptors to the controller). The count is a 16 bit unsigned number, there is no overflow protection. Software should ensure it does not unintentionally cause the count to wrap.
64:16	reserved	48'b0	Reserved

**Table 96: Current Descriptor A Debug Register**

<pre> dma_cur_dscr_a_mac_0_rx_ch_0-00_1006_4820 dma_cur_dscr_a_mac_0_tx_ch_0-00_1006_4C20 dma_cur_dscr_a_mac_0_rx_ch_1-00_1006_4920 dma_cur_dscr_a_mac_0_tx_ch_1-00_1006_4D20 dma_cur_dscr_a_mac_1_rx_ch_0-00_1006_5820 dma_cur_dscr_a_mac_1_tx_ch_0-00_1006_5C20 dma_cur_dscr_a_mac_1_rx_ch_1-00_1006_5920 dma_cur_dscr_a_mac_1_tx_ch_1-00_1006_5D20 dma_cur_dscr_a_mac_2_rx_ch_0-00_1006_6820 dma_cur_dscr_a_mac_2_tx_ch_0-00_1006_6C20 dma_cur_dscr_a_mac_2_rx_ch_1-00_1006_6920 dma_cur_dscr_a_mac_2_tx_ch_1-00_1006_6D20 dma_cur_dscr_a_ser_0_rx - 00_1006_0420 dma_cur_dscr_a_ser_0_tx - 00_1006_04A0 dma_cur_dscr_a_ser_1_rx - 00_1006_0820 dma_cur_dscr_a_ser_1_tx - 00_1006_08A0 </pre>			
<b>READ ONLY</b>			
Bits	Name	Default	Description
63:0	cur_a	64'b0	The current descriptor first double word can be read from this register (intended for debugging only).

**Table 97: Current Descriptor B Debug Register**

<pre> dma_cur_dscr_b_mac_0_rx_ch_0-00_1006_4828 dma_cur_dscr_b_mac_0_tx_ch_0-00_1006_4C28 dma_cur_dscr_b_mac_0_rx_ch_1-00_1006_4928 dma_cur_dscr_b_mac_0_tx_ch_1-00_1006_4D28 dma_cur_dscr_b_mac_1_rx_ch_0-00_1006_5828 dma_cur_dscr_b_mac_1_tx_ch_0-00_1006_5C28 dma_cur_dscr_b_mac_1_rx_ch_1-00_1006_5928 dma_cur_dscr_b_mac_1_tx_ch_1-00_1006_5D28 dma_cur_dscr_b_mac_2_rx_ch_0-00_1006_6828 dma_cur_dscr_b_mac_2_tx_ch_0-00_1006_6C28 dma_cur_dscr_b_mac_2_rx_ch_1-00_1006_6928 dma_cur_dscr_b_mac_2_tx_ch_1-00_1006_6D28 dma_cur_dscr_b_ser_0_rx - 00_1006_0428 dma_cur_dscr_b_ser_0_tx - 00_1006_04A8 dma_cur_dscr_b_ser_1_rx - 00_1006_0828 dma_cur_dscr_b_ser_1_tx - 00_1006_08A8 </pre>			
<b>READ ONLY</b>			
Bits	Name	Default	Description
63:0	cur_b	64'b0	The current descriptor second double word can be read from this register (intended for debugging only).

**Table 98: Current Descriptor Address Register**

<pre> dma_cur_daddr_mac_0_rx_ch_0 - 00_1006_4830 dma_cur_daddr_mac_0_tx_ch_0 - 00_1006_4C30 dma_cur_daddr_mac_0_rx_ch_1 - 00_1006_4930 dma_cur_daddr_mac_0_tx_ch_1 - 00_1006_4D30 dma_cur_daddr_mac_1_rx_ch_0 - 00_1006_5830 dma_cur_daddr_mac_1_tx_ch_0 - 00_1006_5C30 dma_cur_daddr_mac_1_rx_ch_1 - 00_1006_5930 dma_cur_daddr_mac_1_tx_ch_1 - 00_1006_5D30 dma_cur_daddr_mac_2_rx_ch_0 - 00_1006_6830 dma_cur_daddr_mac_2_tx_ch_0 - 00_1006_6C30 dma_cur_daddr_mac_2_rx_ch_1 - 00_1006_6930 dma_cur_daddr_mac_2_tx_ch_1 - 00_1006_6D30 dma_cur_daddr_ser_0_rx - 00_1006_0430 dma_cur_daddr_ser_0_tx - 00_1006_04B0 dma_cur_daddr_ser_1_rx - 00_1006_0830 dma_cur_daddr_ser_1_tx - 00_1006_08B0 </pre>			
<b>READ ONLY</b>			
Bits	Name	Default	Description
39:0	dscr_addr	40'b0	The current descriptor address can be read from this field. If count is nonzero the address is the descriptor being used. If count is zero the address is where the next descriptor will be fetched from.
55:40	count	16'b0	The current count of descriptors owned by the DMA engine can be read from this field. (It provides the same information as reading the <b>dma_dscr_count</b> register).

**Table 98: Current Descriptor Address Register (Cont.)**

dma_cur_daddr_mac_0_rx_ch_0 - 00_1006_4830 dma_cur_daddr_mac_0_rx_ch_1 - 00_1006_4930 dma_cur_daddr_mac_1_rx_ch_0 - 00_1006_5830 dma_cur_daddr_mac_1_rx_ch_1 - 00_1006_5930 dma_cur_daddr_mac_2_rx_ch_0 - 00_1006_6830 dma_cur_daddr_mac_2_rx_ch_1 - 00_1006_6930 dma_cur_daddr_ser_0_rx - 00_1006_0430 dma_cur_daddr_ser_1_rx - 00_1006_0830		dma_cur_daddr_mac_0_tx_ch_0 - 00_1006_4C30 dma_cur_daddr_mac_0_tx_ch_1 - 00_1006_4D30 dma_cur_daddr_mac_1_tx_ch_0 - 00_1006_5C30 dma_cur_daddr_mac_1_tx_ch_1 - 00_1006_5D30 dma_cur_daddr_mac_2_tx_ch_0 - 00_1006_6C30 dma_cur_daddr_mac_2_tx_ch_1 - 00_1006_6D30 dma_cur_daddr_ser_0_tx - 00_1006_04B0 dma_cur_daddr_ser_1_tx - 00_1006_08B0 <b>READ ONLY</b>	
Bits	Name	Default	Description
56	ch_pause_on	1'b0	Only valid if System revision indicates PERIPH_REV3 or greater and only valid for Ethernet transmit channels. This bit is set if the channel has paused because the cpu_pause_en bit is set in the <b>dma_config1</b> register or a flow control frame has been received the fc_pause_en bit is set in the <b>dma_config1</b> register and the ch_base_fc_en bit is set in the <b>mac_vlan_tag</b> register. Note that this bit is set when the channel is actually pausing, so when the cpu_pause_en bit is set the ch_pause_en will remain clear until any packet in progress has been sent and the channel is stopped between packets.
63:57	reserved	7'b0	Reserved

**Table 99: Ethernet Receive Packet Drop Registers (Only if System Revision >= PERIPH\_REV3)**

dma_oodpktlost_mac_0_rx_ch_0 - 00_1006_4838 dma_oodpktlost_mac_0_rx_ch_1 - 00_1006_4938 dma_oodpktlost_mac_1_rx_ch_0 - 00_1006_5838 dma_oodpktlost_mac_1_rx_ch_1 - 00_1006_5938 dma_oodpktlost_mac_2_rx_ch_0 - 00_1006_6838 dma_oodpktlost_mac_2_rx_ch_1 - 00_1006_6938			
Bits	Name	Default	Description
15:0	oodlost	16'b0	This is the count of packets that were dropped by the channel because it was out of descriptors when the start of the packet was received. (The count will not be incremented if the channel has descriptors available but is disabled, or if the channel has descriptors but the drop bit is set.) The counter will stick at 16'hFFFF to indicate 65535 or more packets dropped.
23:16	eop_count	8'b0	This field contains the current value of the counter that controls the completion interrupt.
63:24	reserved	40'bx	This field is reserved. Reads will return UNPREDICTABLE data.



## ETHERNET AND SERIAL DMA DESCRIPTORS

**Table 100: DMA Descriptor First Doubleword**

dscr_a		
Bits	Name	Description
4:0	offset	The offset in the buffer that the header should start at.
39:5	a_addr	The base address of the data buffer (cache block aligned). In ring mode this is the base address of the first data buffer.
48:40	a_size	The size of the data buffer (in cache blocks). If this field is zero the buffer is 512 cache blocks. In a transmit channel which has the <code>tbx_en</code> configuration bit set bit [40] must be zero.
49	interrupt	If this bit is set an interrupt will be raised when the DMA engine has finished using this descriptor.
50	offset_b	If this bit is set then the offset field will be applied to the first buffer pointed to by descriptor B in a packet rather than the first buffer pointed to by descriptor A. The DMA controller behavior is UNPREDICTABLE if this bit is set in chain mode. If this bit is set, the <code>b_valid</code> bit must be set in the <code>dscr_b</code> or the behavior of the DMA engine will be UNPREDICTABLE.
63:51	status	In the first descriptor of a received packet this field will be updated with the packet status at the end of reception. The flags differ between the interfaces and are described in <a href="#">Section: "Option and Flag Bits for Ethernet MACs" on page 171</a> and <a href="#">Section: "Control and Flag Bits for Synchronous Serial Interface" on page 174</a> .

**Table 101: DMA Descriptor Second Doubleword**

dscr_b		
Bits	Name	Description
3:0	options	This field sets the per packet options that are sent to the interface. The options depend on the channel, and are outlined below.
4	addr	In chain mode this bit provides the additional address bit that is used with bits 39:5 to form the pointer to the next descriptor. In ring mode the bit is reserved and should be zero.
39:5	b_addr	In ring mode this gives the base address of the second data buffer (cache block aligned). In chain mode it and bit 4 contain a pointer to the next descriptor.
48:40	b_size	The size of the second data buffer (in cache blocks). If this field is zero the buffer is 512 cache blocks. In a transmit channel which has the <code>tbx_en</code> configuration bit set bit [40] must be zero.
49	b_valid	If this bit is set the buffer is valid and will be used. If this bit is clear the B buffer is not used and the <code>b_addr</code> and <code>b_size</code> fields are ignored. (Ring mode only.)
63:50	pkt_size	In the first descriptor of a packet this field contains the packet length. In a transmit channel the length is written by the CPU as part of setting up the descriptor and is passed to the interface. In a receive channel the length is written back into the descriptor at the end of reception.

**Table 102: Unaligned Buffer Format DMA Descriptor First Doubleword**

dscr_a		
Bits	Name	Description
39:0	addr	The base address of the data buffer.
47:40	unused	Unused. If extra_status is enabled the receive DMA engine will overwrite this field.
48	reserved	Reserved.
49	interrupt	If this bit is set an interrupt will be raised when the DMA engine has finished using this descriptor.
50	reserved	Reserved.
63:51	status	In the first descriptor of a received packet this field will be updated with the packet status at the end of reception. The flags are described in <a href="#">Section: "Option and Flag Bits for Ethernet MACs" on page 171</a> .

**Table 103: Unaligned Buffer Format DMA Descriptor Second Doubleword**

dscr_b		
Bits	Name	Description
3:0	options	This field sets the per packet options that are sent to the interface. The options depend on the channel, and are outlined below.
7:4	reserved	Reserved.
21:8	size	This field specifies the buffer size in bytes. This field must be set to the number of bytes of data in the buffer plus bits [4:0] of the start address in the dscr_a addr field. The DMA engine behavior is UNPREDICTABLE if this size is set to be less than the value of bits [4:0] of the address in dscr_a.
47:22	reserved	Reserved.
49:48	pkt_size_msb	This field provides bits [15:14] of the packet size. (Note that if packets greater than 16K-1 are used only the low 14 bits of the length is added to the RMON counters.)
63:50	pkt_size	In the first descriptor of a packet this field contains the packet length. In a transmit channel the length is written by the CPU as part of setting up the descriptor and is passed to the interface. In a receive channel the length is written back into the descriptor at the end of reception. In unaligned buffer mode this field is extended to 16 bits using the pkt_size_msb field as the upper two bits.



## OPTION AND FLAG BITS FOR ETHERNET MACS

**Table 104: Status Flags for Ethernet Receive Channel**

Bits	Name	Description
<b>Flags Written to dscr_a size_a field if extra_status is enabled</b>		
47:40	dscr_cnt	Number of descriptors used by this packet. If more than 255 descriptors were used to receive the packet then this field will be 0 and software must compute the actual number. This field is valid for Ethernet and Packet Fifo modes.
48	reserved	This bit may be used for status information in the future.
<b>Flags Written to dscr_a options field</b>		
51	bad_ip4cs	This bit is clear if at the offset specified in the iphdr_offset field of the <b>mac_adfilter_cfg</b> register the receive packet has a standard length (no options) IPv4 header with a valid header checksum. Otherwise it is set. This flag is only valid in Ethernet mode.
52	dscr_err	Descriptor error. During packet reception the channel ran out of descriptors, the tail of the packet is lost. When this bit is set, the length field can be used to determine the number of bytes transferred before the receiver ran out of descriptors. If the system_revision indicates PERIPH_REV3 or later, the length field indicates the number of bytes transferred. In older revisions the field must be corrected to account for the offset at the start of the packet. Software should compensate by subtracting the low three bits of the offset field: $\text{ActualBytesTransferred} = \text{dscr\_b.pkt\_size} - 32 - (\text{dscr\_a.offset} \& 7)$ This flag is valid for Ethernet and Packet Fifo modes.
54:53	rx_ch	Receive channel number (always 2'b00 for DMA channel zero) This field is valid for Ethernet and Packet Fifo modes.
57:55	pkt_type	Packet type. Contains an encoded version of the Ethernet Type field. See <a href="#">Section: "Packet Type Identification" on page 281</a> . This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)
58	match_hash	This bit is set if the packet passed the hash match filter. This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)
59	match_exact	This bit is set if the received packet passed the exact addresses filter. (The packet will pass if it matched an address and the filter was true, or the packet did not match and the filter was inverted). This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)
60	bcast	This bit is set if the received packet has a broadcast address. This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)
61	mcast	This bit is set if the received packet has a multicast address. This field is valid for Ethernet and Packet Fifo modes. (It depends on the packet format so it may not be useful in Packet Fifo mode.)
62	bad	This bit is set if the received packet has errors (length error or dribble error or code error or CRC error or is a runt packet or is an oversize packet). See <a href="#">Table 162 on page 275</a> for a description of the errors. In Packet Fifo mode, the error bit will be set by a CRC error or the packet is terminated with a link Error (only possible in GMII style and Encoded modes).

**Table 104: Status Flags for Ethernet Receive Channel (Cont.)**

Bits	Name	Description
63	SOP	This bit is set to indicate the start of the packet. Software should ensure this bit is clear when it sets up the descriptor, the DMA controller will only set it when packet reception has been completed.
<b>Flags Written to dscr_b options field</b>		
0	bad_tcpcs	The value of this bit is only valid if the interface is in Ethernet mode and the bad_ip4cs bit is clear. When valid, this bit is clear if the packet received has a correct TCP checksum, and set if the packet has a bad TCP checksum. Since UDP uses the same checksum algorithm this flag also applies to UDP packets.
1	vlan_flag	This bit is only valid if the system revision indicates PERIPH_REV3 or greater and for Ethernet operation and the vlan_det_en bit is set in the <b>mac_adfilter_cfg</b> register. This bit will be set when a packet with a VLAN tag is received and clear if there is no VLAN tag.
2	crc_flag	This bit is only valid if the system revision indicates PERIPH_REV3 or greater. Ethernet mode only: this flag is set (in addition to the bad bit) when a CRC error is detected.
3	reserved	This bit may be used for status information in the future.

**Table 105: Option Flags for Ethernet Receive Channel**

Receive Options		
Bits	Name	Description
None		

**Table 106: Status Flags for Ethernet Transmit Channel**

Transmit Status Flags		
Bits	Name	Description
62:51	ignored	Preserved when the descriptor is written.
63	SOP	This bit must be set for the first descriptor in a packet. The DMA engine will report a descriptor error if it is starting a new packet and this bit is clear in the corresponding descriptor. This bit is cleared when the descriptor is written.





**Table 107: Option Flags for Ethernet Transmit Channel**

Transmit Commands		
Bits	Name	Description
3:0	pkt_mod	<p>This field indicates how the transmitter should modify the packet.</p> <p>0000: Descriptor is not start of packet.  0001: Append CRC.  0010: Replace CRC.  0011: Append CRC and pad.  0100: Append VLAN tag and replace CRC.  0101: Remove VLAN tag and replace CRC.  0110: Replace VLAN tag and replace CRC.  0111: No Modifications.  1000: Reserved  1001: Replace source address and append CRC.  1010: Replace source address and replace CRC.  1011: Replace source address, append CRC and pad.  1100: Replace source address, append VLAN tag and replace CRC.  1101: Replace source address, remove VLAN tag and replace CRC.  1110: Replace source address, replace VLAN tag and replace CRC.  1111: Reserved</p> <p>Descriptors with the SOP bit set in their transmit status flags must have a nonzero value in this field. If the SOP bit is clear this field must be zero.</p> <p>In packet FIFO modes all codes are reserved except for:</p> <p>0000: Descriptor is not start of packet.  0001: Append CRC.  0111: No Modifications.</p>

**CONTROL AND FLAG BITS FOR SYNCHRONOUS SERIAL INTERFACE**

**Table 108: Status Flags for Synchronous Serial Receive Channel**

Receive Status Flags		
Bits	Name	Description
55:51	reserved	Reserved
56	crc_error	This bit is set if the received packet has a bad CRC.
57	abort	This bit is set if the received packet ended with the abort flag.
58	octet_error	This bit is set if the size of the received packet was not a multiple of 8 bits.
59	longframe_error	This bit is set if the size of the received packet is bigger than the maximum frame size.
60	shortframe_error	This bit is set if the size of the received packet is shorter than the minimum frame size.
61	overrun_error	This bit is set if the received packet over ran the FIFO and is therefore invalid.
62	good	This bit is set if the received packet has no errors.
63	SOP	This bit is set to indicate the start of the packet. Software should ensure this bit is clear when it sets up the descriptor, the DMA controller will only set it when packet reception has been completed.

**Table 109: Option Flags for Synchronous Serial Receive Channel**

Receive Options		
Bits	Name	Description
		None

**Table 110: Status Flags for Synchronous Serial Transmit Channel**

Transmit Status Flags		
Bits	Name	Description
62:51	ignored	Preserved when the descriptor is written.
63	Flag	This bit is cleared when the descriptor is written. Software can set this bit in descriptors associated with the start of a packet and the bit will be cleared when the packet has been transmitted. (Unlike the Ethernet there is no requirement that this bit be set for a SOP).

**Table 111: Option Flags for Synchronous Serial Transmit Channel**

Transmit Commands		
Bits	Name	Description
0	reserved	Reserved
1	appendCRC	If this bit is set the computed CRC will be appended to the packet.
2	append PAD	If this bit is set the packet will be padded to the minimum packet size.
3	abort	If this bit is set the packet will be ended with an abort instead of a standard flag.



---

This Page is left blank for notes



## DATA MOVER

The Data Mover can perform transfers between arbitrary addresses. The source for a transfer can be either memory or an I/O device, and the destination of the transfer can be either memory or an I/O device. The alignment of the source and destination can differ, in which case the engine will participate in the MESI coherence protocol to ensure the correct merge is done at the start and end of the transfer. The Data Mover has four channels (in many cases this is sufficient for them to be allocated to processors/processes to avoid the need for locking accesses to the control registers). When multiple channels have requests a modified round robin service schedule is applied, this provides fair access to all the channels while optimizing requests to memory to allow open-bank page mode access. The buffering in the Data Mover is sufficient that it can run at the full memory bandwidth.

### DATA MOVER OPERATION

The Data Mover channels can only be configured for ring mode. The descriptor A pointer points to the destination of the transfer and the descriptor B pointer points to the source of the transfer. These addresses can have any alignment. There is no buffer size, just the length parameter which may specify up to 1 MByte of data. The Data Mover reads and writes in 32 byte blocks as much as possible, either using the full coherence protocol or as uncached but merged data (similar to the CPU uncached accelerated). If the full coherence protocol is used and a smaller block needs to be written the Data Mover will read the block exclusive, merge in the new data and write the block back. If uncached accelerated mode is used then the byte enable signals will indicate the valid data and the Data Mover does not do a merge.

The source and destination addresses can be incremented, decremented or held constant for each 32 byte block. This allows movement of overlapping regions; if the source address is less than the destination the data should be moved from the top end of the buffer with both addresses decrementing, if the source address is greater then the data should be moved starting at the bottom of the region and incrementing the addresses. Holding the address constant is useful for transfers to or from I/O devices.

If the source and destination move in different directions the Data Mover will not reorder the bytes within the 32 byte line, and the addresses must be cache aligned. In this case if A[4:0] of either address is nonzero the behavior of the move is UNPREDICTABLE.

There is a level 2 cache flag associated with both the source and destination addresses. This is used to request the data be allocated in the L2 cache on a miss. In most cases it would only be used on the destination side (if at all) since there is little point in caching data that is being moved.

One case where the L2 cacheability attribute would be set on the source is when the prefetch bit is set. In this case the destination is ignored and only the reads are done. If the reads are done with the L2 allocation bit set this will prefetch the data into the L2 cache. In uncacheable I/O space this mode could be used to flush an external FIFO.

There is also a zero\_mem bit. If this is set the source address is ignored and destination is written with zeros.

If the data mover is used to perform accesses to the HyperTransport space it can only use a subset of reads. For any access the HyperTransport interface will only accept reads that are less than 8 bytes or are aligned to a 4 byte boundary and a multiple of 4 bytes (up to the full 32 bytes in the block), any other sizes will result in UNDEFINED behavior. Thus a move may need to be broken up into the odd bytes needed to align to a 4 byte boundary, then the main block (a multiple of 4 bytes), then the tail bytes. Any size of write will work correctly.

Channels of the Data Mover can be individually enabled and reset. The reset bit should only be written when the channel is currently disabled or being disabled (i.e. it can be written with either the enable or disable command). The act of writing the reset bit causes the current descriptor pointer to be reset to the ring base address. Note that the reset bit for a channel should always be set to initialize the channel when it is enabled for the first time. There are two ways to disable a channel. The first is to clear the enable bit, which causes the engine to complete the transfer that is in progress and stop until re-enabled when it will continue with the next descriptor (unless reset). The second method is to set the abort bit in the same write that clears the enable bit, this causes the current transfer to be abandoned and the engine stops immediately. When re-enabled after being aborted the Data Mover will start with the next descriptor, so the end of the transfer that was aborted will be lost. The active status bit indicates that the channel is currently in use. If the enable bit is cleared the active bit will remain set until the channel has finished any transfer and is disabled.

The round robin weights are only used when more than one channel is busy. The data mover will service requests from a channel until it has completed the number of transfers in the weight parameter for that channel or there are no descriptors queued. The next channel is then serviced. Note that the weights are by transfer, not by number of bytes transferred. To achieve true bandwidth sharing the transfer sizes need to be similar (as do the latencies for the reads and writes). If it is important for system behavior to have a better sharing of the data mover, software should break down large transfers into a series of smaller transfers to enable finer granularity on the round robin.

The data mover descriptors include an interrupt bit. If this is set then when the transfer specified in the descriptor is complete the interrupt bit will be set in the **dm\_dscr\_base** register and the channel interrupt will be raised. If an error is flagged on any data read (either for descriptors or for data blocks) the error bit will be set, the channel interrupt raised and channel will abort and disable. Both the error and interrupt bits are cleared by reads to the **dm\_dscr\_base** register, clearing the interrupt.

A high load can be put on the system by the data mover. As soon as the data returns for one of its four outstanding reads it will write the data and issue another read. To prevent this, data transfers can be throttled on either the read or the write (or both), this is a parameter of the transfer set in the descriptor. When enabled the data mover will check the blocking signal from the agent and will only request if the blocker has been clear for the previous four ZBbus cycles. To avoid starvation there is a timeout, if the data mover is forced to backoff 16 times (e.g. the blocker deasserts but some other agent inserts a request causing the block to assert in less than four cycles) it will request the next time the blocker is clear.

In parts with system revision indicating PERIPH\_REV3 or greater the data mover will prefetch descriptors by overlapping the descriptor fetch with the data transfer of the previous transaction. This improves performance if small transactions are used. A side effect of this is that the current descriptor and count may be one ahead of the transfer in progress (although the count will not go to zero until the channel is idle). If the channel is disabled a prefetched transaction may be completed before the channel goes inactive. An abort will stop the channel immediately and it is UNPREDICTABLE if it stops during the original or prefetched transaction.

## CRC AND CHECKSUM GENERATORS

In parts with system revision indicating PERIPH\_REV3 or greater the Data Mover includes a programmable CRC generator and a checksum engine that computes the ones-complement checksum used by the TCP and UDP protocols. These can be used as part of a data copy or with the prefetch (null destination) parameter to read the data and generate the result without a copy. Each of the four channels has its own register to store the current state information for the CRC and checksum in progress in that channel, allowing the channels to be operated independently and with arbitrary interleave.

The CRC and Checksum operations are only supported when the addresses increment, the results of enabling CRC or checksum with a decrementing source or destination address are UNDEFINED.

The basic operation is the same for both CRC and checksum. The generator is controlled by the data mover descriptors and a single CRC or checksum computation may be split over multiple descriptors (there can even be unrelated moves that do not use the generators mixed in). For both generators each descriptor includes an enable flag, a reset flag and an append flag. If the enable flag is set then the function is enabled for this move. When the engine is enabled if the reset flag is set then the partial result is set to the initial value before the move is started and if the append flag is set the final result is written to the destination address after the move has completed. In all cases the software readable partial result register is updated at the end of the move. The partial result register may be saved by software if a computation is interrupted and can be written to restore the result before the computation is resumed (this requires control of the DMA channel and in many cases can be avoided since a different computation can be performed in other channels). The data in the partial result register is only valid when the channel is not currently computing a checksum or CRC, and the register should only be written when the channel is not computing.

If the data mover channel is disabled then the CRC and checksum partial result registers will contain valid data when the active bit becomes clear, and the computation may be resumed when the channel is enabled again. If the channel is aborted then the partial result registers and the results of any in-progress generation become UNPREDICTABLE and it is not possible to resume the computation.

### Checksum Generation

If the checksum is enabled then for every 16 bits in the move the data is added to the partial sum using the TCP/UDP ones-complement maths. The value that the sum should be initialized to at the start of a computation must be set in the `tcpcs_init` field, in most cases this will be zero. At the start of moves that have the reset flag set the current sum is set from this initial value. At the start of any move with the checksum enabled that does not have the reset flag set the current sum is updated from the partial sum register for that channel.

At the end of any move that has the checksum enabled the partial sum register associated with a channel is updated. If there is a byte left over and the append bit is set then the final byte and a byte of zeros is added to the partial sum before the result is appended. If there is a byte left over and the append bit is not set then the byte is included in the sum and a flag set to ensure the first byte of the next description is used to complete the 16-bit sum.

When the sum is appended it behaves just as if the move were two bytes longer, and any partial cache line is merged with a read-modify-write cycle if cacheable. If the prefetch bit is set then no regular data is moved and just the checksum is written to the destination (again with a read-modify-write merge if required).

Note that if the prefetch flag is set no data will be moved and the destination address is used just to write the TCP sum if the append flag is set. If the Prefetch flag is clear then data will be moved as the checksum calculation is done and the TCP sum is appended if the append flag is set. If the zero\_mem flag is set then zeros are added to the sum, this has no effect unless there is a retained byte from a previous merge to combine in to the sum. If the prefetch and zero\_mem flags are both set no memory is zeroed but zeros will be added to the sum, again this will have no effect unless there is a retained byte or a retained byte is created.

### CRC Generation

The programmable CRC generator can be used to generate any CRC up to 32 bits (output fieldwidths of 8, 16 and 32 bits are supported). There are two sets of CRC definition registers and each datamove specifies which should be used. The CRC is programmed using a set of parameters:

- `crc_init`        32 bit initial value
- `crc_poly`        32 bit polynomial
- `crc_width`       Sets width of output CRC field (32, 16 or 8 bit)
- `crc_txor`        32 bit value to xor the final CRC
- `crc_bit_order`   Sets the order of bits within a byte for the computation (0 for bit 0-7, 1 for 7-0)

When the CRC width is less than 32 bits the definition should be put in the high end of the registers. Thus for a 16 bit CRC the width is set to 16, the polynomial must be placed the upper half of the `crc_poly` field (i.e. bits 63:48 of the `crc_def` register), the initial value will come from the upper half of the `crc_init` field (i.e. bits 31:16 of the `crc_def` register) and the partial result will be in the upper bits (31:16) of the `crc_partial` register. For an 8 bit CRC the width is set to 8 the polynomial must be placed in the upper byte (63:48) of the `crc_poly` field, the initial value will come from the upper byte (31:24) of the `crc_init` field and the partial result will be in the upper byte (31:24) of the `crc_partial` register. For other CRC widths the next larger output field width should be used and the CRC result will be in the upper bits of the field and the upper bits of the registers are used for the definition. In all cases unused bits must be set to zero or the result will be UNPREDICTABLE.

Before the result is written to memory it is XORed with the `crc_txor` value. Many CRCs require this to prevent all zero data having a zero CRC.

The polynomial arithmetic behind the CRC algorithms splits a message (and the attached CRC) into the coefficients of a polynomial. This is normally done in the order the bits flow serially over the wire. While all interfaces send bytes in the order of ascending memory address, some choose to send the least significant bit of the byte first and some the most significant bit first. This changes the order of the bits needed for the CRC computation. The generator can compute with either order as set by the `crc_bit_order` flag. There is a similar issue when the CRC result is written to memory. The data is always written in big endian manner (i.e. the high bits of the result get written to the lower memory addresses) to match the final packet being sent in the order of ascending memory address. But again a swap may be needed within the bits. This final swap is specified in the DMA descriptor by setting the `crc_xbit` flag. This should only be done in the last descriptor of the computation.

**Table 112: Result in memory of appending the result CRC[31:0]**

	Addr+0	Addr+1	Addr+2	Addr+3
<code>crc_xbit = 0. Big Endian Order</code>	CRC[31:24]	CRC[23:16]	CRC[15:8]	CRC[7:0]
<code>crc_xbit = 1. Big Endian Order, bit swapped bytes</code>	CRC[24:31]	CRC[16:23]	CRC[8:15]	CRC[0:7]

Table 113 gives some example values for some common CRCs (in a particular use care will need to be taken with the final bit order).

**Table 113: Example CRC configurations**

	<b>crc_width</b>	<b>crc_poly</b>	<b>crc_init</b>	<b>crc_txor</b>	<b>crc_bit_order</b>	<b>crc_xbit</b>
CRC32 (Ethernet)	32	04C11DB7	FFFFFFFF	FFFFFFFF	1	1
CRC32C (iSCSI)	32	1EDC6F41	FFFFFFFF	FFFFFFFF	1	1
CRC-CCITT (HDLC)	16	80050000	00000000	00000000	1	1

If the CRC is enabled then for every byte in the move the data is merged in to the partial CRC. At the end of any move that has CRC enabled the partial sum register associated with a channel is updated. At the start of any move with CRC enabled that do not have the reset flag set the current sum is updated from the partial sum register for that channel. At the start of any move that has the reset flag set the current sum is set to the `crc_init` value for the selected CRC definition. At the end of a move that has the append bit or the `crc_xbit` flag set the partial sum register associated with the channel is written with the final CRC value after the XOR and bit-flip have been done.

If the append bit is set then the CRC is appended BEFORE the checksum and is included in the checksum. This matches protocols such as iSCSI where the CRC is internal to the TCP payload. The number of bytes appended will be 1, 2 or 4 to match the field width of the CRC. If required a read-modify-write is done. Note that when a checksum is appended it is not included in the CRC.

When the prefetch flag is set no data will be moved and the destination address is used just to write the CRC if the append flag is set. If the Prefetch flag is clear then data will be moved as the CRC calculation is done and the CRC is appended if the append flag is set. If the zero\_mem flag is set then zeros are added to the CRC, this will have an effect on the result. If both prefetch and zero\_mem flags are set then the zeros will be added to the CRC but no memory will be zeroed.

**Computation Sizes and Bandwidth**

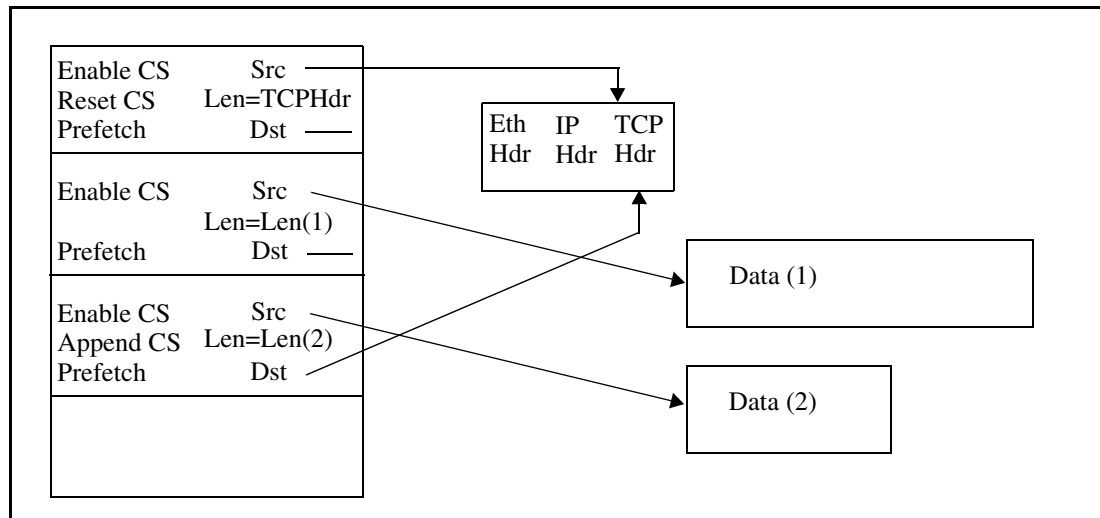
The TCP checksum is done in 16-bit operations, and CRC calculation is done in 12-bit operations so for a full 32 byte cache line 16 or 24 ZBbus cycles are needed. There will be a few clocks overhead, but this gives a theoretical max sum/crc bandwidth of about 6 or 4 Gbps.





## Examples

The following examples illustrate uses of the checksum and CRC engine. These are intended to illustrate use of the data mover generators, in some cases it will be more efficient to do more processing of small buffers on the CPU rather than expending more effort on managing descriptors.



**Figure 33: Example 1 - TCP checksum a packet**

Example 1 illustrated in [Figure 33](#) shows preparing a packet for transmission by computing its TCP checksum. The example has a packet made up of a header buffer (with Ethernet, IP and TCP headers) and two data buffers. Prior to passing the descriptors to the data mover software should compute the checksum of the pseudo-header (the source and destination IP addresses, the protocol type and the TCP Length) and write this (rather than zero) to the checksum field in the TCP header. The descriptors are set up so the first resets the sum and adds the bytes in the TCP header. The pseudo-header sum is included (this takes advantage of the fact that the order of the adds does not matter). The second descriptor adds the first data block, and the third descriptor the second data block. Because the Prefetch flag is set the buffers are only read and no copy is made. The third descriptor has the Append CS flag set so the destination address is used to write out the final checksum. The destination points to the checksum field in the TCP header, so the final checksum is automatically written into the packet. If the packet is going to be sent over the Ethernet interface after only a short delay then consideration should be given to setting the `l2ca_src` flag in all the descriptors so the packets are only read from memory once and staged in the L2 cache for transmission. In this example `l2ca_src` should certainly be set in the first descriptor since the buffer will be at most two cache blocks and the data mover will soon be doing a read-modify-write to update the two checksum bytes.

Example 2 illustrated in Figure 34 shows preparing an iSCSI packet for transmission. The example has a packet made up of a single buffer, although split buffers could also be used. Again, software must compute the pseudo-header sum and put it in the TCP header checksum field. The first descriptor initializes the TCP checksum and computes the CRC of the iSCSI header information, when this is inserted into the packet it will be included in the TCP checksum. The second descriptor causes the CRC of the data to be computed and inserted. In this case the TCP checksum is just continued. The third move adds the TCP header (and pseudo-header sum) to the checksum and stores the result in the TCP checksum field. (In practice it may be more effective to just use descriptors 1 and 2 and have the checksum appended after the packet by descriptor 2. The software can then collect the checksum and do the pseudo-header and TCP header computation.)

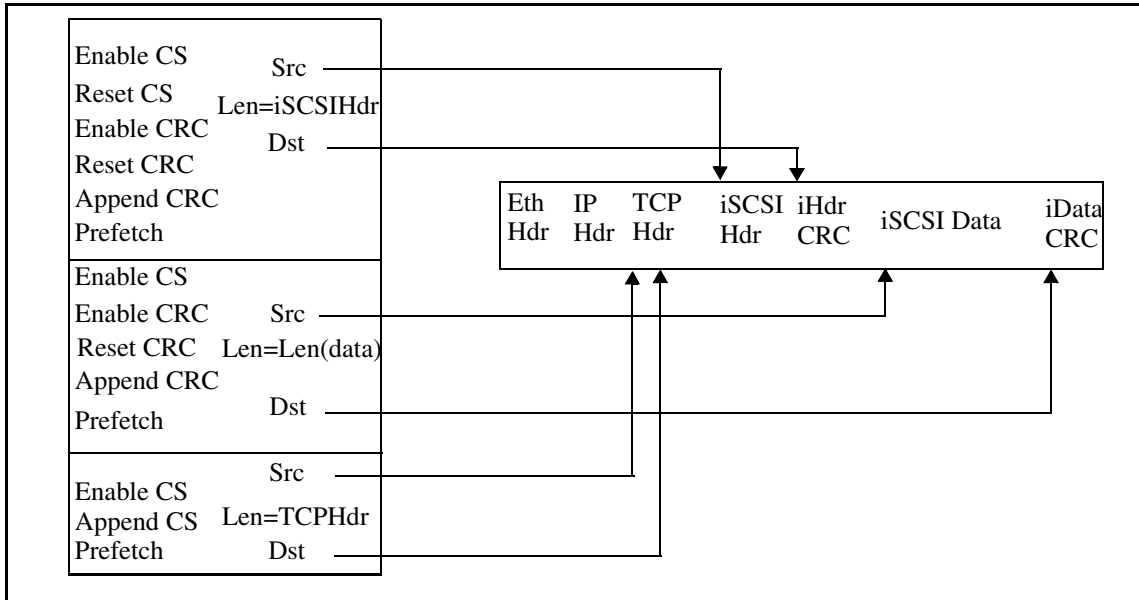


Figure 34: Example 2 - Preparing an iSCSI packet

Example 2 can be extended into a more extreme form to also have the Data Mover schedule the packet for transmission. Assume all iSCSI packets get processed through one channel of the Data Mover and they are the only packets added to one transmit channel of a MAC. If the CPU prepares a MAC DMA descriptor at the same time as the data mover descriptors and places the MAC descriptor in the MAC DMA ring (without incrementing the count) then an additional data move could do a single doubleword copy of the value 1 to the MAC DMA descriptor count register (using an uncached destination to ensure only one doubleword is written). This is a somewhat inefficient use of the data mover but has offloaded the CPU from fielding the datamover interrupt and writing the MAC register.

Example 3 illustrated in Figure 35 on page 183 shows preparing a large iSCSI block for fragmentation before transmission. The example has a packet made up of a single buffer, although split buffers could also be used. The data mover is used to compute the iSCSI CRCs and the partial TCP checksums that cover the data in each fragment. The CRCs are inserted into the data block, the checksums are written to a separate array. The first descriptor resets the checksum for the first fragment and computes and inserts the iSCSI header CRC, the appended CRC will be included in the TCP sum. The second descriptor resets the CRC for the data computation, but keeps the checksum running. At the end of the first fragment the TCP checksum is written into the array (the appended checksum is not included in the CRC so this does not disturb the running CRC). Descriptor 3 covers the second fragment which needs its own checksum, but continues the CRC (if there were more fragments this style of descriptor could be repeated). The fourth descriptor covers the final fragment and will append the datablock CRC and include this CRC in the final checksum. There is then a problem because



if the checksum were appended in this descriptor it would be put on the end of the buffer after the CRC (software could do this and pick the final checksum from there). In the example this is solved using the observation that adding zeros does not change the checksum. In the example a final descriptor is used to add 2 zeros to the checksum (i.e. a 16-bit halfword of zero) and place the final checksum into the array. The software can combine the partial checksums from the array with the header checksums when it computes the headers prior to transmitting the packets. This example could be modified to a more extreme form in a similar manner to example 2: if the header buffers were prepared by software before the data mover were started then rather than having the TCP checksums written to an array the Append CS would be removed from the second and third descriptors, and additional descriptors inserted after the second and third and to replace the final one. These descriptors would have the CRC disabled and would add and append the TCP sum in the same way as the third descriptor in example 2. With appropriate setup of the MAC DMA ring moving a single doubleword to the MAC DMA count register would allow the packets to be scheduled for transmission.

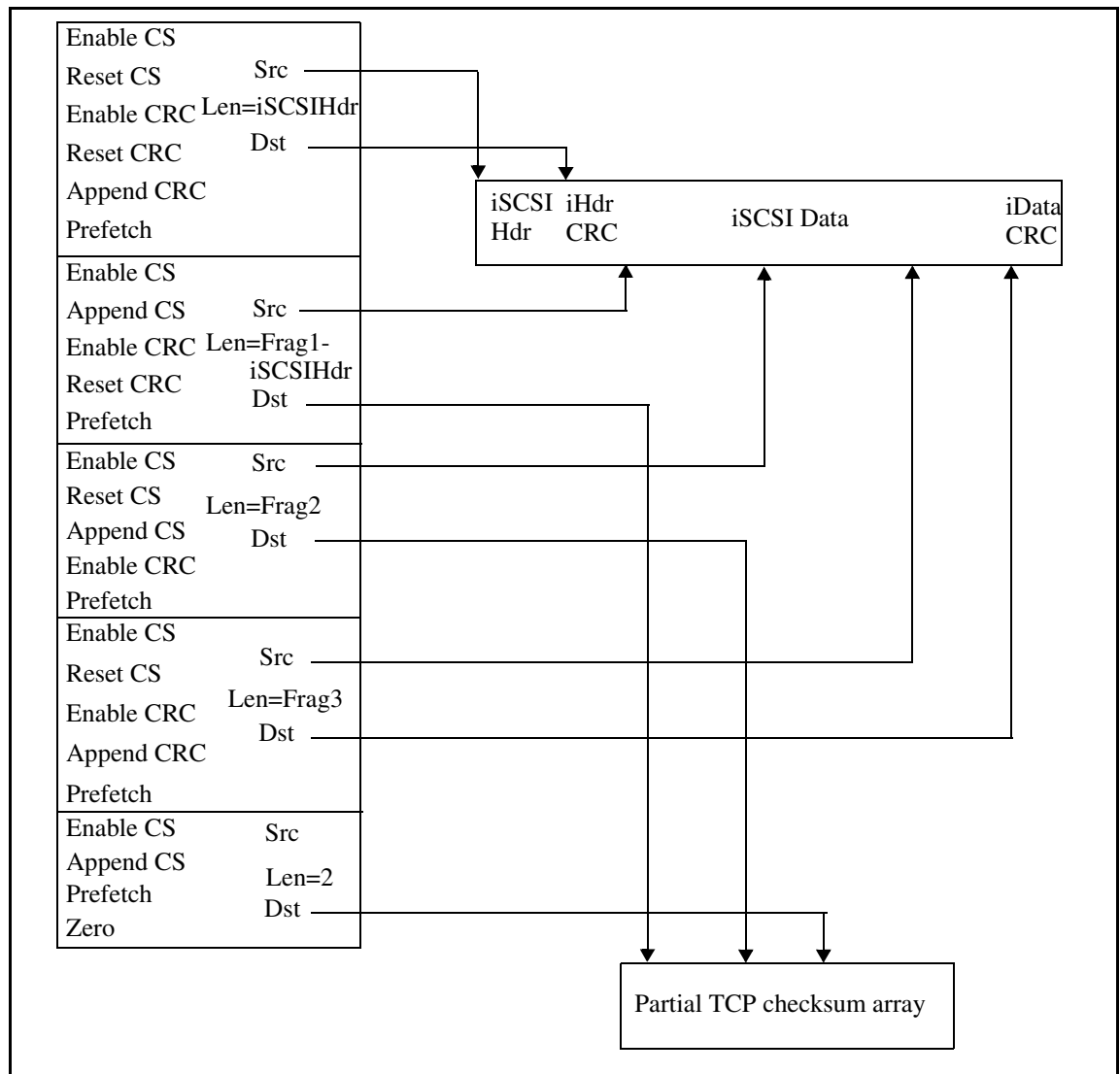


Figure 35: Example 3 - Fragmenting an iSCSI packet

**DATA MOVER CONTROL REGISTERS**

**Table 114: Data Mover Descriptor Base Address Register**

dm_dscr_base_0 - 00_1002_0B00 dm_dscr_base_1 - 00_1002_0B20 dm_dscr_base_2 - 00_1002_0B40 dm_dscr_base_3 - 00_1002_0B60 Read clears some bits			
Bits	Name	Default	Description
3:0	zero	4'b0	These bits must be zero.
39:4	base	36'bx	This is the base address of the descriptor ring.
55:40	ring_size	16'bx	This field sets the total number of descriptors in the ring. If this field is set to 0 the ring will contain 65536 descriptors.
58:56	priority	3'bx	This field gives the weight for this channel. It sets how many descriptors are processed from this channel before advancing round robin to the next channel. 000: 1 descriptor. 001: 2 descriptors. 010: 4 descriptors. 011: 8 descriptors. 100: 16 descriptors. 101-111: Reserved
59	active	1'b0	Read Only. This bit is set when a descriptor from the current channel is actively being used. When the enbl bit is cleared the channel will remain enabled until this bit is clear.
60	interrupt	1'b0	Read Only. This bit is set when the channel is interrupting because of the end of a transfer with the descriptor interrupt bit set. This bit is cleared by a read from the <b>dm_dscr_base</b> register.
61	error (R/O)	1'b0	On a read this bit is set when the channel is interrupting because of a data transfer error (uncorrectable ECC, Bus Error or Fatal bus error signalled on the D_CODE). If such an error occurs the channel will abort. This bit is cleared by a read from the <b>dm_dscr_base</b> register.
	reset (W/O)		If this bit is written with a 1 the current descriptor pointer is reset to the base address of the descriptor ring (bits 39:0). This should always be done the first time a channel is enabled.
62	abort(W/O)	1'b0	If this bit is written with a 1 the DMA engine will abort the current transfer and disable the channel. If both this bit and the enbl bit (bit 63) are set in the same write, this bit will override and the channel will be aborted.
63	enbl	1'b0	This bit must be set to enable the DMA channel. If this bit is cleared while the channel is running the current transfer is completed before the engine stops. (Use the abort bit to cause the engine to stop immediately).

**Table 115: Debug Data Mover Descriptor Base Address Register**

dm_debug_dscr_base_0 - 00_1002_0B18 dm_debug_dscr_base_1 - 00_1002_0B38 dm_debug_dscr_base_2 - 00_1002_0B58 dm_debug_dscr_base_3 - 00_1002_0B78 READ ONLY			
Bits	Name	Description	
63:0	dscr_base	Reading from this register gives the same data as reading from the <b>dm_dscr_base</b> register (Table 114), but without the side effect of clearing the interrupt and error status bits. It is intended for debugger access to the status.	



**Table 116: Data Mover Descriptor Count Register**

Bits	Name	Default	Description
<b>dm_dscr_count_0 - 00_1002_0B08</b> <b>dm_dscr_count_1 - 00_1002_0B28</b> <b>dm_dscr_count_2 - 00_1002_0B48</b> <b>dm_dscr_count_3 - 00_1002_0B68</b>			
15:0	count	16'b0	This is the number of descriptors owned by the DMA engine. Reads will return the number of unused descriptors. Data that is written to this register will be added to the count (assigning that number of additional descriptors to the controller).
63:16	reserved	48'b0	Reserved

**Table 117: Data Mover Current Descriptor Address**

Bits	Name	Default	Description
<b>dm_cur_dscr_addr_0 - 00_1002_0B10</b> <b>dm_cur_dscr_addr_1 - 00_1002_0B30</b> <b>dm_cur_dscr_addr_2 - 00_1002_0B50</b> <b>dm_cur_dscr_addr_3 - 00_1002_0B70</b> <b>READ ONLY</b>			
39:0	dscr_addr	40'bx	The current descriptor address can be read from this field.
47:40	reserved	8'b0	Reserved
63:48	cur_count	16'b0	The current count of descriptors owned by the DMA engine can be read from this field.

**Table 118: Data Mover CRC Definition Registers (Only if System Revision >= PERIPH\_REV3)**

Bits	Name	Default	Description
<b>crc_def_0 - 00_1002_0B80</b> <b>crc_def_1 - 00_1002_0B90</b>			
31:0	crc_init	32'bx	This is the initial value for the partial CRC and is loaded into the partial result register before a move that has the crc_reset flag set. For CRCs smaller than 32 bits the initial value should be put into the high bits of this field and the low bits should be zeros.
63:32	crc_poly	32'bx	This is the polynomial used to define the CRC. For CRCs smaller than 32 bits the polynomial value should be put into the high bits of this field and the low bits should be zeros.

**Table 119: Data Mover CRC/Checksum Definition Registers (Only if System Revision >= PERIPH\_REV3)**

ctcp_def_0 - 00_1002_0B88 ctcp_def_1 - 00_1002_0B98			
Bits	Name	Default	Description
31:0	crc_txor	32'bx	This value is XORed with the partial result of the CRC computation before the final result is written to memory. For CRCs smaller than 32 bits the value should be put into the high bits of this field and the low bits should be zeros.
47:32	tcpcs_init	16'bx	This is the initial value for the checksum and is loaded into the partial result register before a move that has the tcpcs_reset flag set.
49:48	crc_width	2'bx	This sets the field width used when the CRC is written to memory: 00 - 4 bytes (32 bits) 01 - 2 bytes (16 bits) 10 - 1 byte (8 bits) 11 - Reserved
50	crc_bit_order	1'bx	This selects the bit order used when the CRC engine is reading bytes. 0 - Bit 0 is used first and bit 7 last 1 - Bit 7 is used first and bit 0 last
63:51	reserved	13'bx	Reserved.

**Table 120: Data Mover Channel Partial Result Registers (Only if System Revision >= PERIPH\_REV3)**

dm_partial_0 - 00_1002_0BA0 dm_partial_1 - 00_1002_0BA8 dm_partial_2 - 00_1002_0BB0 dm_partial_3 - 00_1002_0BB8			
Bits	Name	Default	Description
31:0	crc_partial	32'bx	Current partial CRC result. After a transfer with the crc_ap append bit set this register will contain the final CRC result (after XOR and bit-flip). For CRCs smaller than 32 bits the value is in the high bits of this field. If the value is saved it may be written back later to allow the CRC computation to resume.
47:32	tcpcs_partial	16'bx	Current checksum partial result. If the value of this field and the odd_byte bit are saved they may be written back later to allow the checksum computation to resume.
48	odd_byte	1'bx	This bit is set when the current checksum finished on an odd byte boundary. Since the checksum is calculated in 16 bit halfwords this bit being set indicates that the first byte of the next buffer must be added to the low half of the partial result before 16 bit additions are used. If the value of this bit and the tcpcs_partial field are saved they may be written back later to allow the checksum computation to resume.
63:49	reserved	15'bx	Reserved.



## DATA MOVER DESCRIPTORS

**Table 121: Data Mover Descriptor First Doubleword**

dm_dscr_a		
Bits	Name	Description
39:0	dst_addr	The destination address of the transfer (may be any alignment).
40	un_dest	This bit should be set for an uncached destination, and clear if the destination is cacheable coherent. Uncached accelerated operations will be used.
41	un_src	This bit should be set for an uncached source, and clear if the source is cacheable coherent. Uncached accelerated operations will be used.
42	interrupt	If this bit is set an interrupt will be generated at the end of the transfer.
43	reserved	Reserved
45:44	dir_dest	This indicates the direction the destination address should be moved: 00: The address is incremented 01: The address is decremented 10: The address is held constant 11: Reserved
47:46	dir_src	This indicates the direction the source address should be moved: 00: The address is incremented. 01: The address is decremented. 10: The address is held constant. 11: Reserved
48	zero_mem	If this bit is set the source parameters will be ignored, and zeros will be transferred to the destination address.
49	prefetch	If this bit is set the destination parameters will be ignored, the data will just be read from the source. This can be used with the L2C_SRC bit to prefetch lines from memory into the L2 cache. Note: when used in this manner, it is most efficient if the src address is 32 byte aligned.
50	l2c_dest	If this bit is set writes to the destination buffer are marked L2_cacheable and will therefore be allocated in the L2 on an L2 miss. (For cacheable transfers the L2 is always checked, and will always be updated if hit).
51	l2c_src	If this bit is set reads from the source buffer are marked L2_cacheable and will therefore be allocated in the L2 on an L2 miss. (For cacheable transfers the L2 is always checked, and will supply the data if hit).
52	rd_bkoff	If this bit is set the data mover will backoff from a read transfer until the destination blocker has been clear for four ZBbus cycles.
53	wr_bkoff	If this bit is set the data mover will backoff from a write transfer until the destination blocker has been clear for four ZBbus cycles.
54	tcpcs_en	Set to enable TCP checksum (System revision PERIPH_REV3 or greater)
55	tcpcs_res	Set to reset TCP partial sum to init value at start of move (System revision PERIPH_REV3 or greater)
56	tcpcs_ap	Set to append TCP checksum at end of move (System revision PERIPH_REV3 or greater)
57	crc_en	Set to enable CRC engine (System revision PERIPH_REV3 or greater)
58	crc_res	Set to reset partial CRC to init value at start of move (System revision PERIPH_REV3 or greater)
59	crc_ap	Set to append CRC at end of move, but before checksum (System revision PERIPH_REV3 or greater)
60	crc_dfn	Selects which TCP/CRC definition to use (System revision PERIPH_REV3 or greater)
61	crc_xbit	Set to swap the bits within the CRC bytes prior to checksumming and appending to the packet and/or updating the partial register. Typically only set for last block of CRC (if the CRC needs the bitflip). (System revision PERIPH_REV3 or greater)

**Table 121: Data Mover Descriptor First Doubleword (Cont.)**

dm_dscr_a		
Bits	Name	Description
63:62	reserved	Reserved

**Table 122: Data Mover Descriptor Second Doubleword**

dm_dscr_b		
Bits	Name	Description
39:0	src_addr	The source address of the transfer (may be any alignment).
59:40	length	The length of the transfer in bytes. If this field is zero then 2 <sup>20</sup> bytes will be transferred.
63:60	reserved	Reserved





This Page is left blank for notes

## Section 8: PCI Bus and HyperTransport Fabric

### INTRODUCTION

The PCI bus and HyperTransport (formerly called “Lightning Data Transport” or “LDT”) fabric provide the main general expansion buses for the BCM1250 and BCM1125/H parts. PCI is the standard bus used for peripheral connection in many systems and there are a wide variety of peripheral devices that connect to it. HyperTransport (HT) is a high performance replacement for PCI on system boards, it uses a chain of unidirectional point to point links to form a fabric of devices running at much higher data rate. The HyperTransport devices use the standard PCI configuration process, so devices can be migrated from the PCI to HyperTransport with few software changes. Initial devices available for the HyperTransport fabric include bridges to PCI, PCI-X and AGP. The BCM1250 and BCM1125H have both HT and PCI interfaces, the BCM1125 has only PCI.

The PCI interface conforms to revision 2.2 of the PCI standard. It is a 32 bit wide interface, can run up to 66MHz and uses 3.3V signal levels (the interface is **not** 5V tolerant). PCI special cycles, dual address cycles and the LOCK protocol are not supported. The interface can act as either the host bridge or as a device configured and controlled by another host. These are referred to as Host Mode and Device Mode. When operating in Host Mode an internal arbiter can be used, providing support for four external PCI devices; alternatively an external arbiter can be used to allow more external devices.

The HyperTransport interface conforms to revision 1.03 of the HyperTransport specification although to ensure software backwards compatibility with the earlier implementation (based on 0.17 of the HT specification) there are a few differences, mainly in the mechanisms for error reporting (this was not specified in the earlier version of the standard). The HyperTransport transmit and receive clocks are usually the same frequency (the receive link frequency may be lower than the transmit frequency), generated from the 100 MHz reference clock. The HyperTransport transmit clock can be the reference clock x2, x3, x4, x5 or x6. The interface runs 8 bits wide in each direction, with data sent on both edges of the clock. The HyperTransport interface runs as a host bridge, and therefore is always on one end of the HyperTransport fabric. Support is provided for fabrics with host bridges at each end.

Figure 36 shows the physical and logical organization of the PCI and HyperTransport interfaces.

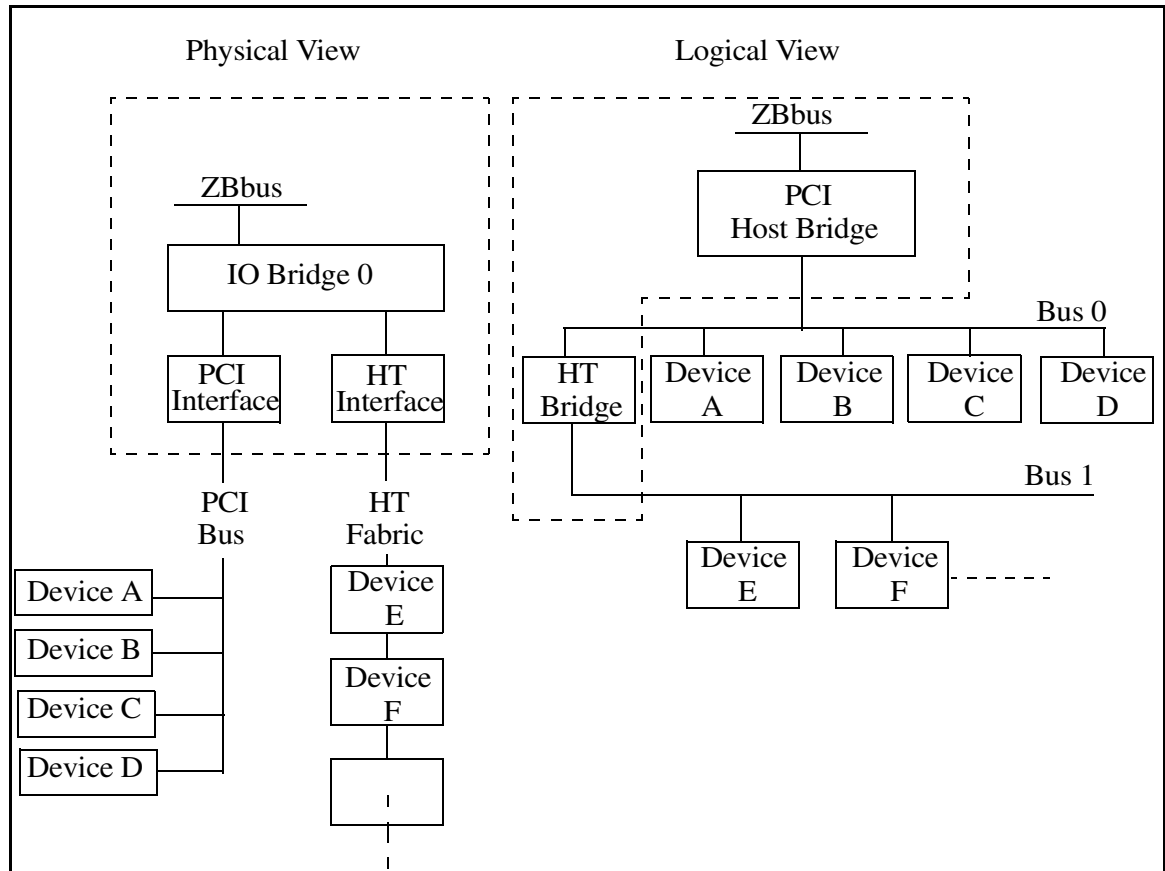


Figure 36: PCI and HyperTransport Organization

Physically the PCI and HyperTransport interfaces are both connected to ZBbus through I/O bridge 0 (as shown on the left in Figure 36). The I/O bridge implements the ZBbus protocol, and includes the address and data buffers for transactions between the ZBbus and each interface, the buffers for peer-to-peer transfers between the PCI and HyperTransport, and the buffer for merging partial cache line writes. The PCI and HyperTransport interface units implement their respective bus protocols, and include additional data and command buffering. The bandwidths of the connections between the I/O bridge and interface units are sized appropriately for the interfaces, so the HyperTransport is not constrained by the PCI bandwidths.

However, the interfaces are presented logically to the system as if the HyperTransport was bridged from the PCI bus (shown on the right of Figure 36). This makes the PCI always bus zero to the device enumeration code (this is required by some software) and allows configuration to be based on existing code.

## PCI AND HYPERTRANSPORT ADDRESS RANGE

This section describes how devices on the PCI bus and HyperTransport fabric (and any buses bridged from them) are mapped into the memory space. There are two complexities in the mapping. The first is supporting the different types of access that need to be done, and the second is supporting endian swapping when the part is running as a big endian system (both PCI and HyperTransport are little endian). A full discussion of the endian issues is in [Section: "Endian Policies" on page 201](#), the two endian options match bit lanes and match byte lanes are defined there.

All the areas for mapping PCI and HyperTransport devices, except a HyperTransport only expansion area, are put in the low 4 GBytes of the address space so they can be addressed using 32 bits of address in systems that only use 32 bit addressing.

Accesses to the PCI and HyperTransport ranges of the address map should be done as uncacheable or cacheable non-coherent. These addresses pass outside the coherence domain of the system. If cacheable coherent accesses are made to the PCI or HyperTransport space the behavior of the system will become UNDEFINED.

The PCI and HyperTransport portions of the memory map are shown in Figure 37.

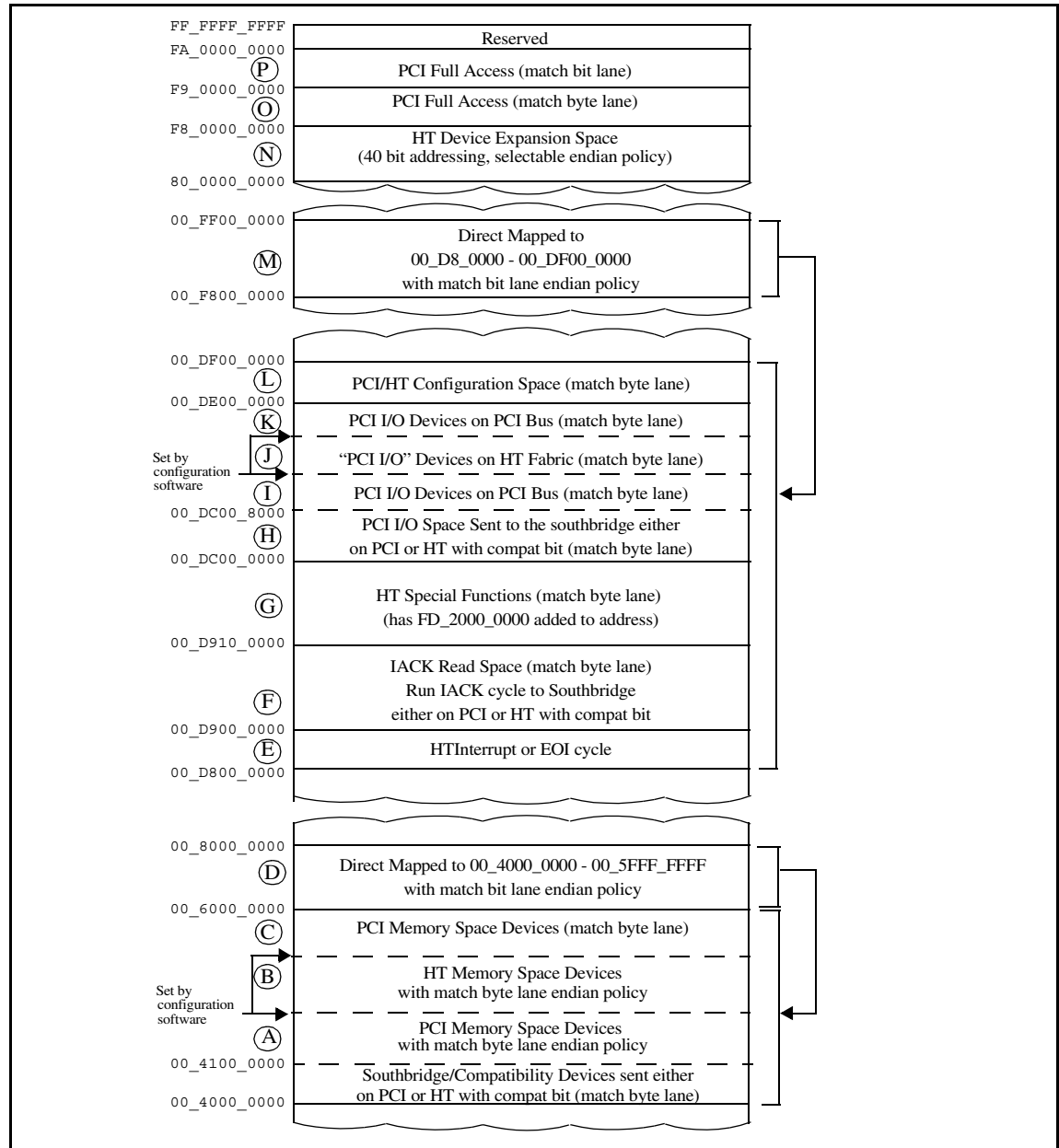


Figure 37: Address Ranges for CPU Access to PCI and HyperTransport

## MEMORY MAPPED DEVICES

There is a 512MB region for mapping memory mapped PCI and HyperTransport devices. Logically this all maps to the PCI and a segment of it is bridged to the HyperTransport. (On the BCM1125 there is no HyperTransport interface and all accesses to this space are PCI accesses.) Thus the area is divided up into the three regions A, B and C in the address map. Region A contains all the devices on the PCI bus (including bridges) that the configuration code allocated before the HyperTransport bridge. Region B contains the addresses bridged to the HyperTransport fabric, accesses to this address range are detected by the I/O Bridge 0 and sent to the HyperTransport controller directly. Region C consists of the address range allocated to devices on the PCI bus that were configured after the HyperTransport bridge and empty space that was not allocated. When the part is running in big endian mode this area of memory accesses the PCI and HyperTransport using the match byte lane policy.

Region D (offset from the main area by setting address bit 29) is directly mapped onto the A, B and C regions, but uses the match bit lane policy.

## HYPERTRANSPORT EXPANSION SPACE

There is a large region of the address space allocated to HyperTransport expansion. This is marked as region N in the address map above ([Figure 37 on page 193](#)). This space is Reserved on the BCM1125. Use of this area requires the use of full 40 bit physical addresses. Configuration software will also need modification to use this area (by default it will attempt to put all HyperTransport peripherals in region B since they will seem to be bridged from the PCI bus). The endian policy used in the expansion space is determined by the `exp_endian` bit in the SRI Command Register. If this bit is clear then all of the expansion space will use the match byte lane policy. If the bit is set then address bit [38] is used to select the endian policy and the address bit is cleared as the request passes through the bridge. Thus:

- `exp_endian` bit clear:
  - `80_0000_0000 - F7_FFFF_FFFF` use the match bytes policy.
- `exp_endian` bit set:
  - `80_0000_0000 - BF_FFFF_FFFF` and `E0_0000_0000 - F7-FFFF-FFFF` use the match bytes policy.
  - `C0_0000_0000 - DF_FFFF_FFFF` map to `80_0000_0000 - 9F_FFFF_FFFF` and use the match bits policy.

## CONFIGURATION SPACE

Configuration cycles can be performed on the PCI bus and HyperTransport fabric by accessing the configuration space. Again, with address bit [29] clear the match byte lane policy is used and with a[29] set the match bit lane policy is selected. See [Section: "Configuration of PCI and HyperTransport" on page 234](#) for a full discussion of configuration.

## PCI I/O SPACE

PCI, and for compatibility HyperTransport, has both memory space and I/O space corresponding to the standard and I/O instructions in the x86 architecture. Most PCI devices are mapped into memory space, as memory mapped I/O devices (indeed the PCI standard encourages this) but some use I/O space addresses. The MIPS architecture has no distinction between memory and I/O address ranges, so a range of memory addresses has been defined to map to PCI I/O space.

There is a 32 MB region allocated for I/O space transactions, covering regions H, I, J and K in the address map (Figure 37). When the part is running in big endian mode these regions use the match byte lane endian policy. The I/O address consists of only the low 25 bits of the address, for a 32 bit PCI I/O address bits 31:25 are set to zero. Region H is special and is discussed in the next section. Region I is the space allocated to I/O devices on the PCI that are configured before devices on the HyperTransport, these are accessed using PCI I/O reads and writes. Region J is allocated to devices that are behind the HyperTransport bridge and use I/O addresses, these are accessed by adding the 25 bit I/O address to the base address (FD\_FC00\_0000) of the special area on the HyperTransport defined for signalling I/O transactions. Region K covers any PCI devices that were configured after the HyperTransport ones.

Region M (offset from the main I/O space area by setting address bit 29) is directly mapped onto the H-K regions, but uses the match bit lane policy.

## THE SOUTHBRIDGE, VGA AND SUBTRACTIVE DECODE

In most x86 systems there is a device called the Southbridge that connects to the PCI bus and provides the interface to a number of slow speed I/O devices (serial ports, parallel ports, keyboard, mouse, more recently USB) and legacy buses (normally the ISA bus), it also contains the legacy interrupt controller (often known as the PIC, after the original Peripheral Interrupt Controller chip).

The interface supports the use of a single southbridge device, which can either be on the PCI bus, bridged from the PCI bus, on the HyperTransport fabric or bridged off the HyperTransport fabric.

There are two problems with addressing I/O devices in a legacy southbridge:

- The I/O addresses of peripherals in the southbridge are normally at the location they have historically been. These addresses are not contiguous, so do not fit well into the PCI address allocation model.
- If the southbridge connects to an ISA bus, there is no easy way to know the memory or I/O addresses of devices on the ISA bus.

On PCI buses subtractive decode is used to access these addresses. Any address that is not claimed by a device on the bus is claimed by the southbridge and used by its internal devices or passed to the ISA bus. As of PCI 2.2 the southbridge may be behind a (or a series of) PCI-PCI bridge(s), the configuration software notices this and sets up all the bridges on the path to act as subtractive decode bridges.

On HyperTransport there is no way to do subtractive decode. Normally commands whose address is not actively decoded and claimed will reach the end of the fabric and an NXA error will be raised (revision 1.0 and later of the HyperTransport standard allows the device at the end of the chain to perform a subtractive decode). Instead, the standard allows packets to be marked with the COMPAT bit, this overrides the active address decoding and will always pass the access to the southbridge (for subtractive decode).



There is a reset time configuration bit southOnLDT (on generic bus IO\_AD[21]) that sets whether the route to the southbridge is via the PCI bus or HyperTransport fabric. This is done as a hardware configuration option to allow use of the southbridge before the PCI bus and HyperTransport fabric have been configured (this is in line with section C.3 of the HyperTransport specification rev 0.17, and section E.3.1 of the HyperTransport specification rev 1.0). The PCI and HyperTransport device enumeration will discover the southbridge as it configures the system and can check that the southOnLDT bit is set correctly (if it is not correct there is a serious hardware error, or an option card has been added with a new southbridge -- both these events should be reported). If the southbridge is reached through the HyperTransport bridge then the configuration software should check (and possibly configure, if they are not also setup in hardware) any HyperTransport-HyperTransport and HyperTransport-PCI bridges on the route to accept and forward packets with the COMPAT bit set (and from there on set any PCI-PCI bridges to do subtractive decode). If the southbridge is on the direct PCI bus or is bridged from there by anything other than the internal HyperTransport bridge (which logically appears as if on the PCI), then the southOnLDT bit must be clear (and any bridges off the PCI must be set to do subtractive decode).

Since subtractive decode is only used for legacy devices it will only be used for I/O addresses in the bottom of the range. In this interface, locations in the first 32 KB of the I/O address space can **only** be used for the legacy subtractive decode devices. This area is shown as region H in [Figure 37 on page 193](#). Any access to this range will be directed according to the southOnLDT bit, if it is set they will be sent out as HyperTransport requests with the COMPAT bit set, if it is clear they will be sent to the PCI for subtractive decode. (On the BCM1125 the access is always made to PCI.)

Similarly, the bottom 16 MB of the memory space region is used for subtractive decode only. If the southOnLDT bits is set any accesses in this region will be sent on the HyperTransport bus as a 24 bit address with the COMPAT bit set, otherwise they are sent on the PCI for subtractive decode.

The other legacy device that the PCI and HyperTransport specifications have as special is the VGA display controller. This has historically used memory addresses 0A\_0000-0B\_FFFF in the address map and registers X3B0-X3BB and X3C0-X3DF in I/O space (per PCI Spec AD [15:10] are ignored for this decode). These addresses are in the compatibility regions. Rather than being routed based on the SouthOnLDT configuration bit, the VGA range is routed using the VgaEn bit in the HyperTransport Bridge Control Register. If the VgaEn bit is clear (indicating the bridge should not forward VGA accesses) VGA accesses are done to the PCI bus, if the VgaEn bit is set the access is sent to the HyperTransport Fabric but the COMPAT bit is not set allowing the request to be routed by address to the VGA controller. As with all the compatibility space accesses the address is masked to a 24 bit address with the upper bits zero.



Compatibility space routing is summarized in the pseudo-code below:

```
if (00_4000_0000 <= zbaddr <= 00_7FFF_FFFF)
{
    pcildtaddr = zbaddr & DFFF_FFFF
    endian = zbaddr[29] ? match bit lanes : match byte lanes
    if (pcildtaddr < 00_4100_0000) /* Note after a[29] removal */
    {
        if (pcildtaddr[23:17] == 7'b0000101) /* 0A_xxxx/0B_xxxx */
        {
            if (Bridge Control VgaEn)
                send (pcildtaddr & 00FF_FFFF) to HT with COMPAT bit clear
            else
                send (pciaddr & 00FF_FFFF) to PCI
        }
        else
        {
            if (Southbridge is on LDT)
                send (pcildtaddr & 00FF_FFFF) to HT with COMPAT bit set
            else
                send (pciaddr & 00FF_FFFF) to PCI
        }
    }
    else if (HT mem base <= pcildtaddr <= HT mem limit)
        send to HT
    else
        send to PCI
}
```

For I/O space:

```

if (00_4000_0000 <= zbaddr <= 00_7FFF_FFFF)
  ioaddr = (zbaddr & 1FF_FFFF)
  endian = zbaddr[29] ? match bit lanes : match byte lanes
  if (ioaddr < 000_8000)
  {
    if ((ioaddr[24:16] == 0) &&
        (((ioaddr[9:4]==3B) && (ioaddr[3:2]!=2'b11)) ||
         (ioaddr[9:5]==5'b11110))
        )
    { /* 3b0-3bb or 3c0-3df */
      if (Bridge Control VgaEn)
        send to HT (FD_FC00_0000 + ioaddr) with COMPAT bit clear
      else
        send to PCI ioaddr as I/O access
    }
    else
      if (Southbridge is on LDT)
        send to HT (FD_FC00_0000 + ioaddr) with COMPAT bit set
      else
        send to PCI ioaddr as I/O access
    }
  }
  else /* ioaddr >= 000_8000 */
  {
    if (HT bridge header io base <= ioaddr <= HT header io limit)
      send to HT (FD_FC00_0000 + ioaddr)
    else
      send to PCI ioaddr as I/O access
  }
}

```

## HYPERTRANSPORT END OF INTERRUPT (EOI) SIGNALING SPACE

The HyperTransport interrupt scheme is discussed in detail in [Section: “HyperTransport Interrupts” on page 48](#), which describes all the interrupts on the part. The HyperTransport level sensitive interrupt messages require an End Of Interrupt (EOI) acknowledgement to clear them when the interrupt processing is complete. This is sent as a broadcast message in the HyperTransport address range reserved for interrupt messages.

The interface allows an EOI to be generated by doing a write to the EOI section of the address space, which is marked as region E on the address map ([Figure 37 on page 193](#)). Bits 23:16 of the address accessed should be the vector number being acknowledged, bits 15:5 should be zero, bits 4:2 are the message type (MT) field and must be set to 3'b111 to indicate an EOI. The data written is discarded. (Any access to this region will be converted into the broadcast, but it is inadvisable to do a read to generate the EOI since the data returned will be UNPREDICTABLE and since the processor would be doing an uncached access it is likely to stall the CPU for no good reason).

In addition to generating EOI this space can be used to generate HyperTransport interrupt messages. If bits [4:2] of the address are not equal to 3'b111 (i.e. the message is encoding something other than an EOI) rather than sending a Posted Broadcast message the bridge will send a Posted Sized Byte Write with a count of zero, the COMPAT bit clear and a doubleword of zeros as byte masks (the extended destination bits from rev 1.01 and later of the HyperTransport standard are not supported). This is the format of an interrupt message. The address bits 23:16 indicate the vector, bits 15:8 the destination, bit 7 is Reserved and should be zero, bit 6 indicates the destination mode, bit 5 should be zero to indicate an edge interrupt (level cannot be used because the interface will not accept the inbound EOI), and bits 4:2 are the interrupt type.

## LEGACY INTERRUPT ACKNOWLEDGE (IACK) SPACE

Legacy interrupts from a PIC style interrupt controller in the southbridge require an interrupt acknowledgement (IACK) cycle, this is a byte read that will return the interrupt source vector number. As with the subtractive decode I/O space, the part supports the southbridge being on either the HyperTransport bus or the PCI as set by the southOnLDT configuration bit.

If the southbridge is on the PCI then it will signal the interrupt using its INTR pin, which should be routed to one of the general interrupt inputs. If the southbridge is on the HyperTransport then it signals the interrupt using an Ext. Int. message, which is translated into interrupt 54 being raised in the CPU interrupt mappers (see [Section: "HyperTransport Interrupts" on page 48](#)). In either case the interrupt must be acknowledged with an IACK cycle. This is a byte read that returns the interrupt vector information. The address map defines the region F (in [Figure 37 on page 193](#)) for the CPU to read from to perform an IACK access; the address is ignored and any access in this range has the same effect. If the southOnLDT bit is set this gets run as a cycle on the HyperTransport to the special address range (FD\_F900\_0000 - FD\_F90F\_FFFF) with the COMPAT bit set. If the southOnLDT bit is clear, the read gets run as an IACK cycle on the PCI bus.

## PCI FULL ACCESS SPACE

Regions O and P in [Figure 37 on page 193](#) are provided to allow the CPU to generate any memory space address on the PCI bus. When the interface is configured in Device Mode the address map used on the PCI bus matches the host system and not the internal address map. The full access space allows the device to access any host system address, as would be expected for a PCI peripheral. The bottom 32 bits of the address are used directly to form the PCI bus address. Address bit [32] is used to select between the two regions and therefore the endian policy used for accesses. The full access space is available when the interface is in Host Mode, but it is less useful. The PCI interface will not respond to requests that it generates, any access through Full Access space to an address with a destination in the part will result in a master abort.

There is no equivalent I/O full access space, since the normal I/O space can be used in Device Mode to generate 25 bit I/O addresses on the PCI bus.

## SPECIAL HYPERTRANSPORT SPACE

The HyperTransport defines a range of address space for other special cycles. None are directly supported by the interface, however access is provided to the special range using region G in [Figure 37 on page 193](#). Use of addresses in this region requires understanding of the HyperTransport specification. Since no assistance is provided for the special use, accesses in this region may not be useful, and can result in UNDEFINED behavior. Address bit [29] is used to select between the endian policies.



## HYPERTransport READ RESTRICTIONS

The HyperTransport interface maps read requests from the ZBbus into read commands on the HyperTransport fabric. The interface supports all requests that can be generated by CPU instructions (reads of 1-8 bytes and 32 bytes) and any request that includes an aligned number of 32-bit words. Care must be taken when using uncached accelerated accesses from the CPU or the Data Mover, since these can generate accesses that are not supported and will have UNDEFINED results. For best performance the ZBbus read should translate into a single HyperTransport request, accesses that are 1-4 bytes translate directly to a Read (sized) Byte request, and accesses that are an aligned number of 32-bit words translate to the Read (sized) Doubleword request. The interface is able to split ZBbus reads into two HyperTransport requests allowing the 5-7 byte reads that the load doubleword left (LDL) and load doubleword right (LDR) instructions can generate, but the interface is not optimized to give these unusual accesses high performance.

There is no constraint on writes, the interface supports arbitrary writes of 1-32 bytes. Any ZBbus write can be converted into a Write (sized) HyperTransport request.

There are no constraints on inbound accesses. All legal HT read and write requests are supported. (Requests larger than 32 bytes will be converted into two requests to the ZBbus but this is transparent to the HT fabric.)

## ENDIAN POLICIES

The system can be run either as a big endian system or as a little endian system. Both the PCI bus and HyperTransport fabric are little endian. When the part is running big endian there are two policies that are used for connecting to the interfaces.

Selection of endian policy is made based on address bits for requests going both from the system to the PCI bus or HyperTransport fabric and from the PCI bus or HyperTransport fabric to the system. Peer-to-peer accesses between the PCI bus and HyperTransport fabric are always passed directly. In most cases address bit [29] being clear indicates that the match byte lane policy will be used, and address bit [29] being set indicates the match bit lane policy will be used. The address bit used to select the endian mode is zeroed as the request passes through the interfaces so that the target always see the match bytes' address.

### LITTLE ENDIAN SYSTEM: NO SWAPS

When the part is run as a little endian system there is no need to do any swapping between the system and PCI or HyperTransport. If the system configuration register is set for little endian then no swapping is done, and the two access addresses become aliases.

This is illustrated in [Figure 38](#). At the top this shows a double-word with the bit numbers and byte addresses used by the CPU and ZBbus. Below it shows how the bytes of the double-word will appear on the PCI byte lanes. The PCI uses byte enables to indicate both the transfer size and the low two address bits, these are shown as the BE#[3:0] signals.

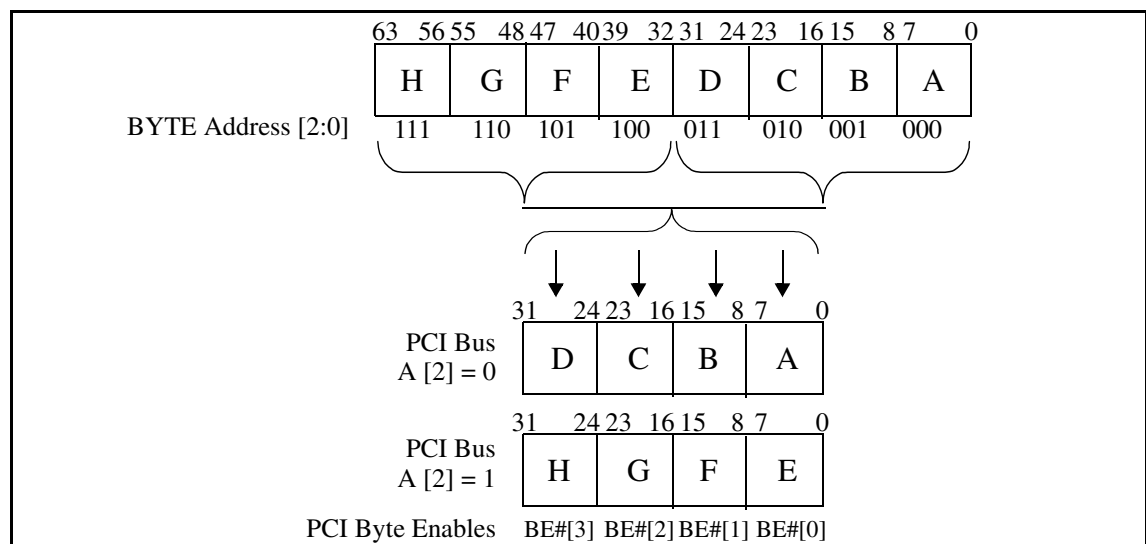


Figure 38: Little Endian System

### BIG ENDIAN SYSTEM: MATCH BYTE LANES

The match byte lanes endian policy will match the byte lanes of 32 bit values on either side of the interface. This preserves the memory address ordering between the ZBbus and PCI or HyperTransport. However, it will scramble the bit ordering. Consequently a 32 bit value that is written into a PCI register from the CPU will have a different interpretation.

The policy is illustrated in the Figure 39. This is similar to Figure 38 on page 201, except that from the CPU the mapping from bytes of the double-word onto addresses is reversed. When the data is passed to the PCI/HyperTransport the address order is maintained and the low two address bits directly set the PCI byte enables. The little endian nature of the PCI is exposed to the processor. If a value is stored from a CPU register into a peripheral device register its bytes will be reversed.

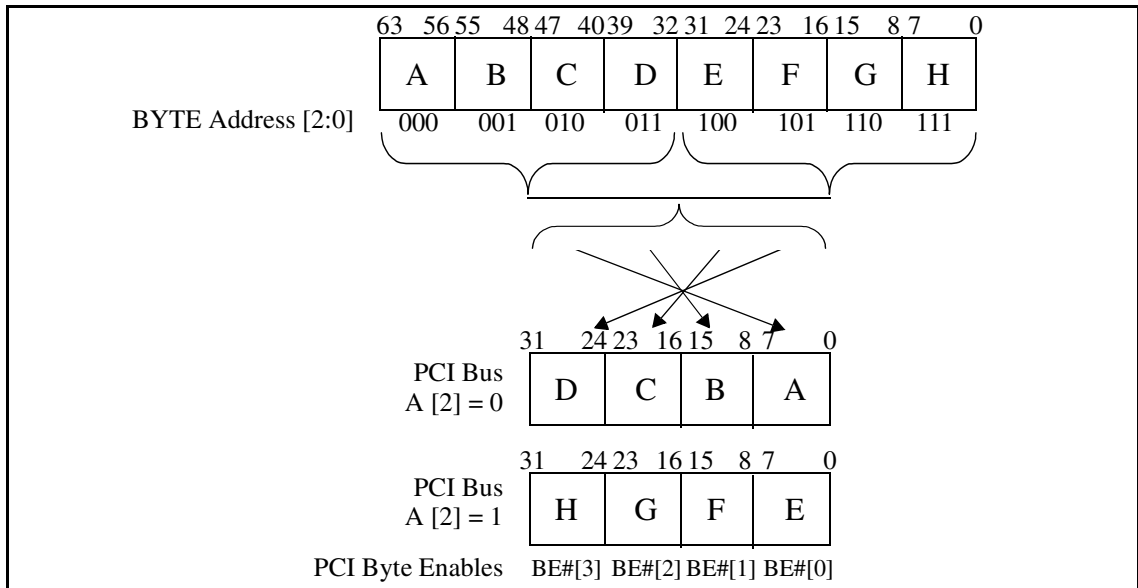


Figure 39: Match Byte Lane Endian Policy

If a sequence of bytes is moved through the interface the address of each byte is maintained, so the order of the sequence is maintained between the PCI or HyperTransport device and memory. Therefore this is the correct endian policy to use for most DMA transfers.

This policy is also the correct one to use for configuration and control register accesses if the software is written to explicitly endian swap the values in big-endian systems. However, since most drivers will be ported from little endian systems they are unlikely to have the endian swap code. It will normally be better to avoid the software overhead and use the match bits policy for control functions.

## BIG ENDIAN SYSTEM: MATCH BIT LANES

The match bit lanes endian policy will match the bit numbers of 32 bit values on either side of the interface. In byte lane terms this is an endian swap of the data. Consequently a 32 bit value that is written into a PCI register from the CPU will have the same interpretation. This policy is therefore the correct one to use when setting up configuration registers, passing addresses, and most control functions.

The policy is illustrated in Figure 40. Compared to Figure 39 on page 202, the bit ordering is maintained going from the system to the PCI. The PCI byte enables are generated from the inverse of the bottom two system address bits reflecting the different byte significance between the two sides of the interface.

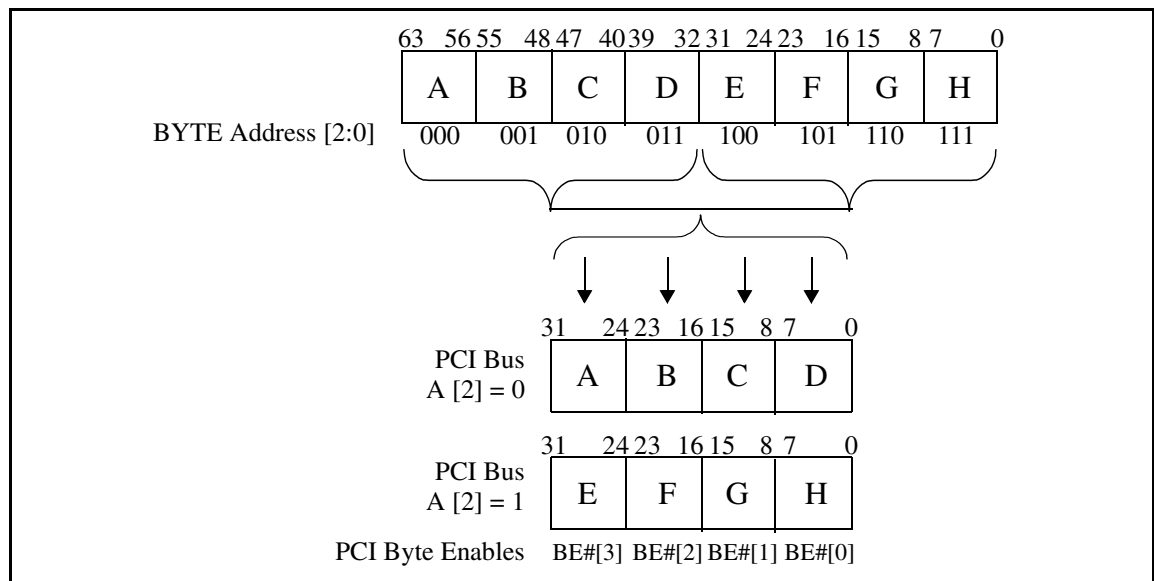


Figure 40: Match Bit Lane Endian Policy

By matching the bit number in 32 bit words, the bits that the CPU uses will match with the register descriptions in the PCI peripheral data sheet. Similarly, an address written into a control register from a CPU register will have the same meaning. Note that the situation is more complicated for 64-bit accesses (see below), but these are forbidden by the HyperTransport specification for configuration accesses to the HyperTransport space, and forbidden by the bridge for accesses to the PCI configuration registers.

If a byte access is performed the programmer will get the expected result for a big-endian context, i.e. the most significant bits of the register will be at the lowest CPU byte address. Similarly a 16 bit access will be mapped in the way expected for a big-endian machine. A sixty-four bit access performed by the CPU will be split into two thirty two bit accesses, again the data will end up in the positions consistent with a big-endian model.

The problem with this policy is that the byte memory ordering is scrambled since the byte lanes are swapped. This affects bulk data transfer, where the device on the PCI or HyperTransport assumes that the order on the byte lanes matches the memory order with BE#[0] associated with the lowest memory address. Therefore this mapping is unlikely to be useful for devices doing DMA accesses into the system memory. It is useful for access to control registers (where it will save endian swapping in software). The match bits policy can also be used if the peripheral can be configured for big endian DMA operation.

As an example consider accessing control registers with the same layout as the standard PCI header (this layout is used only as a familiar one for the example, the interface does not allow 64 bit accesses in configuration space so this cannot be used directly). Looking at the fields at offsets +0 and +4 with a big endian viewpoint the expected behavior for a 64 bit read should give the data at low memory address in the high (big) end of the register:

```
63:48 Device ID
47:32 Vendor ID
31:16 Status
15:0  Command
```

This is what a 64 bit read using the Match Bits policy will give. Looking at a PCI 64 bit BAR instead of the control registers (or equivalently a 64 bit DMA address register) shows that 64 bit accesses will not work directly for access to 64 bit addresses in HyperTransport peripherals. A 64 bit BAR would span two 32 bit words, but has the low address bits in the first word and the high address bits in the second word. Looking at this picture with the big endian viewpoint (as above) the 64 bit read will give:

```
63:32 Low address bits
31:0  High address bits
```

This is word swapped compared to what would be ideal, but is still consistent with the 64 bit world-view because it was originally defined as two 32 bit quantities.

## VIEWING ENDIAN POLICY AS AN OPTIMIZATION

One way to view the endian policy selection is as a way to optimize code. This section considers the software viewpoint assuming a portable operating system is being used.

To summarize the issue: The PCI/HyperTransport bus and peripherals are little endian, thus when storing a 32 bit value in a register the least significant byte (i.e. bits [7:0]) are assigned to the lowest memory address. When an access is done from a big endian CPU, which stores a 32 bit value with the most significant byte in the lowest memory address, the data appears swapped. Consider a 32 bit control register in the PCI device that contains the 32 bit value 32'h12345678, on the PCI bus the byte containing '78' will be on the byte lane with the lowest memory address marked with CBE[0] asserted. If the CPU does a read it will place the lowest memory address into the high bits of the register so will see the 32 bit value 32'h78563412.

In most multi-platform operating systems the first cut way of fixing this is to put an endian swap in if required:

```
#ifdef BIG_ENDIAN /* Need to swap the bytes */
    result = endian_swap(*pci_reg_address);
#else /* on a LITTLE_ENDIAN system can read directly */
    result = *pci_reg_address;
#endif
```

This will work as expected using the Match Bytes access area of the PCI/HyperTransport regions. The Match Bits space allows the hardware to automatically do the swapping. A single physical address bit (bit [29] in the standard memory mapped I/O space) is set to switch from Match Bytes to Match Bits. So the code can become:

```
#ifdef BIG_ENDIAN /* Get hardware to swap the bytes */
    result = *(pci_reg_address | BYTES_TO_BITS_ADDRESS_BIT);
#else /* on a LITTLE_ENDIAN system can read directly */
    result = *pci_reg_address;
#endif
```



This is an optimization because the endian swap is several instructions compared to the single OR (and in many cases all of the control registers need the swap so the base address of the control block can just be offset once; or if all the control registers are mapped in a single TLB entry and the data accesses in another then the virtual to physical mapping could set the Match Bits address bit for control and not for the data ranges).

The standard address is the Match Bytes since this makes memory order correct and thus causes unoptimized (but endian aware) code to work correctly.

## ACCESSING THE SIBYTE FROM PCI DEVICES

The normal PCI Base Address Register (BAR) access scheme is used to accept accesses from the PCI bus and map them to internal accesses. The PCI bus uses 32 bit addresses. To allow PCI masters to do transfers to memory, other devices in the part and peer-to-peer transfers to devices on the HyperTransport a full 40 bit address must be constructed. With the exception of expansion memory space and expansion HyperTransport space the part places everything in the low 4G of the address space (physical address bits [39:32] are zero) allowing a direct map from the 32 bit address on the PCI bus. In addition there is a map table that allows each of sixteen 1 MB regions of PCI space to map to a full 40 bit address anywhere in the internal or HyperTransport space.

The PCI controller has a type 0 (device) configuration header. This allows for up to 6 BARs, 5 are used when the interface is in Host Mode and 3 are used when the interface is in Device Mode. In addition a PCI expansion ROM may be accessed through its special BAR.

Table 123 shows the BARs and the internal addresses that they map to. In Host Mode they default to the values shown and are enabled following reset (if transparent access from PCI is all that is needed then no changes are required by the configuration code). When the interface is run in Device Mode the BARs will be setup by the host (using addresses in the host space). Note that the table shows the addresses. Since the spaces are prefetchable the value in the PCI BAR register will have bit 2 set.

**Table 123: PCI Base Address Register Use**

Reg	Offset in Config Header	PCI Size	Default PCI Base Address in Host Mode	PCI Base Address in Device Mode	Internal Base Address	Internal Device
BAR0	+10	16M	6000_0000 (R/O)	xx00_0000	Table Lookup	Region controlled by map table.
BAR1	+14	0	0000_0000	0000_0000	-	Reserved
BAR2	+18	4K	7000_0000 (R/O)	xxxx_x000	00_1002_1xxx	Mailbox CPU 0. Match Bytes mode is used if the system is configured Big Endian.
BAR3	+1c	4K	7100_0000 R/O	xxxx_x000	00_1002_3xxx	Mailbox CPU 1. Match Bytes mode is used if the system is configured Big Endian.
BAR4	+20	1G	0000_0000 (R/O)	Reserved	00_0000_0000	Memory below PCI space (PCI a[29] indicates endian policy).
BAR5	+24	2G	8000_0000 (R/O)	Reserved	00_8000_0000	Memory above PCI space (PCI a[29] indicates endian policy).
ROM	+30	64K	7300_0000 (R/O)	xxxx_0000	00_1FD0_xxxx	Expansion ROM. Match Bytes mode is used if the system is configured Big Endian.

Addresses on the PCI that match in BAR0 are mapped into internal addresses using a mapping table illustrated in Figure 41 on page 206. This allows a PCI device to access any area of in the internal address map, and can be used to ensure PCI devices can only access specified memory areas. The 16 MB region is divided into 1 MB chunks. PCI Address bits [23:20] are used to index into a table which provides address bits [39:20] of the internal address to use and an enable bit. The low bits [19:0] for the internal address are copied directly from the PCI address. If the table entry is enabled then the request on the PCI bus is accepted (DEVSEL# asserted) and the transaction is made to the internal address. If the table entry is not enabled then the bus request will not be accepted (DEVSEL# will not be asserted) and the requester will see an error. There is also a bit in the entry that controls whether the access is sent to the HyperTransport fabric or used internally (this bit overrides the destination implied by the address, so software must take care to set it correctly). If the bit is set to send the request internally and the address is in the HyperTransport range the result is UNPREDICTABLE, but in most cases a target abort error will be returned. If the bit is set to send the access to the HyperTransport and the address is in the internal range then the result depends on the devices on the HyperTransport, in most cases the request will reach the end of chain without being accepted, an NXA error will be returned and be turned in to a target abort, but if there is a host at the other end of the chain then it may unexpectedly reply. If the request is sent in to the part then two more bits in the map entry set the endian policy to be used for the transfer and the state for the level 2 cache allocate on miss (A\_L2CA) flag that should accompany the transfer. The mapping table is placed in configuration registers h44 - h80 in the PCI configuration header, however they can only be set by accesses from the ZBbus side of the bridge (even when in Device Mode).

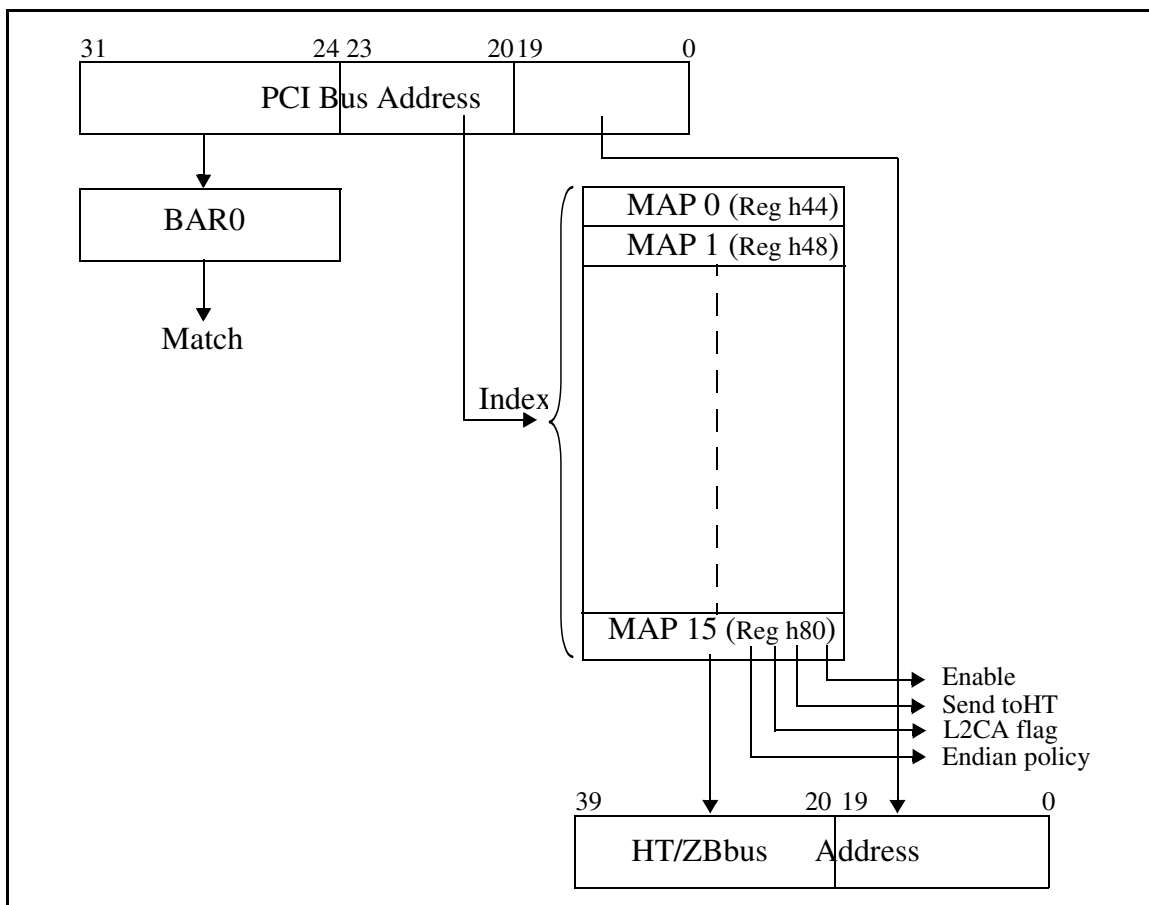


Figure 41: PCI BAR0 Address Mapping Table



BAR2 and BAR3 provide access to the CPU mailbox registers (see [Section: "Mailbox Registers" on page 46](#)). The PCI device can read the current value of the mailbox and do writes to set bits in the mailbox, but it is not able to clear the mailbox. The PCI device can therefore raise interrupts to the CPUs, send small messages, and read status, but is prevented from directly clearing any CPU interrupts. This access is provided in both Host and Device Modes (but will most likely be used in Device Mode).

BAR4 is only available in Host Mode. There is an enable/disable bit for this BAR in configuration register h40, it is enabled after system reset. The BAR provides a direct map from the PCI to the bottom 512 MB of the system memory using the match byte lane endian policy, and (by setting a[29]) a map from the PCI to the bottom 512 MB using the match bit lane policy. (This area covers the low memory area, internal devices and some of the generic bus space). The interface will use pre-fetches for any reads in this area.

BAR5 is only available in Host Mode. There is an enable/disable bit for this BAR in configuration register h40, it is enabled after system reset. The BAR provides a direct map from the PCI to the two 512 MB sections of memory 00\_8000\_0000 - 00\_9FFF\_FFFF and 00\_C000\_0000 - 00\_DFFF\_FFFF using the match byte lane endian policy, and (by setting a[29]) a map from the PCI to the same two areas using the match bit lane policy. (These areas cover the rest of the memory that is addressable with 32 bit addresses, they also cover the cache test space and special access addresses that should not be accessed by normal PCI devices).

The PCI expansion ROM access BAR is mapped to 64 KB of the space that is allocated to the boot ROM at reset time. This allows an external PCI host to access an expansion ROM with no setup required by the CPU (preventing reset ordering problems). The bottom bit of the BAR must be set to enable the region (as required by the PCI specification).

Reads on the PCI do not explicitly indicate a length. To optimize performance the PCI bridge will always prefetch a cache line (i.e. do a read on the ZBbus with all byte enables set) when a read to the part comes in from a PCI device. When memory accesses are made the I/O bridge will do a coherent read and this will work as expected. If an access is made (through BAR4 or BAR5 or the ROM BAR or mapped to through BAR0) to any non-memory space in the part the I/O bridge will do an uncacheable read with all byte enables set. The result of this read will depend on the device accessed: memories on the generic bus will be read correctly, reads of internal registers that are cacheline aligned (i.e. a[5:3]=0) will work as expected, but reads of internal registers that are not cacheline aligned will give UNPREDICTABLE results.

Writes on the PCI do not explicitly indicate a length. The PCI bridge will gather a burst write targeted at the system into a buffer and do a write with the appropriate byte enables set. If the write is to memory space and all the byte enables are set a write-invalidate is sent on the ZBbus to write the data and invalidate old copies that are in any caches. If the write is to memory space and less than the full 32 byte enables are set the I/O bridge will perform a coherent read exclusive to get ownership of the block, the write data will be merged and the block written back. The bridge acts as a full participant in the coherency protocol, so if there is any request for the line it will respond as owner and will provide the data. If the write is to non-memory space an uncacheable write is performed on the ZBbus with the byte enables reflecting the bytes that were written in the PCI transaction. Writes to internal registers or the generic bus will therefore work as expected.

[Figure 42 on page 209](#) shows the memory map that is seen from the PCI bus after reset on with the interface configured in PCI Host Mode.

Once the PCI interface has accepted a transaction that hits in one of the BARs (by asserting P\_DEVSEL\_L) it will complete or be terminated with a Target Abort. There are four classes of problems that will cause Target Abort:

- 1 An error is detected on the PCI bus. This could be a parity error, an illegal opcode or illegal byte masks.
- 2 An error is returned from a read request that has been sent to the ZBbus (this will be logged by the Bus Watcher) or to the HyperTransport interface (this will be logged in the interface CSRs).
- 3 The request hit in a BAR and targets the HyperTransport interface but the ptp\_en bit in the Feature Control register is clear, disabling peer-to-peer transfers from PCI to HT.
- 4 The request hit in a BAR (in particular BAR0) and was targeted at the ZBbus, but the (mapped) address targets a PCI or HyperTransport space (i.e. one serviced by I/O Bridge 0).

An access that hits in BAR0, but to a disabled map entry, will not be accepted (P\_DEVSEL\_L will not be asserted). This results in a Master Abort.

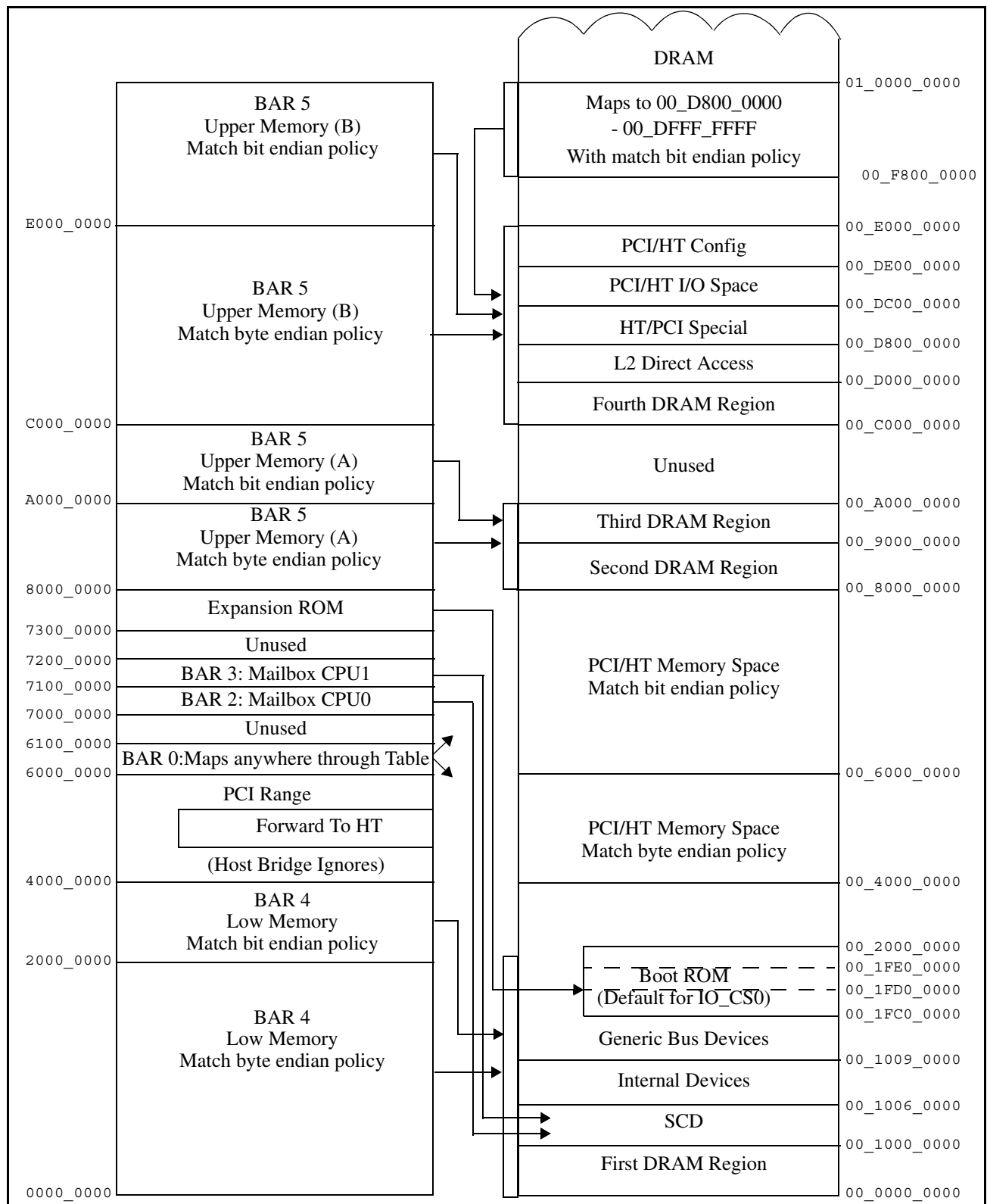


Figure 42: Default Host Mode Memory Map from PCI Bus Master

## ACCESSING THE SiByte FROM HYPERTRANSPORT DEVICES

The mapping from HyperTransport addresses to those in the part is much simpler than from PCI. It is shown in Figure 43.

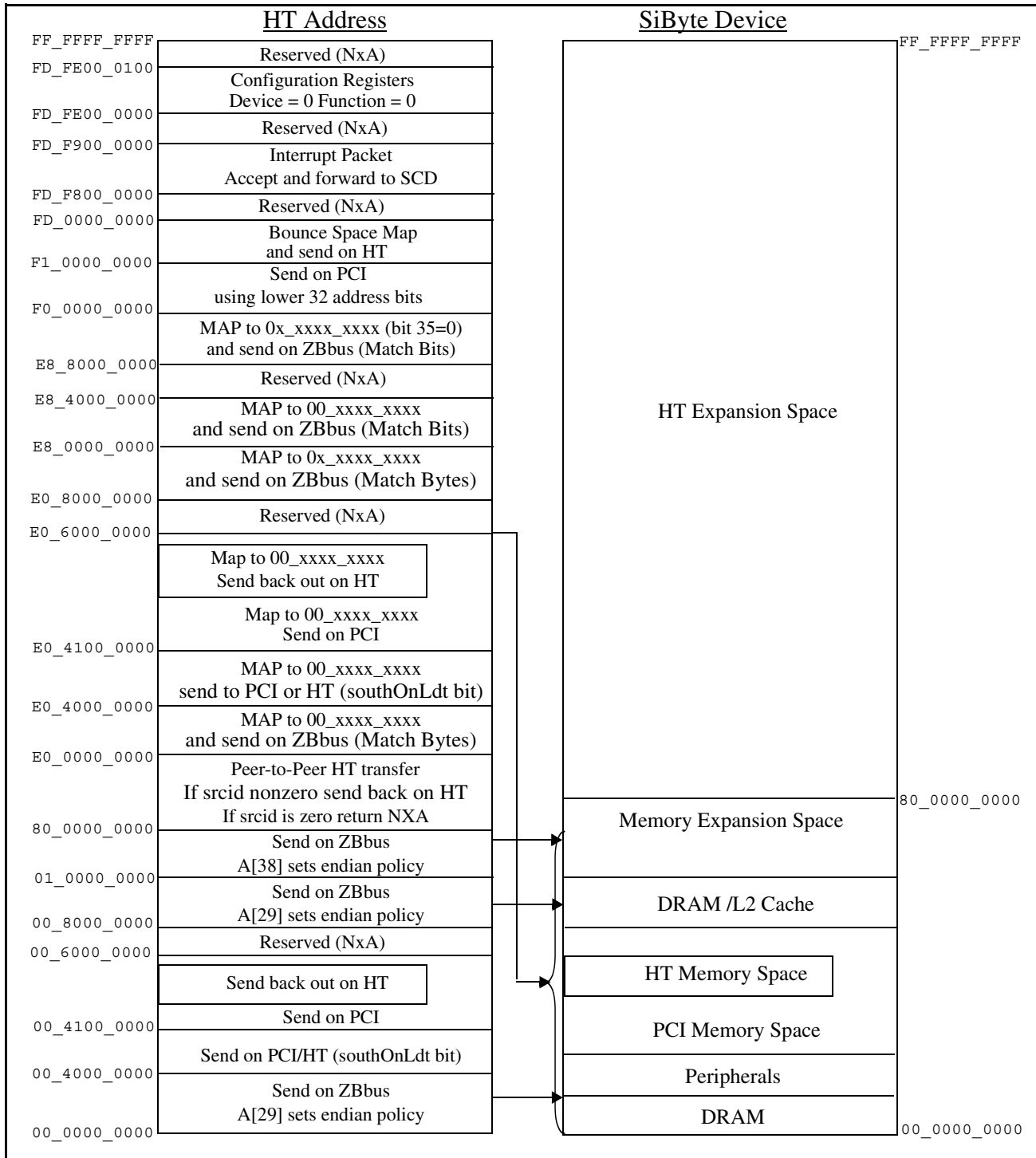


Figure 43: Memory Map From HyperTransport Device

The HyperTransport interface always runs as a host bridge, and has a PCI type 1 configuration header. The base and limit registers in the header define the region of the PCI/HyperTransport memory space that is allocated to devices on the HyperTransport fabric (these devices can be accessed by a system that uses 32 bit addresses). The value of the southOnLDT configuration bit determines whether the 00\_4000\_0000 to 00\_40FF\_FFFF range is allocated to the HyperTransport or PCI (note that these addresses are not changed when they are forwarded, nor will the COMPAT bit be set if the request is sent back out on the HT). In addition the HyperTransport is allocated the top half of the full 40 bit physical address space.

If the Bus Master Enable bit is clear in the HyperTransport interface bridge header the only requests that will be accepted from the HyperTransport fabric are to the configuration space. All other requests will be discarded or receive an NxA response. If this bit is set the bridge will accept transactions to be forwarded to the ZBbus, PCI bridge and back to the HyperTransport.

Peer to peer operations on the HyperTransport fabric must pass through the host bridge, any incoming addresses that match a HyperTransport region are therefore forwarded to the outgoing link. Requests with srcid=0 to HyperTransport peer-to-peer are responded to with an NXA (non existent address) error since they must have come all the way from a host bridge at the other end of the link and not been accepted by any of the devices on the link.

Requests that match a PCI address, either using standard addresses in the low 32 bit range of the address map or in the special F0\_0000\_0000 - F0\_FFFF\_FFFF range, are forwarded to the PCI interface and a PCI cycle is run. A PCI cycle is also run for accesses to the southbridge space if the southOnLDT bit is clear. The data is passed directly between the two interfaces regardless of system endian settings. Only a single HyperTransport-PCI read will be issued at a time. The PCI interface will not respond to requests that it generates, any access forwarded from the HyperTransport fabric that match a PCI address with destination inside the part will result in an error.

Requests that match the ZBbus space will be forwarded through I/O Bridge 0 and run as internal cycles. These requests use one address bit to select the endian mode for the transfer. If the system is big endian then setting the address bit will result in the match bit lane policy and leaving it clear will give the match byte lane policy. If the isochronous bit is set in a HyperTransport request that is sent to the ZBbus, it will have the L2 cacheable bit set in the command. This is particularly useful if the HyperTransport device is writing data that one of the CPUs will need immediate access to. If the address is to the memory range then a cacheable coherent access is made. If the access is a write of smaller than a cache line then the I/O bridge will get the current data and ownership of the block by using a read exclusive command, the new data from the write is merged into the block and it is written back to memory or L2. If the address is not in the memory range then an uncacheable access is used, with the appropriate byte enables set.

If an error is returned from a ZBbus read the MstrAbortMode bit in the Bridge Control Register determines the error reported to the HyperTransport fabric. If the bit is set a HyperTransport error with the NxA bit clear is generated for all ZBbus error types. If the bit is clear, ZBbus Bus Errors or Fatal Errors cause UNPREDICTABLE data marked valid to be returned to the HyperTransport fabric, but other ZBbus errors are still returned as HyperTransport errors with the NxA bit clear. Non-posted writes will never return an error, since they are converted to posted writes internally.

The interface does not accept cycles based on the COMPAT bit being set (even if the southOnLDT flag is clear), it therefore cannot be used to forward from the HyperTransport to a southbridge on the PCI bus.

## Force Isochronous Mode Address Range

Allowing the isochronous bit to set the L2 cacheable bit on the ZBbus is useful for new devices that support setting the Isochronous bit during requests, but cannot be used by devices that do not allow control of the Isochronous bit or that are bridged from other buses by a bridge that will not translate requests appropriately. Therefore for revisions RevId 3 or greater of the HT interface there is an additional BAR (the IsocBAR) and mask (the IsocIgnMask) that is used to specify a range of addresses that will be forced to behave as if the Isochronous bit is set. The function is enabled when bit [0] of the IsocBAR is set. The IsocIgnMask is used to indicate which bits should be ignored when the address from a HT request is compared with the IsocBAR. If the other bits match between the received address and the IsocBAR then the isochronous bit in the request is forced to be set and the L2CA flag will be set on the ZBbus. The isochronous bit is forced if:

$$(\text{htAddr}[35:5] \mid \text{IsocIgnMask}[31:1]) == (\text{IsocBAR}[31:1] \mid \text{IsocIgnMask}[31:1])$$

In addition to allowing the PCI BAR style power-of-two windows this allows for more complicated situations. For example consider the configuration:

```
IsocBAR =      32'h08120001
IsocIgnMask = 32'h0200FF02
```

This defines a 1MByte region with base 00\_8120\_0000 in which the Isochronous bit will be forced for the first 64 bytes in every 4K (since addr[11:6] must be zero and addr[5] is ignored). Bit 25 of the IsocIgnMask is set to ignore address[29] since that is only used to set the endian policy. Assuming buffer alignment this could be used to allow packet headers to be cached and bodies to be written to memory.

Note that bits [39:36] are always ignored in the comparison so that the Ex\_XXXX\_XXXX address will have the same L2CA behaviour as the 0x\_XXXX\_XXXX address that it maps to.

## ACCESSING THE SiBYTE FROM A SiBYTE ON A DOUBLE HOSTED CHAIN

The interface provides limited support for double-hosted HyperTransport chains. This is primarily intended for interconnection of two BCM1250s, two BCM1125Hs or a BCM1250 to a BCM1125H. The HT configuration registers are accessible from the HyperTransport fabric as device=0 function=0. At system startup one part must be designated the master, and the other the slave (this could be done by using the reset configuration resistors that are allocated for software use, or by having different bootstrap code on the two devices). The master will configure the chain as described in the HyperTransport specification.

The two parts can communicate over the fabric. Since they have the same address map, a mapping must be provided to allow each access to the other's memory space. This is done in a simple manner using a direct map. Accesses from the ZBbus to the region of the address space E0\_0000\_0000 - EF\_FFFF\_FFFF will be sent to the HyperTransport fabric (since it is in region N in [Figure 37 on page 193](#)). This range will be accepted by the host bridge on the part on the other end of the fabric, and direct mapped into the low address space (00\_0000\_0000 - 0F\_FFFF\_FFFF) and a second level decode performed to determine if the request should be forwarded to the ZBbus, the local PCI or sent back out on the HyperTransport. Accesses to these addresses pass over the I/O fabric and therefore outside the coherency domain. Software will need to manage the coherency of any data between the two parts. The space should be mapped either uncacheable or cacheable non-coherent in the CPU, and will not be cached in the L2 cache.



Accesses from the ZBbus of one part to the F0\_0000\_0000 - F0\_FFFF\_FFFF region of the address map will be sent over the HyperTransport fabric, accepted by the second part and direct mapped to a PCI access on the second device.

There is a restriction on configuration accesses made from one part to another over the HyperTransport link. Both reads and writes to the configuration registers must be 32 bit operations (a HyperTransport full unmasked double-word), if bigger or smaller operations are performed reads will return UNPREDICTABLE values and writes will leave the configuration register in an UNPREDICTABLE state.

Interface RevId 3 and later support the ActAsSlave function defined in the HyperTransport 1.03 specification, which is of use in some double-hosted configurations.

## HyperTransport Bounce Space

In a double-ended chain is used with a master host on one end and a slave on the other, peripherals will identify the master device and direct all transactions to it. Anything that matches a memory address will be accepted by the master. The slave has an identical memory map, so direct transactions will never reach it. The Bounce space provides a way for the peripheral to bounce transactions to the slave via the master.

Any address received in the range F1\_0000\_0000 - FB\_FFFF\_FFFF will be sent back out on the HyperTransport fabric with the address mapped to 01\_0000\_0000 - 0B\_FFFF\_FFFF (the top 4 bits are cleared) and any address received in the range FC\_0000\_0000 - FC\_FFFF\_FFFF will be sent back out on the HyperTransport fabric with the address mapped to 00\_0000\_0000 - 00\_FFFF\_FFFF (the top 8 bits are cleared). This allows the device to access the memory space in the slave.

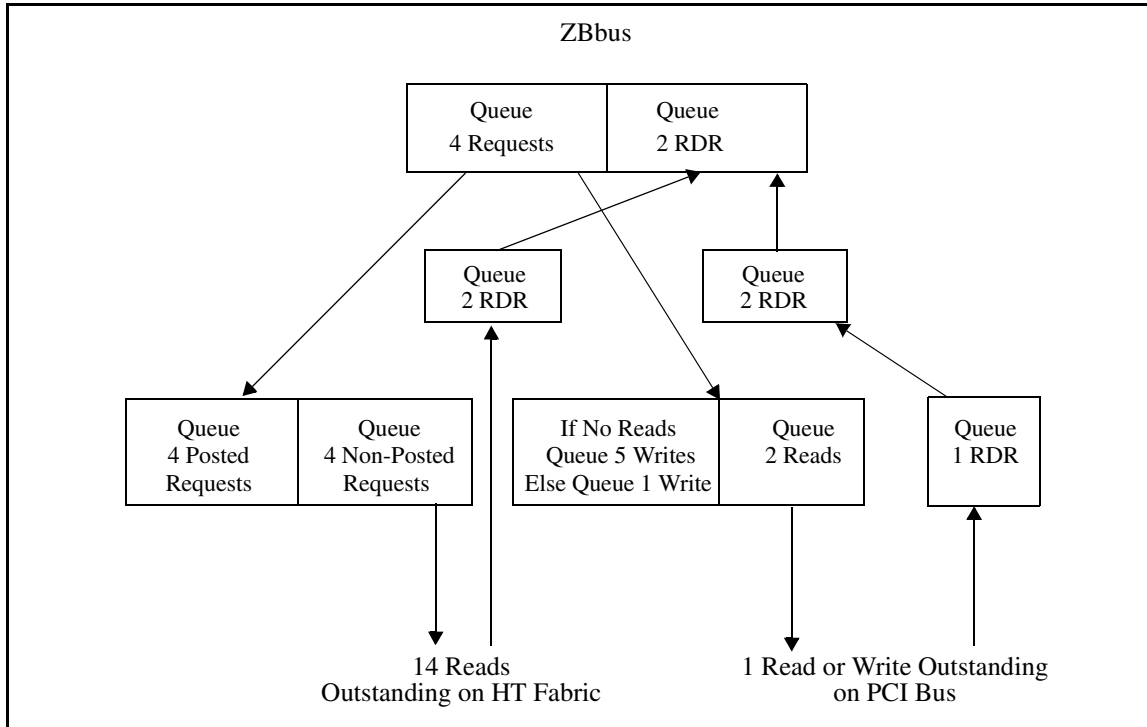
## PERFORMANCE OF THE PCI AND HYPERTRANSPORT INTERFACES

The performance of the PCI and HyperTransport interfaces is strongly dependent on the size of the transfer. The best case is when 32 byte cache blocks are moved. If smaller writes are done from the PCI or HyperTransport into the part then the I/O bridge 0 has to read the cache block exclusive (to fetch any modifications that may be in caches), modify it and write it back.

Writes from the ZBbus to either interface are always posted. Reads consist of two stages: the Read Request, which is a non-posted request; and the Read-Data-Return (RDR) which is a response.

**ACCESSES FROM THE SiBYTE TO THE PCI OR HYPERTRANSPORT**

The data path from the ZBbus to the PCI or HyperTransport is split into three sections. The interface to the ZBbus includes queues of requests and returning data. Inside the I/O bridge there are additional buffers in the routing path to the interface block. Finally, there are additional queues and tables of outstanding requests in the interface blocks.



**Figure 44: Buffers Used for Accesses from the ZBbus to PCI and HyperTransport**

Figure 44 illustrates these queues. The request queues always maintain transactions in the order that their A-phase happened on the ZBbus, for uncached accesses this matches the program order of the loads and stores. Once the requests are on the HyperTransport fabric or PCI bus the normal PCI ordering rules apply, so writes can pass reads; for uncacheable accesses from the SB-1 cores this is not a problem because the CPU will not issue an uncacheable write while there are uncacheable reads outstanding. The internal connections will maintain the order of reads and the SB-1 CPU supports multiple outstanding uncached reads to allow streaming. There is one case where this can cause problems: in the HyperTransport fabric (due to the ordering of the chain), or PCI with bridges (using delayed reads) reads to different devices may reach their destinations in a different order than they were issued. If the reads have side effects that are signalled on a back-channel between the two devices the second read issued by the CPU may or may not see the effect of the first (this is always a danger with back-channels). If this rare situation is encountered in a system then the program must include a SYNC instruction between the reads to ensure they complete in the expected order.

Writes from the ZBbus are always posted and need no reply, they pass through the request queues and are issued to the I/O bus. Reads also pass through the queues to be issued, but also have associated Read-Data-Return (RDR) passing back up through the bridge to the ZBbus.

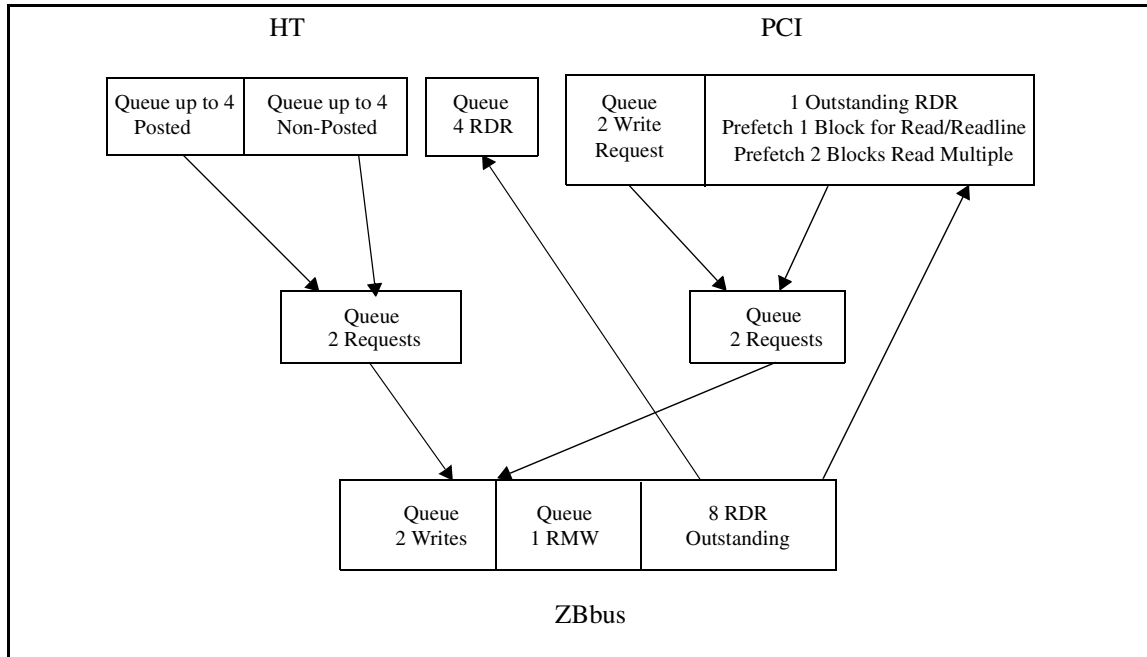
The HyperTransport fabric uses split transaction reads and the interface supports 14 reads outstanding on the fabric. When a HyperTransport response arrives it is matched to the associated request and passes back through the RDR path. If a masked read (a read that is not a multiple of aligned 32-bit words) is encountered then no further reads will be accepted by the interface until it has completed (the `ldl` and `ldr` instructions can result in reads of 5, 6 or 7 bytes, these will be split into two masked reads that can both be outstanding on the link and must both complete before the RDR is returned).

The PCI interface can hold one read pending and have one read in flight on the bus. If a read is retried excessively (more than the number of times set in the Retry Timeout register described in [Table 132 on page 240](#)) the interface will allow any writes or RDRs to pass before retrying the read. If a read exceeds the retry timeout on the second attempt it will be abandoned and a bus error reported.

The restriction on only having one read pending on the PCI bus will limit the performance. Therefore in interface with `RevId` 3 or greater a prefetch mechanism is added. If prefetching is enabled in the PCI Adaptive Extend Register then a full cache block ZBbus read will cause 2 or 4 cache blocks to be prefetched from the PCI. This is particularly useful when the Data Mover is transferring a large block from the PCI. If a read request from the ZBbus is for a full cacheline at an aligned address and the prefetch is enabled and the address is not greater than (4k page boundary minus 4 cacheline addresses), then a read of 2 or 4 cachelines is issued to the PCI. If the next request from the ZBbus is also a full cacheline read to the next consecutive cacheline address, the prefetched data will be returned. This pattern will repeat until all the prefetched data are returned. If the next request does not meet this criteria, the prefetched data will be flushed and the new request will be sent to the PCI bus. (Note that since the prefetched data could be fetched and discarded it is possible for the prefetching to lower performance.) In the error case where the prefetched data (the additional data that is not used by the initial transaction) is timed out for too many retries, the interface will not reissue the prefetch reads. This may cause the device being accessed to deadlock, to prevent this the retry limit should be set high to ensure only actual errors will be captured.

**ACCESSES FROM THE HYPERTRANSPORT TO THE SIBYTE**

Figure 45 shows the queuing for DMA accesses from the HyperTransport fabric and PCI bus into the part.



**Figure 45: Buffers Used for DMA Accesses from the PCI and HyperTransport**

Posted writes from the HyperTransport fabric flow through the queues in the I/O bridge and ZBbus interface and are posted on the ZBbus. Non-posted writes from the HyperTransport become posted writes internally (since the architecture does not have any internal non-posted writes). An acknowledgment response is always sent on the HyperTransport when the write leaves the HyperTransport interface and is inserted in the I/O bridge queue. Since it has become a posted write it may now pass reads. An error response will never be generated.

Reads from the HyperTransport fabric pass through the queues and are sent onto the ZBbus. The I/O bridge interface can have 8 reads outstanding on the ZBbus (including those from both PCI and HyperTransport). When all 8 entries in the RDR buffer are in use further reads are blocked from moving into the ZBbus request queue. The read-data-return is sent back to the CPU response queue in the HyperTransport interface. Read requests from the HyperTransport that are longer than 32 bytes will result in two reads being issued towards the ZBbus.

## ACCESSES FROM THE PCI TO THE SiBYTE

The path for the PCI to access the part is shown in [Figure 45 on page 216](#). It is similar to the HyperTransport flow.

Writes from the PCI are always posted into the part, they flow through the queues shown in the figure.

The PCI interface will accept one outstanding read from the PCI bus. If this read is not serviced after 16 PCI cycles (the debug is programmable as described below) it will be disconnected to allow other PCI devices to make progress (turning it into a Delayed Read Request). Subsequent reads from the PCI bus into the part will be immediately disconnected until the first has been satisfied. PCI reads do not include a length in the request, so a heuristic is used to improve performance. If the read uses a PCI Read or ReadLine command the PCI interface will initially request a single (32 byte) block from the ZBbus interface, when this returns it will be used to satisfy up to 32 bytes of the request. If the request is still in progress an additional request will be made to the ZBbus interface for the next cache block. If the ReadMultiple PCI command is used the interface will immediately launch two read requests for consecutive cache blocks, and will continue to prefetch blocks as the data is sent to the PCI. The reads share the RDR buffer in the ZBbus interface with reads from the HyperTransport fabric.

If a request from the PCI bus requests a burst mode other than Linear Incrementing the interface will disconnect the transfer after a single data phase.

Memory space reads smaller than four bytes will result in the appropriate byte enables being asserted on the PCI bus unless the `dis_memrd_be` bit is set in the PCI Adaptive Extend register. If this bit is set all memory space reads will have all four byte enables asserted (the additional bytes received from the PCI will be ignored). If the `dis_memrd_be` bit is clear then any access to the PCI greater than 4 bytes should be an aligned multiple of four bytes. If the `dis_memrd_be` bit is set then any access may be safely done but the PCI device will always see an aligned multiple of four bytes.

### PCI Adaptive Retry

The PCI specification requires that if a request is not completed within 16 cycles the target should terminate the transaction with a retry to free the bus for other devices. It encourages devices that know they have a long latency to disconnect the transaction as early as possible to avoid dead cycles on the bus. On the host the latency of memory reads is hard to calculate because it depends on what other activity is in the system. Typically in a lightly loaded system reads will complete in 200-400 ns, which is well under the PCI timeout. By default the interface will not retry the transaction early, and will wait the full 16 cycles before issuing the retry.

The interface also has an adaptive retry policy, which is enabled by setting the `adapt_retry_en` bit in the Feature Control Register ([Table 133 on page 241](#)). The actual latency of read requests is measured and used to set the number of cycles that the interface waits before signalling the read should retry. [Table 124](#) shows how the retry delay is changed based on the current memory latency. Two consecutive samples must show the same trend in latency before the algorithm changes behavior.

The adaptive retry parameters are spread across the PCI Feature control register and the PCI Adaptive Extend register. The minimum value is a 3 bit field, nominal and maximum are 7 bit fields. This allows the values to be programmed well outside the compliant PCI range, but allows optimal selection to be made in embedded systems.

Table 124: Adaptive Retry Delay

Condition	New Delay Before Retry
Memory latency > max_tar_retry	retry_delay = min_tar_retry
retry_delay != min_tar_retry AND max_tar_retry > Memory latency > nom_tar_retry	retry_delay = max_tar_retry
retry_delay == min_tar_retry AND nom_tar_retry > Memory latency > min_tar_retry	retry_delay = nom_tar_retry

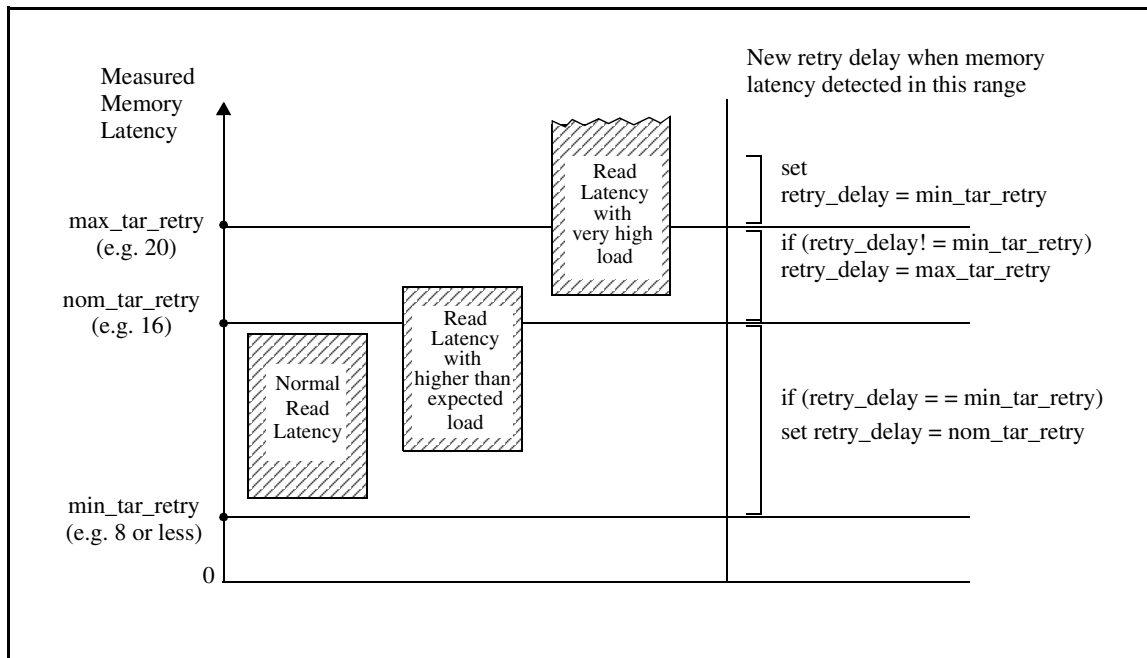


Figure 46: PCI Adaptive Retry Parameters

Figure 46 illustrates the algorithm. (The suggested delay values are appropriate for a host bridge, which is permitted to wait up to 32 cycles before issuing a retry, compared to only 16 cycles in a device). The retry delay is initialized to the value given in the **nom\_tar\_retry** register. This sets the normal retry timeout that should be used and should be set above the delay expected for the memory system. Most reads will be satisfied in this latency (falling in to the Normal Read Latency box in the figure), and the occasional long latency accesses will be disconnected and retried (this behavior is just like the standard case). If the system becomes busier and the latency increases slightly (moving the read latency into the higher than expected load box), to fall between the nominal and maximum values, rather than continually disconnecting transactions just before their data becomes available the adaptive policy will increase the retry delay to the maximum value. This prevents the PCI throughput falling catastrophically when the system experiences a load slightly higher than was expected. However, if the memory latency becomes larger than the maximum (the very high load box) then the retry delay is set to its minimum value causing requests to be rapidly disconnected. Once the overload has been detected the retry delay is left at the minimum until the memory latency returns to its normal range, when the retry delay will also be restored to its normal value.

## PEER-TO-PEER ACCESSES

The interface supports peer-to-peer accesses from devices on the PCI bus to devices on the HyperTransport fabric, and from devices on the HyperTransport fabric to devices on the PCI bus. Requests and data being transferred in this way do not travel on the ZBbus, they are directly routed through special buffers in the I/O Bridge 0. In the master bus interface of a peer-to-peer operation the queues are shared with DMA operations (from I/O devices into the ZBbus) and on the slave side the queues are shared with PIO operations (from the ZBbus to I/O devices).

Peer-to-peer operations in I/O space are not supported. Neither the PCI bus nor the HyperTransport bus will accept an incoming I/O space request.

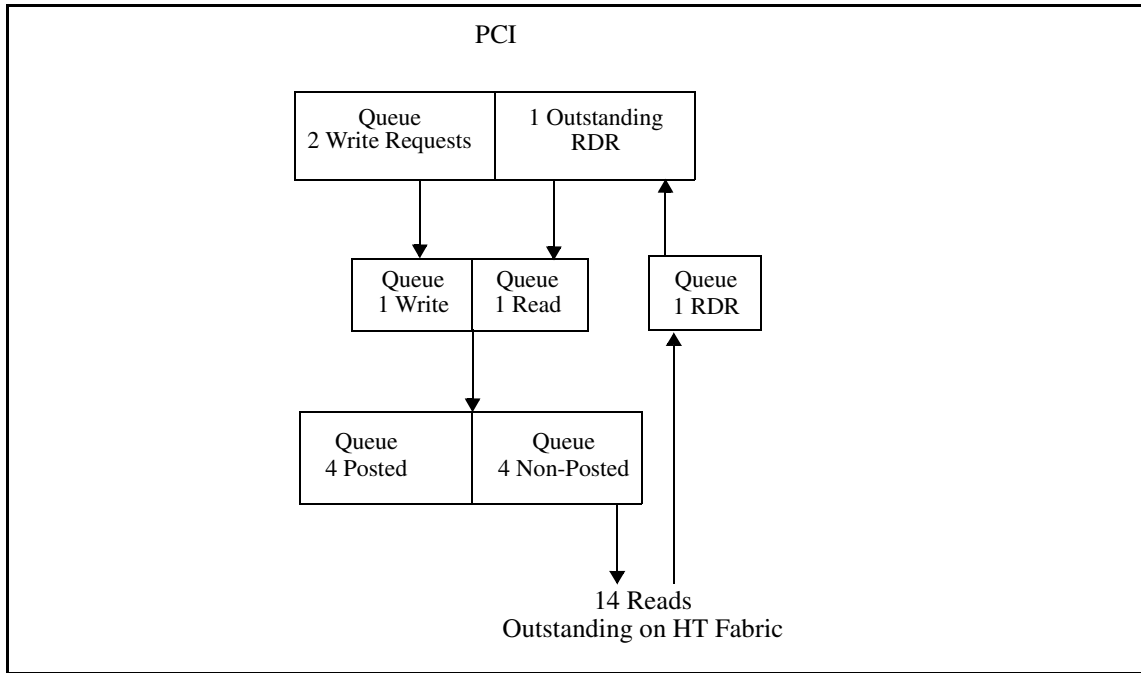
### PCI BUS TO HYPERTRANSPORT FABRIC

When the PCI interface is configured in Host Mode, requests on the PCI that fall into the secondary bus memory range specified in the HyperTransport bridge header will be accepted from the PCI and forwarded to the HyperTransport fabric. This forwarding is disabled by default, and must be enabled in the **Feature Control** register (offset h40 in the PCI configuration header). In addition requests that match BAR0 and have an enabled mapping with the send\_idt bit set are forwarded to the HyperTransport (the send\_idt bit overrides any destination implied by the address). The peer-to-peer bit in the **Feature Control** register must also be set to enable forwarding through the map table.

If the PCI interface is configured in Device Mode and the peer-to-peer bit is set in the **Feature Control** register the only requests that will be forwarded to the HyperTransport fabric are those that hit in BAR0 and have an enabled mapping with the send\_idt bit set. This allows the part to act as a non-transparent bridge (i.e. one with separate address spaces on each side and explicit routing between the sides) from the PCI host to the HyperTransport fabric. Note that in device mode the Feature Control register is written by the host on the PCI bus and cannot be accessed from the ZBbus, thus both the host (through Feature Control ptp\_en) and device (through BAR0 send\_idt) must agree for the PCI-HT access to be permitted.

Figure 47 shows the queues used in peer-to-peer operations with a PCI master.

Peer-to-peer writes will transfer exactly the number of bytes written and will always be posted to the HyperTransport fabric. They flow in their own channel from the PCI interface to the HyperTransport interface and may pass non-posted reads.



**Figure 47: Buffers Used for PCI to HyperTransport Peer-to-Peer Accesses**

Peer-to-peer reads always assume the destination address is prefetchable (this is required for performance, since the PCI bus does not give a good indication of the transfer size at the start of the transaction). Caution should be used when accessing registers using peer-to-peer operations. The same algorithm is used as for reads to the ZBbus, the PCI read is assumed prefetchable, and is a Read or ReadLine is causes single 32 byte request to the HyperTransport fabric while a ReadMultiple will launch two requests.

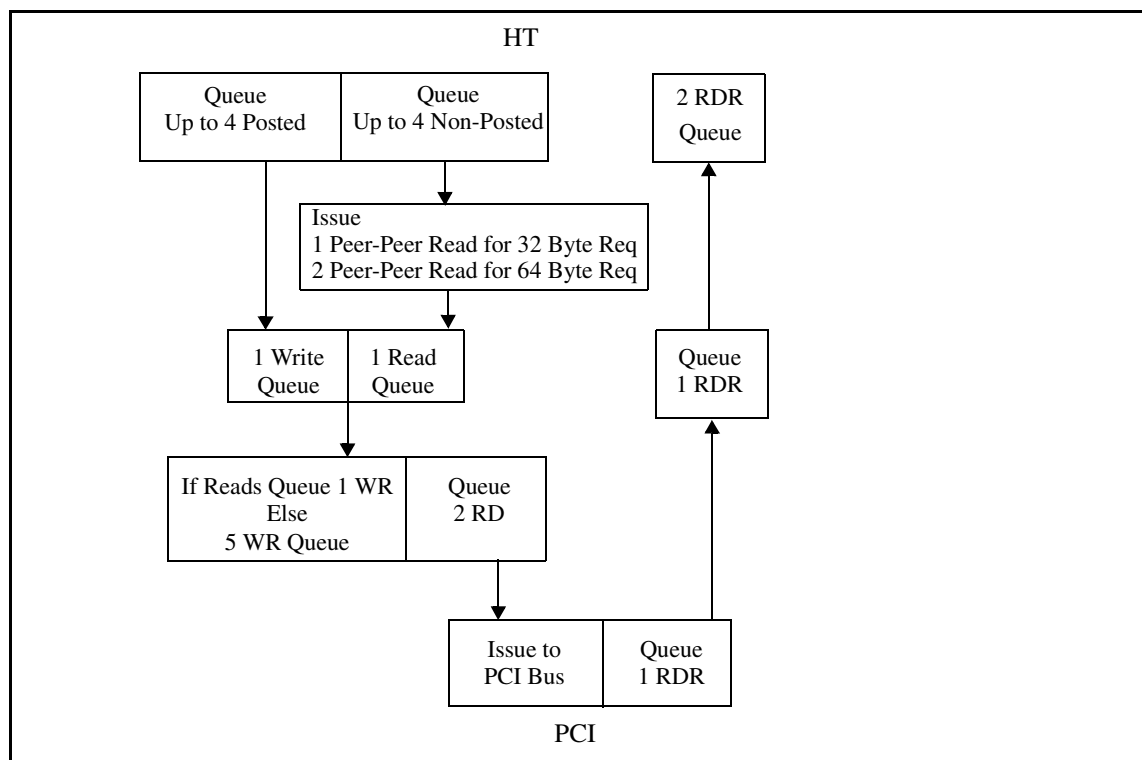


## HYPERTRANSPORT FABRIC TO PCI BUS

Requests received by the HyperTransport interface to any addresses that fall in the space allocated to the PCI memory space (regions A and C in [Figure 37 on page 193](#)) or that fall into the special double ended chain mapped address ranges (E0\_4000\_0000 - E0\_7FFF\_FFFF and F0\_0000\_0000 - F0\_FFFF\_FFFF in [Figure 43 on page 210](#)) will be forwarded to the PCI interface. Since the size of the transfer is specified at the start of a HyperTransport request the access to the PCI bus will always be for the exact number of bytes requested.

[Figure 48](#) shows the queues used in peer-to-peer operations with a HyperTransport master.

Peer-to-peer writes will always be posted to the PCI bus. Non-posted write requests will be acknowledged when they leave the HyperTransport interface. Writes flow in their own channel from the HyperTransport interface to the PCI interface and may pass non-posted reads.



**Figure 48: Buffers Used for HyperTransport to PCI Peer-to-Peer Accesses**

The HyperTransport interface can have two reads outstanding to the PCI interface. If a HyperTransport request is for more than 32 bytes both outstanding reads are needed to service it. Alternatively two smaller requests can proceed together. The buffering and PCI interface will maintain the ordering of the reads. The request path and RDR buffer in the PCI interface are shared with PIO and DMA requests, but there are dedicated return buffers for the peer-to-peer RDR.

## PCI ARBITER

The device incorporates a PCI bus arbiter that may be used when the interface is configured as the host bridge. The arbiter supports four external devices in addition to the host. Arbitration is done using a single-ring round robin scheme with the bus parked on the last granted device when there are no new requests. Except for the hotplug situation described below, the arbiter will wait for up to 32 PCI clocks for the master to assert frame before removing the grant (the "broken" master case of section 3.4.1 of the PCI specification), but requests from the master will continue to be responded to and the arbiter does not have a status register that records this being done.

The arbiter includes support for hotplug systems. When the hotplug\_en bit is set in the Additional Status and Control Register (at offset +88 in the PCI configuration registers) and P\_REQ\_L[3] is asserted, once P\_GNT\_L[3] is granted it will be held in the asserted (granted) state as long as P\_REQ\_L[3] is asserted, independent of higher priority requests. In addition to enabling hot plug systems, this can be used to solve some arbitration problems in some older southbridge devices.

## PCI INTERRUPTS

The PCI interface includes 4 active low interrupt lines P\_INTA\_L-P\_INTD\_L. When used as a host the interface will normally use all of these as the interrupt inputs from the other PCI devices (the PCI specification describes the recommended way of permuting these as they are connected to the INTA#-INTD# pins of slots or devices). In all cases the four lines are monitored as active low interrupt inputs and the results presented to the interrupt mapper as sources 56-59 (see [Table 22: "Interrupt Sources," on page 52](#)). Any of the inputs that are not used by PCI devices can therefore be used as general active low interrupt inputs.

The P\_INTA\_L signal can also be driven (as an open collector output) by the interface (note that when it is driven low the input logic will detect that it is low and raise system interrupt 56). This is intended for use in device mode to allow an interrupt to be driven to the host (see [Section: "Using the PCI in Device Mode" on page 232](#)). However, the output can still be driven when the interface is configured for host mode so a system that does not need four PCI interrupt inputs may choose to use the P\_INTB\_L-P\_INTD\_L for PCI use and use P\_INTA\_L as a general open collector I/O.

## HYPERTRANSPORT DIFFERENCES FROM REVISION 0.17 SPECIFICATION

This section lists the main differences between the HyperTransport Specification Revision 0.17 and the interface implemented in the BCM1250 and BCM1125H. Since there was only limited distribution of the 0.17 specification, most readers will find the next section (differences from revision 1.03) more useful.

- 1 The transmitters drive UNPREDICTABLE data on the CAD and CTL lines during CRC testing mode.
- 2 Non-zero byte masks for double words which are not sent are ignored.
- 3 Five additional error conditions are reported: Protocol error detected on the link; Receive FIFO overflow error; Response received with a srctag that has no corresponding request; End-of-chain NXA error issued to the link; and NXA error issued to a request from a peer host bridge (a request with a source tag of zero).
- 4 The interface has 4 responses to errors. Localized errors result in an error response. The remaining errors can be set to result in sync flooding, raising the HyperTransport fatal error interrupt or raising the HyperTransport non-fatal error interrupt.
- 5 If the isochronous bit is set in a HyperTransport request that is sent to the ZBbus, it will have the L2 cache allocate bit set in the ZBbus command.

- 6 In a dual host topology, if any incoming packet, with an address mapped as to be forwarded to the outgoing link and a srcid = 0, is received by a host it will be NXAed.
- 7 The ISA deadlock prevention mechanism, described in section C.1 of the HyperTransport Specification Revision 0.17, is not supported. This mechanism has been removed in Revision 1.0 of the specification. As a result, and as specified in Revision 1.0, no peer-to-peer transactions involving ISA are allowed.
- 8 The optional secondary bus reset functionality in the Capability command register is implemented.
- 9 The interface only supports an 8 bit CAD width. There is no support for the 2 or 4 bit widths that are described in Revision 1.0.
- 10 The ordering between CPU (or any other internal device) writes to the HyperTransport fabric and DMA Read Data Return to the HyperTransport fabric is imposed in the HyperTransport interface. This ordering is not part of the ZBbus. In particular the I/O bridge and ZBbus behave as if the Response may pass posted Writes bit is always set. If the PCI producer-consumer model is being used read responses should always pass posted writes and this bit should always be clear. The result of not having the ordering rule internally is that if HyperTransport device A polls a flag in memory that the CPU writes after writing HyperTransport device B (which may or may not be the same device), the read response with the flag set may reach device A before the write reaches B. The ordering can be enforced by the CPU doing a PCI or HyperTransport configuration register read between the two writes. This forces the write to device B into the HyperTransport interface (where the ordering rules apply) before the flag gets written in memory. (This also applies for the PCI interface).
- 11 Posted writes have the highest priority with regard to transmission to the HyperTransport link. Starvation of non-posted transactions and read responses is prevented by implementing counters in all entries of the interface transmit FIFOs. The outgoing entry counters will get incremented on each transaction issued. When an entry's counter overflows, its priority is elevated.
- 12 The correct ordering of interrupts with respect to writes from the HyperTransport into the BCM1250 is maintained. An interrupt that follows a HyperTransport write will not be raised inside the BCM1250 until the write is visible inside the ZBbus coherency boundary.
- 13 A non-posted write from the HyperTransport fabric to the ZBbus or PCI bus will be acknowledged when the write leaves the HyperTransport interface, and will continue as a posted write. Error responses are never generated for non-posted writes. Therefore, if the HyperTransport device needs to confirm the success of a non-posted write, it must (1) wait for the write tgtDone response, then (2) issue a read to the write address for confirmation. It cannot issue the read without waiting for tgtDone since non-posted transactions may bypass each other.
- 14 No peer-to-peer I/O space traffic between the HyperTransport and PCI is supported.
- 15 No DMA master is allowed on an ISA bridge on the HyperTransport or PCI buses. It could result in a deadlock.
- 16 PCI devices designed for rev2.0 or earlier may not be bridged off the HyperTransport.
- 17 The LinkFreq register, described in Revision 1.0 of the Specification, has been added to the HyperTransport Capability registers. This frequency set in this register sets both the transmitter frequency and the maximum frequency the receiver can accept.
- 18 Support has been added for frequencies other than those specified in HyperTransport Revision 1.0 by using the sipFrequencyDirect bit in the HyperTransport SRI Command register.
- 19 In a dual hosted system determination of the master and slave ends of the chain must be made before link configuration. Neither the HyperTransport Specification nor the BCM1250 hardware specify the method for determining this. System designers are free to implement it in whatever way matches their system. Three possible methods on the BCM1250 are: use one of the software configuration bits latched into the System Configuration Register from the IO\_AD[31:26] during reset (see [Table 15 on page 43](#)); store the information in the boot ROM; store the information in a configuration EEPROM on the SMBus.

## HYPERTRANSPORT DIFFERENCES FROM REVISION 1.03 SPECIFICATION

This section compares the HyperTransport interface with the HyperTransport I/O Link Protocol Specification Rev 1.03 10/10/2001. This is the first specification released by the HyperTransport Consortium, the specification details and section numbers match the 1.02 release made by AMD. Comments are made by section number in the specification. (The sections were reordered in HT 1.04.)

### 1.1 Terminology

Care is needed with the term Doubleword. In HyperTransport/PCI it is used for four bytes, in MIPS it is used for eight bytes.

### 2 Signaling

The interface only supports 8 bit links (8 CAD pairs, 1 CLK pair, 1 CTL pair). The optional power management signals are not supported.

#### 3.2.1.4 Command Encoding

The interface uses the Isochronous bit in Wr(sized) and Rd(sized) inbound (dma) commands to set the ZBbus L2CA flag to request the access be allocated in the L2 cache on a miss. All memory accesses will be marked cacheable coherent and all others uncached regardless of the coherence bit in the command.

### 4.1 Topology

The part is always a host. If the interface revision is 3 or greater the ActAsSlave mode is supported for double-hosted chains.

#### 4.1.1 Double-Hosted Chains

The interface supports double-hosted chains. Designation of the master and slave end must be done by software. The Host Hide function is not implemented. The ActAsSlave is implemented in interface RevId 3 and greater, but is not available on earlier versions. Care is therefore needed to understand the dataflows in sharing double-hosted chains.

In a chain with two hosts A and B, if the MAC or Serial port DMA engine on A is doing writes to any device through the HyperTransport then accesses from B to any of the peripherals behind I/O bridge 1 on A can cause deadlock.

#### 4.1.2.2 Host Implementations

The interface has LDT\_PWROK and LDT\_RESET\_L separated from the system reset. A system reset will always cause the HyperTransport to reset until released by software. See [Section: "HyperTransport Resets" on page 259](#) for more details.

### 4.2 Transactions and UnitID

The interface is a host and therefore always uses UnitID 0. In interface RevId 3 and greater the ActAsSlave function allows the interface to use the UnitID set in the DevNum field on a double-hosted chain.

#### 4.4.1 Sized Writes

In byte writes nonzero byte masks are ignored for doublewords that are not sent.



#### 4.4.2 Broadcast Message

The only broadcast message the interface will generate is an EOI. No broadcast messages are accepted.

#### 4.4.3 Flush, 4.4.4 Fence

The interface will never generate a Flush or Fence command. They will be accepted and Flush will result in a TgtDone return.

#### 4.4.5 Atomic Read-Modify-Write

The Atomic Read-Modify-Write is neither generated nor accepted by the part.

#### 4.5.2 Target Done

Target done is returned when a request has been sent from the HyperTransport interface to the I/O Bridge. Inbound TargetDone responses cause the target done counter to increment (making the completion visible to the processor).

#### 4.7 Virtual Channels

The three standard Virtual Channels are supported. Since all writes within the part are posted, the only writes that will be in the non-posted channel are configuration space writes.

#### 4.9.4 Host Bridges

1. The interface supports the optional double-hosted memory space region as described in [Section: "Accessing the SiByte from a SiByte on a Double Hosted Chain" on page 212](#). Responses from the part always have the Bridge bit set (rather than the specified clear) and a UnitID of zero.
2. Broadcasts are dropped.
3. Directed requests are accepted as described in the [Section: "Accessing the SiByte from HyperTransport Devices" on page 210](#).
4. This rule is not completely enforced. Responses with the Bridge bit set and a UnitID of zero are treated as responses from a host on the far end of the link and compared to the outstanding requests.
5. A SrcTag error is logged in the HyperTransport Error Status register ([Table 153](#)) if a response is received with no matching request.

### 5 Interrupts

The interrupt usage follows the earlier HyperTransport specification which has now been moved to the x86-specific appendix.

#### 5.1 Interrupt Requests

The IntrInfo[55:32] bits are not supported. IntrInfo[31:2] are interpreted in a similar way to the x86-specific model, as described in [Section: "HyperTransport Interrupts" on page 48](#). Interrupt messages can be generated using the special address range described in [Section: "HyperTransport End Of Interrupt \(EOI\) Signaling Space" on page 198](#).

#### 5.2 End of Interrupt

The interface ignores any EOI messages it receives. EOI messages are generated by software as described in [Section: "HyperTransport End Of Interrupt \(EOI\) Signaling Space" on page 198](#).



## 6.1 Upstream I/O Ordering

The interface follows the ordering rules through the HyperTransport Interface into the I/O Bridge.

## 6.2 Host Ordering Rules

The ordering rules are maintained into the ZBbus domain, which has a different set of rules. The same ordering is used for Cacheable and Non-Cacheable commands, and the rules for these, interrupt messages and responses are maintained correctly. However, because the ZBbus allows responses to pass posted writes care must be taken when using the producer-consumer model as described in point 10 of the previous section.

### 6.2.1 Host Responses to Nonposted Requests

A non-posted write is acknowledged as the transaction leaves the HyperTransport interface and becomes a posted write internally. See point 13 of the previous section.

## 6.4 Ordering in Sharing Double-Hosted Chains

The I/O Bridge and HyperTransport interface will always push posted writes ahead of read responses. But this is not the case internally and care must be taken because the ZBbus always allows read responses to pass posted writes. See the [Section: "Ordering Rules and Device Drivers" on page 14](#).

## 7.1 Configuration Cycle Types

Configuration Cycles are generated as described in [Section: "Configuration of PCI and HyperTransport" on page 234](#).

### 7.3.1.1 I/O Space Enable, 7.3.1.2 Memory Space Enable

These bits are always set because the address decoders on the ZBbus are always active.

### 7.3.8 Interrupt Line

The Interrupt Line register is not writeable.

### 7.3.2.3, 7.3.2.4, 7.3.2.5 Primary Bus Error Status bits

These bits are always clear because errors on the ZBbus side are reported in other ways.

### 7.4.8.3 ISA Enable

The optional ISA address scrambling is not supported so this bit is always 0.

### 7.4.8.4 VGA Enable

The optional VGA address forwarding is supported so this bit is R/W. The implementation is described in [Section: "The SouthBridge, VGA and Subtractive Decode" on page 195](#).

## 7.5 Capability Registers

The interface uses the Host Format from the earlier HyperTransport specifications, these cover the +00, +04 and +08 offsets of the block in Table 29 of the HyperTransport specification. Interface revision 3 uses the format from the 1.03 specification.

### 7.5.3.3 Host Interface Command Bits

The pass 2 BCM1250 does not implement the fields that are new in the revision 1.02 specification: Device Number, Chain Side, Host Hide, Act as Slave, Drop on Uninit. Interface revision 3 and greater, used for the BCM1125H and later versions of the BCM1250, includes these. The new Host Inbound End of Chain error (Bit 11) functionality is incorporated in the MapNxaError bit in the HyperTransport Error Status Register ([Table 153 on page 253](#)).

### 7.5.4 Link Control Register

The interface does not implement the fields that are new in the revision 1.02 specification: Isocronous Enable, LDTSTOP# Tristate Enable, Extended CTL time. None of the functionality that these control is supported.

### 7.5.5 Link Configuration Register

This register reflects the fact that the interface only supports 8 bit links and does not support doubleword flow control.

### 7.5.8 Link Error Register

The error conditions reported in this new register are reported in the HyperTransport Error Status register ([Table 153 on page 253](#)). Note that if the bridge is put into the direct frequency selection mode (allowing link frequencies outside the standard) then bit 4 is used for frequency selection.

### 7.5.9 Link Frequency Capability

The link frequency capability register is only supported on revision 3 and greater of the interface. On revision 2 (BCM1250 pass 2) it would have the value 16'h801F, indicating that frequencies up to 600MHz are supported and there are vendor specific frequencies. On revision 1 (BCM1250 pass 1) it would have the value 16'h800F, indicating that frequencies up to 500MHz are supported and there are vendor specific frequencies.

### 7.5.10 Feature Capability Register

The link feature capability register is only supported on revision 3 and greater of the interface. On older revisions it would have the value 16'h0004.

### 7.5.11 Enumeration Scratchpad

This register is not implemented (its address collides with the SRI registers).

### 7.5.12 Error Handling Register

The functions in this new register are available in the HyperTransport Error Control register ([Table 152 on page 252](#)).

### 7.5.13, 7.5.14 Memory Base/Limit Upper Bits

These are not implemented since the interface constrains PCI style memory operations to the low section of the address map, and has special rules for addresses above the 32-bit region.

## 7.6 Interrupt Discovery and Configuration Capability Block

This new capability block is not implemented. Generation of interrupt messages is done by software, and is restricted to IntrInfo[55:32]=0 and IntrInfo[31:24]=F8 (i.e. compatibility with revision 1.01 and earlier and the x86 specific interrupt restrictions).

### 7.7 Address Remapping Capability Block

The interface does not have an address remapping block, the mappings between the ZBbus and HyperTransport addresses are discussed in the earlier sections of this chapter.

### 8 System Management

The interface does not support any of the legacy x86 system management or special cycles. The optional LDTSTOP# is not supported.

### 9 Address Map

The internal address map ([Table 11 on page 36](#)) reflects the requirements of the HyperTransport map.

Only interrupts with Address[31:24]=F8 are accepted.

### 10.1 Error Conditions

The interface detects and reports the errors described in this section. Since the method of detection and reporting were not specified in earlier versions of the HyperTransport Specification the implementation differs.

#### 10.1.1 Transmission Errors

A CRC error is detected and reported as in the specification. The sync flood control is in the Link Control register as specified. The interrupt enable bits are in the HyperTransport Error Control Register ([Table 152 on page 252](#)) rather than the new Error Handling Register in the extended register set.

When a CRC error is detected the expected and received CRC values are logged in the HyperTransport Diagnostic Receive CRC Expected ([Table 158 on page 254](#)) and Received ([Table 159 on page 254](#)) registers.

#### 10.1.3 Protocol Errors, 10.1.4 Receive Buffer Overflow Errors

Protocol and Overflow errors are reported in the HyperTransport Error Status Register ([Table 153 on page 253](#)) rather than the new Link Error Register, and controlled in the HyperTransport Error Control Register ([Table 152 on page 252](#)) rather than the new Error Handling Register in the extended register set.

#### 10.1.5 End of Chain Errors

The end of chain error is reported in the EocNxaError flag in the HyperTransport Error Status Register ([Table 153 on page 253](#)) and controlled in the HyperTransport Error Control Register ([Table 152 on page 252](#)).

The MapNxaError flag is used to report two variants of the issue described in the spec as Host Inbound End Of Chain error. A MapNxaError is flagged when a posted operation is received that is to an address that is not on the HyperTransport link or PCI and does not decode to a legal address within the BCM1250. It is also flagged and the transaction dropped, for a posted operation if the srcid is zero (i.e. it has come from the other host on a double-hosted chain) and the destination is an address on the HyperTransport link. In this second case the transaction has already been rejected by all devices on the link, and sending it back will just lead to the transaction looping forever.

#### 10.1.6 Chain Down Errors

The host interface discards all state on a link RESET# and does not report a chain down error. Responses for in-flight reads will be lost and the requestor may hang.



### 10.1.7 Response Errors

A response received without a matching source tag (or as a result of a response to a WrSized request) is reported as a SrcTagError in the HyperTransport Error Status Register (Table 153 on page 253) and controlled in the HyperTransport Error Control Register (Table 152 on page 252).

No check is done for a TgtDone response or an incorrect count received for a RdSized request, the requester will receive UNPREDICTABLE data in this case.

The interface does not generate Flush or Atomic Read-Modify-Write cycles.

### 10.2.1 Error Responses

Error responses are controlled by the Master Abort Mode bit as the specification describes. Bit 5 of Table 146 describes the method used to propagate errors from the HyperTransport fabric to the ZBbus.

### 10.2.2 Error Interrupts

The error interrupts are passed directly to the interrupt mapper as interrupt numbers 48 and 49.

### 10.2.2 Error Routing CSRs

The error CSR bits in the implementation are summarised in the Table 125 on page 229 below which follows the format of the Table 50 in the HyperTransport Specification.

**Table 125: Error Routing Registers**

Error Type	Log Bit	Flood Enable	Fatal Error Enable	Nonfatal Error Enable
<b>Protocol</b>	ErrStat/ProtoErr	ErrCtrl/ ProtSyncFlood	ErrCtrl/ ProtFatalEn	ErrCtrl/ ProtNonFatalEn
<b>Overflow</b>	ErrStat/OvfErr	ErrCtrl/ OvfSyncFlood	ErrCtrl/ OvfFatalEn	ErrCtrl/ OvfNonFatalEn
<b>EOC</b>	ErrStat/ EocNXAErr	ErrCtrl/ EocNxaSyncFlood	ErrCtrl/ EocNxaFatalEn	ErrCtrl/ EocNxaNonFatalEn
<b>Inbound EOC (covers more than Spec)</b>	ErrStat/ EocNXAErr	ErrCtrl/ EocNxaSyncFlood	ErrCtrl/ EocNxaFatalEn	ErrCtrl/ EocNxaNonFatalEn
<b>Response</b>	ErrStat/ SrcTagErr	ErrCtrl/ SrcTagSyncFlood	ErrCtrl/ SrcTagFatalEn	ErrCtrl/ SrcTagNonFatalEn
<b>CRC</b>	LinkCtrl/ CrcErr	LinkCtrl/ CrcFloodEn	ErrCtrl/ CrcFatalEn	ErrCtrl/ CrcNonFatalEn
<b>SERR</b>	SecStatus/ SerrDet	BridgeCtrl/ SerrEn	ErrCtrl/ SerrFatalEn	Not Supported

### 11.1 Clocking Mode Definition

The interface supports synchronous and asynchronous mode. Software must configure the interface before setting the SipReady bit, as described in the “[System Reset Initialization of the HyperTransport Interface](#)” on page 255.

### 12 Reset and Initialization

The interface performs the standard link initialization after software has configured the low level parameters and set the SipReady bit.



## 12.2 Low Level Link Initialization and Table 54 Values of CTL and CAD During Link Initialization Sequence

The transmit side will generate 512 cycles with CTL and CAD as zero before asserting CAD and the receive side will accept 512+M cycles. This matches both the older versions of the spec which need 'a minimum of 512 cycles' and can have an arbitrary number of additional bit times, and the newer version '512+4N' where the minimum is still 512 cycles (N=0) but any additional duration of CTL and CAD at zero must be for a multiple of four bit times.

## 12.3 I/O Fabric Initialization

Since the MIPS architecture does not support non-posted writes the TgtDone counter has been added to allow detection of the completion of configuration writes.

In a Double-hosted chain software is responsible for determining which is the master and slave ends of the fabric and performing appropriate configuration.

### 12.3.1 Finding the Firmware ROM

The part cannot boot from a ROM over the HyperTransport link.

## 12.5 Link Frequency Initialization

The interface will normally perform the link frequency change on a link reset, and revert to 200MHz on a cold reset. However, to support other devices that use the HyperTransport specification from before revision 1.0 this behaviour can be prevented by setting the sriLdtPLLCompat bit in the SRI Command Register ([Table 149 on page 251](#)).

### A. Address Remapping

The interface supports the host address map as outlined throughout this chapter, so does not need the address remapping scheme.

### B. Ordering Rules

The HyperTransport interface uses the PCI ordering rules.

### C. Mapping of Other Protocol Ordering Rules

The interface maps from the ZBbus as described in C.1. Non-posted writes are not part of the MIPS architecture and so will never be generated for memory-mapped I/O.

### D. Considerations for Isochronous Traffic

The interface does not use the new isochronous scheme. The isochronous bit from DMA read or write requests is copied into the L2CA flag for the ZBbus transaction. Isochronous requests will therefore cause L2 allocation on a miss.

The part always generates requests with the isochronous bit clear.

### E.1, E.2 ISA/LPC Support

There is no special ISA/LPC support. All warnings and restrictions in these sections must be followed.

Note that PCI devices that only support Revision 2.0 or earlier of the PCI Specification may have similar problems to ISA devices since they do not support the full ordering and transaction acceptance rules.

### E.3 Subtractive Decode

Subtractive Decode is discussed in the [“The SouthBridge, VGA and Subtractive Decode” on page 195](#).

### F.1 Interrupts

The interface follows the x86 interrupt mechanism as outlined in [Section: “HyperTransport Interrupts” on page 48](#). Interrupt messages and EOI are generated by software as described in Section EOI Signalling Space on [Section: “HyperTransport End Of Interrupt \(EOI\) Signaling Space” on page 198](#).

In logical mode IntrDest[31:2] bits are ignored and the low two bits used to select the processor(s) that will receive the interrupt. The MT[3] bit is ignored so legacy PIC NMI and ExtInt get mapped on to the regular NMI and ExtInt interrupts.

### G. CRC Testing Mode

The CRC Testing mode is supported. The data used during the test is UNPREDICTABLE.

### H. Doubleword Based Data Buffer Flow Control

The interface does not support the optional doubleword flow control.

## ORDERING RULES

The HyperTransport fabric and the PCI bus have a set of ordering rules described in Appendix E of the PCI specification revision 2.2. These differ from the more relaxed ordering rules used on the ZBbus. The PCI/HyperTransport rules are imposed in the I/O bridge which will allow posted writes to pass reads and will not allow read responses to pass posted writes, allowing the producer-consumer model to be used. Using this model requires some care, see [Section: “Ordering Rules and Device Drivers” on page 14](#).

A deadlock exists in the ordering boundary between the ZBbus and I/O Bridge 1 (in the HyperTransport terms bridge 1 does not have sufficient virtual channels). This is only provoked when writes from I/O Bridge 1 are targeted at the PCI/HT space (the access must have come from one of the MAC or Serial DMA engines) at the same time as any write is in progress from the PCI/HT to any address serviced by Bridge 1 (see [Figure 5 on page 9](#) for the devices that are serviced by Bridge 1) or a read is in progress from I/O Bridge 1 to bridge 0. This deadlock can be avoided by forbidding the Bridge 0 to Bridge 1 writes and the Bridge 1 to Bridge 0 reads.

## USING THE PCI IN DEVICE MODE

The part can be used in PCI Device Mode in a couple of different situations. The main one is when it is used as a PCI device on an option card and the second is when the PCI is used to connect several parts in a system.

The BAR0, BAR2, BAR3 and ROM BAR are active in Device Mode. Most accesses by an external host will be done through BAR0 and the mapping table, BAR2 and BAR3 will be used for the host to communicate with the SB-1 CPUs via the mailbox registers and the ROM BAR will typically only be used during initialization. When the part needs to make a memory space access on the PCI it will use the Full Access space, which allows any (32 bit) PCI address to be generated, I/O accesses can be made through the normal I/O space range.

In Device Mode the PCI configuration is mostly done by the host in the system. However the BAR0 Map registers, the SubSysSet register (and the vendor IdSet and ClassRevSet) and the INTA control bit can only be set by configuration accesses from the ZBbus side of the PCI interface and are therefore under control of software on the device.

The address map on the PCI bus is different from the address map within the part, and translation is done through the BAR0 address map. The map table allows control of what areas of the local memory map are accessible to an external PCI master. Therefore it is protected and can only be used with configuration accesses from the ZBbus side of the bridge. Configuration accesses to the map table registers will be accepted from the PCI bus, but the writes will be ignored and reads will return UNPREDICTABLE values. The external master will always see the BAR0 region as a 16 MByte space that needs an address allocation, software running on the part should configure the map table to allow access to appropriate internal resources. The peer-to-peer mappings in the map register give the device some of the characteristics of a non-transparent PCI to HyperTransport bridge.

The SubSysSet register can only be written from the ZBbus side of the bridge. It is used to provide a value in the SubSystem Device and Vendor Id registers seen by external configuration reads. In Device Mode software on the part should write the SubSysSet register with a value that identifies the manufacturer of the option card and its part number. This write must be done early in initialization, before the host needs to read the value. On interface revision 3 and greater the Device/Vendor ID register and the Class/Rev register seen by external configuration reads can also be set using the VendorIdSet and ClassRevSet registers.

The INTA control bit can be used by the device to signal a PCI interrupt on the (bidirectional) P\_INTA\_L pin. Again, this bit can only be accessed from the ZBbus side and external accesses have UNPREDICTABLE results.

The P\_RST\_L signal is only an output, and is therefore only appropriate for use when the part is in Host Mode. In Device Mode the pin is not used. There are two options for connecting the PCI reset signal. When the part is used on an option card it should be reset when the PCI bus is reset, therefore the PCI reset should drive the RESET\_L input to the part. When the PCI is used to connect multiple parts in a system they may need to be independently reset, in this case the PCI reset line can be connected to one of the GPIO pins used as an interrupt input. Software running on the Device Mode part can then read the state of the PCI reset and can be interrupted to take appropriate action if the PCI bus is unexpectedly reset during normal operation.

Accesses to the PCI configuration registers in device mode are confused by their being shared between accesses from within the part (ZBbus accesses) and accesses originating as Type 0 configuration cycles on the PCI bus (PCI accesses). In addition the interface must ensure that the host doing PCI accesses does not attempt to read the SubSystem information before local software has written the SubSysSet register. The rd\_host bit, in the WRITE ONLY Device control register, is used to control the access. This bit is only used in device mode. It defaults to being set which allows read and write ZBbus accesses to the static interface identification, the BAR0 map entries, the SubSysSet register and the SignalINTA register. While the bit is set



any PCI access will be issued with a retry. Software on the part should do initial configuration of the map registers and set the subsystem information before writing a zero to the rd\_host bit and flush the write to the interface by following it with a dummy read that will return UNPREDICTABLE data. To be compliant with the PCI specification this must be done within 1 second (0.55 on a 66 MHz bus) of the PCI reset being deasserted.

While the system is running with the rd\_host bit clear ZBbus writes may be done to the map entries and the SignalNTA register. Writes to the SubSysSet register will result in the SubSystem ID being updated, but the results of changing this after device enumeration is system specific: the subsystem information may never be read again or the operating system may assert an error because static information changes. Reads to any of the PCI configuration registers will return UNPREDICTABLE data. The rd\_host bit may be set to give read access to the map entries, SubSysSet and SignalNTA, but while the bit is set any read requests from the PCI bus will be issued retries (read access will probably only be needed for debugging).

Table 131 summarizes access to the PCI configuration registers.

**Table 126: PCI CSR Access Rules**

Register	ZBbus Read	ZBbus Write	PCI Read	PCI Write
<b>Host Mode</b>				
<b>Device/Vendor ID (0)</b> <b>Class/Revision (8)</b>	Valid data	Ignored	Unpredictable	Unpredictable
<b>Map Entries (44-80)</b> <b>SubSysSet (8C)</b> <b>SignalNTA (90)</b>	Valid data	Write register	Unpredictable	Unpredictable
<b>Interface revision 3 or later</b> <b>VendorIdSet (9C)</b> <b>ClassRevSet (A0)</b>	Valid data	Write register	Unpredictable	Unpredictable
<b>Others</b>	Valid data	Write register	Unpredictable	Unpredictable
<b>Device mode with rd_host = 1</b>				
<b>Device/Vendor ID (0)</b> <b>Class/Revision (8)</b>	Valid data	Ignored	Retried	Unpredictable
<b>Map Entries (44-80)</b> <b>SubSysSet (8C)</b> <b>SignalNTA (90)</b>	Valid data	Write register	Retried	Ignored
<b>Interface revision 3 or later</b> <b>VendorIdSet (9C)</b> <b>ClassRevSet (A0)</b>	Valid data	Write register	Retried	Ignored
<b>Others</b>	Unpredictable	Unpredictable	Retried	Unpredictable
<b>Device mode with rd_host = 0</b>				
<b>Device/Vendor ID (0)</b> <b>Class/Revision (8)</b>	Unpredictable	Unpredictable	Valid data	Ignored
<b>Map Entries (44-80)</b> <b>SubSysSet (8C)</b> <b>SignalNTA (90)</b>	Unpredictable	Write register	Unpredictable	Unpredictable
<b>Interface revision 3 or later</b> <b>VendorIdSet (9C)</b> <b>ClassRevSet (A0)</b>	Unpredictable	Write register	Unpredictable	Unpredictable
<b>Others</b>	Unpredictable	Unpredictable	Valid data	Write register

## CONFIGURATION OF PCI AND HYPERTRANSPORT

The PCI bus and HyperTransport fabric both require configuration before use. This is normally done by the startup firmware, but may also be done by the operating system as it starts. There is usually a "PCI-BIOS" abstraction layer above the configuration hardware (this is only well specified for the x86 architecture, but most systems have a version of it). Since HyperTransport devices use the same configuration headers as PCI, only minor changes to the configuration software will be needed to support it. In particular accesses to the PCI/HyperTransport memory or I/O ranges are UNPREDICTABLE until the HyperTransport mem\_base, mem\_limit, io\_base and io\_limit registers have been programmed, and configuration accesses to anything other than bus 0 are UNPREDICTABLE until the HyperTransport secondary and subordinate bus numbers have been programmed. (See [Section: "Systems That Do Not Use HyperTransport" on page 236](#) if HyperTransport is not used.)

During configuration the tree of bridged buses (an entire HyperTransport chain is regarded as a single bus) is built along with a list of devices on each bus and their memory size requirements. The buses are allocated numbers, with bus 0 being at the root of the tree. A bridge gets configured with a bus number for its primary interface (the one that is closest to the root of the tree), a bus number for its secondary interface (the other bus it directly connects to) and a subordinate bus number which is the highest numbered bus that is behind the bridge (all buses in the range secondary - subordinate are accessible through the bridge).

Once all the buses are enumerated the configuration code will assign base addresses to all the devices. There are three separate address ranges: for memory mapped I/O, prefetchable memory, and I/O space. In each range, all addresses allocated behind a bridge must be contiguous.

Configuration uses the logical organization of the interfaces shown in [Figure 36 on page 191](#). The PCI bus on the part is bus 0 at the top of the configuration tree and therefore has its configuration space at bus 0, device 0. The HyperTransport bridge configuration is at bus 0, device 1. (The bus number that will be given to the HyperTransport fabric depends on the enumeration software).

The configuration space for PCI and HyperTransport is based at 00\_FE00\_0000 for match bit lanes (the most useful endian policy for configuration) with a repeat at 00\_DE00\_0000 using the match byte lane policy. To access a configuration register the address is formed from the register, function, device and bus number as shown in [Figure 49](#). Reads and writes of up to 32 bits can be done. The results of doing a 64 bit access to configuration space are UNDEFINED. Note that in Device Mode the PCI interface will mostly be configured by the host in the system, but the device mode part must still configure the BAR0 address mapping table and the HyperTransport fabric.

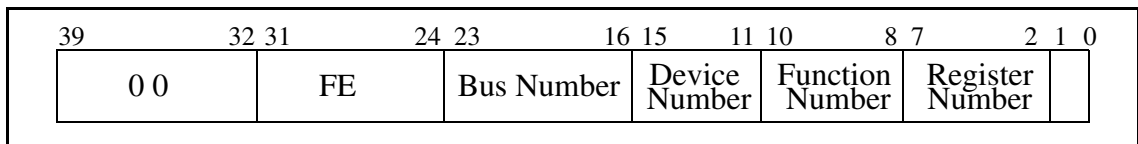


Figure 49: Configuration Space Address



Accesses to the configuration range with bus 0 device 0 and bus 0 device 1 will be directed to the internal configuration registers for the PCI and HyperTransport interfaces. Once the CPU has performed a series of writes to the internal configuration registers it should read back at least one. This will ensure the writes have taken effect before any other operations will proceed. Although the transfers will be in order on the ZBbus the data phase will be behind the address. If there is no read separating a configuration write from a subsequent transaction it is possible for the address of the transaction to reach the interface before the data of the configuration write, in this case the address decode for the transaction would be done based on the old value of the configuration register.

Other accesses to bus 0 devices will be sent as Type 0 configuration cycles on the PCI bus, the mapping used for generating IDSEL is to convert the device number into a single one on AD[31:11]. Since the internal PCI and HyperTransport headers are device 0 and 1 the external devices start with AD[13] being set for device 2, AD[14] being set for device 3 and so on up to AD[31] being set for device 20 (note that the internal PCI arbiter only supports four external devices). During PCI configuration cycles the interface will pre-drive the address on the bus for a cycle (as described in section 3.2.2.3.5 of the PCI Specification revision 2.2), this allows a resistor to be used to connect to the IDSEL inputs of PCI devices. A PCI configuration read to a nonexistent device will be terminated with a master abort and set the status flags to indicate this happened, the data returned will be marked valid and will contain 32'hFFFF\_FFFF (for normal accesses, a master abort will result in a bus error return). When a PCI configuration read is aborted in this way all other reads that are in flight in the PCI bridge will be terminated with a bus error. Software should therefore ensure that there are no reads in progress through the PCI bridge when it is probing the PCI bus for devices. Note that configuration reads that encounter other PCI errors (e.g. a parity error) will also return 32'hFFFF\_FFFF.

If the bus number of an access to the configuration range matches the secondary bus number configured in the HyperTransport Bridge, the request will be forwarded as Type 0 configuration accesses on the HyperTransport fabric. This is done by turning it in to an access to the address range given in the HyperTransport specification for configuration. Accesses that have a bus number greater than the HyperTransport bridge secondary bus and less than or equal to the HyperTransport subordinate bus number are sent as Type 1 configuration accesses on the HyperTransport fabric. Errors generated from configuration accesses to HT will be forwarded to the ZBus rather than the PCI bridge. All other accesses are sent as Type 1 configuration accesses on the PCI bus. This is shown algorithmically in the pseudo-code below.

```

endian = zbaddr[29] ? match bit lanes : match byte lanes
bus = zbaddr[23:16]
device = zbaddr[15:11]
func = zbaddr[10:8]
reg = zbaddr[7:0]
if (bus == 0)
{
  if (dev == 0)
    access PCI config reg
  else if (dev == 1)
    access HyperTransport config reg
  else
    do PCI Type 0 cycle (dev, func, reg)
}
else if (bus == HT secondary bus)
{
  // Form special HT Type 0 config address
  htaddr = FD_FE00_0000 + zbaddr[15:0]
  do HT access
}
else if (HT secondary bus < bus <= HT subordinate bus)
{
  // Form special HT Type 1 config address
  htaddr = FD_FF00_0000 + zbaddr[23:0]
  do HT access
}
else
  do PCI Type 1 cycle (bus, dev, func, reg)

```

## HyperTransport Target Done Counter

Since the MIPS architecture does not support non-posted writes it is hard for software to tell when HyperTransport configuration writes have completed. To help with this a counter has been implemented in the HyperTransport Additional Status Register. This counter can be cleared by software writing a zero and will increment every time a TgtDone message is returned from the fabric. Thus software can clear the counter, do all the writes to configure a HyperTransport device and then poll the counter until it reaches the number of writes performed. This will ensure the final write has been acknowledged by the device.

## SYSTEMS THAT DO NOT USE HYPERTRANSPORT

If a BCM1250 or BCM1125H system does not use the HyperTransport interface the mem\_base, mem\_limit, io\_base and io\_limit registers of the HT bridge must still be programmed. This configures a shadow copy of these registers that is by default UNPREDICTABLE. The mem\_limit should be set to be lower in address than the mem\_base. The io\_base upper 16 bits (bits [15:0] of offset h30 in the HyperTransport configuration header) should be set above the 24 bit range used by the I/O space, for example the register can be written with 32'h0000F200 (note that subsequent reads will show bits [15:12] as zero). For configuration accesses to buses other than bus 0 to work the HyperTransport secondary and subordinate bus numbers should be written with zero. Until these registers are programmed accesses to the PCI memory or I/O space will have UNDEFINED results.

## CONFIGURATION HEADER DESCRIPTIONS

The following sections describe the configuration headers used by the PCI and HyperTransport interfaces. The PCI interface is bus=0, device=0, function=0 so the header can be accessed from within the part starting at the base address is 00\_FE00\_0000 (match bits) or 00\_DE00\_0000 (match bytes). The HyperTransport interface is bus=0, device=1, function=0, with base address 00\_FE00\_0800 (match bits) or 00\_DE00\_0800 (match bytes).

The following terms are used in the register descriptions:

- R/O. Read Only. The field contains a static value and should not be written.
- R/W. Read/Write. The field can be modified by software.
- R/C. Read/Clear. The field can be read to give status information. Bits may be cleared by writing a 1 to them, writes of 0 are ignored.

## PCI CONFIGURATION HEADER

The PCI configuration header is a standard Type 0 header with extensions to support the address map table. The following table shows the fields of the header, along with the values they contain after a system reset.

**Table 127: PCI Interface Configuration Header (Type 0)**

Offset	Register Bits							Description
	31	24	23	16	15	8	7	
00	Device Id R/O 0001			Vendor Id R/O 166D				Identifies the device as a Broadcom SiByte family PCI interface.
04	Status (see <a href="#">Table 129 on page 239</a> ) R/W 02A0			Command (See <a href="#">Table 128 on page 239</a> ) R/W 0002				As defined by PCI standard.



Table 127: PCI Interface Configuration Header (Type 0) (Cont.)

Offset	Register Bits								Description
	31	24	23	16	15	8	7	0	
08	Class Code R/O 060000				Rev Id R/O xx				Class is a host bridge, revision reflects the interface revision code. The revision code is: 1 - for early prototype BCM1250s. 2 - for initial production BCM1250s 3 - for initial production BCM1125/H and later BCM1250s
0C	BIST R/O 00	Hdr type R/O 00	LatTimer (Table 130 on page 240) R/W 00		ClineSz (Table 131 on page 240) R/W 00				The bridge uses a Type 0 device header.
10	BAR 0 - Map Table Host: R/O 6000_0008 Dev: R/W xx00_0008								Hits to this 16 MB BAR will be translated through the mapping table. Bit 3 is set to indicate a prefetchable region.
14	BAR 1 - Reserved R/O 00000000								This BAR is reserved and will always be read as zero
18	BAR 2 - mbox 0 Host: R/O 7000_0008 Dev: R/W xxxx_x008								Hits to this 4 KB BAR are translated into accesses to the mailbox set and value register for CPU0.
1C	BAR 3 - mbox 1 Host: R/O 7100_0008 Dev: R/W xxxx_x008								Hits to this 4 KB BAR are translated into accesses to the mailbox set and value register for CPU1.
20	BAR 4 - low memory Host:R/O 0000_0008 Dev: R/O 0000_0008 (Reserved)								Hits to this BAR are passed through as accesses to the low 512MB of memory.
24	BAR 5 - High Memory Host:R/O 8000_0008 Dev: R/O 0000_0008 (Reserved)								Hits to this BAR are passed through as accesses to the upper 2GB of the low 4GB of the address space.
28	Cardbus CIS R/O 00000000								This register is not used.
2C	SubSystem Id R/O 0000			SubSys Vendor R/O FFFF					These registers are not used internally since this is a host bridge. If the Configuration registers are read by an external PCI master with the bridge in Device Mode the value read from this register will be the value written from the ZBbus to the SubSysSet register (offset 8C).
30	ROM Base Address Host: R/W 73000000 Dev: R/W xxxx0000								Hits to this 64 KB BAR are translated to accesses to the boot rom area of memory.
34	Reserved R/O 000000				Cap Ptr R/O 00				This register is not used.
38	Reserved R/O 00000000								This register is not used.
3C	max_lat R/O 00	min_gnt R/O 00	Int Pin R/O 01		Int Line R/W 00				The Max Latency and Min Grant registers are used to specify the device preferences for the latency timer, they default to indicate no special requirement. The IntPin register indicates that this device uses INTA to interrupt in Device Mode. The IntLine register is used by firmware and system software.
40	FControl (See Table 133 on page 241) R/W 0003			Timeout (See Table 132 on page 240) R/W 8080					This additional register allows configuration of the retry and TRDY timeouts and enable bits for special features.

**Table 127: PCI Interface Configuration Header (Type 0) (Cont.)**

Offset	Register Bits							Description
	31	24	23	16	15	8	7	
44-80	Map Entry Base Address R/W 000000				Flags R/W 00			The address mapping table for BAR0, described in <a href="#">Table 134 on page 241</a> .
84	Error Address R/O xxxxxxxx							This register records the address of every transaction on the PCI bus. When any of the error bits in the status register are set the address in this register is locked, recording the address of the transaction that experienced the error. The register is unlocked when software clears the error status.
88	Additional Status and Command (See <a href="#">Table 135 on page 241</a> ) R/W 00000000							This register contains addition command and status bits.
8C	SubSysSet R/W 00000000							This register is only accessible from the ZBbus. When the interface is configured in Device Mode the value written to this register from the ZBbus will be read by an external read of register 2C.
90	SignalIntA R/W 00000000							Bits 31:1 are R/O 0. Bit 0 can be set from the ZBbus to assert the INTA pin. Writing this bit from the PCI has UNPREDICTABLE results.
94	ReadHost W/O 00000001							This register controls register accesses in device mode. See <a href="#">Table 137 on page 242</a> and <a href="#">"Using the PCI in Device Mode" on page 232</a>
98	Adaptive Extend R/W xxxxxx00							This register controls performance features. See <a href="#">Table 138 on page 242</a> .
9C	VendorIdSet R/W 0001166D							This register is only available if the interface Rev Id is 3 or greater. This register is only accessible from the ZBbus. When the interface is configured in Device Mode the value written to this register from the ZBbus will be read by an external read of register 0.
A0	ClassRevSet R/W 06000003							This register is only available if the interface Rev Id is 3 or greater. This register is only accessible from the ZBbus. When the interface is configured in Device Mode the value written to this register from the ZBbus will be read by an external read of register 4.
A4	BypassCtrl R/W 00000FFF							This register is only available if the interface Rev Id is 3 or greater. This register controls the number of times a ZBbus initiated write can pass a retrying read before the read is discarded and a fatal error signalled.
A8-FC	Reserved							Reserved, accesses will have UNPREDICTABLE results.

**Table 128: PCI Command Register - Offset 4 Bits [15:0]**

Bits	Name	Default	Description
0	IoSpaceEn	R/O 1'b0	I/O Space Enable. This bit is always zero. The bridge never accepts I/O space accesses from the PCI bus.
1	MemSpaceEn	R/W 1'b1 RevId>=3 R/W 1'b0	Memory Space Enable. This bit must be set to allow the bridge to accept memory space accesses from the PCI bus and forward them to either ZBbus or the HyperTransport bridge. In interface revisions 1 and 2 this bit defaults to 1 to enable these accesses after reset, in interface revision 3 and greater this bit defaults to 0 to match the PCI specification.
2	MasterEn	R/W 1'b0	Bus Master Enable. This bit must be set to allow the bridge to act as a master on the PCI bus. This bit should be set before any PCI accesses are made. Any accesses to the PCI space that are received while this bit is clear will result in UNDEFINED behavior in the interface.
3	SpecCycEn	R/O 1'b0	The bridge does not accept Special Cycles, so this bit is always zero.
4	MemWrInvlEn	R/W 1'b0	This bit should be set to allow the bridge to generate Memory Write and Invalidate commands. The bridge will only use Write Invalidate if the start address is cacheline aligned and the transfer is at least one cache line long (using the cache line size set in the ClineSz register). If the latency timer expires while a Write Invalidate command is in progress the access will only be terminated at the next cache line boundary.
5	VgaPalSnpEn	R/O 1'b0	This bit is always zero. The bridge does not snoop VGA palette accesses.
6	ParErrResp	R/W 1'b0	This bit controls the response to PCI parity errors. If it is set then the MstrDParErr bit is set in the status register and the PCI error interrupt is raised when a parity error is detected. If it is clear then no interrupt is raised. In both cases the DetParErr bit is set in the status register. A parity error resulting from a read will return data to the ZBbus with a bus error if this bit is set, and returned as valid data if this bit is clear.
7	StepCtrl	R/O 1'b0	This bit is always clear. The bridge does not do address/data stepping.
8	SerrEn	R/W 1'b0	SERR enable. If this bit is set, the bridge will drive the SERR_L pin if it detects a fatal error. If clear the bridge will never drive SERR_L.
9	FastB2BEn	R/O 1'b0	This bit is always clear. The bridge will never do fast back-to-back cycles to different devices.
15:10	reserved	R/O 6'b0	Reserved

**Table 129: PCI Status Register - Offset 4 Bits [31:16]**

Bits	Name	Default	Description
3:0	reserved	R/O 4'b0	Reserved
4	CapList	R/O 1'b0	Always clear. There is no capabilities list.
5	66MHzCap	R/O 1'b1	This bit is set. The PCI is 66 MHz capable.
6	reserved	R/O 1'b0	Reserved
7	FastB2BCap	R/O 1'b1	This bit indicates that the bridge can accept fast back-to-back transactions when the transactions are to different agents.
8	MstrDParErr	R/C 1'b0	This bit is set if the bridge was a bus master and detected a parity error on read data, or was signalled that a parity error happened on write data. The ParErrEn bit in the command register must be set to allow this bit to be set. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.
10:9	DevselTiming	R/O 2'b01	The bridge generates medium DEVSEL timing.

**Table 129: PCI Status Register - Offset 4 Bits [31:16] (Cont.)**

Bits	Name	Default	Description
11	SigdTgtAbort	R/C 0'b0	This bit is set when the bridge is the target of a PCI transaction and signalled a Target Abort. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.
12	RcvdTgtAbort	R/C 1'b0	This bit is set when the bridge was a master for a transaction and received a Target Abort. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.
13	RcvdMstrAbort	R/C 1'b0	This bit is set when the bridge was a master for a transaction that was not a Special Cycle and signalled a Master Abort. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.
14	SigdSerr	R/C 1'b0	This bit is set when the bridge asserts SERR_L. It will do this when it detects a parity error on an address. If this bit is set the PCI error interrupt is asserted. It is cleared by software writing a 1.
15	DetParErr	R/C 1'b0	This bit will be set whenever the bridge detects a parity error, even if parity error handling is disabled. The parity error is only signalled using the PCI interrupt if the ParErrResp bit is set in the command register. This bit is cleared by software writing a 1.

**Table 130: PCI Latency Timer - Offset 0C Bits [15:8]**

Bits	Name	Default	Description
7:0	LatTime	R/W 8'b0	This contains the value of the Latency Timer for the bridge to use when bus master. If this register is left at its default value of zero the bridge will never do bursts on the PCI. Bits 1:0 are R/O and are always zero.

**Table 131: PCI Cache Line Size - Offset 0C Bits [7:0]**

Bits	Name	Default	Description
7:0	ClineSz	R/W 8'h0	This register sets the cache line size in DWORDS (32 bit words) used by the bridge for PCI transactions. As a master the bridge bases the PCI command used on how the size of the transfer matches the value in this register: word_count < one cacheline: use Read command word_count >= one cacheline: use Read Line command word_count >= two cache lines: use Read Multiple command. This size is also used to enable the Write Invalidate command.

**Table 132: PCI Timeout Register - Offset 40 Bits [15:0]**

Bits	Name	Default	Description
7:0	TrdyTimeout	R/W 8'h80	This field sets the maximum number of PCI cycles that the bridge will wait for TRDY to be asserted. If TRDY is deasserted for longer than this the transfer will be aborted. When a transfer is aborted the TrdyErr bit is set in the Additional Status register and the pci_interrupt may be raised. If a read is aborted UNPREDICTABLE data is returned to the ZBbus marked with a bus error.
15:8	RetryTimeout	R/W 8'h80	This field sets the number of times a read will be retried following a disconnect. The read is first attempted this number of times. The bridge will then allow any outstanding writes and read-data-returns to proceed before retrying the read. In interface revisions less than 3 a second attempt is made, in revisions 3 and later the read is retried the number of times specified in the num_piowr_bypass field of the PCI Bypass Control register (Table 139). If the read fails on the final attempt the transfer is aborted, UNPREDICTABLE data marked with a bus error is returned to the ZBbus and the RetryErr bit is set in the Additional Status register. A write that is retried will be attempted the number of times specified in this field before being discarded and the RetryErr bit set.

**Table 133: PCI Feature Control Register - Offset 40 Bits [31:16]**

Bits	Name	Default	Description
0	bar4_en	R/W 1'b1	This bit must be set to enable accesses through BAR 4 in Host Mode. (default enabled).
1	bar5_en	R/W 1'b1	This bit must be set to enable accesses through BAR 5 in Host Mode. (default enabled).
2	ptp_en	R/W 1'b0	This bit must be set to enable PCI-HyperTransport peer-to-peer transfers (default disabled).
3	adapt_retry_en	R/W 1'b0	This bit must be set to enable the adaptive retry (See <a href="#">Section: "PCI Adaptive Retry" on page 217</a> )
6:4	min_tar_retry	R/W 3'b011	Sets the minimum number of cycles before a retry is issued when the adaptive retry algorithm is used.
10:7	nom_tar_retry	R/W 4'b1011	Along with the upper bits from the PCI Adaptive Extend register this sets the nominal number of cycles before a retry is issued when the adaptive retry algorithm is used.
15:11	max_tar_retry	R/W 5'b01111	Along with the upper bits from the PCI Adaptive Extend register this sets the maximum number of cycles before a retry is issued when the adaptive retry algorithm is used.

Each map table entry contains the mapping address and flags. There are 16 entries in the table, and it is indexed by bits [23:20] of the PCI address.

**Table 134: PCI BAR0 Map Table Entry - Offset 44 – 80**

Bits	Name	Default	Description
0	enable	1'b0	This bit must be set to enable the mapping. If it is clear the PCI interface will not assert DEVSEL# to accept the request.
1	send_ldt	1'b0	If this bit is set then the request will be forwarded to the HyperTransport fabric (regardless of what the address is). If this bit is clear the request is sent into the BCM1250.
2	l2ca	1'b0	If this bit is set then the L2CA flag will be set when an access is made to the ZBbus through this entry. This requests that the cache block be allocated in the L2 cache.
3	endian	1'b0	If the system is configured for big endian operation and this bit is set the match bits policy will be used, if this bit is clear the match bytes policy is used. If the system is configured little endian this bit is ignored.
11:4	reserved	8'b0	Reserved
31:12	addr	20'b0	These bits provide bits 39:20 of the internal address. Bits 19:0 are copied from the PCI address.

**Table 135: PCI Additional Status and Control Register - Offset 88 Bits [31:0]**

Bits	Name	Default	Description
0	hotplug_en	R/W 1'b0	If this bit is set the hotplug function in the internal PCI arbiter is enabled.
1	serr_det	R/C 1'b0	Set when some other device on the PCI bus asserts SERR (a PCI error interrupt is also raised). Software must write a 1 to this bit to clear it and remove the interrupt.
2	TrdyErr	R/C 1'b0	Set when a transfer is aborted because TRDY was not asserted within the TrdyTimeout (See <a href="#">Table 132 on page 240</a> ). The TrdyIntEn bit must be set in order for this bit to be set. Software must write a 1 to this bit to clear it and remove the interrupt.

**Table 135: PCI Additional Status and Control Register - Offset 88 Bits [31:0] (Cont.)**

Bits	Name	Default	Description
3	RetryErr	R/C 1'b0	Set when a transfer is aborted because a read was retried more than the RetryTimeout (See <a href="#">Table 132 on page 240</a> ). The RetryIntEn bit must be set in order for this bit to be set. Software must write a 1 to this bit to clear it and remove the interrupt.
4	TrdyIntEn	R/W 1'b0	If this bit is set the PCI error interrupt will be raised and the TrdyErr bit set on a TRDY timeout. If this bit is clear no interrupt will be raised and the TrdyErr bit will not be set.
5	RetryIntMask	R/W 1'b0	If this bit is set the PCI error interrupt will be raised and the RetryErr bit set on a Retry timeout. If this bit is clear no interrupt will be raised and the RetryErr bit will not be set.
6	DisMulti	R/W 1'b0	This bit should be clear for normal operation.
31:7	reserved	R/O 25'b0	Reserved

**Table 136: PCI INTA Control Register - Offset 90 Bits [31:0]**

Bits	Name	Default	Description
0	signalINTA	R/W 1'b0	If this bit is set the P_INTA_L line will be asserted. This is intended for use in Device Mode when the device needs to interrupt the host. Note that the input from P_INTA_L is still active, so setting the bit will raise a PCI INTA interrupt to the interrupt mapper. This register should only be accessed from the ZBbus side of the PCI interface. Writing to this register from the PCI interface will have UNPREDICTABLE results.
31:1	reserved	R/O 31'b0	Reserved

**Table 137: PCI Read Host Register - Offset 94 Bits [31:0]**

Bits	Name	Default	Description
0	rd_host	W/O 1'b1	This bit controls CPU read access to the PCI configuration registers when the interface is in device mode.  By default this bit is set, allowing read access to the static PCI registers, the BAR0 map, the SubSysSet and SignalINTA registers. While this bit is set any read requests from the PCI bus will be issued a retry.  When this bit is clear the CPU can still write the registers but reads will return UNPREDICTABLE data.
31:1	reserved	W/O 31'b0	Reserved

**Table 138: PCI Adaptive Extend Register - Offset 98 Bits [31:0]**

Bits	Name	Default	Description
0	reserved	1'b0	Reserved
3:1	nom_tar_retry_h	R/W 3'b0	This field is the top three bits of the nom_tar_retry value used by the adaptive retry algorithm. (See <a href="#">Table 133</a> PCI Feature Control Register bits 10:7)
5:4	max_tar_retry_h	R/W 2'b0	This field is the top two bits of the max_tar_retry value used by the adaptive retry algorithm. (See <a href="#">Table 133</a> PCI Feature Control Register bits 15:11)
6	dis_dmar_iow_dep	1'b0	This bit should be zero for normal operation. If set no checking is performed for dependencies between DMA read data return and PIO writes.

**Table 138: PCI Adaptive Extend Register - Offset 98 Bits [31:0] (Cont.)**

Bits	Name	Default	Description
7	dis_memrd_be	1'b0	This bit should be zero for normal operation. If set the byte enables will always be all set for read transactions.
8	prefetch_en	R/W 1'b0	Interface revision 3 or greater: this bit may be set to enable read prefetching to improve performance of block reads from the ZBbus to the PCI.
9	prefetch_sz	R/W 1'b0	Interface revision 3 or greater: if this bit is 0 two cachelines are prefetched when the prefetch_en bit is set, if it is 1 then 4 cachelines are prefetched.
31:10	notimp	22b'x	Not Implemented.

**Table 139: PCI Bypass Control Register - RevId >= 3 Offset A8 Bits [31:0]**

Bits	Name	Default	Description
11:0	num_pior_byp	R/W 12'hFFF	A ZBbus or HT-to-PCI initiated read will be retried a total of num_pior_byp multiplied by the retry_timeout (in the Timeout register). Each retry_timeout there is a chance for one write to bypass the read (this is a workaround for older devices that did not correctly implement the PCI ordering rules).
31:12	notimp	20b'x	Not Implemented.

In Host Mode the header should be configured by the local processor (in many cases the default values will be reasonable and only the map table will need to be initialized), in Device Mode it will be configured by the external host.

The map table allows control of what areas of the internal memory map are accessible to an external PCI master. Therefore it is protected and can only be used with configuration accesses from the ZBbus side of the bridge. Configuration accesses to the map table registers will be accepted from the PCI bus, but the writes will be ignored and reads will return UNPREDICTABLE values. When the interface is run in Device Mode, the external master will always see the BAR0 region as a 16 MByte space that needs an address allocation, software running on the device should configure the map table to allow access to appropriate internal resources. The peer-to-peer mappings in the map register give the part some of the characteristics of a non-transparent PCI to HyperTransport bridge.

The SubSysSet register can only be written from the ZBbus side of the bridge. It is used to provide a value in the SubSystem Device and Vendor Id registers seen by external configuration reads. In Device Mode software on the device should write the SubSysSet register with a value that identifies the manufacturer of the option card and its part number. This write must be done early in initialization, before the host needs to read the value. On interface revision 3 and later the Device/Vendor Id and the Class code that will be read by the external host can also be changed by using the VendorIdSet and ClassRevSet registers.



## HYPERTRANSPORT CONFIGURATION HEADER

The HyperTransport configuration header is a standard PCI Type 1 (bridge) header, as defined in the PCI-to-PCI Bridge Architecture Specification revision 1.1, with the extensions required by the HyperTransport specification and device specific control extensions. The following table shows the fields of the header, along with the values they contain after a system reset. Values marked R/O are read only.

**Table 140: HyperTransport Configuration Header (Type 1)**

Offset	Register Bits							Description
	31	24	23	16	15	8	7	
00	Device Id R/O 0002			Vendor Id R/O 166D				Identifies the device as a Broadcom SiByte family 8 bit HyperTransport interface.
04	Status (See <a href="#">Table 142 on page 247</a> ) R/W 0010			Command (See <a href="#">Table 141 on page 246</a> ) R/W 0003				As defined in the HyperTransport specification.
08	Class Code R/O 060000				Rev Id R/O xx			Class is a host bridge. The revision code is: 1 - for early prototype BCM1250s. 2 - for initial production BCM1250s 3 - for initial production BCM1125H and later BCM1250s
0C	BIST R/O 00	Hdr Type R/O 01	LatTimer R/O 00	ClineSz R/O 00				This is a bridge header. As defined in the HyperTransport specification.
10	BAR 0 R/O 00000000							Not used.
14	BAR 1 R/O 00000000							Not used.
18	SecLatTimer R/O 00	Subord Bus# R/W 00	Sec bus# R/W 00	Pri Bus# R/O 00				Primary bus is fixed in hardware as 0. Secondary and subordinate buses must be set by the PCI/HyperTransport bus enumeration code.
1C	Sec Status (See <a href="#">Table 143 on page 247</a> ) R/C 0000		I/O Limit R/W x1	I/O Base R/W x1				The low 4 bits of the base and limit registers indicate 32 bit I/O addressing and register 30 is used. The upper four bits are used to specify bits [15:12] of the base and limit of the range of I/O addresses that should be sent on the HyperTransport fabric. This sets the address of region J in <a href="#">Figure 37 on page 193</a> .
20	Mem Limit R/W xxx0		Mem base R/W xxx0					The low four bits of the base and limit registers are always zero. The upper 12 bits specify address bits [31:20] of the base and limit of the range of memory addresses that should be sent on the HyperTransport fabric. This sets the address of region B in <a href="#">Figure 37 on page 193</a> . If the limit address is set lower than the base address then region B is disabled.
24	Prefetch Limit R/O 0000		Prefetch base R/O 0000					These registers read as zero, since no special prefetchable memory region is supported.
28	Prefetch base upper 32 bits R/O 00000000							
2C	Prefetch limit upper 32 bits R/O 00000000							
30	I/O limit upper 16 bits R/W 0xxx		I/O base upper 16 bits R/W 0xxx					The upper seven bits of the base and limit are fixed at zero. The lower nine bits are used to specify bits [24:16] of the base and limit of the range of I/O addresses that should be sent on the HyperTransport fabric. This sets the address of region J in <a href="#">Figure 37 on page 193</a> .



Table 140: HyperTransport Configuration Header (Type 1) (Cont.)

Offset	Register Bits								Description
	31	24	23	16	15	8	7	0	
34	Reserved R/O 000000						Cap Ptr R/O 40		Points to HyperTransport capability registers.
38	ROM base address R/O 0								The HyperTransport bridge does not support an expansion ROM.
3C	Bridge Ctrl (See <a href="#">Table 144 on page 248</a> ) R/W 0000			Int Pin R/O 0		Int line R/O 0		<p>The HyperTransport does not use the interrupt registers, they always read as zero.</p> <p>On interface revision 3 and later the Int Line register is R/W but unused by the hardware as recommended by the HT specification.</p>	
40	HTCmd (See <a href="#">Table 145 on page 248</a> ) R/W 2001			Cap Ptr R/O 00		Cap Id R/O 08		This is the HyperTransport capability block, there are no further blocks.	
44	LinkConfig (See <a href="#">Table 147 on page 250</a> ) R/O 0000			LinkCtrl (See <a href="#">Table 146 on page 249</a> ) R/W 0000				As defined in the HyperTransport specification.	
48	Reserved R/O 0000			LDT Freq ( <a href="#">Table 148 on page 250</a> ) R/W 00		LDT Rev Id R/O 11		Interface revId 1 and 2 is based on HyperTransport revision 0.17.	
	RevId >=3 LinkFreqCap R/O 801F					RevId >=3 R/O 23		Interface revision 3 is based on HyperTransport revision 1.03. The LinkFreqCap indicates operation at 200-600MHz and vendor specific frequencies.	
4C	Reserved R/O Unpredictable			Reserved R/O Unpredictable				Reserved on RevId 1 and 2.	
	R/O 0000			RevId >=3 Features R/O 0004				On revision 3 and later, this is the feature register indicating support for CRC test and not other features.	
50	SriCmd (See <a href="#">Table 149 on page 251</a> ) R/W 0000			SriRxDen R/W 10		SriTxDen R/W 10		System Reset Initialization registers. These registers must be configured by the CPU before the HyperTransport fabric comes out of reset. The interface will assert the LDT_RESET_L signal until the SipReady bit in the SriCmd register is set to indicate that these registers have been programmed.	
54	SriTxNumerator R/W 0000FFFF								
58	SriRxNumerator R/W 0000FFFF								
5C	Reserved R/O Unpredictable								Reserved on RevId 1 and 2.
	RevId >=3 IsocBAR R/W 00000000								On RevId 3 and later, the IsocBAR and IsocIcnMask selects an address range that inbound transactions will be allocated in the L2 cache.
60	Reserved R/O Unpredictable								See <a href="#">Section: "Force Isochronous Mode Address Range" on page 212</a>
	RevId >=3 IsocIcnMask R/W 00000000								
64	Reserved R/O Unpredictable								Reserved
68	ErrStatus ( <a href="#">Table 153 on page 253</a> ) R/C 00	ErrCtrl (See <a href="#">Table 152 on page 252</a> ) R/W 000000							Error control and status register. Provides additional control over interface error reporting.
6C	Reserved R/O 00	TxCtrl (See <a href="#">Table 154 on page 253</a> ) R/W 04		DataBufAloc ( <a href="#">Table 155 on page 253</a> ) R/W 1515				Control registers.	

**Table 140: HyperTransport Configuration Header (Type 1) (Cont.)**

Offset	Register Bits						Description
	31	24	23	16	15	8 7 0	
70-C4	Reserved R/O Unpredictable						Reserved
C8	TxBufCountMax (See <a href="#">Table 157 on page 254</a> ) R/W 00FFFFFF						Transmit buffer control register.
CC-D8	Reserved R/O Unpredictable						Reserved
DC	DiagRxCrcE (See <a href="#">Table 158 on page 254</a> ) R/O xxxxxxxx						Expected CRC (diagnostic register).
E0-EC	Reserved R/O Unpredictable						Reserved
F0	DiagRxCrcR (See <a href="#">Table 159 on page 254</a> ) R/O xxxxxxxx						Received CRC (diagnostic register).
F4-FC	Reserved R/O Unpredictable						Reserved

**Table 141: HyperTransport Bridge Command Register - Offset 4 Bits [15:0]**

Bits	Name	Default	Description
0	IoSpaceEn	R/O 1'b1	I/O Space Enable. This bit is always set. The HyperTransport bridge always forwards I/O space transactions that originate on the ZBbus and match the bridge I/O range. Peer-to-Peer I/O is not supported between PCI and HyperTransport.
1	MemSpaceEn	R/O 1'b1	Memory Space Enable. This bit is always set. The HyperTransport bridge always forwards memory space transactions that originate on the ZBbus and match the bridge memory range. Peer-to-Peer transfers from the PCI bus will always be accepted by the HyperTransport bridge, but they can be disabled in the PCI bridge by clearing the ptp_en bit in the PCI Feature Control Register (see <a href="#">Table 133 on page 241</a> ).
2	MasterEn	R/W 1'b0	Primary Bus (i.e. ZBbus) Master Enable. This bit controls acceptance of requests from the HyperTransport fabric. If a request has unitid=0 and contains an address that should be sent back on the HyperTransport fabric (a peer-to-peer operation) it must have come the whole length of the fabric from another host bridge and it will always be responded to with an NXA error. 0: Accept only Configuration Cycles and requests that must be sent back out on the HyperTransport fabric. All other posted requests are dropped and non-posted requests return an NXA error. 1: Accept all requests that match an address range in this part.
3	SpecCycEn	R/O 1'b0	These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.
4	MemWrInvEn	R/O 1'b0	
5	VgaPalSnpEn	R/O 1'b0	
6	ParErrResp	R/O 1'b0	
7	WaitCycCtrl	R/O 1'b0	
8	SerrEn	R/W 1'b0	SERR enable. This controls the SERR on the primary interface. Since the ZBbus has no equivalent of SERR the setting of this bit is ignored.
9	FastB2BEn	R/O 1'b0	These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.
15:10	reserved	R/O 6'b0	

**Table 142: HyperTransport Bridge Primary (ZBbus) Status Register - Offset 4 Bits [31:16]**

Bits	Name	Default	Description
3:0	reserved	R/O 4'b0	Reserved
4	CapList	R/O 1'b1	Always set. There is a capabilities list pointed to by the pointer register.
5	66MHzCap	R/O 1'b0	
6	reserved	R/O 1'b0	
7	FastB2BCap	R/O 1'b0	These bits apply to PCI bridges only. This register is the primary interface status register and contains the details for the ZBbus side of the bridge. These bits always return zero.
8	MstrDParErr	R/O 1'b0	
10:9	DevselTiming	R/O 2'b0	
11	SigdTgtAbort	R/C 0'b0	This bit is set when the HyperTransport interface returns an error to the ZBbus. It is cleared when written with a 1. Note that an error code sent with the data is the ZBbus equivalent of a Target Abort.
12	RcvdTgtAbort	R/O 1'b0	Always clear. Errors from the ZBbus are logged by the Bus Watcher in the SCD and errors from peer-to-peer PCI accesses are logged in the PCI bridge.
13	RcvdMstrAbort	R/O 1'b0	
14	SigdSerr	R/O 1'b0	The interface does not directly signal SERR on the ZBbus interface, so this bit always returns zero. If a Sync packet is seen on the HyperTransport interface it will not be forwarded but it will be reported in the DetSerr bit in the Secondary Status register.
15	DetParErr	R/O 1'b0	Always clear. Errors from the ZBbus are logged by the Bus Watcher in the SCD.

**Table 143: HyperTransport Bridge Secondary (HT) Status Register - Offset 1C Bits [31:16]**

Bit	Name	Default	Description
4:0	reserved	R/O 5'b0	
5	66MHzCap	R/O 1'b0	
6	reserved	R/O 1'b0	These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.
7	FastB2Bcap	R/O 1'b0	
8	MstDParErr	R/O 1'b0	
10:9	DevSelTiming	R/O 2'b0	
11	SigdTgtAbort	R/C 1'b0	This bit is set when the bridge issues a non-NXA error response on the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.
12	RcvdTgtAbort	R/C 1'b0	This bit is set when the bridge receives a non-NXA error response from the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.
13	RcvdMstrAbort	R/C 1'b0	This bit is set when the bridge receives an NXA error response from the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.
14	DetSerr	R/C 1'b0	This bit is set when the bridge detects SYNC packet flooding of the HyperTransport fabric. It is cleared when written with a 1. This bit is persistent through a warm reset.
15	DetParErr	R/O 5'b0	This bit applies to PCI bridges only. For a HyperTransport bridge it is read-only and always returns a zero.

**Table 144: HyperTransport Bridge Control Register - Offset 3C Bits [31:16]**

Bits	Name	Default	Description
0	ParErrRespEn	R/O 1'b0	This bit applies to PCI bridges only. For a HyperTransport bridge it is read only and always returns a zero.
1	SerrEn	R/W 1'b0	This bit must be set to enable detection of Sync flooding on the HyperTransport link.
2	IsaEn	R/O 1'b0	This bit is always zero. The HyperTransport bridge does not support modified ISA address forwarding.
3	VgaEn	R/W 1'b0	This bit controls the routing of VGA address accesses in the compatibility spaces (0A_0000-0B_FFFF in memory space and X3B0-X3BB and X3C0-X3DF in I/O space). If this bit is clear these addresses are sent to the PCI interface and if set they are sent (without the COMPAT bit set) to the HyperTransport fabric.
4	Reserved	R/O 1'b0	Reserved
5	MstrAbortMode	R/W 1'b0	Error response behavior. This bit determines the way errors are forwarded through the HyperTransport bridge. If this bit is set: Any ZBbus error causes a HyperTransport Error response with the NXA bit clear. Any HyperTransport error causes a ZBbus Bus Error. If this bit is clear: Any ZBbus Bus Error or Fatal Error causes a valid HyperTransport response. Any other ZBbus Error causes a HyperTransport error response with the NXA bit clear. Any HyperTransport error with the NXA bit set causes a ZBbus response with valid data that is all ones. Any HyperTransport error with the NXA bit clear causes a ZBbus Bus Error response.
6	SecBusReset	R/W 1'b0	If this bit is set the LDT_RESET_L signal is asserted. If the WarmReset bit in the HyperTransport host interface command register is clear, the LDT_PWROK signal is deasserted to cause a cold reset. When this bit is set, clearing it will bring the fabric out of reset.
7	FastB2BEn	R/O 1'b0	
8	PriDiscard	R/O 1'b0	
9	SecDiscard	R/O 1'b0	These bits apply to PCI bridges only. For a HyperTransport bridge they are read only and always return zero.
10	DiscardStat	R/O 1'b0	
11	DiscardSerrEn	R/O 1'b0	
15:12	reserved	R/O 4'b0	Reserved

**Table 145: HyperTransport Command Register - Offset 40 Bits [31:16]**

Bits	Name	Default	Description
0	WarmReset	R/W 1'b1	If this bit is set then a warm reset of the fabric is done when the SecBusReset bit is set in the Bridge Control Register. If this bit is clear a cold reset is done. Changing the state of this bit when the SecBusReset bit is asserted results in UNDEFINED behavior.
1	DoubleEnded	R/W 1'b0	This bit is set by the master host bridge on other side of the chain to indicate that the slave host bridge is successfully configured on a double ended link. This bit is just used by software, it has no affect on hardware.
6:2	Reserved	R/O 5'b0	On interface RevID 1 and 2 this field is reserved.
	RevID>=3 DevNum	R/W 5'b0	On interface with RevId 3 and greater this field is used to set the Unit ID the part will use if the ActAsSlave bit is set. The default is restored on a cold reset.



**Table 145: HyperTransport Command Register - Offset 40 Bits [31:16] (Cont.)**

Bits	Name	Default	Description
7	Reserved	R/O 1'b0	On interface RevID 1 and 2 this field is reserved.
	RevID>=3 ChainSide	R/W 1'bx	On interface with RevId 3 and greater this bit reads as a 0 from the ZBbus side and 1 from the chain side.
8	Reserved	R/O 1'b0	On interface RevID 1 and 2 this field is reserved.
	RevID>=3 HostHide	R/O 1'b0	The HostHide feature is not supported.
9	Reserved	R/O 1'b0	Reserved
10	Reserved	R/O 1'b0	On interface RevID 1 and 2 this field is reserved.
	RevID>=3 ActAsSlave	R/W 1'b0	On interface with RevId 3 and greater if this bit is set the interface acts as a slave and uses the unit ID set in the DevNum field. The default is forced on a cold reset and any change will only take effect on a link warm reset.
11	Reserved	R/O 1'b0	On interface RevID 1 and 2 this field is reserved.
	RevID>=3 InboundEOC Error	R/O 1'b0	On interface with RevId 3 and greater this bit is not used. The error reporting is compatible with the older version of the interface and differs from 1.03.
12	Reserved	R/O 1'b0	On interface RevID 1 and 2 this field is reserved.
	RevID>=3 DropOnUninit	R/O 1'b0	The DropOnUninit feature is not supported.
15:13	CapType	R/O 3'b001	This field indicates that this is a host/secondary bridge capability type.

**Table 146: HyperTransport Link Control Register - Offset 44 Bits [15:0]**

Bits	Name	Default	Description
0	reserved	R/O 1'b0	Reserved
1	CrcSyncFloodEn	R/W 1'b0	If set CRC errors will cause a sync flood and set the Link Failure bit, and will therefore jam the link. If clear a sync flood is not generated and the bit not set. CRC checking is always enabled and errors are always logged in the CRC error bits.
2	CrcStartTest	R/S 1'b0	If this bit is set the bridge initiates a CRC test sequence on the link (see the HyperTransport specification for details). The bit is cleared by the hardware when the test is complete, software can then check the CRC Error bit to check the status of the test.
3	CrcForceErr	R/W 1'b0	If this bit is set the bridge generates bad CRCs on the outgoing link.
4	LinkFail	R/W 1'b0	This bit is set if a link failure is detected. This bit is persistent through warm reset. If this bit is set link synchronization will not be done and the Initialization Complete bit will not get set on either end of the link. If this bit is changed by software it will only have effect after the next warm reset.
5	InitDone	R/O 1'b0	This bit is set by the bridge to indicate that the low level link initialization has completed successfully.
6	EOC	R/S 1'b0	This bit may be set by software to indicate this is the last device in a logical chain. If set all received packets are ignored. If the XmitOff bit is clear the bridge will generate NOP packets and good CRCs on its outgoing link. Once set this bit can only be cleared by a fabric reset.

**Table 146: HyperTransport Link Control Register - Offset 44 Bits [15:0] (Cont.)**

Bits	Name	Default	Description
7	XmitOff	R/S 1'b0	This bit can be set by the software to shut off a transmitter. When set none of the output pins will toggle. If the EOC bit is set on an active link XmitOff should not be set until the transmitter has driven enough NOP packets to fill the receiver's receive FIFOs. Once set this bit can only be cleared by a fabric reset.
11:8	CrcErr	R/C 4'b0	When a CRC error is received on the incoming link the bridge will set the bit in this field corresponding to the byte lane that had the error. The BCM1250 only supports an 8-bit link, so bit [8] is the only one that will ever be set. Software may clear these bits by writing a 1 to them.
15:12	reserved	R/O 4'b0	Reserved

**Table 147: HyperTransport Link Configuration Register - Offset 44 Bits [31:16]**

Bits	Name	Default	Description
<b>Note that this register matches the HyperTransport 1.0 standard, and is a superset of rev 0.17</b>			
2:0	MaxIn	R/O 3'b000	This field indicates the BCM1250 only supports an 8 bit input link.
3	DwFcln	R/O 1'b0	The bridge is not capable of Doubleword Flow Control.
6:4	MaxOut	R/O 3'b000	This field indicates the BCM1250 only supports an 8 bit output link.
7	DwFcOut	R/O 1'b0	The bridge is not capable of Doubleword Flow Control.
10:8	WidthIn	R/O 3'b000	This field is fixed for an 8 bit link.
11	DwFclnEn	R/O 1'b0	The bridge is not capable of Doubleword Flow Control.
14:12	WidthOut	R/O 3'b000	This field is fixed for an 8 bit link.
15	DwFcOutEn	R/O 1'b0	The bridge is not capable of Doubleword Flow Control.

**Table 148: HyperTransport Link Frequency Register - Offset 48 Bits [15:8]**

Bits	Name	Default	Description
3:0	LinkFreq	R/W 4'b0000	This register sets the maximum transmitter clock frequency for the link. The BCM1250 uses this value to control the PLL that provides both transmit and receive internal clocks using the CLK100 reference. If the sriLinkFreqDirect bit is clear in the sriCmd register, these bits have the encoding given in the HyperTransport 1.0 standard: 0000: 200 MHz (CLK100 x2) 0001: 300 MHz (CLK100 x3) 0010: 400 MHz (CLK100 x4) 0011: 500 MHz (CLK100 x5) 0100: 600 MHz (CLK100 x6) 0101: Not supported (code for 800 MHz) 0110: Not supported (code for 1000 MHz) Others: UNDEFINED
4	LinkFreq	R/W 1'b0	If the sriLinkFreqDirect bit is set in the sriCmd register then the [4:0] field is used to set the PLL frequency directly, using the values shown in <a href="#">Table 9 on page 31</a> . The link frequency will be set to CLK100 * LinkFreq[4:0]/2.  The HyperTransport link clock frequency must not be set faster than four times the frequency of the I/O bridge 0 clock. If the bridge is set (using the configuration resistor on IO_AD[5]) to CPU clock/4 then the HyperTransport clock frequency must be less than or equal to the CPU clock frequency. If the bridge is set to CPU clock/3 then the HyperTransport link clock must be run at less than or equal to 4/3 of the CPU clock frequency.  Note that the sriLinkFreqDirect bit must be set before the frequency code is written to this register for the alternate frequencies to be used.
7:5	Reserved	R/O 3'b0	Reserved

**Table 149: HyperTransport SRI Command Register - Offset 50 Bits [31:16]**

Bits	Name	Default	Description																								
0	SipReady	R/W 1'b0	System Initialization Process Ready. This bit should be set by software when the SRI initialization process is complete. The LDT bridge will be held in reset until this bit is set.																								
1	SyncPtrCtl	R/W 1'b0	This bit should be set to enable synchronous pointer control. If the interface is synchronous the pointers in the receive FIFO are moved according to a fixed pattern based on the ratio of the load and unload clocks. If the interface is asynchronous the pointers are moved by sampling the incoming receive clock and counting the number of edges.																								
2	ReduceSyncZero	R/W 1'b0	If this bit is set only 128 bit times of zeros will be sent during link initialization rather than the standard 512 bit times. Only set during testing.																								
3	DisMultTxVld	R/W 1'b0	If clear (default), all 3 virtual channels can send requests simultaneously. If set, only one channel can make a request at a time (as in RevId). This bit is for debug use only.																								
8:4	RxMargin	R/W 5'h0	This sets the offset between the receive FIFO load and unload pointers.																								
9	sriLdtPLLCompat	R/W 1'b0	This bit should be set for compatibility with 0.17 HyperTransport devices.  It modifies the behavior for updating the LdtLinkFreq register (the one visible to programmer) and moving it into LdtPLLFreq (the shadow copy used to control the PLL). Older fixed frequency devices need this bit set, so a link cold reset does not alter the link frequency. HyperTransport Rev 1.0 devices reset their link frequency to 200 MHz on a link cold reset. sriLdtPLLCompat set to 1 => compatible with API AP1011 / SiPackets SP1011 sriLdtPLLCompat set to 0 => comply with 1.0 spec.  This table summarizes the behavior on reset: <table border="1" style="margin-left: 40px;"> <thead> <tr> <th></th> <th>LdtLinkFreq</th> <th>LdtPLLFreq</th> </tr> </thead> <tbody> <tr> <td><b>System reset</b></td> <td>200MHz</td> <td>200MHz</td> </tr> <tr> <td><b>Link cold reset</b></td> <td></td> <td></td> </tr> <tr> <td>  Compat == 0</td> <td>200MHz</td> <td>200MHz</td> </tr> <tr> <td>  Compat == 1</td> <td>No Change</td> <td>No Change</td> </tr> <tr> <td><b>Link warm reset</b></td> <td></td> <td></td> </tr> <tr> <td>  Compat == 0</td> <td>No Change</td> <td>copied from LdtLinkFreq</td> </tr> <tr> <td>  Compat == 1</td> <td>No Change</td> <td>No Change</td> </tr> </tbody> </table>		LdtLinkFreq	LdtPLLFreq	<b>System reset</b>	200MHz	200MHz	<b>Link cold reset</b>			Compat == 0	200MHz	200MHz	Compat == 1	No Change	No Change	<b>Link warm reset</b>			Compat == 0	No Change	copied from LdtLinkFreq	Compat == 1	No Change	No Change
	LdtLinkFreq	LdtPLLFreq																									
<b>System reset</b>	200MHz	200MHz																									
<b>Link cold reset</b>																											
Compat == 0	200MHz	200MHz																									
Compat == 1	No Change	No Change																									
<b>Link warm reset</b>																											
Compat == 0	No Change	copied from LdtLinkFreq																									
Compat == 1	No Change	No Change																									
10	exp_endian	R/W 1'b0	Selects endian policy for expansion space. If this bit is clear then all of the expansion space will use the match byte lane policy. If the bit is set then address bit [38] is used to select the endian policy and the address bit is cleared as the request passes through the bridge.																								
11	reserved	R/O 1'b0	Reserved																								
14:12	TxInitialOffset	R/W 3'b000	This sets the initial offset between the transmit FIFO unload and load pointers.																								
15	sriLinkFreqDirect	R/W 1'b0	If this bit is clear the HyperTransport link frequency is set according to the HyperTransport specification revision 1.0. If it is clear a wider range of frequencies can be set. See <a href="#">Table 148 on page 250</a> .																								

**Table 150: HyperTransport Isochronous BAR - Offset 5C Bits [31:0]**

Bits	Name	Default	Description
0	isocEn	R/W 1'b0	If this bit is set then the IsocBAR and IsocIgnMask are used to force inbound transactions within the specified range to behave as if their Isochronous bit is set.
31:1	isocBase	R/W 31'b0	This register sets the address bits 35:5 that are compared to force the isochronous bit. See <a href="#">Section: "Force Isochronous Mode Address Range" on page 212</a> .



**Table 151: HyperTransport Isochronous Ignore Mask - Offset 60Bits [31:0]**

Bits	Name	Default	Description
0	reserved	R/W 1'b0	Reserved
31:1	isocMask	R/W 31'b0	This specifies the mask of bits that are ignored when the comparison is done to force the isochronous bit. See <a href="#">Section: "Force Isochronous Mode Address Range" on page 212.</a>

**Table 152: HyperTransport Error Control Register - Offset 68 Bits [23:0]**

Bits	Name	Default	Description
0	ProtFatalEn	R/W 1'b0	If this bit is set a fatal interrupt will be raised when a protocol error is detected.
1	ProtNonFatalEn	R/W 1'b0	If this bit is set a nonfatal interrupt will be raised when a protocol error is detected.
2	ProtSyncFloodEn	R/W 1'b0	If this bit is set a protocol error will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.
3	OvfFatalEn	R/W 1'b0	If this bit is set a fatal interrupt will be raised when a receive buffer overflow is detected.
4	OvfNonFatalEn	R/W 1'b0	If this bit is set a nonfatal interrupt will be raised when a receive buffer overflow is detected.
5	OvfSyncFloodEn	R/W 1'b0	If this bit is set a receive buffer overflow will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.
6	EocNxaFatalEn	R/W 1'b0	If this bit is set a fatal interrupt will be raised when a posted request to a nonexistent address or response that does not match a request is detected. (Non-posted requests will receive an NXA error in this case).
7	EocNxaNonFatalEn	R/W 1'b0	If this bit is set a nonfatal interrupt will be raised when a posted request to a nonexistent address or response that does not match a request is detected. (Non-posted requests will receive an NXA error in this case).
8	EocNxaSyncFloodEn	R/W 1'b0	If this bit is set a posted request needing an NXA or a response that does not match a request will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.
9	CrcFatalEn	R/W 1'b0	If this bit is set a fatal interrupt will be raised when a receive CRC error is detected.
10	CrcNonFatalEn	R/W 1'b0	If this bit is set a nonfatal interrupt will be raised when a receive CRC error is detected.
11	SerrFatalEn	R/W 1'b0	The SerrEn bit in the Bridge Control register determines if an SERR will generate an interrupt. If this bit is set a fatal interrupt will be raised, if clear a nonfatal interrupt is used.
12	SrcTagFatalEn	R/W 1'b0	If this bit is set a fatal interrupt will be raised when a source tag error is detected.
13	SrcTagNonFatalEn	R/W 1'b0	If this bit is set a nonfatal interrupt will be raised when a source tag error is detected.
14	SrcTagSyncFloodEn	R/W 1'b0	If this bit is set a source tag error will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.
15	MapNxaFatalEn	R/W 1'b0	If this bit is set a fatal interrupt will be raised when an NXA error response is sent to a request with zero source id.
16	MapNxaNonFatalEn	R/W 1'b0	If this bit is set a nonfatal interrupt will be raised when an NXA error response is sent to a request with zero source id.
17	MapNxaSyncFloodEn	R/W 1'b0	If this bit is set an NXA error response to a request with zero source id will cause SYNC flooding of the HyperTransport link and the LinkFail bit will be set.
23:18	Reserved	R/O 6'b0	Reserved



**Table 153: HyperTransport Error Status Register - Offset 68 Bits [31:24]**

Bits	Name	Default	Description
0	ProtoErr	R/C 1'b0	This bit will be set if a protocol error is detected on the incoming link. Software may clear this bit by writing a 1 to it. In some circumstances this bit will be set by a HyperTransport link reset.
1	OvfErr	R/C 1'b0	This bit will be set if the receive FIFO overflows. Software may clear this bit by writing a 1 to it.
2	EocNxaErr	R/C 1'b0	This bit will be set if a Non Existent Address (NXA) error is returned to a request from the incoming link because the End Of Chain bit (bit 6 in the link control register) is set. Software may clear this bit by writing a 1 to it. This bit will also be set if a request is rejected because the link has not been established (the InitDone bit is clear).
3	SrcTagErr	R/C 1'b0	This bit will be set if a response packet is received with a source id that does not match any outstanding request. Software may clear this bit by writing a 1 to it.
4	MapNxaError	R/C 1'b0	This bit will be set if a Non Existent Address (NXA) error is returned due to a mapping problem with the request. There are two causes: 1) A packet is received that does not match one of the valid address ranges in <a href="#">Figure 43 on page 210</a> . 2) A packet is received that is decoded as having an address on the HyperTransport link and with a zero source id. This indicates that the packet is a peer-to-peer HyperTransport transaction, but it originated in the host bridge at the far end of the link and has therefore not been accepted by any device on the link. This bit is also set (but no error packet sent) if a response is received that has source id of zero and did not come from this host. Software may clear this bit by writing a 1 to it.
7:5	Reserved	R/O 3'b0	Reserved

**Table 154: HyperTransport SRI Transmit Control Register - Offset 6C Bits [23:16]**

Bits	Name	Default	Description
3:0	BufRelSpace	R/W 4'b0100	This field sets the minimum number of packets that will be sent between buffer release NOP packets. If the link is idle buffer release packets are sent immediately. If the link is busy buffer release messages are inserted into the packet stream with this minimum spacing so that the bandwidth is not consumed with flow control traffic.
7:4	Reserved	R/O 4'b0	Reserved

**Table 155: HyperTransport SRI Data Buffer Allocation Register - Offset 6C Bits [15:0]**

Bits	Name	Default	Description
1:0	NeedResp	R/W 2'b01	This register controls the allocation of the 8 HyperTransport receive data buffers among the 3 virtual channels, to allow performance tuning.
3:2	NeedNpReq	R/W 2'b01	
5:4	NeedPReq	R/W 2'b01	The "Need" fields indicate the minimum allocation at all times to each channel, minus one. That is, a value of 0 indicates a permanent minimum allocation of 1. The total number of buffers "needed" must be less than or equal to 8 or the behavior of the bridge is UNDEFINED. If the number of "needed" buffers is less than 8 the allocator will dynamically allocate them to the three channels, subject to the limits set in the "Want" fields.
7:6	Reserved	R/O 2'b00	
9:8	WantResp	R/W 2'b01	
11:10	WantNpReq	R/W 2'b01	The "Want" fields indicate how many buffers the allocator should try to have released and outstanding to each channel at all times, minus 1.
13:12	WantPReq	R/W 2'b01	
15:14	Reserved	R/O 2'b00	

In the default (reset) case, there are 2 buffers in each category.

Note that these counts must not be set to 2'b11 (i.e. 4 buffers) or the system may behave in UNDEFINED ways.

**Table 156: HyperTransport Additional Status Register - Offset 70**

Bits	Name	Default	Description
7:0	tgt_done	R/W 8'b0	Target Done Counter. This value is incremented every time a Target Done acknowledgement is received from the fabric. These will happen as a result of the NonPosted writes done in configuration space. Software may write this field, incrementing will continue from the value written.
31:8	notimp	26b'x	Not Implemented.

**Table 157: HyperTransport SRI Transmit Buffer Count Max Register - Offset C8 Bits [31:0]**

Bits	Name	Default	Descriptions
3:0	PCmd	R/W 4'b1111	These fields set the maximum number of each class of buffer the transmitter will use. The classes (Posted Command, Posted Data, NonPosted Command, NonPosted Data, Response and Response Data) match the buffers in the receiver and fields in the NOP flow control packet.
7:4	Pdata	R/W 4'b1111	
11:8	NpCmd	R/W 4'b1111	
15:12	NpData	R/W 4'b1111	NOP packets received from the other end of the link will increment the number of buffers that the transmitter believes the receiver has available. If the count reaches the limit set in this register the extra credits will be discarded. This allows a general way to throttle traffic in a particular virtual channel on a link.
19:16	RCmd	R/W 4'b1111	
23:20	RData	R/W 4'b1111	
			The maximum counter values should be setup before the sipReady bit is set. The values may be lowered in a running system, but the link must be reset before any increase takes effect.
31:24	reserved	R/O 8'b0	Reserved

**Table 158: HyperTransport Diagnostic Receive CRC Expected - Offset DC**

Bits	Name	Default	Description
31:0	RxCrcE	R/O 32'bx	This contains the expected value of the receive CRC.

**Table 159: HyperTransport Diagnostic Receive CRC Received - Offset F0**

Bits	Name	Default	Description
31:0	RxCrcR	R/O 32'bx	This register contains the actual value received for the CRC.



### Configuration Flags in the SriCmd Register

There are three flags in the **SriCmd** register. Two of these should be set at the start of initialization.

The ReduceSyncZero flag should only be set during debugging; it reduces the number of bit times of zeros during link initialization from the standard 512 to 128.

The SyncPtrCtl should be set if the interface is running in Synchronous Clock Mode (using the same reference clock source as the other end of the link) and clear for Asynchronous Mode. Selection of the correct mode depends on the system design, it will normally be coded in to the boot ROM but could also be set on the software configuration bits in the system control register.

The third flag is the SipReady flag. This should be left clear while the initialization is done. Once all the registers (including the **SriCmd** register) have been setup this bit should be set.

The SipReady bit is sticky. Once it has been set the only way to clear it is a cold reset. Configuration code should check the state of SipReady to determine if the full initialization sequence or an abbreviated version is needed.

### Timing Registers: SriRxDen, SriTxDen, SriRxNum and SriTxNum

The receive timing registers are used when the interface is running in synchronous clocking mode, the transmit registers must always be configured. The registers describe the relationship between the receive clock from the HyperTransport link ( $f_{RX}$ ) and the internal HyperTransport interface clock ( $f_{LDTINT}$ ), and the relationship between the internal HyperTransport interface clock and the HyperTransport transmit clock ( $f_{TX}$ ). Note that the receive and transmit clocks used in this discussion are the actual clocks on the link and therefore are half of the data rate.

The transmit clock frequency is set in the LinkFreq register in the HyperTransport Capability block. The 100 MHz reference clock is multiplied up by the HyperTransport PLL to give the requested frequency. The internal HyperTransport interface clock is always one quarter of the transmit clock frequency:  $f_{LDTINT} = f_{TX}/4$ .

The receive clock frequency is set by the device the other end of the link. Data is inserted into the receive FIFO byte wide sent on both edges of the receive clock, it is removed from the FIFO 8 bytes wide clocked on the rising edge of the internal HyperTransport interface clock. The data rate ratio is therefore:  $f_{RX}/4 * f_{LDTINT} = f_{RX}/f_{TX}$ .

The **SriRxDen** and **SriRxNum** registers encode this receive ratio. The Numerator is a bit pattern in which a 1 indicates that data should be extracted from the receive FIFO. The Denominator sets the number of bits in the Numerator that are used. Two bits from the Numerator pattern are examined each internal cycle, starting from bit zero and ending at bit (Denominator-1), to determine how many 32 bit words should be read out of the FIFO in that cycle.

For example if the receive clock from the link (i.e. the transmit clock set in the LinkFreq register of the device the other end of the link)  $f_{RX}=300$  MHz and the transmit clock (set in the LinkFreq register)  $f_{TX}=400$  MHz then the receive data rate ratio is 3/4. The **SriRxNum** should therefore be set to 32'b1101 (or some similar pattern with three ones and a zero) and the **SriRxDen** should be set to 4 to indicate only the bottom four bits of the Numerator pattern are used.

In the common case where the clock frequency is the same at both ends of the link then the default values of **SriRxNum** = 32'hFFFF and **SriRxDen** = 16 will work.



The transmitter registers work similarly. Bits in the Numerator indicate that data should be inserted into the transmit FIFO from the internal clock. In the BCM1250 or BCM1125H, the clock ratio is fixed to 1/4. Data is inserted into the FIFO 8 bytes wide on the internal clock and removed byte wide on each edge of the HyperTransport transmit clock. These cancel, so the timing registers must be configured to allow data to be inserted every internal cycle. The defaults of **SriTxNum** = 32'hFFFF and **SriTxDen** = 16 are suitable values.

### Receive Pointer Margin Control in SriCmd Register

The receive FIFO is used to separate the link receive clock domain from the internal HyperTransport interface clock domain. To allow for the data to become stable and valid some time must be provided between writing an entry in the FIFO and reading it. This is done by having an offset (or margin) between the load and unload pointers. The initial separation is configured in the **SriCmd** register.

The value set for the initial margin is computed from the minimum margin that is needed once the interface is running, adjusted for the movement of the load and unload pointers during the initialization sequence. (For implementation reasons the value programmed in the configuration register is twice the basic computation, hence the extra doubling in the discussion below.)

The configuration register must be set to twice the offset that the unload pointer should have from the load pointer when the link is reset. If the pointers did not move during the initialization process this would simply be:

$$2 * (- \text{margin for data to be stable})$$

This is negative because it is measured from the load pointer to the unload pointer, which must always be behind. (Since the pointers cycle around the FIFO the offset must be taken mod 8 to get the actual value programmed.) In asynchronous mode the load and unload pointers are stationary during initialization so the correct value can be obtained directly from this computation. In synchronous mode the load and unload pointers are always being moved according to the ratio set in the **SriRxNum** and **SriRxDen** registers, so a correction must be applied.

In synchronous mode, the received SYNC pattern indication from the link passes through four synchronization stages clocked by the internal HyperTransport clock on its way in to the link initialization logic. The load pointer for the receive FIFO will advance every two receive clocks (the incoming data is on every edge of the clock, so the four byte FIFO width builds up in two clocks). The minimum advance of the FIFO load pointer during the SYNC synchronization is therefore:

$$\begin{aligned} & (\text{time for SYNC propagation to initialization logic}) / (\text{time per load pointer increment}) \\ & = (4 * \text{internal clock period}) / (2 * \text{receive clock period}) \\ & = 2 * f_{RX} / f_{LDTINT} \end{aligned}$$

The FIFO wraps every eight entries, so this value is taken modulo 8.

The pointer offset must also be adjusted to take into account the possible movement of the unload pointer. The worst case movement of the unload pointer must be used, since that brings it closest to the load pointer. The maximum unload pointer movement can be computed from the **SriRxNum** value. If there are two adjacent 1s anywhere (including over the wrap from bit [SriRxDen-1] to 0) in the Numerator then it is possible for two (32 bit) words to be extracted from the FIFO during SYNC. If there are only 10 and 01 patterns for adjacent bits then only one word can be extracted during SYNC.

On the BCM1250 or the BCM1125H, the recommended time allowed for the data to become stable is two cycles.

Putting the three terms together, the value to be programmed for the RxMargin field in synchronous mode is:

$$\begin{aligned} \text{RxMargin} &= 2 * (\text{load pointer updates} - \text{unload pointer updates} - \text{margin for data to be stable}) \\ &= 2 * (2 * f_{\text{RX}} / f_{\text{LDTINT}} - (1 \text{ or } 2) - 2) \end{aligned}$$

In asynchronous mode the RxMargin field should be programmed to twice the number of receive clock cycles for the data to settle. The settle time recommended for the BCM1250 or BCM1125H is 7ns, so in asynchronous mode:

$$\text{RxMargin} = 2 * 7\text{ns} * f_{\text{RX}}$$

### Transmit Pointer Initial Offset in the SriCmd Register

The transmit FIFO pointers must be offset to take account of the seven stages of synchronization using the HyperTransport transmit clock that are applied to the reset pulse, and to allow time for data settle margin. The minimum number of internal HyperTransport clocks taken for the synchronization time is given by:

$$\begin{aligned} \text{minInternalSyncClocks} &= 7 * \text{transmitClkPeriod} / \text{internalClkPeriod} \\ &= 7 * f_{\text{LDTINT}} / f_{\text{TX}} = 7/4 = 1 \text{ (rounding down)} \end{aligned}$$

The minimum advance of the load pointer can be found from the **SriTxNum** value. In this case the Numerator is all 1s so the minimum advance is 2, if the Numerator had 10 or 01 pairs of bits then the minimum advance would be 1.

The recommended data settle margin is 2. This gives:

$$\begin{aligned} \text{TxInitialOffset} &= \text{minInternalSyncClocks} - \text{minPointerAdvance} - \text{Margin} \\ &= 1 - 2 - 2 = -3 \text{ MOD } 8 = 5. \end{aligned}$$

### Error Control Register

The Error Control Register sets the error handling behavior of the HyperTransport interface for both the standard HyperTransport errors and the additional ones reported on the BCM1250 or BCM1125H. The possible responses to errors are: to raise the ldt\_fatal\_int interrupt, to raise the ldt\_nonfatal\_int interrupt and to flood the link with SYNC packets. Any of these may be selected as the response to any of the errors. (The CRC error causing SYNC flooding is set in the standard Link Configuration Register).

### Transmit Control Register

The transmit control register must be configured to set the minimum number of packets that will be sent between NOP packets that are reporting buffer releases to the device on the other end of the link. If the transmitter is idle, the NOP packet will be sent as soon as the buffer is free. If the transmitter is busy and becomes idle the NOP packet will be sent when the link idles. If the transmitter is continually busy the NOP packets will be spaced by the number of packets set in this register.

If the value is set too low then the NOP packets will frequently interrupt transmission of data, so the link utilization will fall. If the value is set too high then buffer releases will not be reported in a timely manner, so the device the other end of the link may throttle its transmissions unnecessarily.

### Buffer Control: TxBufCountMax and DataBufAlloc

The Transmit Buffer Count Max register sets the maximum number of buffers the transmitter will use in each of the six buffer classes described in the HyperTransport Specification. The classes (Posted Command, Posted Data, NonPosted Command, NonPosted Data, Response and Response Data) match the buffers in the receiver and fields in the NOP flow control packet.

NOP packets received from the other end of the link will increment the number of buffers that the transmitter believes the receiver has available. If the count reaches the limit set in this register the extra credits will be discarded. This allows a general way to throttle traffic in a particular virtual channel on a link.

The **DataBufAlloc** register allows allocation of the HyperTransport receive data buffers between the three virtual channels. The minimum number of buffers that must remain allocated to each channel is specified, along with the number of additional buffers that the dynamic buffer allocator should try to make available for that channel.

## HYPERTRANSPORT RESETS

There are four reset conditions associated with the HyperTransport. This section describes them and their actions on the HyperTransport link and interface. The four resets are:

- 1 System cold reset. A system cold reset is caused by the assertion of COLDRES\_L or the setting of the system configuration register system\_reset bit. This reset causes the subsequent assertion of all other resets detailed here.
- 2 System warm reset. A system warm reset is caused by the assertion of RESET\_L or the setting the system configuration register sb\_softres bit and results in the assertion of link reset.
- 3 HyperTransport link power ok (LDT\_PWROK). This open drain output is driven and received by the interface. The HyperTransport link is inactive if this signal is not asserted.
- 4 HyperTransport link reset (LDT\_RESET\_L). This open drain output is driven and received by the interface. The HyperTransport link is reset when this signal is asserted, a cold link reset will be performed if LDT\_PWROK is deasserted and a warm link reset if LDT\_PWROK is asserted.

Following a system cold reset the interface will drive the HyperTransport link controls to ensure LDT\_PWROK will remain deasserted and LDT\_RESET\_L will remain asserted until the SipReady bit has been set in the HyperTransport SRI Command register. Once the bit is set LDT\_PWROK will be asserted and 1ms later LDT\_RESET\_L will be deasserted to start the cold link initialization. These are inputs as well as open drain outputs, so other devices in the chain can hold off starting the link (for example in a double hosted link the parts at each end of the link may take different lengths of time to complete their initialization and set SipReady). Note that writing the SRI configuration registers and setting of the SipReady bit should only be done after a system cold reset, the SipReady bit must not be touched during link resets.

As required by the HyperTransport specification, a deassertion of LDT\_PWROK is caused by clearing the warm reset bit in the HyperTransport Command Register and setting the SecBusReset bit in the HyperTransport Bridge Control Register. This is effectively a cold reset on the HyperTransport link and also causes the assertion of the link reset.

As required by the HyperTransport specification, an assertion of link reset is caused by setting the warm reset bit in the HyperTransport Command Register and setting the SecBusReset bit in the HyperTransport Bridge Control Register. This is effectively a HyperTransport link warm reset.



The registers that make up the HyperTransport Bridge header are reset by both system resets and are unaffected by link reset with the following exceptions:

- 1 The HyperTransport SRI Command Register is reset on system cold reset and persistent across system warm reset.
- 2 The following bits in the HyperTransport Bridge Secondary Status Register are set to zero on a system cold reset and are persistent through a system warm reset: SigdTgtAbort, RcvdTgtAbort, RcvdMstrAbort, DetSerr.
- 3 The following bits in the HyperTransport Link Control Register are reset by link reset: InitDone, EOC, XmitOff.
- 4 The LinkFail bit and CrcErr bits in the HyperTransport Link Control Register are reset to zero on a system cold reset and are persistent through a system warm reset.
- 5 The HyperTransport Link Frequency Register is cleared by system cold reset and is persistent across system warm reset. Beyond that, it's behavior is determined by the sriLdtPLLCompat bit in the HyperTransport SRI Command Register. If the sriLdtPLLCompat bit is set, the Link Frequency implementation becomes backwards compatible with HyperTransport Specification revision 0.17, which does not allow for dynamic frequency negotiation. In this mode, the HyperTransport link must come up at a static frequency and therefore writes to the LinkFrequency Register take effect immediately. It is strongly suggested that this be done prior to setting the SipReady bit in the HyperTransport SRI Command Register and at no other time. If the sriLdtPLLCompat bit is clear, a write to the Link Frequency Register will take effect only on the next link warm reset.



---

This Page is left blank for notes



This Page is left blank for notes



**Broadcom Corporation**

## Section 9: Ethernet MACs

### INTRODUCTION

The BCM1250 includes three Ethernet MACs. The BCM1125 and BCM1125H include two Ethernet MACs. The MACs support the 10 Mbps, 100 Mbps and 1 Gbps Ethernet standards. They can operate in full or half-duplex mode at all speeds. The MII (10/100 Mbps) or GMII (1 Gbps) standard interfaces are used to connect to external PHY (physical interface) chips. The minimum and maximum size of a packet is programmable, giving support for jumbo packets up to 16K-1 Bytes. For half-duplex operation the MAC supports programmable backoff and retransmission following a collision. A set of RMON counters are updated automatically at the end of each packet transmission or reception, these may be read by the CPU at any time to gather interface statistics.

In addition to the standard Ethernet mode, each interface can be put into a bypass mode where the Ethernet protocol processing is disabled and the interface acts as a Packet FIFO. In this the GMII pins are used to provide an 8-bit data path in each direction that can carry packet or unframed data. Selection of Ethernet or 8-bit Packet FIFO is done by software.

The interfaces are identical and independent. The registers and interrupts associated with each are differentiated by appending the interface number to their names. On the BCM1250 the interfaces are `_0`, `_1` and `_2`. On the BCM1125/H the interfaces are `_0` and `_1`. Accesses made to the address range allocated to a non-existent MAC may cause all MACs to exhibit UNPREDICTABLE behaviour. The following sections describe one of the MACs.

The MAC interfaces can be replaced by Packet FIFO interfaces that are 16-bits wide in each direction. Pins from MAC E0 and E1 are reassigned to Packet FIFO interface F0 on the BCM1250 or F on the BCM1125/H, and on the BCM1250 pins from all three MACs are reassigned to Packet FIFO interface F1. Selection between Ethernet and Packet FIFO operation is done by software.

## INTERFACE OVERVIEW

Figure 51 shows the block diagram of a single Ethernet interface. The interface to the system bus is provided through I/O Bridge 1. This allows the CPU to access the interface control registers, the RMON statistical counters and the DMA control interface. It also provides the connection from the DMA engines into the part.

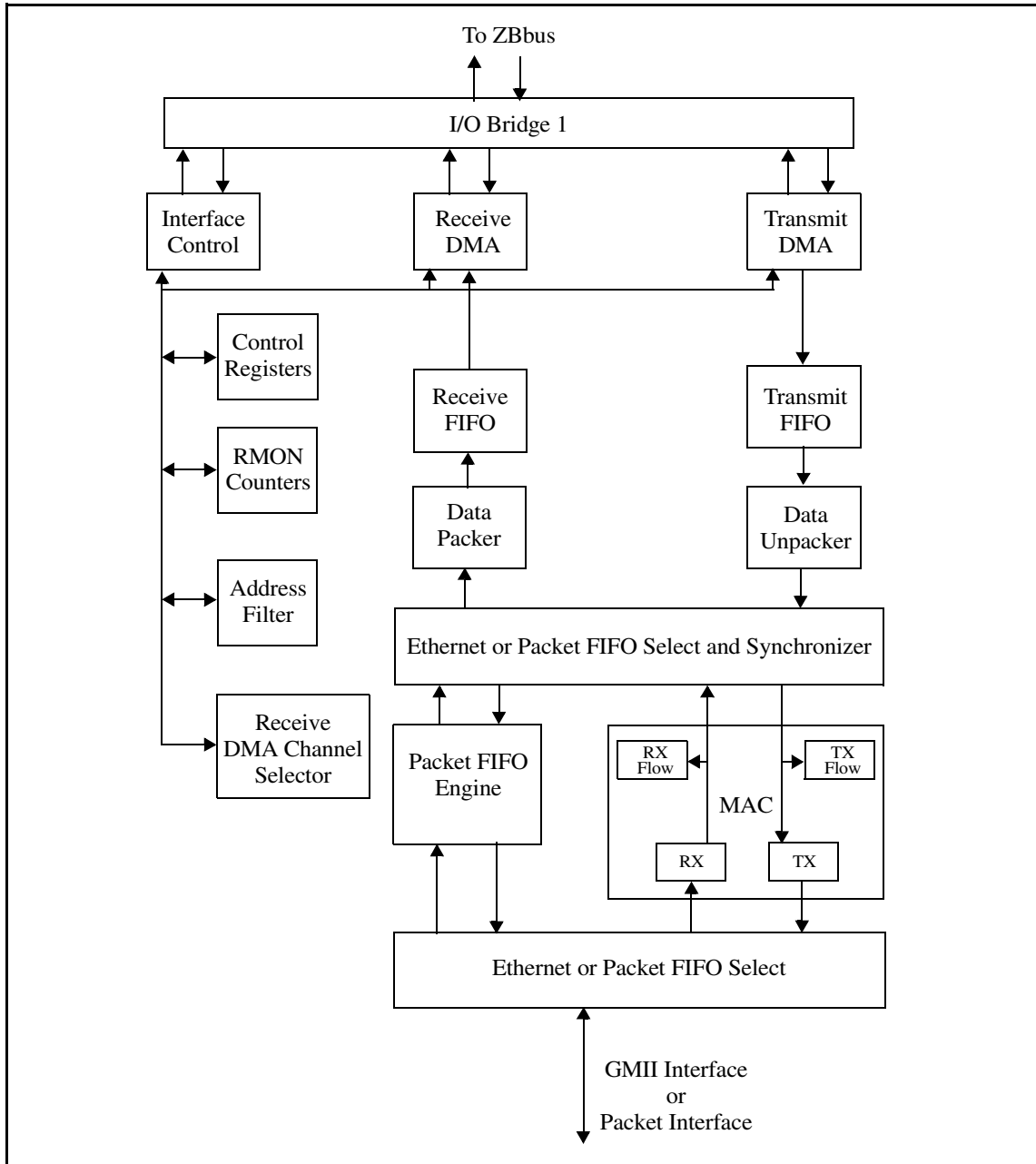


Figure 51: Ethernet Interface Block Diagram

The DMA controllers are described in [Section 7: "DMA" on page 147](#). On the transmit side there are two DMA channels which are serviced using a weighted round robin algorithm. In the receive direction there are also two channels, the packet header is used to select which will be used for delivery.

The transmit packet flow starts with the packet being DMAed from memory into the transmit FIFO. The FIFO is not intended to store the complete packet, it is there to buffer the line rate from the system bus. The DMA controller will fetch 32 byte cache blocks as required to keep the FIFO filled. The controller can be configured to either fetch one or two blocks at a time. The memory performance will be improved by fetching two blocks at a time since the second is likely to hit the memory page opened by the first, however it lowers the granularity of control over the FIFO. The data unpacker extracts 64 bit double-words from the FIFO and forwards them as a byte stream to the protocol engine. Either the Ethernet MAC engine or the Packet FIFO protocol bypass path is selected. For successful transmission one byte must be sent to the protocol engine every cycle, this is enabled by only starting to send a new packet when enough data has been written in to the FIFO to buffer the DMA latency.

The Ethernet MAC protocol engine formats the packet according to the Ethernet specification, executes the MAC protocol to gain access to the transmission medium and sends the packet to the (external) physical layer interface over the GMII pins. It will also respond to flow control requests and block packet transmission as required. The Packet FIFO protocol engine generates simple framing signals for the external interface, but otherwise acts as a bypass path for data to the GMII pins.

The receive packet flow starts with a byte stream being delivered from the physical layer device over the GMII pins. Bytes are sent either to the Ethernet or Packet FIFO protocol blocks. The protocol engine will validate the packet, strip the MAC layer overhead and forward the data to the packing unit. The received bytes are packed into 64 bit double-words and inserted into the receive FIFO. Again, the FIFO just provides a buffering function to cover the DMA latency. The receive DMA engine will extract 32 byte cache blocks from the FIFO and transfer them into memory. The receive protocol engine can also request flow control, it will do this either under software control or if the DMA engine is running out of buffers for incoming packets.

A set of registers in the control section allows the interface to be configured and status to be read. The MAC must be configured before it is used. After the system has been reset the transmit section, receive section and protocol engine are all held in reset until enabled by software.

There is one main interrupt associated with each interface. It combines the interrupts from the transmit and receive sides of each of the two DMA channels and the interrupt from detection of errors. There is a second interrupt that can be enabled to split the DMA channels. When the second interrupt is enabled it signals events from the channel 1 transmit and receive DMA engines, and the main interrupt only signals events from the channel 0 DMA engines and error conditions.

## PROTOCOL ENGINE AND GMII/MII

The Ethernet MAC engine performs all the MAC layer processing needed to comply with the IEEE 802.3 standard at 10 Mbps, 100 Mbps and 1000 Mbps. It interfaces to the receive and transmit FIFOs to move data into and out of the system, and connects to an external physical layer interface (PHY) using the standard MII (media independent interface) connection for 10/100 Mbps operation and the GMII for gigabit operation.

Many of the protocol parameters are programmable. They must be set to the correct values for IEEE 802.3 operation, but may be changed in other applications.

As well as the basic protocol processing during transmission the engine can:

- Automatically retry packets that experience transmission errors during the first few bytes.
- Insert or remove a VLAN tag.
- Append padding to short packets to meet the minimum frame size.
- Overwrite the source address field in the packet with the MAC's own source address.
- Append a CRC to the end of the packet.

During reception the protocol engine can:

- Automatically drop packets that have errors during their first few bytes.
- Filter incoming packets and automatically discard any with a destination address that does not match. Both exact and hash-based matching are available.
- Flag errors and packet type information in the DMA descriptor.

The protocol engine can run the 10 Mbps, 100 Mbps and 1000 Mbps protocol variants and supports full and half-duplex operation at all speeds. Typically the external PHY would support auto-negotiation of the line parameters. During initialization the software will read the negotiated state using the management interface to the PHY and configure the MAC accordingly.

The protocol engine implements flow control. It will back-pressure the link when software explicitly requests or when the number of buffers available to the receive DMA engine falls below a threshold. In half-duplex mode back-pressure is applied by forcing collisions or a fake carrier on the line, in full-duplex mode flow control (pause) frames are sent.

## ETHERNET FRAME FORMAT

The Ethernet frame format is shown in [Figure 52](#).

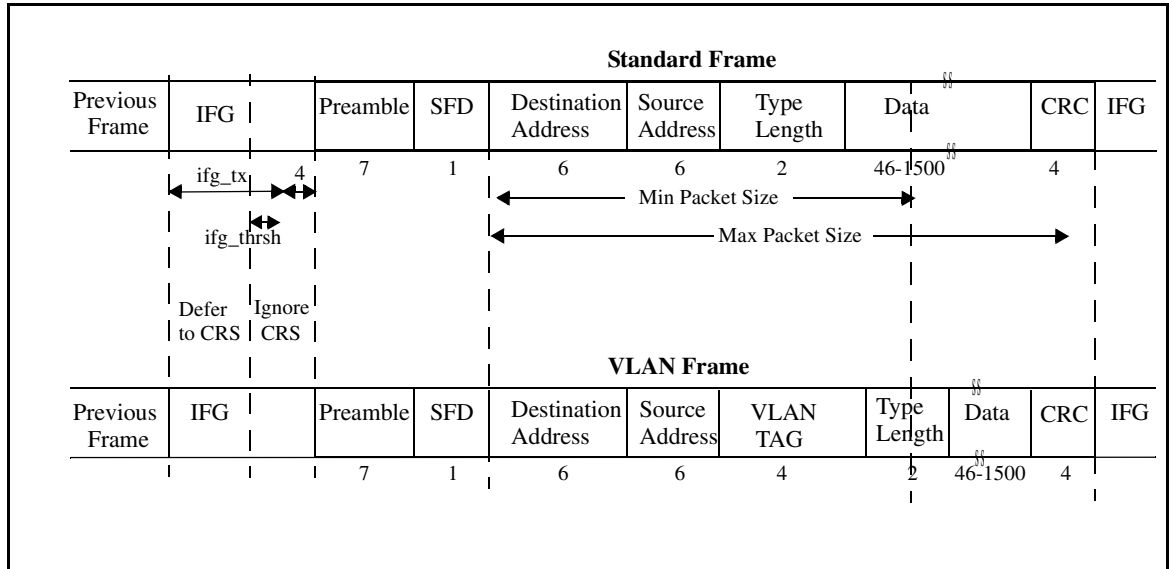


Figure 52: Ethernet Frame Format

The frame is broken up into the following fields:

**Table 160: Ethernet Frame Fields**

Name	Description
IFG	<p>Inter-frame Gap. This is the period that the link is idle between frames.</p> <p>The size of the gap used by the MAC is programmable. The ifg_tx parameters in the <b>mac_frame_cfg</b> register sets the gap that will be used after reception and transmission of a packet. The ifg_thrs threshold sets the size of the end of the gap in which the MAC does not check the carrier before transmitting.</p>
Preamble	<p>The preamble is used to allow other devices to detect the carrier and synchronize with the transmitter. Alternating ones and zeros are sent.</p>
SFD	<p>The Start of Frame Detect byte is used to mark the start of the active frame. It is a single byte with the value 8'hD5.</p>
Destination Address	<p>The destination address is the 6 byte unique Ethernet address that this frame is intended for. Packets with the destination address FF-FF-FF-FF-FF-FF are broadcast packets, and those with the least significant bit of the first byte (i.e. the first bit sent on the wire) set are multicast packets. The interface will filter incoming packets based on this field.</p>
Source Address	<p>The source address is the 6 byte unique Ethernet address of the sender of the frame. During packet transmission the interface can automatically overwrite an old source address with its own.</p>
Type/Length	<p>The meaning of the type/length field depends on whether the packet is sent in 802.3 format or Ethernet format. The field value is a 16 bit number, where the first byte is the most significant 8 bits and the second byte is the least significant 8 bits. If the value of the field is less than or equal to 1500 then this is an 802.3 format frame and this field gives the length of the data field. If the value is greater than 1535 then this is an Ethernet format frame and this field gives the type.</p> <p>The receive interface can be set to check this length against the actual packet length, and report an error if they differ.</p>
Data	<p>The data portion of the packet is the (layer 3) data that is being transported from the source to the destination.</p> <p>The standard gives the minimum and maximum sizes of the data as 46 and 1500 bytes (giving minimum and maximum packet sizes of 64 and 1518 bytes). Both can be changed in the <b>mac_frame_cfg</b> register to support jumbo packets.</p>
CRC	<p>The CRC (also known as Frame Check Sequence or FCS) is a 32 bit cyclic redundancy check calculated over all the bytes in the packet from the first destination byte to the last data byte. It is normally appended to the frame during transmission and checked during reception.</p> <p>The CRC is formed using the CRC-32 polynomial:</p> $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$ <p>The partial CRC accumulator is initialized to 32'hFFFFFFFF at the start of the packet. The 32 bit CRC is transmitted most significant byte first.</p> <p>The interface can automatically insert the CRC during transmission, this is selected on a per-packet basis in the DMA descriptor. If this is disabled the CRC must be provided as part of the data from the transmit FIFO. The MAC will always check the transmitted CRC and log an error if a packet has been sent with a bad CRC.</p>
VLAN TAG	<p>The VLAN Tag is not part of the Ethernet standard. It is an extra four byte header that is used when encapsulating other packets into a "virtual LAN" link.</p> <p>The interface can insert, replace or remove VLAN tags from packets during transmission. This is selected on a per-packet basis in the DMA descriptor, if the VLAN tag is altered then automatic CRC generation is also enabled.</p>





## PREPENDED HEADER FRAME FORMAT

In devices with system revision indicating PERIPH\_REV3 or greater the MAC interface supports a modified frame format that has an additional header prepended before the Ethernet frame. The CRC can be set to cover the prepended header and Ethernet frame or just the Ethernet frame (or a combination). Ignoring the framing (IFG/Preamble/SFD for Ethernet, SOP/EOP marks on first and last bytes for Packet Fifo modes) the prepended header format is shown in Figure 53. This figure shows the CRC offset pointing to part way through the prepended header which is unusual, the standard cases are when the `crc_offset` is zero (i.e. the CRC covers everything) or the `crc_offset` is equal to the `pkt_offset` (i.e. the CRC covers the Ethernet frame and not the prepended header). The `crc_offset` must be less than or equal to the `pkt_offset`, if it is greater the behavior of the interface is UNPREDICTABLE.

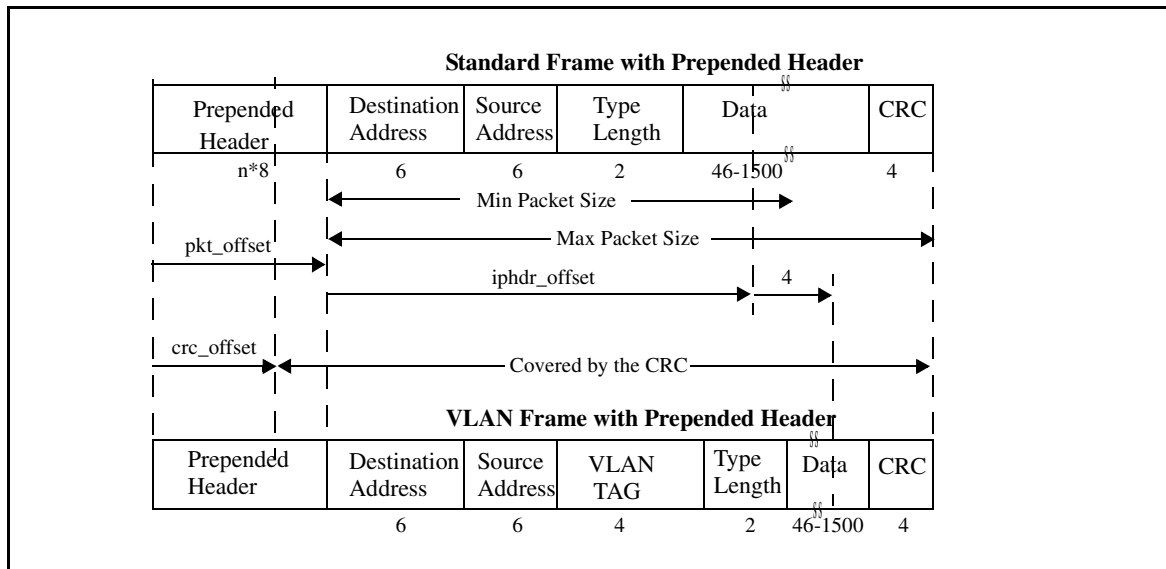


Figure 53: Prepended Header Format

The `pkt_offset` and `crc_offset` must be a multiple of 8 bytes. An offset of zero indicates there is no offset and the start of the packet is used. The maximum offset is 248 (31\*8) bytes. There are separate registers to set the offsets on the transmit and receive sides (and their values may be different). In both cases the CRC offset specifies the first byte of the packet that will be included in the CRC calculation. For the transmit side the `tx_pkt_offset` is used to locate the Ethernet header for replacement of the source address, and the VLAN tag position for automatic insertion, removal or replacement of the VLAN tag. On reception the `rx_pkt_offset` is used to specify the start of the Ethernet header for the address filtering and for extraction of the packet type.

The figure also shows the `iphdr_offset`. This identifies the start of the IP header in the packet for checksum checking (both IPv4 header checksum and TCP checksums are checked) and is discussed in later sections. To allow for networks that have a mix of VLAN and non-VLAN packets the interface can be configured to automatically add 4 bytes to the `iphdr_offset` when a packet with a VLAN tag is received.

## PROTOCOL ENGINE CONFIGURATION

The basic encapsulation parameters are set in the **mac\_frame\_cfg** register. This allows configuring of the inter-frame gap, maximum backoff time, slot size, minimum frame size and maximum frame size. This must be programmed to the correct values for IEEE 802.3 operation at 10 Mbp/s and 100 Mbp/s. For gigabit operation the slot\_size is automatically increased to 512 byte times (rather than 512 bit times) but software must half the IFG values; none of the other parameters should be altered. In networks configured to use jumbo packets the maximum frame size should be increased to match the rest of the network. Parameters should only be altered while the protocol engine is held in reset. Care must be used when altering the values from their standard ones, typically all endpoints on the network must be configured similarly. A detailed description of each of the parameters is given in the register description in [Table 180 on page 307](#).

The effective slot size is adjusted by adding a 10 bit signed offset in bit times to the 512 bit times or 512 byte times selected by the data rate. The offset adjusts the MAC timing to take account of the number of cycles of latency through the PHY and back. Typically the field will be a small positive number.

The lfsr\_seed is the only parameter in the **mac\_frame\_cfg** register that can be altered at any time and that can be altered without violating the 802.3 standard. Whenever it is written the value written sets the lowest eight bits of the linear feedback shift register that generates the pseudo-random backoff following a collision.

The rest of the protocol engine configuration is done in the **mac\_cfg** register. For Ethernet operation the bypass\_sel bit must be clear. The ss\_en bit must always be set unless the part is being run in a test mode.

Ethernet packets are recovered into a byte stream before being put into the 64 bit wide receive FIFO, and data is read in 64 bit chunks from the transmit FIFO and converted to a byte stream before it is sent. The system endian configuration is used to correctly order the bytes so that the first byte on the wire is at the lowest memory location.

The flow control portion of the protocol engine must also be programmed with the flow control mechanism and for full-duplex links the number of slot times to request the peer to pause transmission. These are set in the **mac\_cfg** register in the fc\_cmd and tx\_pause\_cnt fields.

Control of the movement of data between the transmit and receive FIFOs and the protocol engine is also done in the **mac\_cfg** register. These settings are discussed in the transmit and receive sections below.

Errors generated in the interface are signalled in the **mac\_status** register. Bits in this register are set when an error is detected and also reflect the interrupt state from the DMA engines. Whenever the register is read the error bits are cleared. There is a mask register **mac\_int\_mask** associated with the status register. If both the mask and status bits are set for any of the error conditions or DMA channels then the MAC interrupt is raised. Reading the **mac\_status** register will clear the error bits, but the DMA channels must also be serviced before the interrupt is removed.

## INTERFACE TO PHY

The interface to the PHY is done through the standard GMII/MII interface. This provides an 8 bit path each direction for gigabit operation and a 4 bit path for 100 Mbp/s or 10 Mbp/s operation. The PHY does the appropriate encoding for the link for transmission, and will deserialize the incoming data. The PHY reports carrier sense, collision and code violation to the protocol engine.

In most cases software will interrogate the PHY to discover the link speed and duplex settings. These are obtained either from auto-negotiation or based on the PHY capabilities. The settings must be used to configure the **mac\_cfg** register.



## TRANSMITTER OPERATION

### TRANSMITTER CONFIGURATION

The transmitter clock is accepted as an input from the PHY device for 10/100 Mbit/s operation. However, for 1 Gbit/s operation the data lines are clocked at 125 MHz so the clock is forwarded with the data. In this case the reference clock input is used (on the BCM1250, the REFCK01 pin is for interfaces E0 and E1, and the REFCK2 pin for interface E2; on the BCM1125/H, REFCK0 is for E0 and REFCK1 for E1). The clock source is configured by the MAC speed selection in the **mac\_cfg** register.

The transmit FIFO is a 64 bit wide fifo with 128 entries. Figure 54 shows the thresholds that are used to configure it. When the FIFO has space for data it will signal the DMA engine to request data. The **tx\_wr\_thrsh** field in the **mac\_thrsh\_cfg** sets the number of empty entries there must be in the FIFO before it will request data. The DMA engine fetches either 32 bytes or 64 bytes at a time, so this value should be set to 4 or 8.

The data from the transmit FIFO is read out by the protocol engine for transmission. Once transmission of a packet has started the DMA engine must ensure that there is always data in the FIFO when the protocol engine needs it. If the FIFO is empty at any time before the end of the packet then an underflow error is reported and transmission fails (in encoded Packet FIFO mode the transmission will not fail, instead cycles with no valid data will be inserted in the link protocol). To reduce the likelihood of the FIFO becoming empty the **tx\_rd\_thrsh** threshold can be set to ensure a certain number of entries have been written to the FIFO before transmission starts.

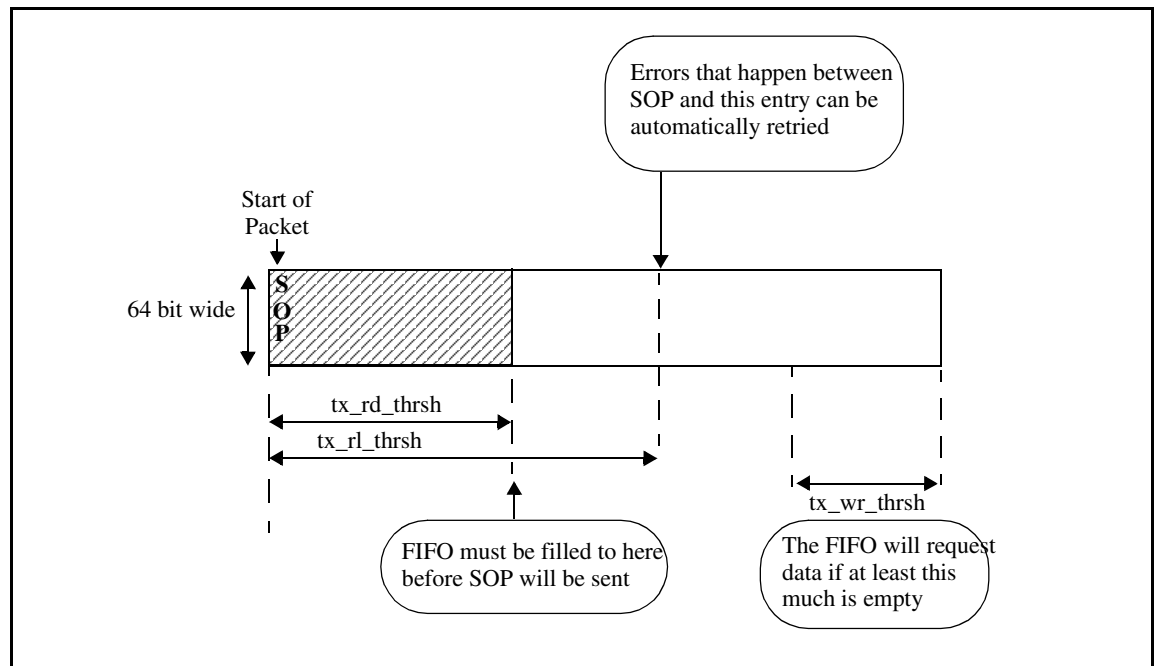


Figure 54: Transmit FIFO Thresholds

In half-duplex operation the Ethernet protocol expects collisions to occur, and relies on them to share access to the transmission medium. These collisions will only occur at the start of packet transmission, and the interface will backoff for a random period before retrying the transmission. The maximum backoff time is exponentially increased (up to a maximum 1024 slot times) each time a collision is encountered. If the collision is detected early in the packet the interface will automatically backoff and retry. If a collision happens late in the transmission then the packet is dropped.

Once transmission has begun entries are retained in the transmit FIFO until tx\_rl\_thrsh entries have been transmitted, if an error is detected during this time the packet can be automatically retried. Retaining the start of the packet in this way is enabled by setting the tx\_hold\_sop\_en bit in the **mac\_cfg** register. The errors that will cause automatic retry are also selected in this register.

Table 161 describes all the transmission error conditions, and lists the bit that must be set to enable automatic retry for them.

**Table 161: Transmission Error Conditions**

Error	Bit to Set for Automatic Retry	Description
Collision	retry_en	Collisions near the start of the frame are an expected part of the Ethernet protocol, and should always be retried.
Excessive Collision	ret_drpreq_en	After 16 attempts to transmit a frame have resulted in a collision the interface will report that excessive collisions have happened. This condition normally indicates a serious problem with the link (although it will occasionally be encountered on a large very busy Ethernet), in most cases the packet should not be automatically retried. Note that this error and late collisions use the same bit to enable automatic retry. Excessive collisions will set the excol_err bit in the <b>mac_status</b> register.
Late Collision	ret_drpreq_en	The Ethernet protocol sets the minimum packet size so that all collisions will be seen during the transmission of a minimum length packet. Any collision after the minimum length has been sent is a late collision and indicates a serious error. Late collisions can be caused by the Ethernet segment being longer than allowed by the specification, or by some other station on the segment turning on or off or violating the standard. In most cases the packet should not be automatically retried. Note that this error and excessive collisions use the same bit to enable automatic retry. Late collisions will set the ltcol_err bit in the <b>mac_status</b> register.
Underflow	ret_ufl_en	An underflow error is reported when the transmit protocol engine finds the transmit FIFO empty during transmission of a frame. This will happen if the DMA (or external agent in direct mode) has been unable to write data to the FIFO at the rate of the transmission. The tx_rd_thrsh threshold can be adjusted to avoid underflows by providing a data buffer in the FIFO to cover the request latency. In normal operation the underflow error may occasionally occur if the DMA engine is locked out from the bus or memory, and this could be automatically retried. If automatic retry is enabled care must be taken to ensure it is not being applied to every packet. If the read threshold is set too low, then all packets will get an underflow shortly after transmission starts (leading to a runt packet being sent). The time they take to be retried could cover the extra time needed to fetch data into the FIFO so they succeed on a second attempt. This situation should be fixed by increasing the read threshold. A FIFO underflow will set the tx_undrfl bit in the <b>mac_status</b> register.
Overflow	None Cannot be retried	An overflow error is reported when the DMA engine (or external agent in direct access mode) attempts to write to the transmit FIFO when the FIFO is full. The data written is lost. During DMA this error will only be seen if the tx_wr_thrsh parameter is incorrectly set. A FIFO overflow will set the tx_ovrfl bit in the <b>mac_status</b> register.



**Table 161: Transmission Error Conditions (Cont.)**

Error	Bit to Set for Automatic Retry	Description
CRC error	None Cannot be retried	The transmitted CRC is compared to the CRC that was computed over the frame, and this error is raised if they differ. If the interface is automatically appending the CRC, then this error will never happen. If the CRC is being supplied with the data then this error indicates that the supplied CRC was bad, and that the packet will be rejected by the recipient.

Automatic retry would normally only be enabled for collisions since the other conditions could indicate a more serious problem (once the system is known to work automatic retry could be added for occasional underflows).

The Gigabit Ethernet standard allows bursts of packets to be transmitted without requiring the link be released and reacquired. This is enabled by setting the `burst_en` configuration bit, which should only be set when the MAC is configured for half-duplex 1000 Mbps operation.

## TRANSMIT PATH

Each MAC has two transmit DMA channels associated with it. If only one is enabled or only one has descriptors available then it is selected for transmission. If both channels are enabled and have packets available then the weights set in the `mac_txd_ctl` register are used to select which is used. The number of packets selected by the weight are sent from one DMA channel then the other is serviced for the number of packets set in its weight; if a channel becomes empty then the other channel immediately starts. (Note that if both channels are enabled and given weights of zero then no packets will be transmitted.) If the packets in each DMA channel are the same length (or on average the same length) then this will share the transmission bandwidth according to the weights. If the packets are on average of very different lengths the weights can be skewed accordingly. The selected DMA channel will write a packet into the transmit fifo, with the start and end specially marked for the protocol engine.

The packet to be transmitted is encapsulated in the Ethernet frame described above. The inter-frame gap, preamble and SFD are automatically prepended before data read from the FIFO. The data is then copied from the FIFO to form the body of the packet. There are six manipulations that the protocol engine can perform, configured per packet by options in the DMA descriptor:

- Replacement of source address. The source address that is in the packet supplied by the DMA engine can be overwritten with the address of this interface that has been set in the `mac_ethernet_addr` register.
- Removal of VLAN tag. The four bytes of VLAN tag can be dropped from the packet, converting a VLAN packet into a regular Ethernet packet.
- Insertion of VLAN tag. The four bytes in the `mac_vlan_tag` register can be inserted into the packet just after the source address, converting a standard Ethernet packet into a VLAN packet.
- Replacement of VLAN tag. The four bytes in the `mac_vlan_tag` register overwrite the four bytes in the VLAN tag position in the packet.
- Padding short packets to the minimum frame size.
- Insertion of CRC. The four byte CRC can be automatically replaced in the packet or appended to it. If any of the other modifications are done the CRC will be replaced to ensure the interface uses the correct CRC for the data sent.

During packet transmission the protocol engine always calculates the CRC of the frame. This value will be inserted if the interface is configured to automatically append the CRC. The CRC that is transmitted will always be compared to the CRC that was calculated. If the CRC was automatically generated they will always match. If the CRC was supplied through the transmit FIFO and it does not match the calculated one then a transmit CRC error is reported.

In Packet FIFO mode the only packet modification that can be done by the transmitter is to append the CRC.

# RECEIVER OPERATION

## RECEIVER CONFIGURATION

The receive fifo is a 64 bit wide fifo with 64 entries. When the FIFO contains data it will signal the DMA engine to request emptying. The rx\_rd\_thrsh field in the **mac\_thrsh\_cfg** register sets the number of valid entries that must be in the FIFO to request emptying. The DMA engine transfers in blocks of 32 bytes so this field should normally be set to four entries. The threshold is overridden when the end of a packet is in the FIFO, since these are unlikely to be correctly aligned.

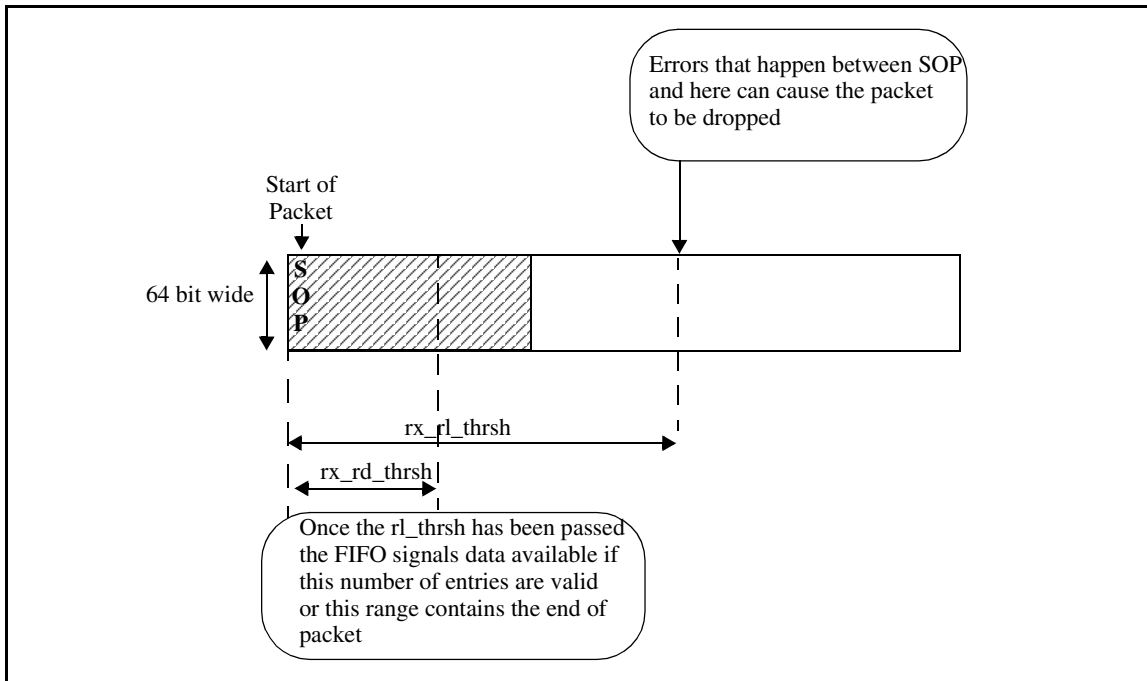


Figure 55: Receive FIFO Thresholds

The receiver will hold off informing the DMA engine of the arrival of a new packet until successful reception of the first few bytes. The rx\_rl\_thrsh field in the **mac\_thrsh\_cfg** register sets the number of entries that must be written to the FIFO before it signals that data is available. If the end of the packet is received the data is always reported to the DMA engine. If there is any reception error before this point then the packet can be automatically dropped by flushing the data in the FIFO. The error will still be counted in the error statistics. If the drp\_errpkt\_en bit in the **mac\_cfg** register is set then all packets with errors will be thrown on the ground in this way. If this bit is clear then more selective bits in the **mac\_cfg** determine whether packets with errors are dropped or delivered with an error status. Regardless of any of these settings, errors that are detected after the rx\_rl\_thrsh threshold has passed are always delivered with an error status.

For example, if the rx\_rl\_thrsh is set to 8 and the MAC detects a runt packet (shorter than the 64 byte minimum size Ethernet packet) the DMA engine will never be told about the packet and it will be automatically discarded.



Automatic discard can be triggered by any error that happens (FCS error, code error, dribble error, runt error, oversize error or length error) while the FIFO is locked (i.e. less than the number of entries set by `rx_rl_thrsh` have been written into the receive FIFO after the start of a given packet). [Table 162](#) lists the bits in the `mac_cfg` register that should be set to enable automatic dropping selectively for each of the error types.

**Table 162: Receiver Error Conditions**

Error	Bit to set for automatic dropping	Description
CRC Error	<code>drp_fcserpkt_en</code>	<p>A Frame Check Sequence or CRC error happens when the CRC field in the final four bytes of the received packet does not match the CRC that has been calculated for the data that was received. This error indicates that the data in the packet has been corrupted during transmission.</p> <p>Note that because the automatic packet dropping can only be done early in the packet (set by <code>rx_rl_thrsh</code>) longer packets with a CRC error cannot be dropped and will be received marked with an error.</p>
Code Error	<code>drp_codeerrpkt_en</code>	<p>A code error is reported for any error signalled by the PHY device. The details depend on the encoding used and the particular transmission medium. This error will usually be signalled because received data did not match a valid encoding, but it could also be signalled by loss of light in an optical medium, or the clock recovery block losing lock. A single code error can be ignored, but when multiple code errors are seen the PHY status should be checked using the management interface.</p>
Dribble Error	<code>drp_dbrlerrpkt_en</code>	<p>In 10 Mbp/s and 100 Mbp/s operation the data received from the PHY over the GMII interface is smaller than a byte (1 bit per clock for 10 Mbp/s and 4 bits per clock for 100 Mbp/s). However, the length of a valid Ethernet packet is always an integral number of bytes. If a packet is received where this is not the case a dribble error is reported (this error is sometimes called an alignment error). This error can never happen during gigabit operation because the interface from the PHY is 8 bits wide (any similar error would be reported by the PHY as a code error).</p>
Runt Packet	<code>drp_rntpkt_en</code>	<p>A runt packet is any packet that is shorter than the minimum packet size. The minimum packet size that is used by the MAC is configurable in the <code>mac_frame_cfg</code> register, for compliance with IEEE 802.3 this size will be set at 64 bytes.</p>
Oversize Packet	<code>drp_oszpkt_en</code>	<p>An oversize packet is any packet that is larger than the maximum packet size. The maximum packet size that is used by the MAC is configurable in the <code>mac_frame_cfg</code> register, for compliance with IEEE 802.3 this size will be set at 1518 bytes. If a packet is received that is longer than this size it will be truncated at one byte longer than the maximum frame size and marked with an error. i.e. If the max frame size is set to 1518 bytes and a packet of length 4000 bytes is received from the PHY, the DMA engine will transfer 1519 bytes to memory and in the start of packet DMA descriptor status information software will see the bad packet flag set and a length of 1519 bytes.</p> <p>Note that because the automatic packet dropping can only be done early in the packet (set by <code>rx_rl_thrsh</code>) in most cases oversize packets cannot be dropped and will be received marked with an error.</p>
Length Error	<code>drp_lenerrpkt_en</code>	<p>If the packet type/length field is less than 1500 then it holds the length of the data portion of the packet. A length error is signalled if this does not match the number of bytes that were actually received for the data section.</p> <p>Note that because packets may be padded to the minimum frame size a length error is not reported if the packet that is received is longer than the size given in the header.</p>

**Table 162: Receiver Error Conditions (Cont.)**

Error	Bit to set for automatic dropping	Description
Underflow	None	<p>An underflow error is reported when the receive DMA engine (or external agent in direct mode) reads the receive FIFO when it is empty. The data returned is UNPREDICTABLE and the FIFO pointers will not change. During DMA this error will only be seen if the rx_rd_thrsh parameter is incorrectly set.</p> <p>An underflow will set the rx_undrfl bit in the <b>mac_status</b> register.</p>
Overflow	None	<p>An overflow error is reported when the protocol engine attempts to write to the receive FIFO and it is full. This indicates that the DMA engine (or external agent in direct mode) is not emptying the FIFO fast enough. The data written is lost.</p> <p>A FIFO overflow will set the rx_ovrfl bit in the <b>mac_status</b> register.</p>

At the end of the packet the protocol engine can write one additional entry into the FIFO containing status for the packet. This is used by the DMA controller to update the length and flags field in the first DMA descriptor of the packet. Appending of this status word is enabled by setting the ap\_stat\_en bit in the **mac\_cfg** register, this bit must be set when the interface is configured for Ethernet operation and must be clear in Packet FIFO mode.

## RECEIVE PATH

The protocol engine processes all packets arriving from the PHY, removes the physical layer encapsulation and passes them into the receive FIFO. The receive DMA engine will remove data from the FIFO and transfer it into packet buffers in memory. The receiver is always active, regardless of the full- or half-duplex selection.

The packet is checked for errors during reception, and the CRC is computed and compared with the value in the packet. If an error is detected early in packet reception it will not have started to be DMA'ed and the packet can be dropped. If an error is detected after DMA has started, the DMA will be completed and the error flagged in the status bits written back to the descriptor. The dropping of packets is enabled by holding the first few entries in the receive FIFO and not informing the DMA logic that there is data to extract until a threshold has been reached. If the error happens before the threshold then the FIFO pointer can be restored to the entry that holds the start of the packet, and the packet is dropped.





## DESTINATION ADDRESS FILTERING

The MAC will filter received packets based on their Ethernet destination address. Packets that pass the filter are received, those that do not are never delivered to the DMA engine.

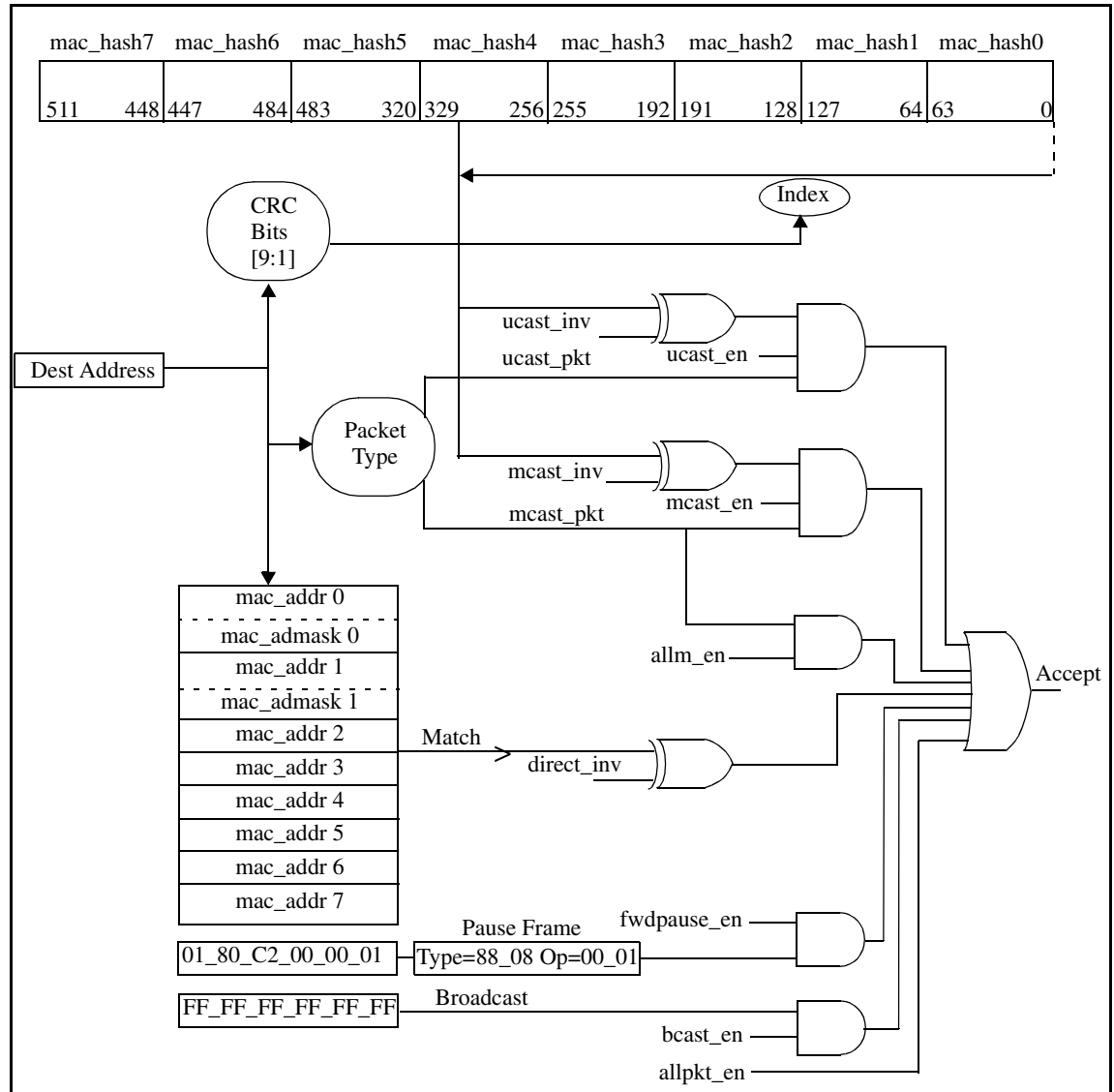


Figure 56: Receive Address Filter

There are five components to the filter.

- 1 Broadcast packet detection. Broadcast packets are either all accepted or all rejected.
- 2 Exact match. The incoming packet address is compared to eight addresses (which may be unicast or multicast addresses), if it matches any of them then the packet is accepted, otherwise it is rejected. In parts with system revision PERIPH\_REV3 or greater two of the addresses have masks associated with them, if a bit is clear in the mask it is excluded from the comparison. The sense of this section of the filter can be reversed, so that packets will only be accepted if it does not match any of the addresses. All exact match addresses are always enabled, if less than 8 entries are needed the extra ones can be written with copies of one of the accepted addresses, or with 00-00-00-00-00-00 which is not a valid address.
- 3 Hash match. Nine bits are extracted from the partial CRC of the packet for the six destination address bytes, this provides a hash value. This value is extracted from the standard CRC-32 described in [Table 160 on page 268](#). There are eight 64-bit hash table registers, these are concatenated to form a 512 bit bitmap (map bit 511 is hash\_table7 bit 63... map bit 0 is hash\_table0 bit 0). The hash value is used as an index into this bitmap, the packet is accepted if the bit is set and rejected if the bit is clear. The sense of the match can be inverted so the packet will only be accepted if the bit is clear.
- 4 Multicast Match. The filter can be configured to accept all multicast packets. If this is enabled then the only reason to put multicast addresses in the Exact Match filter or enable multicast hash matches is to make use of the match\_exact and match\_hash flags in the DMA status information.
- 5 Pause Frame. If the header of the frame indicates that it is a Pause Frame (flow control frame) then it will normally be consumed by the interface and the transmitter flow controlled as requested. In parts where the system revision indicates PERIPH\_REV3 or greater if the fwdpause\_en control is enabled then the hardware will not act on Pause Frames and the address filter will accept them for reception.

If the destination address of the packet is accepted by any of the three filters or the promiscuous mode bit is set then the packet will be received. If there is no match then the packet is dropped. (Note that the filter will not operate properly if the fifo release threshold rx\_rl\_thrsh is set smaller than the position required for the packet header, since the threshold has a minimum of two this is only a concern when there is a prepended header.)

The receive address filter is controlled by the **mac\_adfilter\_cfg** register. There are a set of enable bits and invert bits to control the filter, as shown in [Figure 56 on page 277](#). The address registers and hash map must also be written.

A packet will be accepted if any of the filters indicate it should be accepted. If the broadcast address FF-FF-FF-FF-FF-FF is put in an exact match address register then broadcast packets will be accepted regardless of the state of the broadcast enable bit. However, setting the hash entry corresponding to the broadcast address will not cause acceptance because a hash match is qualified by the packet being unicast or multicast.

The receive address filter is still applied in Packet FIFO mode. Since the data is not expected to be formatted as an Ethernet frame most applications will need to set the allpkt\_en bit to accept all packets.

The value for the hash filter for a particular Ethernet address may be computed using the mchash macro and standard CRC generator given in the following code:

```

/*****
* eth_hwrcrc32(buf,len)
*
* Calculate a CRC-32 of the specified bytes. This is used to
* generate the masks for the hash filter on the receive side.
* For the hash filter the bits must be swapped to match the hardware
*
* Input parameters:
*   buf - buffer
*   len - length of buffer
* Return value:
*   CRC as held in MAC hardware
*****/

#define mchash(mca) ((eth_hwrcrc32(mca, ENET_ADDR_LEN) >> 1) & 0x1FF)

static unsigned eth_hwrcrc32(unsigned char *databuf,int datalen)
{
    unsigned int idx, crc = 0xFFFFFFFFUL, tmp;

    static unsigned int crctab[] = {
        0x00000000, 0x1db71064, 0x3b6e20c8, 0x26d930ac,
        0x76dc4190, 0x6b6b51f4, 0x4db26158, 0x5005713c,
        0xedb88320, 0xf00f9344, 0xd6d6a3e8, 0xcb61b38c,
        0x9b64c2b0, 0x86d3d2d4, 0xa00ae278, 0xbdbdf21c
    };

    for (idx = 0; idx < datalen; idx++) {
        crc ^= *databuf++;
        crc = (crc >> 4) ^ crctab[crc & 0xf];
        crc = (crc >> 4) ^ crctab[crc & 0xf];
    }
    /*****
    ** Flip [31:0] to [0:31] to match hardware **
    *****/
    tmp = 0;
    for (idx=0; idx<32;idx++)
        tmp |= ((crc<<idx)>>31)<<idx;

    return tmp;
}

```



### RECEIVE DMA CHANNEL SELECTION

If a packet is accepted by the address filter the received DMA channel is selected. This is done by extracting eight bits out of the first 128 bytes received (if the rx\_rl\_thrsh threshold is set smaller than 128 then the eight bits must be in the rx\_rl\_thrsh bytes), and using them to lookup a two bit number in a small table formed using the **mac\_chup** and **mac\_chlo** registers. The offset into the packet is specified in nibbles (half bytes) in the **mac\_cfg** register, enabling extraction of the DiffServ code point from the IP v6 traffic class field as well as the (byte aligned) IP v4 TOS field (see Figure 57). If the two bit number is zero the packet will be received on DMA channel zero, otherwise DMA channel 1 is used. Both bits get written in to the packet receive status information.

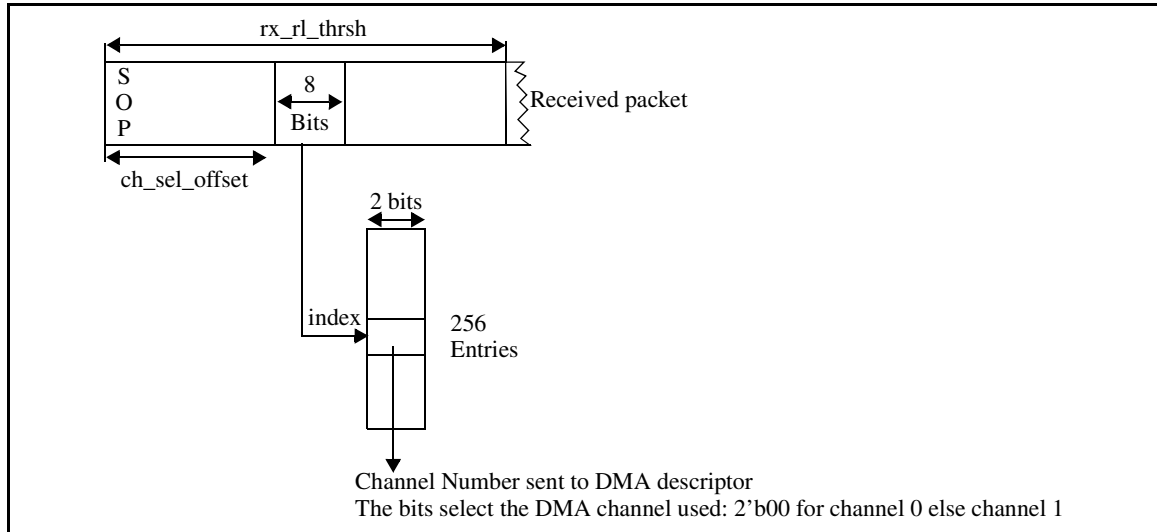


Figure 57: Receive Channel Selection

The channel select offset can be used to select any nibble in the first 128 bytes of the packet. For example if selection is to be made based on the low byte of the Ethernet packet type (which is the second byte of the field sent on the wire) the offset should be (decimal) 26.

Figure 58 shows how this offset is obtained. Even offsets refer to bytes in the received packets, odd offsets straddle bytes: the high four bits of the index byte come from the low four bits of the later byte in the frame and the low four bits of the index byte are the high four bits of the earlier byte in the frame. In the example a complete byte from the frame is used so the bottom bit of the offset will be zero. The upper bits are simply the byte offset (starting from 0) of the desired byte which is 13 for the second byte of the length/type field.

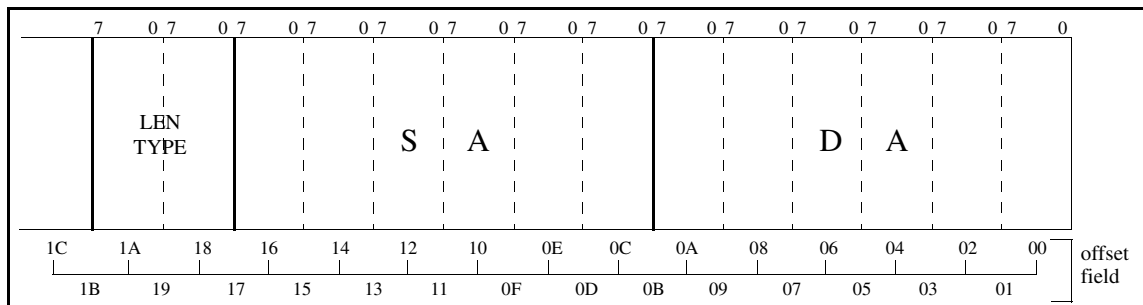


Figure 58: Selecting the Channel Offset



The receive channel selection is used in the same way in Packet FIFO mode.

In devices where the system revision indicates PERIPH\_REV3 or greater the 8 bit index into the channel number table can be formed from two four bit fields extracted from the packet. The `split_ch_en` bit in the `mac_cfg` register enables this feature. The lower 4 bits of the index are extracted from the packet at the offset specified by the `{rx_ch_sel_msb,rx_ch_sel}` value and the upper 4 bits of the index are extracted from the offset specified by the `rx_ch_msn_sel` field in the `mac_adfilter_cfg` register. Both these are nibble offsets as described above. The `rx_ch_msn_sel` must point later in the packet, if `rx_ch_msn_sel <= {rx_ch_sel_msb,rx_ch_sel}` the channel selected is UNPREDICTABLE. When `split_ch_en` is zero the `rx_ch_msn_sel` value is ignored and the upper 4 bits of the index are the nibble after the `{rx_ch_sel_msb,rx_ch_sel}` offset as described above.

## PACKET TYPE IDENTIFICATION

The receiver reads the Ethernet packet type field in the inbound packet. Some common packet types are detected and encoded in the status that is written to the descriptor. Four of the types are fixed, the other four are programmable in the `mac_type_cfg` register. The packet type encodings are shown in [Table 163](#).

**Table 163: Ethernet Type Mappings**

Ethernet Type Field	Status[57:55] Encoding	Packet Type
16'h0800	3'b000	IP v4
16'h0806	3'b001	ARP
<=1500	3'b010	802.3 length
Other	3'b011	No match
<code>mac_type_cfg[15:0]</code>	3'b100	Configurable types.
<code>mac_type_cfg[31:16]</code>	3'b101	
<code>mac_type_cfg[47:32]</code>	3'b110	
<code>mac_type_cfg[63:48]</code>	3'b111	

In Packet FIFO mode the 13th and 14th bytes of the received packet will be checked against the same values and the three bit encoding will be placed in the descriptor status word. In most cases this value should be ignored, but it is possible that the four configurable types could be used to identify special packets.

## IPv4 Header Checksum

The receiver will check packets for an IPv4 header with no options and a correct header checksum. The `iphdr_offset` field in the `mac_adfilter_cfg` register sets the offset into the received packet that the header starts (this will normally be 15, since the IP header will start after the Ethernet header, but it can be adjusted to allow encapsulation headers to be skipped). Note that this offset is specified with the first byte in the destination address as 1. The first byte of the header must be 8'h45 indicating the IP version number is 4 and the header is the standard 5 (32 bit) words with no options. If this byte matches and the header checksum of the 5 words is correct (per RFC791) then the `bad_ip4csbit` will be clear in the status written back to the packet DMA descriptor.

If the `bad_ip4cs` bit is set, the packet may not be an IPv4 packet, it may have header options, or it may have an incorrect checksum. Software must determine which of these is the case. Note that the fast path of good packet, Ethernet type IPv4, IPv4 header with no options and good checksum can be detected with an AND mask and BEQ since all the status bits will be low in this case.

The IPv4 header checksum is only computed in Ethernet mode. In Packet FIFO mode the `bad_ip4cs` bit will always be set.

## TCP Checksum Checking

The receiver will check packets for a valid TCP checksum. The result of this check is only valid if the IPv4 Header Checksum is correct. The `iphdr_offset` (described in the previous section) is used to locate the start of the IP header. The pseudo-header containing the source and destination IP addresses, IP protocol type and TCP length (IP length-20) is summed then the checksum continues for the TCP length of the packet (padded with a byte of 0 if required). The `bad_tcpcs` flag (written back to the b portion of the descriptor) will be set if the checksum is not 16'hFFFF. The same pseudo-header and checksumming algorithm is used for UDP as TCP (the UDP header itself is slightly different but since the header is included in the checksum this does not matter). The checksum is optional in UDP so if the `bad_tcpcs` flag is set then software must check for a packet without checksum (the field is zero) before rejecting the packet.

The TCP checksum is only computed in Ethernet mode. In Packet Fifo mode the `bad_tcpcs` flag will be UNPREDICTABLE.

## Packets Dropped by the DMA Channel

A packet will be dropped by the DMA channel for one of three reasons:

- 1 The channel is disabled.
- 2 The channel is enabled but has no descriptors to receive the packet into.
- 3 The Drop bit is set for the channel (for example if one of the DMA channels is used for best effort traffic, that channel may have its drop bit set, so during overload no best effort packets are delivered to the system).

In devices with system revision indicating `PERIPH_REV3` or greater the receive channel provides a counter in the `dma_oodpktlost` register that records the number of times a packet was lost and the channel was out of descriptors. The count is incremented every time a packet is received and the channel is out of descriptors, the other state of the channel is not factored in. Thus the count will still be incremented if the channel is disabled and out of descriptors when a packet is received. And it will be incremented if the drop bit is set and the channel is out of descriptors when a packet is received. But the count will not be incremented if there are descriptors available and the packet is dropped because the channel is disabled or the drop bit set.

## FLOW CONTROL

The MAC supports flow control in both full-duplex and half-duplex modes. In full-duplex mode pause frames are used to request the peer to stop sending for a particular length of time. In half-duplex mode back-pressure is applied by forcing the physical layer into not allowing transmissions.

Flow control can be explicitly invoked by setting the `fc_sel` bit in the `mac_cfg` register, or it can be requested automatically by the interface when the number of receive DMA buffers falls below the low watermark (see [Section: "Descriptor Count Watermarks" on page 157](#)). Flow control is invoked if any of the sources request it.

The flow control method used when the interface is unable to accept data is set by the `fc_cmd` field in the `mac_cfg` register, as shown in [Table 164](#).

**Table 164: Back Pressure Methods in Half-Duplex Operation**

<code>fc_cmd</code>	Half-Duplex Back Pressure Method
2'b00	Disabled
2'b01	Force Collision
2'b10	False Carrier
2'b11	Reserved

If the interface is operating in half-duplex mode and flow control is invoked back pressure will be applied. If the `fc_cmd` field is set to 2'b01 back pressure is generated by the collision method. The interface forces collisions on the link by transmitting the collision data pattern. If the carrier is not currently being detected, or the protocol engine has not yet received more than the number of bytes set in the `ifg_rx` field of the `mac_frame_cfg` register then the collision transmission begins immediately. If more than the threshold number of bytes have been received in the current packet the collision will not be forced until after the incoming packet has completed (to avoid a late collision). Packet transmission is still possible when back pressure is applied in this way. If there is an outgoing packet it will contend for access in the usual manner. If the transmitter gains the link then the packet can be sent normally, if it loses then the back pressure machine will cause collisions with the packet that another source is trying to send.

In half-duplex mode if `fc_cmd` is set to 2'b10 the interface will apply back pressure by causing a false carrier sense. This is done by sending the collision data pattern (i.e. generating a carrier) after only 1/3rd of inter-frame gap time has elapsed since the carrier was last detected. This causes all other sources to sense a carrier and backoff. The carrier must be continually generated to ensure the other stations continue to detect it, so it is not possible to transmit packets while using this method.

If the interface is operating in full-duplex mode, flow control is applied by transmitting a Pause Frame. A frame is created with the Pause Count (the number of slot times that the peer should not transmit) set to the value encoded in the `tx_pause_cnt` field in the `mac_cfg` register. When a pause is requested the slot times are also counted locally. Towards the end of the period if the interface can still not accept packets an additional pause frame will be sent, requesting another pause in transmission by the peer. If flow control is no longer needed because DMA buffers have been added or the `fc_sel` bit has cleared, or if flow control is disabled by the `fc_cmd` field being set to 2'b00, then a Pause Frame is immediately sent with a zero Pause Count to restart the peer.

In full-duplex mode the reception of a valid Pause Frame always suspends transmission at the end of the packet currently being sent, transmission will be resumed after the requested pause time unless another flow control packet is received. Pause frames will be detected if their destination address matches the address in the **mac\_ethernet\_addr** register or the multicast address 01-80-C2-00-00-01 and the packet type matches the MAC Control type (88\_08) and the pause frame opcode (00\_01) is found. Normally valid Pause frames will be consumed by the MAC and not delivered to the DMA engine. However in some situations (for example where a system requires that high priority traffic ignore the flow control) it is useful for the Pause Frame passed to the software (which could disable the low priority DMA channel and keep the high priority one active). On devices with the system revision indicating PERIPH\_REV3 or greater the fwdpause\_en bit can be set in the **mac\_adfilter\_cfg** register to cause the hardware to pass Pause frames to the DMA engine instead of acting on them.

Hardware response to pause frames is normally done at the MAC level. This ensures that transmission stops as soon as possible. On devices with the system revision PERIPH\_REV3 or greater the flow control may be performed in the DMA engines allowing each DMA channel to either obey or ignore the directive. Since there may be packets queued in the transmit fifo (up to 1KB of data between the DMA engine and MAC) there is a longer round-trip for the flow control loop in this case. The ch\_base\_fc\_en bit must be set in the **mac\_vlantag** register to enable channel based flow control. Once this bit has been set if the fc\_pause\_en bit is set in the **dma\_config1** register for a channel then it will pause on flow control, otherwise it continues to transmit even when flow control is requested. On devices with the system revision PERIPH\_REV3 or greater when the fwd\_pause\_en bit is clear and hardware is acting on the pause frame the current pause state can be read from the tx\_pause\_on bit in the **mac\_status** register, this bit will be set if the MAC is still in the pause interval set by the previous pause frame.

There are a number of ways pause frames can be managed. The standard one is to have the MAC act on them and consume them. Alternatively, the DMA engine can act on them and stop only one channel. Or they are passed to software and the hardware will not respond to them. Note that in an entirely software controlled flow control system the cpu\_pause\_en bits in the **dma\_config1** register can be used to pause DMA channels, but that the control loop will be much longer than having hardware act on the pause frames. A mix of software and hardware may be best, for example if channel based flow control is being used the DMA engine can be configured to stop both channels on receipt of a pause frame but software can poll the tx\_pause\_on bit and may choose to release one of the channels.

**Table 165: Pause Frame Options**

fwdpause_en	ch_base_fc_en	Ch 0 fc_pause_en	Ch 1 fc_pause_en	Description
0	0	x	x	Standard operation. The MAC will pause on receipt of a pause frame and will consume the frame.
0	1	0	0	Ignore pause frames. The pause information is given to the DMA engine but is applied to neither channel.
0	1	1	0	Pause only channel zero. The pause information is given to the DMA engine but is only applied to channel 0.
0	1	0	1	Pause only channel one. The pause information is given to the DMA engine but is only applied to channel 1.
0	1	1	1	Pause both channels. The pause information is given to the DMA engine and is applied to both channels. The Standard operation setting will achieve the same result with a shorter flow control loop time, so this setting is only likely to be used in systems that switch between it and selective channel pause.





**Table 165: Pause Frame Options (Cont.)**

<b>fwdpause_en</b>	<b>ch_base_fc_en</b>	<b>Ch 0_fc_pause_en</b>	<b>Ch 1_fc_pause_en</b>	<b>Description</b>
1	x	x	x	Software pause frames. The pause frame will be received by software which may use the <code>cpu_pause_en</code> bit to pause channels, or may use it to cause discard from or reordering of internal queues. Note that the interrupt service latency will be part of the flow control loop time, so the link peer will need larger buffers.

If the interface is operating in 8 or 16 bit encoded Packet FIFO mode (see [Section: “8-Bit Packet FIFO Operation” on page 293](#) and [Section: “16-Bit Packet FIFO Operation” on page 298](#)) the flow control is invoked by asserting the RXFC signal. In all other Packet FIFO modes flow control is ignored.

## INTERRUPTS

There are two methods of signalling interrupts from the Ethernet and Packet Fifo interface. The standard method uses a single system interrupt for all DMA and management interrupts, the split method uses a special interrupt for DMA channel 1 and the standard interrupt covers channel 0 and management.

The **mac\_status** register contains the status information for a MAC. The low 32 bits contain the receive and transmit status for each of the DMA channels, the upper bits contain error and RMON counter information. There is a mask register **mac\_int\_mask** associated with the status register. If a bit is set in both the mask and status then an interrupt will be raised. Reads to the **mac\_status** register have the side effect of clearing latched interrupt information. The information in the status register may be read with no side effects through the **mac\_status\_debug** register.

The `split_ch1` bit in the `mac_int_mask` register is used to select between the standard and split methods of signalling and clearing interrupts. It defaults to 0 to give the standard behaviour.

### STANDARD INTERRUPT SIGNALING

In the standard mode the main interrupt for the MAC is raised whenever any status bit is set and the corresponding mask bit is set. Reads from the **mac\_status** register will clear all latched bits.

### SPLIT INTERRUPT SIGNALING

In the split mode the main interrupt for the MAC is raised whenever any channel 0 DMA status bit (from bits 23:16, 7:0) is set and the corresponding mask bit is set, or when any error status bit (from bits 46:40) and corresponding mask bit is set. Reads from the **mac\_status** register will clear all latched bits associated with channel 0 or errors. The channel 1 interrupt for the MAC will be raised if any channel 1 DMA status bit (from bits 31:24, 15:8) is set and the corresponding mask bit is set. Reads from the **mac\_status1** register will clear all latched bits associated with channel 1.

## MANAGEMENT INTERFACE TO PHY

There is a simple serial management interface between the MAC and the PHY device (or devices). It supports up to 32 PHY chips (although some PHY devices attach special meaning to address 0) each of which have 32 16-bit registers.

The CPU runs the serial protocol through reading and writing the **mac\_mdio** register.

The management data clock (MDC) is generated by the interface and is an input to the PHY. Data written to the mdc bit in the **mac\_mdio** register is put out on the MDC pin.

The management data i/o line (MDIO) can be driven by either the MAC or the PHY. It should have an external 1.5K pull-up resistor to pull the line to a 1 when neither is driving. The state of the pin can always be read on the mdio\_in bit in the **mac\_mdio** register. The interface can drive the line by clearing the mdio\_dir bit and writing the data to output to the mdio\_out bit.

The MDIO data is latched by the PHY on the rising edge of MDC, so the mdio\_dir and mdio\_out bits should never be changed when the mdc bit is changed from a zero to a one. To send data to the PHY the mdio\_dir bit should be cleared, a single write to the **mac\_mdio** register can be used to set mdc low and put the data on mdio\_out. The same data should be written to mdio\_out when mdc is set high.

There is a general purpose output pin (GENO) associated with each MAC that is also controlled through the **mac\_mdio** register. This pin is not part of the standard G/MII connection.

Each MAC interface has its own set of MDIO/MDC/GENO pins. This allows the MAC drivers to be completely independent with no shared resources, which works well in designs where the MACs are partitioned across the processors with each running its own operating system. In a system running a single operating system (or where all the MACs are controlled from a single driver) the PHY addressing scheme can be used to attach the PHYs for all three MACs to the MDIO/MDC lines of a single MAC. In this case the other MDC lines become available as general outputs, and the other MDIO lines as GPIOs.

Figure 59 shows the flow for the main loop for sending and reading bits.

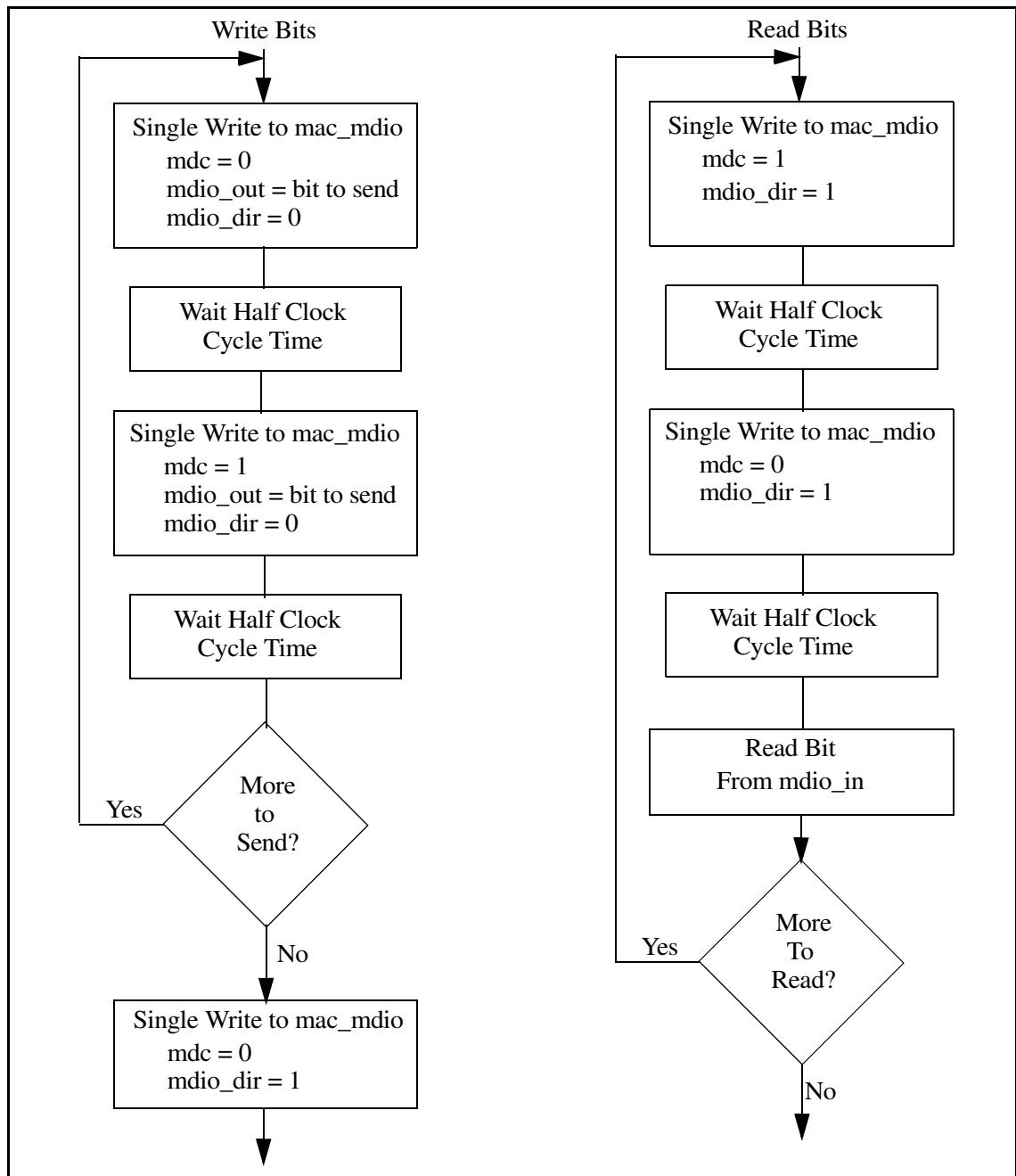


Figure 59: MDIO Flows

The management protocol is shown in Table 166. This shows the two sides of the read (the MAC request and the PHY reply) and the full sequence sent by the MAC for a write. Bits are marked Z where the line is not being driven. Bits marked A form the five bit address of the PHY that the MAC is directing the command to (typically the PHY chip will have some input pins to set the address it responds to). The bits marked R form the five bit register address. The bits marked x are the transfer of the 16 bits being read or written. The two addresses and data are sent most significant bit first.

Table 166: MAC to PHY Management Protocol

Protocol	Idle	Start	Op Code	Device Addr	Register Addr	Turn Around	Data	Idle
Read (MAC)	Idle	01	10	AAAAA	RRRRR	ZZ	ZZZZ ZZZZ ZZZZ ZZZZ	Idle
Read (PHY)	Idle	ZZ	ZZ	ZZZZZ	ZZZZZ	Z0	XXXX XXXX XXXX XXXX	Idle
Write (MAC)	Idle	01	01	AAAAA	RRRRR	10	XXXX XXXX XXXX XXXX	Idle
Write (PHY)	Idle	ZZ	ZZ	ZZZZZ	ZZZZZ	ZZ	ZZZZ ZZZZ ZZZZ ZZZZ	Idle

Some PHY devices can signal interrupts when the link status changes. These may be connected through one of the GPIO pins.

## RMON COUNTERS

Each MAC has 32 RMON statistics gathering registers. They are all 32 bits wide except the byte counts which are 64 bits wide. Zeros will be read in the top 32 bits if a 64 bit access is made to the 32 bit registers, so software can treat them as all being 64 bit except for computing overflow. The registers may be cleared by software writing 0 to them. The counters must be written when the interface is initialized, until a write has been performed their behavior will be UNPREDICTABLE. A write of any value other than zero will set the counter to that value and it will increment from there. If any counter overflows it will wrap to zero and continue to count. The `cntr_ovrfl_err` bit is set and the counter number is written to the `counter_addr` field in the `mac_status` register. The RMON counters are only updated in Ethernet Mode.

Table 167: RMON Counters

Number (offset from 00_1006_0000)	Counter	Description
0 _0 - +4000 _1 - +5000 _2 - +6000	Tx Byte Counter	This Counter counts the total number of bytes successfully transmitted by the MAC. All bytes in the frame from the first destination address byte to the last CRC byte are included in the count. Successful transmission of a packet is defined by transmission of a packet without any errors such as collision, underrun or detection of invalid byte valid in control field of FIFO. This is a 64 bit count.
1 _0 - +4008 _1 - +5008 _2 - +6008	Tx Collision Counter	This Counter counts the total number of collisions experienced by the MAC since the counter was last cleared. The count includes late collisions and will increment by 16 on an excessive collision.
2 _0 - +4010 _1 - +5010 _2 - +6010	Tx Late Col. Counter	This Counter counts the total number of late collisions experienced by the MAC since the counter was last cleared.
3 _0 - +4018 _1 - +5018 _2 - +6018	Tx Ex. Col. Counter	This Counter counts the total number of excessive collisions experienced by the MAC since the counter was last cleared. An excessive collision is defined as 16 consecutive collisions during the attempt to transmit a particular packet. The MAC will abort the packet transmission when it suffers excessive collisions.



Table 167: RMON Counters (Cont.)

Number (offset from 00_1006_0000)	Counter	Description
4 _0 - +4020 _1 - +5020 _2 - +6020	Tx FCS Error Counter	This Counter counts the total number of packets transmitted by the MAC with FCS error. This can only happen when MAC is not appending the CRC to the transmitted packet. The MAC checks the CRC on all transmitted packets and flags all packets with FCS Errors.
5 _0 - +4028 _1 - +5028 _2 - +6028	Tx Abort Error Counter	This Counter counts the total number of packets which were aborted during transmission. Transmission can be aborted by a FIFO underrun.
6	Reserved	
7 _0 - +4038 _1 - +5038 _2 - +6038	Tx Bad Packet Counter	This Counter counts the total number of bad packets transmitted by the MAC. Bad packets are those packets which are not successfully transmitted. A packet which experienced late collision or excessive collision or is aborted by MAC during its transmission is considered bad packet.
8 _0 - +4040 _1 - +5040 _2 - +6040	Tx Good Packet Counter	This Counter counts the total number of successfully transmitted packets.
9 _0 - +4048 _1 - +5048 _2 - +6048	Tx Runt Packet Counter	This Counter counts the total number of runt packets (packets smaller than the minimum frame size) transmitted by the MAC. The minimum frames size for IEEE 802.3 compliance is 64 bytes, if a different minimum frame size has been set in the <b>mac_frame_cfg</b> register then runt packets will be any packet smaller than that number.
10 _0 - +4050 _1 - +5050 _2 - +6050	Tx Over Size Packet Counter	This Counter counts the total number of oversize packets (packets larger than the maximum frame size) transmitted by the MAC. The maximum frames size for IEEE 802.3 compliance is 1518 bytes, if a different maximum frame size has been set in the <b>mac_frame_cfg</b> register then oversize packets will be any packet larger than that number.
11	Reserved	
12	Reserved	
13	Reserved	
14	Reserved	
15	Reserved	
16 _0 - +4080 _1 - +5080 _2 - +6080	Rx Byte Counter	This Counter counts the total number of bytes received by the MAC regardless of the status of a packet (good or bad, as defined below). This is a 64 bit count.
17 _0 - +4088 _1 - +5088 _2 - +6088	Rx Multicast Packet Counter	This Counter counts the total number of packets received by the MAC with a multicast destination address (broadcast packets are not counted).
18 _0 - +4090 _1 - +5090 _2 - +6090	Rx Broadcast Packet Counter	This Counter counts the total number of packets received by MAC with the broadcast destination address.
19 _0 - +4098 _1 - +5098 _2 - +6098	Rx Bad Packet Counter	This Counter counts the total number of bad packets received by the MAC. Bad packets are those packets which have at least one of the following errors: FCS Error Length Error Dribble Error (Only in 10/100 Mbp/s) Code Error Runt Packet Error Oversize Packet Error

Table 167: RMON Counters (Cont.)

Number (offset from 00_1006_0000)	Counter	Description
20 _0 - +40A0 _1 - +50A0 _2 - +60A0	Rx Good Packet Counter	This Counter counts the total number of good packets received (packets with no errors) by the MAC.
21 _0 - +40A8 _1 - +50A8 _2 - +60A8	Rx Runt Packet Counter	This Counter counts the total number of runt packets (packet smaller than the minimum frame size) received by the MAC. The minimum frames size for IEEE 802.3 compliance is 64 bytes, if a different minimum frame size has been set in the <b>mac_frame_cfg</b> register then runt packets will be any packet smaller than that number.
22 _0 - +40B0 _1 - +50B0 _2 - +60B0	Rx Over Size Packet Counter	This Counter counts the total number of oversize packets (packets larger than the maximum frame size) received by the MAC. The maximum frames size for IEEE 802.3 compliance is 1518 bytes, if a different maximum frame size has been set in the <b>mac_frame_cfg</b> register then oversize packets will be any packet larger than that number.
23 _0 - +40B8 _1 - +50B8 _2 - +60B8	Rx FCS Error Packet Counter	This Counter counts the total number of packets received by the MAC with an FCS Error (i.e. a bad CRC).
24 _0 - +40C0 _1 - +50C0 _2 - +60C0	Rx Length Error Packet Counter	This Counter counts the total number of packets received by the MAC with Length Error (where the actual number of bytes received did not match the length given in the Ethernet header).
25 _0 - +40C8 _1 - +50C8 _2 - +60C8	Rx Code Error Packet Counter	This Counter counts the total number of packets received by the MAC that had a Code error signalled by the PHY.
26 _0 - +40D0 _1 - +50D0 _2 - +60D0	Rx Dribble Error Packet Counter	This Counter counts the total number of packets received by the MAC with a dribble error. A dribble error (also referred to as an alignment error) is when a packet is received that does not contain an integral number of bytes. This counter is not used for 1000 Mbps operation.
27	Reserved	
28	Reserved	
29	Reserved	
30	Reserved	
31	Reserved	



## PACKET FIFO INTERFACES

In addition to the standard Ethernet mode, the network interfaces can be put into Packet FIFO mode. In this mode the Ethernet protocol processing is disabled and the pins are used to provide a simple interface with either an 8-bit or 16-bit data path each way. The Packet FIFO interface can be clocked at up to 208 MHz or higher (subject to part speed grade, for full details refer to the hardware data sheet). As can be seen in [Figure 51 on page 264](#) the Packet FIFO engine sits to the side of the Ethernet MAC unit, but most of the network interface logic is common to the Ethernet and Packet FIFO modes. The same DMA controller is used to move data between the packet interface and memory.

There are several modes of Packet FIFO operation, these select the data width and method of framing packets. The minimum packet size supported in the Packet Fifo modes is 10 bytes, using smaller packets than this will have UNPREDICTABLE results. The maximum packet size that will be correctly reported is 16K - 1 bytes (because of the field width in the DMA descriptor) however the interface does not impose a maximum and will continue receiving as long as there are DMA descriptors available.

The 8-bit Packet FIFO modes simply replace the Ethernet MAC with the simpler Packet FIFO logic. The data pins remain the same and the data valid (RXDV, TXEN) and error (RXER, TXER) pins are used to signal framing information. In addition in the encoded mode the collision (COL) and management data (MDIO) pins get used to provide flow control on the link. Switching between Ethernet and any 8-bit Packet mode only affects the interface being switched, so there can be any mix of Ethernet and 8-bit Packet interfaces.

The 16-bit Packet FIFO modes do a larger re-configuration of the pins. The Data Sheet gives the mapping from the Ethernet use to the 16-bit mode uses of the pins. Interface 1 does not support 16-bit modes, the system behavior will be UNDEFINED if it is configured for 16-bit operation. The reuse of pins causes some restrictions on the configurations that can be supported when 16-bit Packet FIFO mode is used. If Interface 0 is put in 16-bit Packet FIFO mode then interface 1 may not be used. If Interface 2 is put in 16-bit Packet FIFO mode then interface 1 may not be used, and interface 0 can only be used in a Packet FIFO mode.

On the BCM1125/H the interfaces may be configured as two 8-bit interfaces or a single 16 bit interface.

**Table 168: BCM1125 Ethernet/Fifo Pin Usage**

	E0_ pins	E1_ pins
<b>All Interfaces Ethernet or 8-bit Fifo</b>	—GMII or 8 bit Fifo E0_—	—GMII or 8-bit Fifo E1_—
<b>One 16-bit Packet Fifo</b>	—————16-bit Packet Fifo F_—————	

The valid combinations for a BCM1250 are:

- Interface 0 in 16-bit Packet mode, Interface 1 disabled, Interface 2 in Ethernet mode.
- Interface 0 in 16-bit Packet mode, Interface 1 disabled, Interface 2 in 8-bit Packet mode.
- Interface 0 in 16-bit Packet mode, Interface 1 disabled, Interface 2 in 16-bit Packet mode.
- Interface 0 in 8-bit Packet mode, Interface 1 disabled, Interface 2 in 16-bit Packet mode.

The pin mapping is summarized in the following table:

**Table 169: BCM1250 Ethernet/Fifo Pin Usage**

	E0_ pins	E1_ pins	E2_ pins
<b>All Interfaces Ethernet or 8-bit Fifo</b>	—GMII or 8-bit Fifo E0_—	—GMII or 8-bit Fifo E1_—	—GMII or 8-bit Fifo E2_—
<b>One 16-bit Packet Fifo, One Ethernet or 8-bit Fifo</b>	—————16-bit Packet Fifo F0_—————		—GMII or 8-bit Fifo E2_—
<b>Two 16-bit Packet Fifos</b>	—————16-bit Packet Fifo F0_—————		
		—————16-bit Packet Fifo F1_—————	
<b>One 16-bit Packet Fifo, One 8-bit Packet Fifo</b>	— 8-bit Fifo E0_—	—————16-bit Packet Fifo F1_—————	

Other than the external data width and framing options (described below) the Packet FIFO modes use the same internal data path. The following rules/differences apply in all modes.

- The speed selection in the **mac\_cfg** register must be set for gigabit operation to use the reference input clock to generate the transmit clock.
- The IPv4 header checking is not performed. The iphdr\_offset field in the **mac\_adfilter\_cfg** register is not used and should be set to zero.
- The address filter is not normally useful in Packet FIFO mode and can be configured to accept all packets by setting the allpkt\_en bit in the **mac\_adfilter\_cfg** register.
- The 13th and 14th bytes of the incoming packet are still compared as if they contained an Ethernet type and the DMA status word in bits [57:55] will still be set as described in [Section: “Packet Type Identification” on page 281](#).
- If the bypass\_fcs\_chk bit is set in the **mac\_cfg** register then received packets will have their final four bytes checked for a valid CRC-32 (using the same algorithm as for the Ethernet) and the bit in the receive DMA status word will be set accordingly. If the bypass\_fcs\_chk bit is clear the status bit will always indicate the CRC check passed.
- The receive channel selection is done in the same way as for the Ethernet mode as described in [Section: “Receive DMA Channel Selection” on page 280](#).
- The only transmit options (see [Section: “Data Buffers and Descriptors” on page 147](#) and [Table 107 on page 173](#)) that can be used in the start of packet descriptor are No Modification (4'b0111) and Append CRC (4'b001). All other options will give UNPREDICTABLE results.
- Some modes of the Packet FIFO interface allow packet errors to be signalled. These packets will be marked as bad in the receive DMA status information. The automatic discard of error packets (described in the [Section: “Receiver Configuration” on page 274](#)) should be disabled in Packet FIFO modes.
- The RMON counters are not incremented when the interface is in Packet FIFO modes.





## Flow Control In Encoded Packet FIFO Modes

Flow control is available in the encoded packet fifo modes. In addition to the DMA descriptor based flow control there is a link level flow control.

The transmitter can be controlled by the COL input in 8 bit mode and the TXFC input in 16 bit mode (note that these are the same physical pin). Since the input is synchronized internally there is some uncertainty in the number of additional bytes that can be transmitted following assertion of the flow control signal. The transmitter will stop:

- 2-4 cycles after TXFC assertion in 16 bit mode no CRC (4-8 bytes)
- 2-6 cycles after TXFC assertion in 16 bit mode + CRC (4-12 bytes)
- 2-4 cycles after TXFC assertion in 8 bit mode no CRC (2-4 bytes)
- 2-8 cycles after TXFC assertion in 8 bit mode + CRC (2-8 bytes)

The receiver will assert its flow control output (MDIO in 8 bit mode, RXFC in 16 bit mode) either because the DMA watermark state requests flow control or to prevent the receive fifo overflowing. The `enc_fc_thrsh` field in the `mac_thrsh_cfg` register sets the flow control threshold. Flow control will be requested when there are fewer than this number of doublewords free in the fifo. There is a 1 cycle delay from the number of entries in the fifo falling below the threshold and the external signal being asserted, and there can be up to 10 cycles of data (10 bytes in 8 bit mode, 20 bytes in 16 bit mode) in flight between the receive data pins and the fifo.

As an example consider connecting two parts back-to-back. With the default threshold of 4 double words (32 bytes) in 16 bit mode there can be 22 bytes (2 from the one cycle delay and 20 in flight) added to the receive fifo if the transmitter stopped instantaneously. However, there is additional delay. If CRCs are not being appended by the transmitter an additional 8 bytes could be sent, leaving two bytes margin. But if the transmitter was appending CRC then the additional 12 bytes would cause the receive fifo to overflow, so the threshold must be increased to 5 doublewords.

Note that the direction of the MDIO/RCFC pin is controllable by software using the `mdio_dir` bit in the `mac_mdio` register. This must be set for output (which is the default) when receive flow control is being used.

## 8-BIT PACKET FIFO OPERATION

There are two models for running the Packet FIFO. In the packet model Start of Packet (SOP) and End of Packet (EOP) data are flagged and all data between these marks are counted as a single packet and passed through the DMA engine in the same way an Ethernet packet would be. In this packet based model once a SOP has been seen further SOP signals are ignored until an EOP has been triggered. The second model is to have an unframed data stream, in this case the DMA will run through filling DMA buffers in sequence for as long as there are descriptors available.

The receive filtering options are described in [Section: "Packet FIFO Interfaces" on page 291](#). If an unframed data stream is being received software must ensure that the `mac_chup` and `mac_chlo` tables select the same DMA channel at all indices.

Packets transmitted in any of the modes can have a CRC-32 appended by setting the append CRC bit in the transmit packet descriptor. Packets received have their final four bytes checked for a valid CRC-32 and if the `bypass_fcs_chk` bit is set in the `mac_cfg` register the status bit in the receive descriptor will be set accordingly (if the `bypass_fcs_chk` bit is clear the status word will never flag a CRC error). If the CRC-32 is used on the receive side there must be at least seven clock cycles between the end of packet and the start of the next packet. If this inter-frame gap is not provided then the CRC checker will give UNPREDICTABLE results.



There are four formats that are available for the 8 bit Packet FIFO to signal data validity and packet boundaries. In all cases data and control signals are sampled on the rising edge of the clock. On the transmit side the GMII TXEN (enable) and TXER (error) signals are used as two control signals that are used to send out the framing information, and on receive the RXDV (data valid) and RXER (error) pins are used to receive the framing information.

### 8-BIT GMII STYLE PACKET FIFO

The GMII style of Packet FIFO mode is shown in [Figure 60 on page 294](#). The packet data is framed by the TXEN or RXDV signal. The first byte that has it active is marked as the start of a packet and the last byte that has it active is the end of the packet, all bytes between are valid and part of the packet. The TXER or RXER signal can be used to signal an error whenever the frame signal is active.

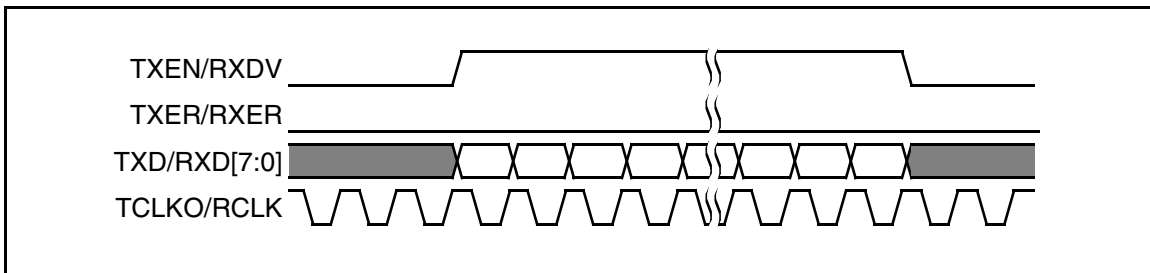


Figure 60: 8-bit Packet FIFO GMII Style

The error pin TXER/RXER may be used to signal that a packet has an error. It should be asserted when the error is detected and remain high until the enable signal falls.

Table 170: Codes for GMII Packet FIFO Mode

	TXEN / RXDV	TXER / RXER
Valid Data, Start of Packet	0->1 at start of cycle	0
Valid Data	1	0
Valid Data, End of Packet	1->0 at end of cycle	0
Error	1	1 held until TXEN/RXDV falls

## 8-BIT ENCODED PACKET FIFO

Encoded mode, shown in Figure 61, uses the control lines to signal the four states that can be associated with the data. This mode should be used for packet data where there can be invalid bytes between the start and end of packet marks. Table shows the encodings used.

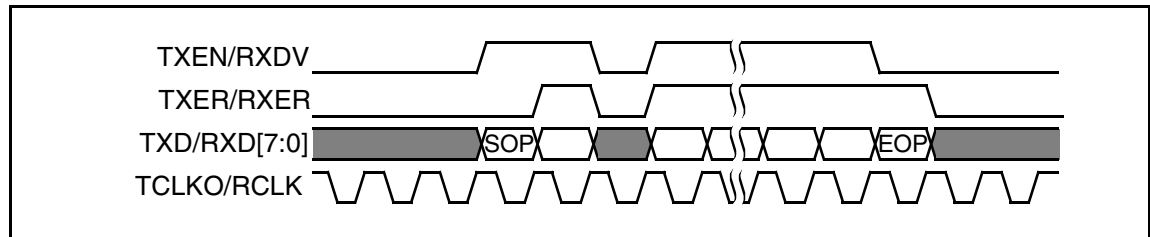


Figure 61: 8-Bit Packet FIFO Encoded Style

Table 171: Codes for 8-Bit Encoded Bypass Mode

	TXEN / RXDV	TXER / RXER
Valid Data, Start of Packet	1	0
Valid Data	1	1
Valid Data, End of Packet	0	1
Data Not Valid	0	0

In this mode flow control is provided in each direction. Transmit flow control is input on the COL pin (normally the collision sense from the PHY), when this is asserted the interface will stop sending data (and drive the data not valid control code) two rising clock edges later. The interface will assert the MDIO pin (normally the management interface to the PHY) as a receive flow control output if (a) the CPU sets the force flow control configuration bit; (b) the number of descriptors on a receive channel falls below the low watermark and automatic flow control is enabled for that channel; or (c) there are less than 4 64 bit words left free in the receive FIFO. If the device the other end of the link honours the flow control request within a few clock times data will never be lost on the link.

In this mode a transmit FIFO underflow during transmission of a packet causes bubbles to be inserted in the data stream (the data not valid code will appear during a packet) but transmission will not be aborted. (In all other modes a transmit FIFO underrun causes the packet to be aborted).

### 8-BIT SOP FLAGGED PACKET FIFO

SOP flagged Packet FIFO mode uses one control line as a data valid signal and the other to flag start of packet. The end of packet is one cycle before the SOP or whenever the data goes invalid (this extra EOP is required to push out the last data at the end of a valid sequence).

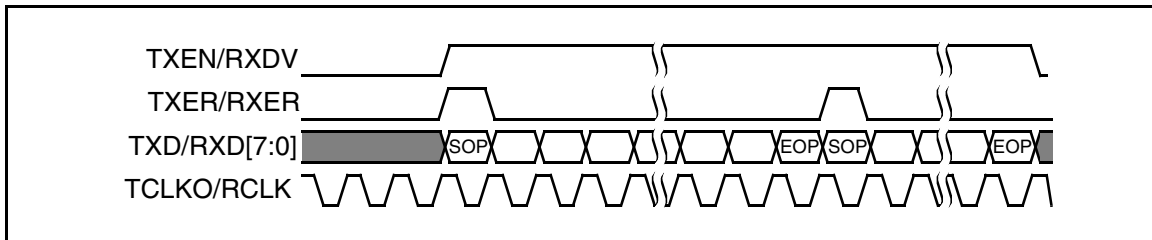


Figure 62: 8-Bit Packet FIFO SOP Style

Table 172: Codes for 8-Bit SOP Packet FIFO

	TXEN / RXDV	TXER / RXER
Valid Data, Start of Packet	1	1
Valid Data	1	0
Valid Data, End of Packet	1	next cycle 1
	1->0 at end of cycle	anything
Data Not Valid	0	anything



## 8-BIT EOP FLAGGED PACKET FIFO

EOP flagged Packet FIFO mode uses one control line as a data valid signal and the other to flag end of packet. The start of packet is one cycle after an end of packet, or the first data after the valid signal has risen. Any SOPs that occur between an SOP and EOP are ignored, so in Figure 63 there is not an SOP flagged in the cycle marked X, one cycle after TXEN/RXDV was low. This mode is good for streaming data that uses only the data valid signal. If the EOP flag is always low the interface will transfer a single never-ending packet for as long as DMA buffers are provided.

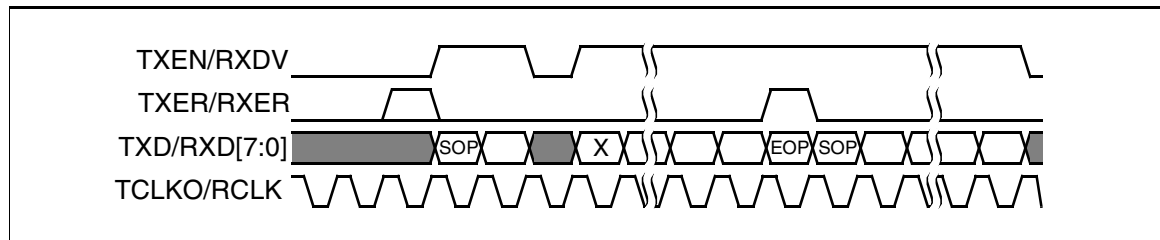


Figure 63: 8-Bit Packet FIFO EOP Style

Table 173: Codes for 8-Bit EOP Bypass Mode

	TXEN / RXDV	TXER / RXER
<b>Valid Data, Start of Packet</b>	0->1 at start of cycle	anything
<b>Valid Data</b>	1	1 previous cycle
<b>Valid Data, End of Packet</b>	1	0
<b>Data Not Valid</b>	0	1
		anything

## 16-BIT PACKET FIFO OPERATION

The 16-bit Packet FIFO mode provides an interface that can run bidirectionally at OC-48 data rates. It can externally be converted into a POS-PHY level 3 interface using a simple gearbox PLD or FPGA. Alternatively, it may be connected directly to an ASIC.

On the BCM1250, there are two Packet FIFO interfaces. On the BCM1125/H there is one. Interface F0 (F on the BCM1125/H) replaces the Ethernet MACs E0 and E1, and reuses pins from both. It uses the DMA controller associated with interface E0. On the BCM1250 Packet interface F1 reuses pins from all three Ethernet MACs (E0, E1 and E2) and uses the DMA engine associated with E2. It is possible to run the F0 packet interface along with the E2 Ethernet interface.

The receive filtering options are described in section [Section: “Packet FIFO Interfaces” on page 291](#). If an unframed data stream is being received software must ensure that the **mac\_chup** and **mac\_chlo** tables select the same DMA channel at all indices.

Packets transmitted in any of the Packet FIFO modes can have a CRC-32 appended by setting the append CRC bit in the transmit packet descriptor. Packets received have their final four bytes checked for a valid CRC-32 and if the **bypass\_fcs\_chk** bit is set in the **mac\_cfg** register the status bit in the receive descriptor will be set accordingly (if the **bypass\_fcs\_chk** bit is clear the status word will never flag a CRC error). If the CRC-32 is used on the receive side there must be at least three clock cycles between the end of packet and the start of the next packet. If this inter-frame gap is not provided then the CRC checker will give UNPREDICTABLE results.

The 16 bit Packet FIFO mode uses three control signals that accompany the data and, in encoded mode, a flow control signal in the reverse direction. The 16 bit wide data path is treated as two bytes where the data on bits [7:0] is earlier in the packet (and therefore will be DMAed to/from a lower memory address) than the data on [15:8]. The last 16 bit half-word of a packet that is an odd number of bytes long will have valid data on bits [7:0] and an UNPREDICTABLE value on bits [15:8].

## 16-BIT GMII STYLE PACKET FIFO

The 16-bit GMII style of Packet FIFO interface is shown in [Figure 64](#). The packet data is framed by the TXC[0] or RXC[0] signal. The first byte that has it active is marked as the start of a packet and the last byte that has it active is the end of the packet, all bytes between are valid and part of the packet. The TXC[1] or RXC[1] signal can be used to signal an error whenever the frame signal is active, it should be asserted when the error is detected and held high until the frame signal falls.

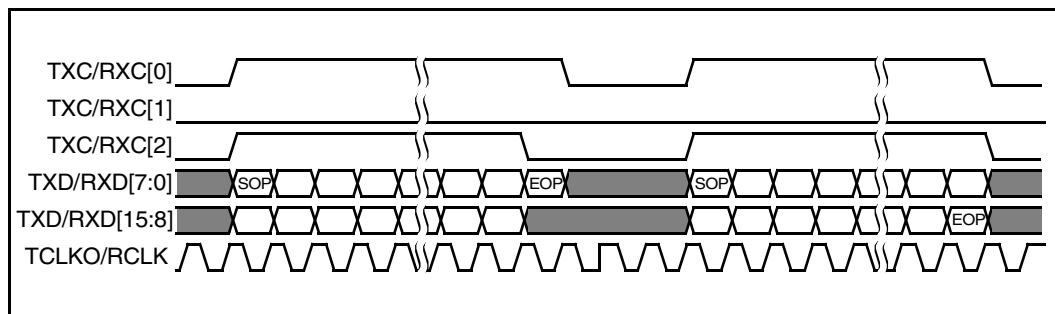


Figure 64: 16-Bit GMII Style Packet FIFO

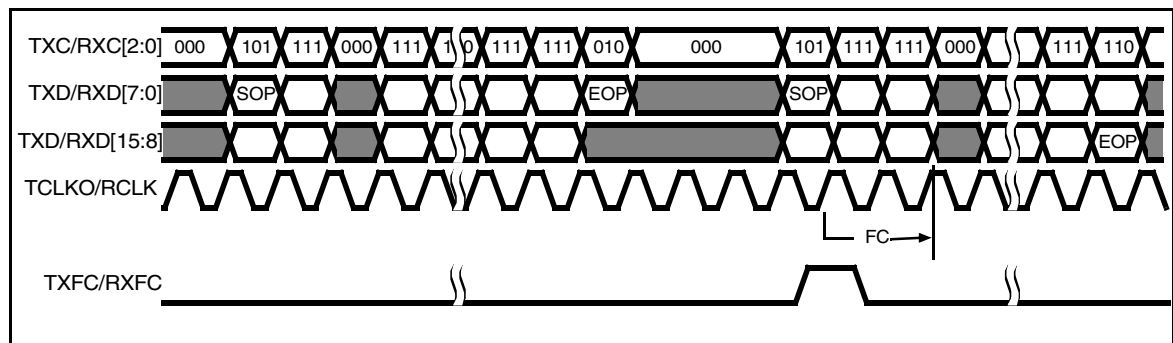
TXC/RXC[2] is used to indicate the validity of the upper 8-bits of data. It should be the same as TXC/RXC[0] except for the final data in a packet that has an odd number of bytes.

**Table 174: Codes for 16-Bit GMII Style Packet FIFO**

	TXC/RXC[2:0]
Data Not Valid	000
Start of Packet, 2 Bytes Valid	000->101 at start of cycle
Middle of Packet, 2 Bytes Valid	101
End of Packet, 1 Byte Valid	001->000 at end of cycle
End of Packet, 2 Bytes Valid	101->000 at end of cycle
Error	111, bit [1] held set until 000

## 16-BIT ENCODED PACKET FIFO

The 16-bit encoded Packet FIFO mode is shown in Figure 65. The TXC[2:0] signals indicate the status of the data they accompany. The encoding is chosen to allow easy conversion to and from POS-PHY Level 3 signalling, which is a wider slower packet bus. The flow control signal runs in the opposite direction to the data. Figure 65 shows two packets being sent. In the first the sender is unable to supply data for one cycle and removed the valid signal. In the second packet the receiver asserts the flow control signal for a cycle requesting a pause from the sender. The transmitter will see the flow control request on the next rising edge of the clock and will send data for 2-4 cycles before suspending transmission one cycle later. The interface will assert its flow control output when receive channel flow control is invoked either by one of the DMA channels having fewer descriptors than the low watermark or by an explicit processor request.



**Figure 65: 16-Bit Encoded Packet FIFO**

**Table 175: Codes for 16-Bit Encoded Bypass Mode**

	TXC/RXC[2:0]
Data Not Valid	000
Reserved	001
End of Packet, 1 Byte Valid	010
End of Packet With Error	011
Reserved	100

Table 175: Codes for 16-Bit Encoded Bypass Mode

	TXC/RXC[2:0]
Start of Packet, 2 Bytes Valid	101
End of Packet, 2 Bytes Valid	110
Middle of Packet, 2 Bytes Valid	111

In this mode flow control is provided in each direction. Transmit flow control is input on the TX\_FC pin, when this is asserted the interface will stop sending data (and drive the data not valid control code) two rising clock edges later. The part will assert the RX\_FC pin as a receive flow control output if (a) the CPU sets the force flow control configuration bit; (b) the number of descriptors on a receive channel falls below the low watermark and automatic flow control is enabled for that channel; or (c) there are less than four 64 bit words left free in the receive FIFO. If the device the other end of the link honours the flow control request within a few clock times data will never be lost on the link.

In this mode a transmit FIFO underflow during transmission of a packet causes bubbles to be inserted in the data stream (the data not valid code will appear during a packet) but transmission will not be aborted. (In all other modes a transmit FIFO underrun causes the packet to be aborted).

## RESTRICTIONS WHEN RESETTING THE INTERFACE

The MAC and Packet Fifo logic is automatically reset when the part is reset. However, on BCM1250s prior to PERIPH\_REV3, correct reset of the logic requires that there be a few rising edges on REFCLK01 (for MACs E0 and E1) and REFCLK2 (for MAC E2) before software releases the interface from reset. This is not normally a problem since the clocks will be free running. If the clocks are not provided then the first access to the registers for the corresponding interface will place the ZBbus in an UNDEFINED state. If an interface is not used there are two options: (1) ensure software never accesses the unused interface registers; (2) provide some edges on the reference clock before software can access the registers. Clearly (2) is preferable, since otherwise a bug in the software (or even a debugger collecting the system state) could cause a hang that would be hard to track down. There is no need to provide a periodic clock. One possible solution would be to connect the unused REFCLK input to the boot ROM chip select (or SCL if the system boots from SMBus) since this is guaranteed to toggle when the part comes out of reset (before any instructions run the SB-1 will have fetched 16 bytes from the boot ROM, thus there will be 4 (for a 32 bit ROM) or 16 (for an 8 bit ROM) edges on the REFCLK even in the unlikely case where the first instruction accesses the MAC registers).

The interface can also be reset by software. Again, for parts prior to PERIPH\_REV3 the clock needs to be running for correct operation (note that if the clock was running during the chip reset and the MAC is not being used then it is safe to perform a software reset without a clock). This can be done by setting the (self-clearing) p\_reset bit in the **mac\_enable** register. After this bit has been written with a 1, the software should delay for at least six times the clock period of the receive clock then write the bit with a 1 again. Following the second write the interface will be fully reset. (If this sequence is not followed the interface will reset but it is possible for a junk packet to be received when the receiver is enabled again). Before software resets the MAC (or a DMA channel) it should ensure that there is no outstanding DMA activity (failure to do this can cause resources to be lost between the MAC and ZBbus and can degrade performance). The DMA should be stopped by clearing the descriptor count by writing the complement of the current count into the count register (the hardware will add this value and discard the carry so the count will become zero), then reading any of the DMA registers, then waiting for twice the transmission time of a maximum length packet (this ensures any in-progress transmission or reception has completed). Once the DMA has stopped the reset can be safely written.





## MAC REGISTERS



**Note** All of the MAC registers need to be initialized by software before it will function. The BCM1125/H has MACs 0 and 1. Accesses made to the address range allocated to MAC 2 may cause all MACs to exhibit UNPREDICTABLE behavior.

**Table 176: MAC Configuration Registers**

Bits	Name	Default	Description
<b>mac_cfg_0 - 00_1006_4100</b> <b>mac_cfg_1 - 00_1006_5100</b> <b>mac_cfg_2 - 00_1006_6100</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
0	ss_tmode	1'b0	This bit must always be zero for normal operation (Broadcom Use Only test bit).
1	tx_hold_sop_en	1'b0	<p>When this bit is set the interface will retain the first few bytes of each transmitted packet in the transmit FIFO. This enables the interface to automatically retry any packet that encounters an error during the initial part of its transmission. The tx_rl_thrsh parameter (in the mac_threshold register) sets the number of 64 bit FIFO entries that are retained, once this number of entries have been successfully transmitted the FIFO is unlocked and no data is retained. For example if tx_hold_sop_en is set and tx_rl_thrsh is set to 8, then if the MAC detects a collision before 8 entries (64 bytes) are read from the transmit FIFO, the FIFO can be restored the start of the packet and the packet can be automatically retried.</p> <p>Automatic retry can be triggered by any error that happens (collision, underrun error, excessive collision error, or late collision) while the FIFO is in locked state. (i.e. less than the number of entries set by tx_rl_thrsh have been read from the transmit FIFO for a given packet.). The retry_en, ret_drprep_en and ret_ufl_en bits enable automatic retry selectively for each of the error types.</p> <p>If the tx_hold_sop_en bit is clear or the number of entries set by tx_rl_thrsh have been read from the FIFO then the FIFO will be unlocked and automatic retry is not possible. In this case the packet with the error will be dropped.</p> <p>Used in Ethernet and Packet FIFO modes.</p>
2	retry_en	1'b0	<p>If this bit is set then automatic retry is enabled when collisions are detected (but not late or excessive collisions). The tx_hold_sop_en bit (bit [1]) must be set to allow automatic retry, see the description above for more details.</p> <p>Used in Ethernet and Packet FIFO modes.</p>
3	ret_drprep_en	1'b0	<p>If this bit is set then automatic retry is enabled when late collisions or excessive collisions are detected. The tx_hold_sop_en bit (bit [1]) must be set to allow automatic retry, see the description above for more details.</p> <p>Used in Ethernet and Packet FIFO modes.</p>
4	ret_ufl_en	1'b0	<p>If this bit is set then automatic retry is enabled when a transmit FIFO underflow is detected. The tx_hold_sop_en bit (bit [1]) must be set to allow automatic retry, see the description above for more details.</p> <p>Used in Ethernet and Packet FIFO modes.</p>
5	burst_en	1'b0	<p>This bit should be set to enable burst operation when the MAC is running in 1000 Mbp/s mode. Burst mode enables transmission of multiple packets without releasing the network between them.</p> <p>Used in Ethernet mode only.</p>

**Table 176: MAC Configuration Registers (Cont.)**

mac_cfg_0 - 00_1006_4100 mac_cfg_1 - 00_1006_5100 mac_cfg_2 - 00_1006_6100 This register is used in both Ethernet and Packet FIFO modes			
Bits	Name	Default	Description
8:6	tx_pause_cnt	3'b0	Transmit pause count. This field sets the number of slot times pause that will be requested when the interface transmits a pause frame. It is also used to initialize the local pause counter which indicates when the next flow control packet needs to be sent. The encodings are: 000: 512 001: 1K 010: 2K 011: 4K 100: 8K 101: 16K 110: 32K 111: 64K  Used in Ethernet mode only.
16:9	reserved	8'b0	Reserved
17	ap_stat_en	1'b0	Append packet status enable. When this bit is set the packet status is written to the receive FIFO following the last entry in the packet. This bit should be set for Ethernet mode and clear for Packet FIFO mode.  Used in Ethernet and Packet FIFO modes.
18	reserved	1'b0	Reserved
19	drp_errpkt_en	1'b0	If this bit is set then automatic dropping is enabled for all error packets (bits [31:26] are ignored).  Used in Ethernet mode only.
20	drp_fcserpkt_en	1'b0	If this bit is set then packets with an FCS (CRC) error are automatically dropped.  Used in Ethernet mode only.
21	drp_codeerrpkt_en	1'b0	If this bit is set then packets that are signalled with a code error by the PHY are automatically dropped.  Used in Ethernet mode only.
22	drp_drblerpkt_en	1'b0	If this bit is set then packets with a dribble error are automatically dropped.  Used in Ethernet mode only.
23	drp_rntpkt_en	1'b0	If this bit is set then runt packets (those shorter than the minimum packet size configured in the <b>mac_frame_cfg</b> register) are automatically dropped.  Used in Ethernet mode only.
24	drp_oszpkt_en	1'b0	If this bit is set then oversize packets (those longer than the maximum packet size configured in the <b>mac_frame_cfg</b> register) are automatically dropped. (Unless a non-standard protocol is used the maximum packet size will always be larger than the receive FIFO, so this is not likely to happen in practice.)  Used in Ethernet mode only.
25	drp_lenerrpkt_en	1'b0	If this bit is set then packets with a length error are automatically dropped. A length error is signalled when the length/type field of the packet is less than or equal to 1500 (indicating it is being used as the length) and its value does not match the packet length.  Used in Ethernet mode only.
31:26	reserved	6'b0	Reserved



Table 176: MAC Configuration Registers (Cont.)

mac_cfg_0 - 00_1006_4100 mac_cfg_1 - 00_1006_5100 mac_cfg_2 - 00_1006_6100 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
32	bypass_sel	1'b0	When this bit is set the MAC is bypassed and Packet FIFO mode is enabled. This provides a simple FIFO interface, backed by the DMA engine. See <a href="#">Section: "8-Bit Packet FIFO Operation" on page 293</a> for details.
33	hdx_en	1'b0	When this bit is set the MAC will operate in half-duplex mode, when the bit is clear it will run full-duplex. The MAC supports both full and half-duplex operation at all speeds.  Used in Ethernet mode only.
35:34	speed_sel	2'b0	This field sets the speed the MAC will operate. 00: 10 Mbps - the TCLK input should be 2.5 MHz. 01: 100 Mbps - the TCLK input should be 25 MHz. 10: 1000 Mbps - the transmit clock will be supplied by the interface. The reference clock input should be 125 MHz. 11: Reserved  Used in Ethernet mode only. Must be set to 10 for Packet FIFO modes.
36	tx_clk_edge	1'b0	This selects the edge of the transmit clock that is used to send the data. Normally the rising clock edge should be used. The falling edge can be used to support old 10 Mbps PHY chips that require the full hold time specified in the original IEEE specification. 0: The rising edge of the clock is the active one. 1: The falling edge of the clock is the active one.  Used in Ethernet and Packet FIFO modes.
37	loopback_sel	1'b0	When this bit is set an internal loopback path is enabled, connecting the transmitter and receiver GMII pins.  Used in Ethernet and 8 bit Packet FIFO modes.
38	fast_sync	1'b0	This bit must be set for normal operation. (It defaults to 0 so software should be sure to set it before starting the MAC.)  Used in Ethernet and Packet FIFO modes.
39	ss_en	1'b0	This bit must always be set. (Broadcom Use Only debug bit). (It defaults to 0 so software should be sure to set it before starting the MAC.)  Used in Ethernet and Packet FIFO modes.
41:40	bypass_cfg	2'b0	This field sets the strobe signals that are used on both transmit and receive interfaces in Packet FIFO mode. 00: GMII style 01: Encoded 10: SOP flagged (not valid in 16 bit Packet FIFO mode) 11: EOP flagged (not valid in 16 bit Packet FIFO mode)  Used in Packet FIFO modes.
42	bypass_16	1'b0	Set for 16 bit Packet FIFO mode, clear for 8 bit Packet FIFO mode. The interface behavior will become UNDEFINED if this bit is set for MAC 1. Setting this bit for MAC 0 will cause the pins to change from their eight bit E0 and E1 configuration to the 16 bit F0 configuration. Setting this bit for MAC 2 will cause the pins to change from their eight bit E0, E1 and E2 configuration to the 16 bit F1 configuration. This bit must be set to reflect the external connection.  Used in Packet FIFO modes.

**Table 176: MAC Configuration Registers (Cont.)**

mac_cfg_0 - 00_1006_4100 mac_cfg_1 - 00_1006_5100 mac_cfg_2 - 00_1006_6100 This register is used in both Ethernet and Packet FIFO modes			
Bits	Name	Default	Description
43	bypass_fcs_chk	1'b0	If this bit is set then the interface will perform the FCS check on packets received in Packet FIFO mode. The last four bytes are checked for a valid CRC-32 using the same algorithm as for Ethernet. The status will be recorded in the receive DMA descriptor status field. If this bit is clear no FCS check is done and all packets will be marked as having a correct CRC.  Used in Packet FIFO modes.
44	rx_ch_sel_msb	1'b0	This is the top bit of rx_ch_sel (see bits 63:57).  Used in Ethernet and Packet FIFO modes.
45	split_ch_sel	1'b0	System revision < PERIPH_REV3: Reserved System revision PERIPH_REV3 or later: If this bit is set the 8 bit index used to select the receive channel number is made up from nibbles extracted from two different places in the packet, {rx_ch_sel_msb,rx_ch_sel} specify the offset of the lower 4 bits of the index, rx_ch_msn_sel (in the mac_adfilter register) specify the upper 4 bits. If split_ch_sel is zero then {rx_ch_sel_msb,rx_ch_sel} is used to select the low 4 bits of the index and the upper 4 bits is always the next nibble.  Used in Ethernet and Packet FIFO modes.
53:46	bypass_ifg	8'b0	This field gives the number of clock cycles of inter-frame gap that is inserted in Packet FIFO mode. If the transmitter is set to append a CRC then this field has a minimum value of 3 in 16 bit FIFO mode and a minimum value of 7 in 8 bit FIFO mode. If CRCs are never inserted then packets may be transmitted back to back.  Used in Packet FIFO modes.
54	fc_sel	1'b0	This bit is used by software to force flow control. If it is set then flow control will be asserted on the link (Pause frames in full duplex Ethernet, backpressure by the method encoded in the fc_cmd bits for half duplex Ethernet, or the link level flow control pin RXFC for encoded Packet Fifo mode). Flow control will remain asserted until this bit is cleared. Note that if this bit is set while the mac is disabled the results are UNPREDICTABLE.  Used in Ethernet and Packet FIFO modes.
56:55	fc_cmd	2'b0	This field sets the flow control style to be used for both software flow control (set by bit 54) and automatic flow control from the receive DMA channels. This field is only used for Half Duplex links. 00: Disabled 01: Enabled using collisions 10: Enabled using false carrier 11: UNPREDICTABLE  Used in Ethernet mode only.
63:57	rx_ch_sel	7'b0	This field along with the rx_ch_sel_msb bit sets the offset into received packets to extract 8 bits to select the receive DMA channel. This offset is in nibbles. If bit [57] is zero then bits [44, 63:58] give the byte of the packet (starting from zero) that will be used to index the channel select table. If bit [57] is one then the top four bits of the byte indexed by [44, 63:58] will be used as the low four bits to index the channel select table, and the lower four bits from the next byte of the packet will be used as the upper four bits to index the channel select table.  Used in Ethernet and Packet FIFO modes.



**Table 177: MAC Enable Registers**

mac_enable_0 - 00_1006_4400 mac_enable_1 - 00_1006_5400 mac_enable_2 - 00_1006_6400 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
1:0	rxdma_en	2'b0	Receive DMA channel enable. Must be set to enable a channel. Bit 0 corresponds to channel 0, bit 1 to channel 1.
3:2	reserved	2'b0	Reserved
5:4	txdma_en	2'b0	Transmit DMA channel enable. Must be set to enable a channel. Bit 4 corresponds to channel 0, bit 5 to channel 1.
7:6	reserved	2'b0	Reserved
8	p_reset	1'b0	Port reset. Writing a 1 to this bit will reset the entire MAC and Packet FIFO interface as though it had come out of a system reset. All registers are restored to their default values thus disabling and resetting the DMA engines, Ethernet mode logic and Packet FIFO logic. The default for this bit is zero, so it will self-clear as part of the reset sequence.
9	reserved	1'b0	Reserved
10	mac_rx_en	1'b0	If this bit is clear the receive Ethernet media access engine will be held in reset. If this bit is set it will run normally.
11	mac_tx_en	1'b0	If this bit is clear the transmit Ethernet media access engine will be held in reset. If this bit is set it will run normally.
12	byp_rx_en	1'b0	If this bit is clear the receive Packet FIFO engine will be held in reset. If this bit is set it will run normally.
13	byp_tx_en	1'b0	If this bit is clear the transmit Packet FIFO engine will be held in reset. If this bit is set it will run normally.
63:14	reserved	50'b0	Reserved

**Table 178: MAC Transmit DMA Control Register**

mac_txd_ctl_0 - 00_1006_4420 mac_txd_ctl_1 - 00_1006_5420 mac_txd_ctl_2 - 00_1006_6420 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
3:0	weight_ch0	4'b0	Transmit channel 0 round robin weight. Note that if both channels are enabled with weight zero (the default) then no packets will be transmitted.
7:4	weight_ch1	4'b0	Transmit channel 1 round robin weight. See note above.
63:8	reserved	56'b0	Reserved

**Table 179: MAC FIFO Threshold Registers**

mac_thrsh_cfg_0 - 00_1006_4108 mac_thrsh_cfg_1 - 00_1006_5108 mac_thrsh_cfg_2 - 00_1006_6108 This register is used in both Ethernet and Packet FIFO modes			
Bits	Name	Default	Description
6:0	tx_wr_thrsh	7'b0	Transmit FIFO write threshold. Sets the number of free 64 bit entries the transmit FIFO must have before it signals that space is available. For DMA operation this field must be set to 4 or 8 entries depending on the state of the tbx_en bit in the dma_config0 register. See <a href="#">Section: "Transmitter Configuration" on page 271</a> .
7	reserved	1'b0	Reserved
14:8	tx_rd_thrsh	7'b0	Transmit FIFO read threshold. Sets the number of valid 64 bit entries the transmit FIFO must hold before the MAC will start transmitting the packet. See <a href="#">Section: "Transmitter Configuration" on page 271</a> . Note that the tx_wr_thrsh + tx_rd_thrsh must be less than the size of the fifo or the transmitter behaviour is UNPREDICTABLE (tx_wr_thrsh + tx_rd_thrsh <= 32 for parts with System Revision of 1, tx_wr_thrsh + tx_rd_thrsh <=128 for parts with revision greater than 2).
15	reserved	1'b0	Reserved
21:16	tx_rl_thrsh	6'b0	Transmit FIFO release count. This sets the number of 64 bit FIFO entries that will be held in the FIFO when the MAC is configured to hold the start of packets. See the tx_hold_sop_en bit in the mac_cfg register. See <a href="#">Section: "Transmitter Configuration" on page 271</a> .
23:22	reserved	2'b0	Reserved
29:24	reserved	6'b0	Reserved
31:30	reserved	2'b0	Reserved
37:32	rx_rd_thrsh	6'b0	Receive read threshold. This field sets the number of entries that must be in the receive FIFO for it to indicate data is available to be read. For DMA operation this field must be set to 4 entries. See <a href="#">Section: "Receiver Configuration" on page 274</a>
39:38	reserved	2'b0	Reserved
45:40	rx_rl_thrsh	6'b0	Receive release threshold. This field sets the number of FIFO entries that must be written at the start of a packet before any of the data is made available for reading. The data is also made available if the packet ends before the threshold is reached. See <a href="#">Section: "Receiver Configuration" on page 274</a> . This field must be greater than 2. If it is zero the MAC receive behavior will be UNPREDICTABLE.
55:46	reserved	10'b0	Reserved
61:56	enc_fc_thrsh	6'h4	In encoded Packet Fifo modes link level flow control will be requested when the number of free doublewords in the receive fifo falls below this threshold.
63:62	reserved	2'b0	Reserved



**Table 180: MAC Frame Configuration Registers**

mac_frame_cfg_0 - 00_1006_4118 mac_frame_cfg_1 - 00_1006_5118 mac_frame_cfg_2 - 00_1006_6118 <b>This register is only used in both Ethernet mode no fields apply for Packet FIFO modes</b>			
Bits	Name	Default	Description
5:0	ifg_rx	6'b0	<p>System Revision &lt; PERIPH_REV3: The ifg_rx is not used in the current implementation. The ifg_tx value is used to set both receive to transmit (on half-duplex links) and transmit to transmit gap.</p> <p>System Revision PERIPH_REV3 and later: If bit [5] is clear a standard 7 byte preamble is transmitted. If bit [5] is set bits [3:0] set the number of bytes of preamble that will be transmitted ([3:0]=0 will result in one byte of preamble in gigabit mode, no preamble in 10/100 mode). The single SFD byte is always sent after the preamble.</p>
11:6	ifg_tx	6'b0	<p>ifg_tx determines inter-frame gap in interface cycles that the MAC uses when doing back to back transmissions (or between a reception and transmission in a half-duplex link). When the MAC has completed a packet the ifg counter loaded with the ifg_tx value and counts down. When the ifg counter reaches zero and four more cycles have passed the inter-frame gap time has elapsed and the MAC is free to transmit. If the value written in this register is less than 4 then the gap will be 7 cycles, otherwise the gap will be ifg_tx+4 cycles (See Figure 39). A value of zero gives 68 cycles.</p> <p>For Gigabit operation this is in bytes, and for 10/100 in nibbles. Including the offset of 4 for a standard 12 byte gap this field should be set to 8 for Gigabit operation and 20 for 10/100.</p> <p>This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE.</p>
17:12	ifg_thrsh	6'b0	<p>This sets the threshold time in interface cycles for checking carrier sense during the inter-frame gap. At the start of the IFG the MAC will monitor the carrier sense and defer if a carrier is detected, the IEEE standard suggests this should last 2/3 of the gap. During the remainder of the IFG the MAC ignores the carrier sense and transmission (potentially with a collision) will begin as usual. The carrier is ignored for the final ifg_thrsh+4 cycles of the IFG.</p> <p>To use the standard 2/3rd:1/3rd split for Gigabit operation the MAC should defer during the first 8 byte times, thus ifg_thrsh+4=4 and this field should be set to zero. For 10/100 operation the 8 byte times become 16 nibbles to defer, thus ifg_thrsh+4=8 and this field should be set to 4.</p> <p>This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE.</p>
21:18	backoff_sel	4'b0	<p>This field sets the maximum number of slot times the MAC will backoff following a collision. If a collision happens when the MAC is transmitting a packet it will abandon the transmission attempt, and backoff for a pseudo-random period before retrying. The length of the backoff is chosen randomly by a LFSR from a range from zero to a maximum value. The maximum backoff period depends on the number of times the packet has previously collided (to give an exponential backoff) and the value of this field. This field sets the maximum value the backoff count can grow to, regardless of the number of times the packet has collided.</p> <p>The encoding of the backoff time is:  backoff_sel = 0 - Backoff disabled, the backoff time will always be zero  1 &lt;= backoff_sel &lt;= 10 - Backoff limit is 0 - (2^n - 1)  11 &lt;= backoff_sel &lt;= 15 - Backoff limit is 0 - (2^10 - 1)</p> <p>This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE.</p> <p>This field should be set to 10 for standard operation.</p>

**Table 180: MAC Frame Configuration Registers (Cont.)**

mac_frame_cfg_0 - 00_1006_4118 mac_frame_cfg_1 - 00_1006_5118 mac_frame_cfg_2 - 00_1006_6118 This register is only used in both Ethernet mode no fields apply for Packet FIFO modes			
Bits	Name	Default	Description
29:22	lfsr_seed	8'b0	This field is the partial seed value for the LFSR that generates the random backoff. When the register is written this value is loaded into the 8 lowest bits of the LFSR (the upper 2 bits are loaded with the LFSR output). This value may be changed at any time.
39:30	slot_offset	10'b0	This field adjusts the slot size for the MAC to match for delays in the PHY. It is a signed value added to the base slot size. For 10/100 operation the slot size is (128 + slot_offset) nibble-clocks, giving the standard 512 bit times when the slot_offset is zero. For 1000 operation the slot size is (512 + slot_offset) byte-clocks, giving the standard 512 byte times when slot_offset is zero. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE.
47:40	min_framesz	8'b0	This field sets the minimum frame size in bytes that is used. The MAC will report a runt packet error for any packet that is received that is smaller than this number of bytes (after preamble and SFD have been stripped). For IEEE 802.3 compliance this value should be 64 for all speeds of operation. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE. This size must be greater than 9.
63:48	max_framesz	16'b0	This field sets the maximum frame size in bytes that is used. If a packet is received that is longer than this (after preamble and SFD have been stripped) the MAC will report an oversize packet error and discard excess bytes. For IEEE 802.3 compliance this value should be 1518 for all speeds of operation. Setting this size above 1518 allows use of "jumbo" packets. This value may only be changed while the protocol engine is held in reset, if it is changed while the engine is active the results are UNPREDICTABLE. This size must be greater than the min_framesz.





**Table 181: MAC VLAN Tag Registers**

mac_vlan_tag_0 - 00_1006_4110 mac_vlan_tag_1 - 00_1006_5110 mac_vlan_tag_2 - 00_1006_6110 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
31:0	tag	32'b0	VLAN tag. This 32 bit tag is inserted into the packet after the destination and source Ethernet addresses if the packet is marked for VLAN tag insertion. The tag thus becomes bytes 12-15 of the packet. The low byte in this register is the first one put on the wire. Note that for normal VLAN operation the low two bytes should be 16'h8100.  Used only in Ethernet mode.
39:32	tx_pkt_offset	8'b0	System Revision PERIPH_REV3 and later only. Sets the offset the Ethernet frame in transmitted packets. Bits 34:32 MUST be 3'b000.  Used in both Ethernet and Packet FIFO modes.
47:40	tx_crc_offset	8'b0	System Revision PERIPH_REV3 and later only. Sets the offset for CRC generation to begin in transmitted packets. Bits 42:40 MUST be 3'b000.  Used in both Ethernet and Packet FIFO modes.
48	ch_base_fc_en	1'b0	System Revision PERIPH_REV3 and later only. If this bit is clear Pause frame flow control is done in the MAC, if set the flow control is done by the DMA engine.  Used in both Ethernet and Packet FIFO modes.
63:49	notimp	32'b0	Not Implemented.

**Table 182: MAC Status Registers**

mac_status_0 - 00_1006_4408 mac_status_1 - 00_1006_5408 mac_status_2 - 00_1006_6408 <b>READ ONLY - Reading this register will clear all latched bits</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
0	rx_ch0_eop_count	1'b0	Set if the EOP interrupt was raised as a result of the packet count being reached.
1	rx_ch0_eop_timer	1'b0	Set if the EOP interrupt was raised as a result of the packet timer triggered.
2	rx_ch0_eop_seen	1'b0	Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).
3	rx_ch0_hwm	1'b0	This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched nor cleared by a read of the status register. It always reflects the state at the time the register is read. Therefore it is possible that the bit may become set and cause an interrupt but be cleared by an in-flight write that adds descriptors before the interrupt is taken.
4	rx_ch0_lwm	1'b0	This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).
5	rx_ch0_dscr	1'b0	Set if the interrupt is triggered by a descriptor with the interrupt bit set.

**Table 182: MAC Status Registers (Cont.)**

mac_status_0 - 00_1006_4408 mac_status_1 - 00_1006_5408 mac_status_2 - 00_1006_6408 <b>READ ONLY - Reading this register will clear all latched bits</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
6	rx_ch0_derr	1'b0	Set if any error is marked on data returned by a read. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
7	rx_ch0_drop	1'b0	Set if a packet was dropped because there were no descriptors available when the start of packet was received. (If there are descriptors available for the start of packet, but the controller runs out of descriptors during the packet reception then the dscr_err bit will be set in the receive status word for the packet.) The interface will continue dropping data until there are descriptors available when a start of packet is received.
8	rx_ch1_eop_count	1'b0	Set if the EOP interrupt was raised as a result of the packet count being reached.
9	rx_ch1_eop_timer	1'b0	Set if the EOP interrupt was raised as a result of the packet timer triggered.
10	rx_ch1_eop_seen	1'b0	Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).
11	rx_ch1_hwm	1'b0	This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched (see bit 3).
12	rx_ch1_lwm	1'b0	This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).
13	rx_ch1_dscr	1'b0	Set if the interrupt is triggered by a descriptor with the interrupt bit set.
14	rx_ch1_derr	1'b0	Set if any error is marked on data returned by a read. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
15	rx_ch1_drop	1'b0	Set if a packet was dropped because there were no descriptors available when the start of packet was received. (If there are descriptors available for the start of packet, but the controller runs out of descriptors during the packet reception then the dscr_err bit will be set in the receive status word for the packet.) The interface will continue dropping data until there are descriptors available when a start of packet is received.
16	tx_ch0_eop_count	1'b0	Set if the EOP interrupt was raised as a result of the packet count being reached.
17	tx_ch0_eop_timer	1'b0	Set if the EOP interrupt was raised as a result of the packet timer triggered.
18	tx_ch0_eop_seen	1'b0	Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).
19	tx_ch0_hwm	1'b0	This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched (see bit 3).
20	tx_ch0_lwm	1'b0	This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).
21	tx_ch0_dscr	1'b0	Set if the interrupt is triggered by a descriptor with the interrupt bit set.
22	tx_ch0_derr	1'b0	Set if data marked with an error is returned for any read (descriptor or data buffer). The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
23	tx_ch0_dzero	1'b0	Set if a descriptor has a packet length of zero or the SOP flag bit is not set in the first descriptor of a packet. This bit is also set if the controller runs out of descriptors during a packet transmission. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
24	tx_ch1_eop_count	1'b0	Set if the EOP interrupt was raised as a result of the packet count being reached.
25	tx_ch1_eop_timer	1'b0	Set if the EOP interrupt was raised as a result of the packet timer triggered.



Table 182: MAC Status Registers (Cont.)

mac_status_0 - 00_1006_4408 mac_status_1 - 00_1006_5408 mac_status_2 - 00_1006_6408 <b>READ ONLY - Reading this register will clear all latched bits</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
26	tx_ch1_eop_seen	1'b0	Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).
27	tx_ch1_hwm	1'b0	This bit will be set if the current descriptor count is less than the high watermark. This bit is not latched (see bit 3).
28	tx_ch1_lwm	1'b0	This bit will be set if the current descriptor count is less than the low watermark. This bit is not latched (see bit 3).
29	tx_ch1_dscr	1'b0	Set if the interrupt is triggered by a descriptor with the interrupt bit set.
30	tx_ch1_derr	1'b0	Set if data marked with an error is returned for any read (descriptor or data buffer). The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
31	tx_ch1_dzero	1'b0	Set if a descriptor has a packet length of zero or the SOP flag bit is not set in the first descriptor of a packet. This bit is also set if the controller runs out of descriptors during a packet transmission. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
39:32	Reserved	8'b0	Reserved
40	rx_undrfl	1'b0	This bit is set by the receive FIFO underflowing. If this happens when DMA is being used then the rx_rd_thrsh is incorrectly set.
41	rx_ovrfl	1'b0	This bit is set by the receive FIFO overflowing. This happens if the DMA (or reader in direct mode) has delayed reading data from the FIFO so there is no space for the next received data.
42	tx_undrfl	1'b0	This bit is set if the transmit FIFO underflows. This happens if the DMA (or sender in direct mode) has delayed inserting data into the FIFO and it was empty when more data was needed to be transmitted.
43	tx_ovrfl	1'b0	This bit is set by the transmit FIFO overflowing. This happens if the DMA (or sender in direct mode) writes to the FIFO when it is full. This will only happen for DMA if the tx_wr_thrsh threshold is incorrectly set.
44	ltxcol_err	1'b0	This bit will be set when a packet experiences a late collision. A late collision is one that happens after the minimum packet size has been sent. Collisions can never happen in Packet FIFO modes.
45	excol_err	1'b0	This bit is set if a packet experiences excessive collisions. It is signalled when attempted transmission of the packet collides 16 times. It normally indicates a hardware problem, but can also be caused by a very busy network or in the unlucky case of the random backoff LFSR being synchronized with another host (normally this will be fixed by the different length of time the two devices take to respond and clear the excessive collision error, but it is also possible to set a different lfsr_seed in the mac_cfg register). Collisions can never happen in Packet FIFO modes.
46	cntr_ovrfl_err	1'b0	This bit is set if any of the RMON counters overflow. Whenever a counter overflows this bit is set and the counter number is written into the counter_addr field. The RMON counters are not used in Packet FIFO modes.
51:47	counter_addr	5'b0	This is the number of the most recent RMON counter that overflowed. It is only valid if the cntr_ovrfl_err bit is set. The RMON counters are not used in Packet FIFO modes.

**Table 182: MAC Status Registers (Cont.)**

mac_status_0 - 00_1006_4408 mac_status_1 - 00_1006_5408 mac_status_2 - 00_1006_6408 <b>READ ONLY - Reading this register will clear all latched bits</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
52	tx_pause_on	1'b0	System revision PERIPH_REV3 or greater. This bit is set when the MAC has received a pause frame and the pause time indicated has not expired. It is cleared when the pause time expires (or if a pause frame with zero time is received). No interrupt can be generated from this bit. Pause frames are not used in Packet FIFO modes.
63:53	reserved	11'b0	Reserved

**Table 183: MAC Status 1 Register**

mac_status1_0 - 00_1006_4430 mac_status1_1 - 00_1006_5430 mac_status1_2 - 00_1006_6430 <b>READ ONLY</b> <b>Reading this register will clear the channel 1 latched bits</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
63:0	status	64'h0	Reading this register gives the same result as reading the status register, except that only the latched bits associated with channel 1 transmit and receive DMA are cleared. It is for use in split interrupt mode.

**Table 184: MAC Debug Status Registers**

mac_debug_status_0 - 00_1006_4448 mac_debug_status_1 - 00_1006_5448 mac_debug_status_2 - 00_1006_6448 <b>READ ONLY</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
63:0	status	64'b0	Reading this register gives the same result as reading the status register, except the bits are not cleared and none of the other side effects happen. It is intended for debug accesses to the status.



**Table 185: MAC Interrupt Mask Registers**

mac_int_mask_0 - 00_1006_4410 mac_int_mask_1 - 00_1006_5410 mac_int_mask_2 - 00_1006_6410 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
46:0	mask	47'b0	Setting a bit in this register enables generation of an interrupt when the corresponding bit is set in the mac_status register.
47	split_en	1'b0	When this bit is set the channel 1 interrupts will be signalled on the channel 1 interrupt instead of standard interrupt, and reads to the <b>mac_status</b> register will not clear channel 1 state. When this bit is clear all interrupts are signalled on the standard interrupt and reading the <b>mac_status</b> register clears all state.
63:48	reserved	19'b0	Reserved

**Table 186: MAC FIFO Pointer Registers**

mac_rx_fifo_ptrs_0 - 00_1006_4120 mac_rx_fifo_ptrs_1 - 00_1006_5120 mac_rx_fifo_ptrs_2 - 00_1006_6120 mac_tx_fifo_ptrs_0 - 00_1006_4128 mac_tx_fifo_ptrs_1 - 00_1006_5128 mac_tx_fifo_ptrs_2 - 00_1006_6128 <b>READ ONLY - Broadcom Use Only</b> <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
63:0	status	64'hx	Status for fifo and counts (Broadcom Use Only)

**Table 187: MAC Receive Address Filter Exact Match Registers**

mac_addr0_0 - 00_1006_4280    mac_addr0_1 - 00_1006_5280    mac_addr0_2 - 00_1006_6280 mac_addr1_0 - 00_1006_4288    mac_addr1_1 - 00_1006_5288    mac_addr1_2 - 00_1006_6288 mac_addr2_0 - 00_1006_4290    mac_addr2_1 - 00_1006_5290    mac_addr2_2 - 00_1006_6290 mac_addr3_0 - 00_1006_4298    mac_addr3_1 - 00_1006_5298    mac_addr3_2 - 00_1006_6298 mac_addr4_0 - 00_1006_42A0    mac_addr4_1 - 00_1006_52A0    mac_addr4_2 - 00_1006_62A0 mac_addr5_0 - 00_1006_42A8    mac_addr5_1 - 00_1006_52A8    mac_addr5_2 - 00_1006_62A8 mac_addr6_0 - 00_1006_42B0    mac_addr6_1 - 00_1006_52B0    mac_addr6_2 - 00_1006_62B0 mac_addr7_0 - 00_1006_42B8    mac_addr7_1 - 00_1006_52B8    mac_addr7_2 - 00_1006_62B8 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
47:0	address	48'bx	The destination address to be exactly matched as part of input packet filtering. The incoming address is always compared to all entries. The low byte in this register corresponds with the first byte of the address to be received.
63:48	zero	16'b0	Reads as zero, writes ignored.

**Table 188: MAC Receive Address Filter Mask Registers (Only if System Revision >= PERIPH\_REV3)**

<b>mac_admask0_0 - 00_1006_4218 mac_admask0_1 - 00_1006_5218 mac_admask0_2 - 00_1006_6218</b> <b>mac_admask1_0 - 00_1006_4220 mac_admask1_1 - 00_1006_5220 mac_admask1_2 - 00_1006_6220</b> This register is used in both Ethernet and Packet FIFO modes			
Bits	Name	Default	Description
47:0	mask	48'b FFFF_ FFFF_ FFFF	The mask qualifies the destination address set in the corresponding mac_addr register. When the incoming address is compared to the address any bits with a mask of zero are excluded from the comparison. The low byte in this register corresponds with the first byte of the address to be received.
63:48	zero	16'b0	Reads as zero, writes ignored.

**Table 189: MAC Receive Address Filter Hash Match Registers**

<b>mac_hash0_0 - 00_1006_4240 mac_hash0_1 - 00_1006_5240 mac_hash0_2 - 00_1006_6240</b> <b>mac_hash1_0 - 00_1006_4248 mac_hash1_1 - 00_1006_5248 mac_hash1_2 - 00_1006_6248</b> <b>mac_hash2_0 - 00_1006_4250 mac_hash2_1 - 00_1006_5250 mac_hash2_2 - 00_1006_6250</b> <b>mac_hash3_0 - 00_1006_4258 mac_hash3_1 - 00_1006_5258 mac_hash3_2 - 00_1006_6258</b> <b>mac_hash4_0 - 00_1006_4260 mac_hash4_1 - 00_1006_5260 mac_hash4_2 - 00_1006_6260</b> <b>mac_hash5_0 - 00_1006_4268 mac_hash5_1 - 00_1006_5268 mac_hash5_2 - 00_1006_6268</b> <b>mac_hash6_0 - 00_1006_4270 mac_hash6_1 - 00_1006_5270 mac_hash6_2 - 00_1006_6270</b> <b>mac_hash7_0 - 00_1006_4278 mac_hash7_1 - 00_1006_5278 mac_hash7_2 - 00_1006_6278</b> This register is used in both Ethernet and Packet FIFO modes			
Bits	Name	Default	Description
63:0	map	64'bx	The hash map bits for a hash based match as part of input packet filtering. Set the bit to zero to not match that hash value, and to one to match on the hash value.

**Table 190: MAC Transmit Source Address Registers**

<b>mac_ethernet_addr_0 - 00_1006_4208</b> <b>mac_ethernet_addr_1 - 00_1006_5208</b> <b>mac_ethernet_addr_2 - 00_1006_6208</b> This register is only used Ethernet mode			
Bits	Name	Default	Description
47:0	address	48'b0	The source address to be put in outgoing packets. The low byte of this register will be the first byte of the address transmitted. Pause Frames will be accepted on the address configured in this register (in addition to the well known multicast address). The address in this register will be used as the source address when the MAC sends a Pause Frame.
63:48	zero	16'b0	Reads as zero, writes ignored.



**Table 191: MAC Packet Type Configuration Registers**

mac_type_cfg_0 - 00_1006_4210 mac_type_cfg_1 - 00_1006_5210 mac_type_cfg_2 - 00_1006_6210 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
15:0	type0	16'b0	If a received packet has this packet type the receive descriptor will be written with type = 3'b100. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.
31:16	type1	16'b0	If a received packet has this packet type the receive descriptor will be written with type = 3'b101. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.
47:32	type2	16'b0	If a received packet has this packet type the receive descriptor will be written with type = 3'b110. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.
63:48	type3	16'b0	If a received packet has this packet type the receive descriptor will be written with type = 3'b111. The high byte is compared with the first byte received in the type field, the low byte with the second type byte.

**Table 192: MAC Receive Address Filter Control Registers**

mac_adfilter_cfg_0 - 00_1006_4200 mac_adfilter_cfg_1 - 00_1006_5200 mac_adfilter_cfg_2 - 00_1006_6200 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
0	allpkt_en	1'b0	When this bit is set all packets will be accepted (promiscuous mode). Used in both Ethernet and Packet FIFO modes.
1	ucast_en	1'b0	When this bit is set unicast packets are checked in the hash map. Used in both Ethernet and Packet FIFO modes.
2	ucast_inv	1'b0	When this bit is set unicast packets that miss in the hash map will be accepted. If it is clear unicast packets must hit to be accepted. Used in both Ethernet and Packet FIFO modes.
3	mcast_en	1'b0	When this bit is set multicast packets are checked in the hash map. Used in both Ethernet and Packet FIFO modes.
4	mcast_inv	1'b0	When this bit is set multicast packets that miss in the hash map will be accepted. If it is clear multicast packets must hit to be accepted. Used in both Ethernet and Packet FIFO modes.
5	bcast_en	1'b0	When this bit is set all broadcast packets are accepted. When it is clear all broadcast packets are dropped. Used in both Ethernet and Packet FIFO modes.
6	direct_inv	1'b0	When this bit is set packets will be accepted if they do not match any of the entries in the exact address match registers. When this bit is clear packets will be accepted if they match any entry in the direct address match registers. Used in both Ethernet and Packet FIFO modes.

**Table 192: MAC Receive Address Filter Control Registers (Cont.)**

mac_adfilter_cfg_0 - 00_1006_4200 mac_adfilter_cfg_1 - 00_1006_5200 mac_adfilter_cfg_2 - 00_1006_6200 This register is used in both Ethernet and Packet FIFO modes			
Bits	Name	Default	Description
7	allm_en	1'b0	When this bit is set all multicast packets are accepted. When it is clear multicast packets will only be accepted if they match in the Exact or Hash filters.  Used in both Ethernet and Packet FIFO modes.
15:8	iphdr_offset	8'b0	Byte in the packet (counting from 1) of the IP header for checking the IPv4 header checksum. This is normally 15, but may be different if encapsulation headers are prepended. This offset is also used by the TCP checksum checking.  Used only in Ethernet mode.
23:16	rx_crc_offset	8'b0	System Revision PERIPH_REV3 and later only. Sets the offset for CRC checking to begin in received packets. Bits 18:16 MUST be 3'b000.  Used in both Ethernet and Packet FIFO modes.
31:24	rx_pkt_offset	8'b0	System Revision PERIPH_REV3 and later only. Sets the offset for the Ethernet frame in received packets. Bits 26:24 MUST be 3'b000.  Used in both Ethernet and Packet FIFO modes.
32	fwdpause_en	1'b0	System Revision PERIPH_REV3 and later only. If this bit is set then Pause Frames will not be processed by the hardware and will be accepted and passed to the DMA engine as though they had passed the address filter. This allows software to do pause frame flow control (for example to only disable the best-effort tx DMA channel when a pause is requested, but allow the priority traffic to continue).  Used only in Ethernet mode.
33	vlan_det_en	1'b0	System Revision PERIPH_REV3 and later only. If this bit is set then 4 is added to the ip_hdr_offset if the type field in the MAC header (i.e. at mac_hdr_off +12 and +13) is the VLAN type 81-00. This allows the correct IP header offset to be used for a network with a mix of VLAN and non-VLAN packets. In addition when the 81-00 type is detected the vlan_pkt bit (written to bit 1 of the dscr_b "options" field i.e. just above the bad_tcpcls bit) will be set and the pkt_type field in the status will come from decoding the two bytes following the VLAN tag (i.e. the packet type from the untagged packet).  Used only in Ethernet mode.
41:34	rx_ch_msn_sel	8'b0	System Revision PERIPH_REV3 and later only. This field specifies the offset of the upper four bits of the channel selection index when the split_ch_sel bit is set in the mac_cfg register. Note that if rx_ch_msn_sel <= {rx_ch_sel_msb,rx_ch_sel} the channel selected is UNPREDICTABLE.  Used in both Ethernet and Packet FIFO modes.
63:42	reserved	22'b0	Reserved





**Table 193: MAC Receive Channel Select Map Registers**

<b>mac_chlo0_0</b> - 00_1006_4300 <b>mac_chlo0_1</b> - 00_1006_5300 <b>mac_chlo0_2</b> - 00_1006_6300 <b>mac_chlo1_0</b> - 00_1006_4308 <b>mac_chlo1_1</b> - 00_1006_5308 <b>mac_chlo1_2</b> - 00_1006_6308 <b>mac_chlo2_0</b> - 00_1006_4310 <b>mac_chlo2_1</b> - 00_1006_5310 <b>mac_chlo2_2</b> - 00_1006_6310 <b>mac_chlo3_0</b> - 00_1006_4318 <b>mac_chlo3_1</b> - 00_1006_5318 <b>mac_chlo3_2</b> - 00_1006_6318 <b>mac_chup0_0</b> - 00_1006_4320 <b>mac_chup0_1</b> - 00_1006_5320 <b>mac_chup0_2</b> - 00_1006_6320 <b>mac_chup1_0</b> - 00_1006_4328 <b>mac_chup1_1</b> - 00_1006_5328 <b>mac_chup1_2</b> - 00_1006_6328 <b>mac_chup2_0</b> - 00_1006_4330 <b>mac_chup2_1</b> - 00_1006_5330 <b>mac_chup2_2</b> - 00_1006_6330 <b>mac_chup3_0</b> - 00_1006_4338 <b>mac_chup3_1</b> - 00_1006_5338 <b>mac_chup3_2</b> - 00_1006_6338 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
63:0	map	64'b0	These registers form the table for mapping the 8 bits from a received packet into a two bit channel number. The 256 entry table has bit 0 register 0 as entry 0, and bit 63 register 3 as entry 255. The chup register provides the msb and the chlo register the lsb of the channel number.

**Table 194: MAC MII Management Interface Registers**

<b>mac_mdio_0</b> - 00_1006_4428 <b>mac_mdio_1</b> - 00_1006_5428 <b>mac_mdio_2</b> - 00_1006_6428 <b>This register is used in both Ethernet and Packet FIFO modes</b>			
Bits	Name	Default	Description
0	mdc	1'b0	MII Management interface clock, this bit is copied to the MDC pin.
1	mdio_dir	1'b0	MII Management Data Direction, when set the mdio pin is an input, when clear the mdio pin is driven from the mdio_out bit. This bit must be clear to allow receive flow control to be output when the interface is configured for encoded packet fifo mode operation.
2	mdio_out	1'b0	MII Management interface data out, this bit is copied to the MDIO pin if the mdio_dir bit is 0.
3	genc	1'b0	General output, this bit is copied to the GENO pin.
4	mdio_in	1'b0	MII Management interface data input, this read only bit gives the value on the mdio pin.
63:5	reserved	59'b0	Reserved

---

This Page is left blank for notes



---

This Page is left blank for notes



# Section 10: Serial Interfaces

## INTRODUCTION

The part incorporates two identical serial ports that provide full-duplex interfaces to a variety of serial devices. Each port is separately configured, and can be run as an asynchronous serial link from 1200 baud to 5 Mbaud or a synchronous serial link at up to 55 Mbps.

In asynchronous mode the programming model is based on a DUART. Some registers share information between the A and B channels, but the two ports can be independently switched to the synchronous mode.

In synchronous mode the serial interface includes an internal HDLC engine and externally provides a PCM highway style link.

Selection between asynchronous and synchronous interfaces is normally made at reset time using the configuration input on IO\_AD[12] (channel A) and IO\_AD[14] (channel B). However, the CPU can write the corresponding bits in the **system\_cfg** register to change the selection. Modifying this bit will switch the output pins between the asynchronous drivers and the synchronous ones. If the interface is switched between modes software will need to re-initialize the interface and may need to reset the external device.

Each serial port has 8 pins associated with it. In addition if an interface is in synchronous mode it can use one of the GPIO pins as an output (software cannot change the use of this pin, it can only be set by the reset configuration). The following table shows their use in each mode, and for asynchronous mode the correspondence between the pins and input or output registers for channel A and channel B.

**Table 195: Serial Interface Signal Names**

<i>Pin Name</i>	<i>Direction</i>	<i>A</i>	<i>B</i>	<i>Asynchronous Mode</i>	<i>Synchronous Mode</i>
DOUT	Output			Transmit data output	Transmit data output
DIN	Input			Receive data input	Receive data input
RTS_TSTROBE	Output	op[0]	op[1]	RTS output or general output	Transmit strobe output
COUT	Output	op[2]	op[3]	General output (e.g. for DTR handshake) or baud rate clock output	Clock output
CTS_TCLKIN	Input	ip[0]	ip[1]	CTS input, or general input with transition detector	Transmit clock input
CIN_RCLKIN	Input	ip[2]	ip[3]	General input with transition detector (e.g. for DSR handshake)	Receive clock input
TIN	Input	ip[4]	ip[5]	General input (e.g. for DCD)	Transmit enable or synchronization input
RIN	Input	ip[6]	ip[7]	General input (e.g. for RI)	Receive enable or synchronization input
RSTROBE (shared with GPIO)	Output				Receive strobe output (if this is not enabled the pins are used for GPIO[0] or GPIO[1])



## ASYNCHRONOUS MODE

The asynchronous interface is provided using a DUART. The two channels are separately programmable and each has its own baud rate generator. Each channel has a 16 byte transmit FIFO and a 16 byte receive FIFO. In addition to the data path there are 4 inputs and 2 outputs per channel that can either be used for flow control or are available for general use. The inputs are readable through the input port register (IPR), and the outputs are set through the output port register (OPR). Even numbered I/O lines are associated with channel A, and odd numbered lines with channel B. The UART supports RTS/CTS flow control in hardware, other control must be done in software. When serial port 0 is set in asynchronous mode it is driven from channel A of the DUART, serial port 1 in asynchronous mode is driven from channel B.

## BAUD RATE GENERATORS

The baud rate is generated on chip by dividing down from the 100MHz reference clock. Each channel can have a different baud rate, but for a channel the transmit and receive rates must be the same. The baud rate is selected by setting the **duart\_clk\_sel** register to

$$(100 \text{ MHz}/(\text{baud\_rate} * 20)) - 1$$

Some popular baud rates are shown in the table below.

**Table 196: Baud Rate Counter Values**

Baud Rate	Count	Actual	% Error
1200	4095	1220.703	1.72526
2400	2082	2400.384	0.016003
4800	1040	4803.074	0.064041
9600	519	9615.385	0.160256
19200	259	19230.77	0.160256
38400	129	38461.54	0.160256
57600	85	58139.53	0.936693
115200	42	116279.1	0.936693
230400	21	227272.7	-1.35732
500000	9	500000	0
1000000	4	1000000	0

The baud clock can be output on the Cout pin by setting the appropriate bit in the output port configuration register **duart\_opcr**.

The baud clock is also used for the synchronous serial interface when it is configured to use an internal clock. In this case Cout should be configured to pass the clock signal to the external devices.

## OPERATION

The two channels of the DUART are identical in terms of configuration and operation. They are controlled by a set of memory mapped registers, in most cases separate registers are provided for channel A and channel B (the register names are identical with either **\_a** or **\_b** indicating the channel); in a few cases the two channels are combined.

Prior to operation a channel must be configured. The **duart\_mode\_reg\_1** is used to set the number of data bits in a character (either 7 or 8) and whether parity should be added. The parity bit may be set to give either an even or odd number of ones, or to be fixed as a high (mark) or low (space). The number of stop bits are set in the **duart\_mode\_reg\_2**. These registers are also used to enable hardware RTS/CTS handshake separately in each direction. The baud rate should be set in the **duart\_clk\_sel** register. The mode and clock registers should only be written while the interface is inactive, changing the parameters while the UART is active results in UNPREDICTABLE behavior. Once configured the transmitter and receiver are enabled by writing the **duart\_cmd** register.

The transmitter indicates that it is able to accept data into the transmit FIFO by setting the **duart\_tx\_rdy** bit in the **duart\_status** register. For as long as this bit is set, the CPU can write characters to the **duart\_tx\_hold** register and they will be inserted in the transmit FIFO. If the transmitter is enabled characters are extracted from the FIFO, have a (low) start bit prepended, parity and (high) stop bits appended and are serialized (least significant bit first) and sent to the DOUT pin.

The transmitter may be disabled or reset by issuing a command to the **duart\_cmd** register. Disabling transmission will cause transmission to stop and the line idle (high) when transmission of the current character is complete, any characters in the FIFO will remain there to be sent when the transmitter is re-enabled. The break command is similar to disable except when transmission of the current character (including stop bits) is complete the line is driven low. The break condition is cleared with the stop-break command, which removes the break synchronously with the transmit bit clock. Resetting the transmitter will cause the current transmission to be aborted and all characters in the FIFO to be discarded, all transmitter state is cleared and it is left in the disabled state. Until the FIFO becomes full characters can continue to be inserted into the FIFO even if the transmitter is disabled (if the disabled state was reached through the reset command any characters in the FIFO when the reset was issued are lost, after this characters can be inserted into the now empty FIFO).

The receiver will detect the (low) start bit on the DIN line, and use it to synchronize the local bit clock. Data bits are then sampled in the middle of the bit time until the data and parity bits have been received. One additional bit is sampled, and an error reported if it is not the expected (high) stop bit. Once a character has been received it is put in to the FIFO along with the frame and parity error flags. If the FIFO becomes full the character is discarded and the first character to be received that can be put in to the FIFO will have the overrun status flag set (thus the overrun flag indicates that characters were lost before the one that is marked).

There are two error conditions may result if the stop bit was not detected. If the data received is all zeros and a zero is seen in place of the stop bit then the break condition is detected. A zero character is put in the FIFO along with the break-detected flag, the receiver will not resume operation until the break condition has been removed by the line being high for two bit times. If the data received is not all zeros but a zero is seen in place of the stop bit then a frame error is reported, the input data continues to be sampled for half a bit time. During this period, if the input is sampled high it is considered as a late stop bit and the line is monitored for the next start bit. If the input remains low for the full half bit time then it is considered to be an early start bit and the receiver will synchronize at the half bit time point and start assembly of the next character.



When there is a character in the receive FIFO the `duart_rx_rdy` bit in the `duart_status` register is set, and the CPU can read it from the `duart_rx_hold` register. Reading the `duart_rx_hold` register removes the character from the FIFO. The top four bits of the status register reflect the flags (overrun error, parity error, frame error and break) associated with the character (since reading the character pops it out of the FIFO the flags should be read before the character). If the `duart_rx_hold` register is read at a time when the `duart_rx_rdy` bit is not set the result is UNPREDICTABLE.

The receiver may be disabled or reset by issuing a command to the `duart_cmd` register. Both of these have immediate effect, any character in the process of being received is discarded. Reset will also discard all data in the FIFO, disabling the receiver will retain it and reading from the FIFO can continue. In both cases the receiver must be enabled by writing the command register.

Handshaking using the RTS/CTS protocol is implemented in hardware. It is enabled separately for the transmitter (in `duart_mode_reg_2`) and the receiver (in `duart_mode_reg_1`). When enabled the transmitter will monitor the state of the CTS\_TCLKIN pin, and will only start sending a character if the signal is low. Once transmission of a character has started it will continue until the character has been sent regardless of the state of CTS\_TCLKIN. When enabled the RTS\_TSTROBE pin is driven by the receiver. It will be set low if the receiver is enabled and is able to receive a character. On receiving a start bit that will cause the receiver to become full the RTS\_TSTROBE line will be deasserted (set high), to indicate that no further characters should be sent.

If hardware handshaking is disabled the RTS\_TSTROBE output is controlled from software, and will reflect the inverse of the `op[0]` bit for channel A or the `op[1]` bit for channel B. The `op` bits can be set by writing to the `duart_set_opr` register and cleared by writing to the `duart_clear_opr` register. For convenience of systems implementing software flow control the `op[0]` and `op[1]` bits (that control RTS\_TSTROBE) can also be set and cleared through the `duart_out_port` register. The CTS\_TCLKIN pin can always be read through the `duart_in_port` register.

The baud rate clock that the transmitter is using can be output on the COUT pin, alternatively the pin can be driven from software as the inverse of `op[2]` or `op[3]`. This is set in the output port configuration register `duart_opcr`.

The CIN\_RCLKIN, RIN and TIN pins are provided in the `duart_in_port` register for software to use either as separate handshake lines or as general inputs. The CTS\_TCLKIN and CIN\_RCLKIN lines have transition detectors associated with them and the upper four bits of the `duart_inport_chng` register will latch the fact the line changed since the last time the register was read (the read automatically clears the bits).

# INTERRUPTS

The DUART interrupts are provided as system interrupts 8 (for channel A) and 9 (for channel B). The conditions that can cause an interrupt are signalled in the interrupt status register **duart\_isr**. This contains the channel A status in the lower four bits and the channel B status in the upper four bits. For convenience the information for each channel is available in the **duart\_isr\_a** and **duart\_isr\_b** registers where the status for the channel is always in the lower four bits and the upper bits are zeros. The conditions that can cause an interrupt are: transmitter ready, receiver ready or receiver FIFO full, break detected or removed and change detected on the handshake lines. Corresponding to each bit in the interrupt status register there is a bit in the interrupt mask register **duart\_imr**, if a status bit is set and the corresponding mask bit is also set then an interrupt will be raised. Again, for programming convenience, aliases of the lower and upper bits of the mask are provided in the lower four bits of the **duart\_imr\_a** and **duart\_imr\_b** registers (these are aliases - internally there is only a single copy of each mask bit).

Figure 66 shows the interrupt generation for a single channel.

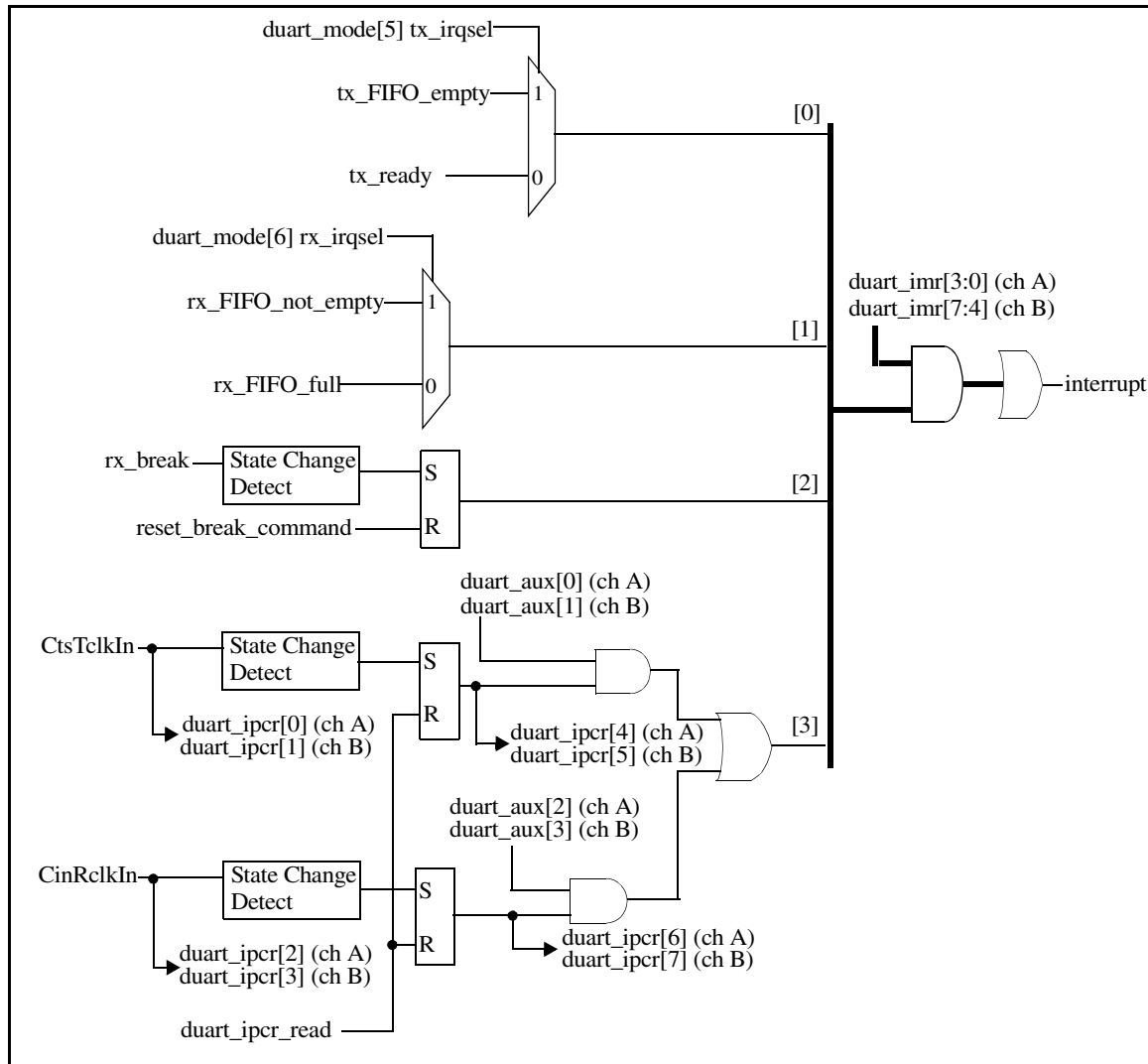


Figure 66: UART Interrupt Generation





The transmit interrupt can be generated either by there being at least one free entry in the fifo (i.e. when the `duart_tx_rdy` status bit is set) or when the fifo is empty. The `tx_irq_sel` bit in the `duart_mode_reg_1` is used to select between them. The `isr` status bit is cleared by the write of the `duart_rx_hold` register that removes the condition (causes the fifo to go full or not-empty depending on the mode).

The receive interrupt can be generated either by there being at least one character in the fifo (i.e. when the `duart_rx_rdy` status bit is set) or when the fifo is full (or almost full). The `rx_irq_sel` bit in the `duart_mode_reg_1` is used to select between them. The `isr` status bit is cleared by the read of the `duart_rx_hold` register that removes the condition (causes the fifo to go empty or not-full depending on the mode).

If the “FIFO full” interrupt source is selected the `duart_sig_full` field in the `duart_full_ctl` register can be used to set the number of characters that must be in the receive fifo for the interrupt to be raised. The interrupt is asserted when the fifo contains more than this number of characters. By default this field is 15, causing the interrupt when the fifo is completely full. With this setting software must respond to the interrupt and read from the fifo within one character time to avoid the fifo overflowing and characters being lost. The level at which the interrupt is raised can be lowered to provide more character times of interrupt latency.

If the `duart_int_time` field in the `duart_full_ctl` register is non-zero and the uart is set to interrupt on rx full (`duart_rx_irq_sel` bit =1) then an interrupt will be raised if the rx fifo is not empty and the link has been idle for `duart_int_time*16` bit times. If `duart_int_time` is zero then there is no timeout. This timeout ensures that characters received when the fifo is below threshold will not suffer long latencies before being serviced.

If either the threshold or timeout mechanism are used the receive interrupt status bit in the `duart_isr` register will be set based on the threshold or timeout. However the `duart_rx_ffull` bit in the `duart_status` register will continue to reflect the real full state of the fifo. The fifo will continue to accept characters until it is really full regardless of threshold and timeout settings.

The change in break interrupt status is set by the detection of the start of a break and by the detection of the end of a break. It will remain set until the CPU issued a reset break change command to the `duart_cmd` register.

The input line transition detector interrupt combines the state change information for both input lines of the channel. They are individually enabled in the `duart_aux_ctrl` registers. The interrupt status is cleared by reading the `duart_inport_chng` register.

## LOOPBACK

There are two loopback modes for diagnostics. They are enabled by setting the `duart_chan_mode` bits in the `duart_mode_reg_2`.

In local loopback mode the transmitter data output is internally connected to the receiver data input, so any character transmitted will be received. In this mode the DOUT pin is held idle (high) and the DIN pin is ignored. During local loopback the transmit module will not check the CTS\_TCLKIN pin and the receive module will not set the RTS\_TSTROBE pin even if their enable bits (`duart_tx_cts_ena` in `duart_mode_reg_2` and `duart_rx_rts_ena` in `duart_mode_reg_1`) are set. While the interface is in local loopback mode the receiver will be active even if it has not been enabled. The transmitter must still be enabled for characters to be sent.

In remote loopback mode data received on the DIN pin is re-clocked and transmitted on the DOUT, the received character is not put in to the FIFO and no error checking is done. In remote loopback mode received data is always sent out, the transmit module does not check the CTS\_TCLKIN pin even if `duart_tx_cts_ena` bit is set in the `duart_mode_reg_2`.

## DUART REGISTERS

**Table 197: DUART Mode Registers**

duart_mode_reg_1_a - 00_1006_0100 duart_mode_reg_1_b - 00_1006_0200			
Bits	Name	Default	Description
1:0	duart_bits_per_char	2'b00	0: 8 1: Reserved 2: 7 3: 8
2	duart_parity_type	1'b0	Parity type: 0: Even/Low 1: Odd/High
4:3	duart_parity_mode	2'b0	Parity mode. 0: Add Parity (even/odd selected by bit 2) 1: Add fixed parity bit (low/high selected by bit 2) 2: No Parity 3: No Parity
5	duart_tx_irq_sel	1'b0	Transmitter interrupt select. 0: Interrupt when the transmitter is ready (there is space in the fifo for at least one character). 1: Interrupt when the transmit fifo is empty.
6	duart_rx_irq_sel	1'b0	Receiver Interrupt select. 0: Interrupt on receiver ready (at least one character in the FIFO). 1: Interrupt on receiver FIFO full
7	duart_rx_rts_ena	1'b0	Receiver Request-to-Send Control enable. Channel A: If set the S0_RTS_TSTROBE pin will be used as hardware controlled RTS, if clear the pin will output the inverse of op[0]. Channel B: If set the S1_RTS_TSTROBE pin will be used as hardware controlled RTS, if clear the pin will output the inverse of op[1].
63:8	notimp	56'bx	Not implemented.

**Table 198: DUART Second Mode Registers**

duart_mode_reg_2_a - 00_1006_0110 duart_mode_reg_2_b - 00_1006_0210			
Bits	Name	Default	Description
2:0	ignored	3'b000	These bits are ignored.
3	duart_stop_bit_len	1'b0	Set for 2 Stop bits, clear for 1 stop bit
4	duart_tx_cts_ena	1'b0	Set to cause the transmitter to check CTS, clear to ignore CTS. Channel A CTS is on the S0_CTS_TCLKIN pin and is also readable as ip[0]. Channel B CTS is on the S1_CTS_TCLKIN pin and is also readable as ip[1].
5	reserved	1'b0	Always 0
7:6	duart_chan_mode	2'b00	Channel Mode. 0: Normal 1: Normal 2: Local loopback 3: Remote loopback
63:8	notimp	56'bx	Not implemented.

**Table 199: DUART Command Registers**

duart_cmd_a - 00_1006_0150 duart_cmd_b - 00_1006_0250			
Bits	Name	Default	Description
0	duart_rx_en	1'b0	Enable receiver. Setting this bit at the same time as duart_rx_dis results in no change in the receiver.
1	duart_rx_dis	1'b0	Disable receiver immediately (a character being received will be lost) Setting this bit at the same time as duart_rx_en results in no change in the receiver.
2	duart_tx_en	1'b0	Enable transmitter. Setting this bit at the same time as duart_tx_dis results in no change in the transmitter.
3	duart_tx_dis	1'b0	Disable transmitter when it next becomes idle (i.e. complete any in progress character) Setting this bit at the same time as duart_tx_en results in no change in the transmitter.
6:4	duart_misc_cmd	3'b000	0: No action. 1: No action. 2: Reset receiver. 3: Reset transmitter. 4: No action. 5: Reset channel's break-change interrupt. 6: Start break (drive data output line low when the transmitter becomes idle, and prevent further transmission). 7: Stop break (if currently sending a break, drive line high for at least one bit time and resume normal transmission).
7	reserved	1'b0	Reserved, write as zero.
63:8	notimp	56'bx	Not implemented.

**Table 200: DUART Status Registers**

duart_status_a - 00_1006_0120 duart_status_b - 00_1006_0220 READ ONLY			
Bits	Name	Default	Description
0	duart_rx_rdy	1'b0	Receiver ready: at least one character can be read. This bit is set whenever there are characters in the receive FIFO even if the receiver is disabled.
1	duart_rx_fful	1'b0	Receive FIFO full: if any further characters are received they will be discarded. This signal is asserted when the start bit is detected for the character that will cause the receiver to become full, if handshaking is disabled the CPU has one character-time to read data from the FIFO before characters are lost.
2	duart_tx_rdy	1'b1	Transmitter ready: there is room for at least one character to be written into the transmitter. This bit is set whenever there is space in the transmit FIFO even if the transmitter is disabled (characters inserted in the FIFO will be transmitted when the transmitter is next enabled).
3	duart_tx_empt	1'b1	Transmitter empty: there are no characters to send and the transmitter is idle
4	duart_ovrun_err	1'b0	Overrun error: This flag tags a received character to indicate that characters were lost prior to this one.
5	duart_parity_err	1'b0	Parity error: This flag tags a received character to indicate that this character was received with incorrect parity

**Table 200: DUART Status Registers (Cont.)**

duart_status_a - 00_1006_0120 duart_status_b - 00_1006_0220 READ ONLY			
Bits	Name	Default	Description
6	duart_frm_err	1'b0	Frame error: This flag tags a received character to indicate that this character (including parity bit) was non-zero and did not have a valid stop bit.
7	duart_rcvd_brk	1'b0	Received break: This flag tags a received character (which will be zero) to indicate that a the start of a break was detected.
63:8	notimp	56'bx	Not implemented.

**Table 201: DUART Baud Rate Clock Registers**

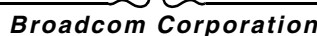
duart_clk_sel_a - 00_1006_0130 duart_clk_sel_b - 00_1006_0230			
Bits	Name	Default	Description
11:0	uart_clk_counter	12'h0	Baud rate counter. This register sets the baud rate for both the transmitter and receiver. Set to $(100 \text{ MHz}/(\text{baud rate} * 20)) - 1$ .
63:12	notimp	52'bx	Not implemented.

**Table 202: DUART Full Interrupt Control Registers**

duart_full_ctl_a - 00_1006_0140 duart_full_ctl_b - 00_1006_0240			
Bits	Name	Default	Description
3:0	duart_sig_full	4'hF	This field sets the threshold for the receive fifo full interrupt. The interrupt is raised when the number of characters in the fifo is greater than the value set. With the default of 15 the interrupt is only raised when the fifo is completely full.
7:4	duart_int_time	4'h0	If this field is non-zero then the fifo full interrupt is raised if there are any characters in the receive fifo and the receive data line has been idle for $\text{duart\_int\_time} * 16$ bit times. If this field is zero the interrupt is never raised by a timeout. The timeout interrupt is cleared by any read of the <b>rx_hold</b> register.
63:8	notimp	56'bx	Not Implemented.

**Table 203: DUART Received Data Registers**

duart_rx_hold_a - 00_1006_0160 duart_rx_hold_b - 00_1006_0260 READ ONLY, Read pops character from FIFO			
Bits	Name	Default	Description
7:0	rx_data	8'hx	Read Only. Received data. The character is only valid if the duart_rx_rdy bit is set in the status register prior to this register being read, when this register is read the next character is made available and its flag bits will be made available in the duart_status register.
63:8	notimp	56'bx	Not implemented.



**Table 204: DUART Transmit Data Registers**

duart_tx_hold_a - 00_1006_0170 duart_tx_hold_b - 00_1006_0270 WRITE ONLY			
Bits	Name	Default	Description
7:0	tx_data	W/O	Write only. Data to transmit. Writes will be ignored if the duart_tx_rdy bit is clear.
63:8	notimp	56'bx	Not implemented.

**Table 205: DUART Input Port Register**

duart_in_port - 00_1006_0380 READ ONLY			
Bits	Name	Default	Description
This register gives the current value of the input port pins.			
0	duart_in_pin0_val	ext	Input pin level (0= Low 1= High) ip[0] S0_CTS_TCLKIN.
1	duart_in_pin1_val	ext	Input pin level (0= Low 1= High) ip[1] S1_CTS_TCLKIN.
2	duart_in_pin2_val	ext	Input pin level (0= Low 1= High) ip[2] S0_CIN_RCLKIN.
3	duart_in_pin3_val	ext	Input pin level (0= Low 1= High) ip[3] S1_CIN_RCLKIN.
4	duart_in_pin4_val	ext	Input pin level (0= Low 1= High) ip[4] S0_TIN.
5	duart_in_pin5_val	ext	Input pin level (0= Low 1= High) ip[5] S1_TIN.
6	duart_rin0_pin	ext	Input pin level (0= Low 1= High) ip[6] S0_RIN.
7	duart_rin1_pin	ext	Input pin level (0= Low 1= High) ip[7] S1_RIN.
63:8	notimp	56'bx	Not implemented.

**Table 206: DUART Input Port Change Status Register**

duart_inport_chng - 00_1006_0300 READ ONLY, Read Clears All Change of State bits			
Bits	Name	Default	Description Port Pins
3:0	duart_in_pin_val	ext	Input pin level (0= Low 1= High) ip[3:0]. These bits match duart_in_port[3:0].
7:4	duart_in_pin_chng	4'b0	Input pin change of state ip[3:0] 0= No 1= Yes. These bits record whether any transitions were detected on the input pins since this register was last read. The bits are cleared when the register is read.
63:8	notimp	56'bx	Not implemented.

**Table 207: DUART Debug Access Input Port Change Register**

<b>duart_inport_chng_debug - 00_1006_03F0</b> <b>READ ONLY, Reads have no side effects</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
7:0	debug		These bits provide the same information as is in the <b>duart_inport_chng</b> register, but reads do not have side effects.
63:8	notimp	56b'x	Not Implemented.

**Table 208: DUART Input Port Change Status Register for Channel A**

<b>duart_inport_chng_a - 00_1006_03D0</b> <b>READ ONLY, Read Clears Channel A Change of State bits</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description Port Pins</b>
3:0	duart_in_pin_val	ext	Input pin level (0= Low 1= High) ip[3:0]. These bits match duart_in_port[3:0].
7:4	duart_in_pin_chng	4'b0	Input pin change of state ip[3:0] 0= No 1= Yes. These bits record whether any transitions were detected on the input pins since this register was last read. Bits 4 and 6 (the change of state bits for ip[0] and ip[2]) are cleared when the register is read.
63:8	notimp	56'bx	Not implemented.

**Table 209: DUART Input Port Change Status Register for Channel B**

<b>duart_inport_chng_b - 00_1006_03E0</b> <b>READ ONLY, Read Clears Channel B Change of State bits</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description Port Pins</b>
3:0	duart_in_pin_val	ext	Input pin level (0= Low 1= High) ip[3:0]. These bits match duart_in_port[3:0].
7:4	duart_in_pin_chng	4'b0	Input pin change of state ip[3:0] 0= No 1= Yes. These bits record whether any transitions were detected on the input pins since this register was last read. Bits 5 and 7 (the change of state bits for ip[1] and ip[3]) are cleared when the register is read.
63:8	notimp	56'bx	Not implemented.

**Table 210: DUART Output Port Control Register**

<b>duart_opcr - 00_1006_0370</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
0	reserved	1'h0	Not used, always zero.
1	opc2_sel	1'h0	Controls the S0_COOUT pin, when clear the pin is the complement of op[2], when set the pin outputs the 1X baud clock for the channel A transmitter.
2	reserved	1'h0	Not used, always zero.
3	opc3_sel	1'h0	Controls the S1_COOUT pin, when clear the pin is the complement of op[3], when set the pin outputs the 1X baud clock for the channel B transmitter.
7:4	reserved	4'h0	Not used (no support for output bits 7:4).
63:8	notimp	56'bx	Not implemented.



**Table 211: DUART Per Channel Output Control Registers**

<b>duart_opcr_a - 00_1006_0180</b> <b>duart_opcr_b - 00_1006_0280</b>			
Bits	Name	Default	Description
0	reserved	1'b0	Reserved
1	opc_sel	1'b0	Controls Cout pin for the channel. An alternative access path for the corresponding bit in the <b>duart_opcr</b> . Channel A: This controls the opc2_sel bit. Channel B: This controls the opc3_sel bit.
7:2	reserved	6'b0	Reserved
63:8	notimp	56b'x	Not Implemented.

**Table 212: DUART Aux Control Register**

<b>duart_aux_ctrl - 00_1006_0310</b>			
Bits	Name	Default	Description
0	duart_ip0_chng_ena	1'b0	ip[0] S0_CTS_TCLKIN change of state enable.
1	duart_ip1_chng_ena	1'b0	ip[1] S1_CTS_TCLKIN change of state enable.
2	duart_ip2_chng_ena	1'b0	ip[2] S0_CIN_RCLKIN change of state enable.
3	duart_ip3_chng_ena	1'b0	ip[3] S1_CIN_RCLKIN change of state enable.
7:4	reserved	4'h0	Unused, always zero.
63:8	notimp	56'bx	Not implemented.

**Table 213: DUART Per Channel Aux Control Registers**

<b>duart_aux_ctrl_a - 00_1006_0190</b> <b>duart_aux_ctrl_b - 00_1006_0290</b>			
Bits	Name	Default	Description
0	duart_cts_chng_ena	1'b0	CTS_TCLKIN change of state enable for the channel: Channel A controls duart_ip0_chng_ena Channel B controls duart_ip1_chng_ena
1	reserved	1'b0	Reserved
2	duart_cts_chng_ena	1'b0	CIN_RCLKIN change of state enable for the channel: Channel A controls duart_ip2_chng_ena Channel B controls duart_ip3_chng_ena
7:3	reserved	1'b0	Reserved
63:8	notimp	56b'x	Not Implemented.

**Table 214: DUART Interrupt Status Register**

<b>duart_isr - 00_1006_0390</b> <b>READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
0	duart_isr_tx_a	1'b1	Transmitter Ready
1	duart_isr_rx_a	1'b0	Receiver Ready / FIFO Full
2	duart_isr_brk_a	1'b0	Change in Break
3	duart_isr_in_a	1'b0	Input Port 0,2 changes Status (input port changes for channel A)
4	duart_isr_tx_b	1'b0	Transmitter Ready
5	duart_isr_rx_b	1'b0	Receiver Ready / FIFO Full
6	duart_isr_brk_b	1'b0	Change in Break
7	duart_isr_in_b	1'b0	Input Port 1,3 changes Status (input port changes for channel B)
63:8	notimp	56'bx	Not Implemented.

**Table 215: DUART Channel A Only Interrupt Status Register**

<b>duart_isr_a - 00_1006_0320</b> <b>READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Convenience register, only gives channel A ISR			
0	duart_isr_tx	1'b1	Transmitter Ready
1	duart_isr_rx	1'b0	Receiver Ready / FIFO Full
2	duart_isr_brk	1'b0	Change in Break
3	duart_isr_in	1'b0	Input Port 0 or 2 Changes Status
7:4	Reserved	4'h0	Reserved
63:8	notimp	56'bx	Not Implemented.

**Table 216: DUART Channel B Only Interrupt Status Register**

<b>duart_isr_b - 00_1006_0340</b> <b>READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Convenience register, only gives channel B ISR			
0	duart_isr_tx	1'b1	Transmitter Ready
1	duart_isr_rx	1'b0	Receiver Ready / FIFO Full
2	duart_isr_brk	1'b0	Change in Break
3	duart_isr_in	1'b0	Input Port 1 or 3 Changes Status
7:4	Reserved	4'h0	Reserved
63:8	notimp	56'bx	Not Implemented.





**Table 217: DUART Interrupt Mask Register**

<b>duart_imr - 00_1006_03A0</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Bits in this register must be set to cause an interrupt to be generated when the corresponding bit in the <b>duart_isr</b> is set.			
0	duart_imr_tx_a	1'b0	Mask Transmitter ready A
1	duart_imr_rx_a	1'b0	Mask Receiver Ready / FIFO Full A
2	duart_imr_brk_a	1'b0	Mask Change in break A
3	duart_imr_in_a	1'b0	Mask Input port 0,2 change interrupt (input port changes for channel A)
4	duart_imr_tx_b	1'b0	Mask Transmitter Ready B
5	duart_imr_rx_b	1'b0	Mask Receiver Ready / FIFO Full B
6	duart_imr_brk_b	1'b0	Mask Change in Break B
7	duart_imr_in_b	1'b0	Mask Input Port 1,3 Changes Status (input port changes for channel B)
63:8	notimp	56'bx	Not Implemented.

**Table 218: DUART Channel A Only Interrupt Mask Register**

<b>duart_imr_a - 00_1006_0330</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Convenience register, only writes channel A IMR bits			
0	duart_imr_tx	1'b0	Mask Transmitter Ready A
1	duart_imr_rx	1'b0	Mask Receiver Ready / FIFO Full A
2	duart_imr_brk	1'b0	Mask Change in Break A
3	duart_imr_in	1'b0	Mask Input Port 0,2 Changes Status
7:4	reserved	4'h0	Reserved
63:8	notimp	56'bx	Not Implemented.

**Table 219: DUART Channel B Only Interrupt Mask Register**

<b>duart_imr_b - 00_1006_0350</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Convenience register, only writes channel B IMR bits			
0	duart_imr_tx	1'b0	Mask Transmitter Ready
1	duart_imr_rx	1'b0	Mask Receiver Ready / FIFO Full
2	duart_imr_brk	1'b0	Mask Change in Break
3	duart_imr_in	1'b0	Mask Input Port 1,3 Changes Status
7:4	reserved	4'h0	Reserved
63:8	notimp	56'bx	Not Implemented.

**Table 220: DUART Output Port Set Register**

<b>duart_set_opr - 00_1006_03B0</b> <b>WRITE ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Note: the output pins are the inverse of the op			
0	duart_set_op[0]	W/O	op[0] 1= Set to High 0= No change
1	duart_set_op[1]	W/O	op[1] 1= Set to High 0= No change
2	duart_set_op[2]	W/O	op[2] 1= Set to High 0= No change
3	duart_set_op[3]	W/O	op[3] 1= Set to High 0= No change
7:4	reserved	W/O	Reserved (only 4 outputs)
63:8	notimp	56'bx	Not Implemented.

**Table 221: DUART Output Port Clear Register**

<b>duart_clear_opr - 00_1006_03C0</b> <b>WRITE ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Note: the output pins are the inverse of the op			
0	duart_clr_op[0]	W/O	op[0] 1= Clear to low 0= No change
1	duart_clr_op[1]	W/O	op[1] 1= Clear to low 0= No change
2	duart_clr_op[2]	W/O	op[2] 1= Clear to low 0= No change
3	duart_clr_op[3]	W/O	op[3] 1= Clear to low 0= No change
7:4	Reserved	W/O	Reserved (only 4 outputs)
63:8	notimp	56'bx	Not Implemented.

**Table 222: DUART Output Port RTS Register**

<b>duart_out_port - 00_1006_0360</b> <b>WRITE ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
Note: the output pins are the inverse of the op / Convenience register, for software RTS control			
0	duart_out_pin_set[0]	W/O	0: No change 1: Set output pin S0_RTS_TSTROBE to High (i.e. clear op[0])
1	duart_out_pin_set[1]	W/O	0: No change 1: Set output pin S1_RTS_TSTROBE to High (i.e. clear op[1])
2	duart_out_pin_clr[0]	W/O	0: No change 1: Clear output pin S0_RTS_TSTROBE to Low (i.e. set op[0])
3	duart_out_pin_clr[1]	W/O	0: No change 1: Clear output pin S1_RTS_TSTROBE to Low (i.e. set op[1])
7:4	Reserved	W/O	Reserved
63:8	notimp	56'bx	Not Implemented.



---

This Page is left blank for notes



## SYNCHRONOUS MODE

In synchronous mode, the serial data stream is accompanied by a clock and an optional gating or framing signal. There are two sub-modes, HDLC and transparent. In HDLC sub-mode, frames within the bit stream are recognized and processed according to ISO/IEC-3309 (High-level data link control (HDLC) procedures - Frame structure). In transparent sub-mode, serial data is transmitted and received without modification.

In both sub-modes, the serial stream can be qualified by a gating signal. Typically, such gating would select a subset of the bits at the physical interface for internal processing. The gating signal can be provided externally. Alternatively, it can be generated by a table-driven sequencer that is synchronized to an external framing pulse. Configuration options allow a choice of polarities and delays for the gating or synchronizing signals.

The synchronous serial ports can operate at speeds of 0 to 55 Mbp/s and thus can support up to T3, E3 and OC-1 data rates. In such applications, an external framer or similar interface would typically be used to connect to the serial line and would itself be controlled from the CPU via the generic bus interface (see [Section 11: "Generic/Boot Bus" on page 361](#)).

Each port is associated with a DMA channel for transmit and another for receive. In synchronous mode, the port and the pair of DMA channels are collectively called a serial channel.

The interface provides two identical and independent serial channels, 0 and 1. The registers and interrupts associated with each are differentiated by appending `_0` or `_1` to their names.

## FUNCTIONAL OVERVIEW

The serial channels are part of the I/O subsystem on the part and connect to the ZBbus through I/O bridge 1. [Figure 67 on page 337](#) shows a block diagram. Each consists of 3 major functional blocks:

- a pair of built-in DMA controllers with FIFOs
- a protocol engine that can operate in either HDLC or transparent mode
- a programmable line interface

For each serial channel, there are two built-in DMA controllers, one for transmit and one for receive. The DMA controllers interface with the I/O Bridge. They connect to the protocol engine via separate FIFOs for transmit (TxFIFO) and receive (RxFIFO).

Except for the interpretation of link-specific option and status bits, which are summarized below, the serial DMA channels function identically to those provided for the Ethernet MACs. Note that each serial DMA channel supports only a single chain or ring. For a full description of DMA configuration and programming, refer to [Section 7: "DMA" on page 147](#).

The line interface unit controls the signals connecting the part to external serial devices, which might range from codecs to simple line drivers to sophisticated external framers. The line interface is described in the next section.

The protocol engine provides the link-level processing. It maps between DMA's byte streams and the line interface's bit streams according to the selected protocol. [Section: "Framing Parameters" on page 344](#) and [Section: "HDLC Transmitter" on page 344](#) describe its operation for the HDLC and transparent protocols respectively.

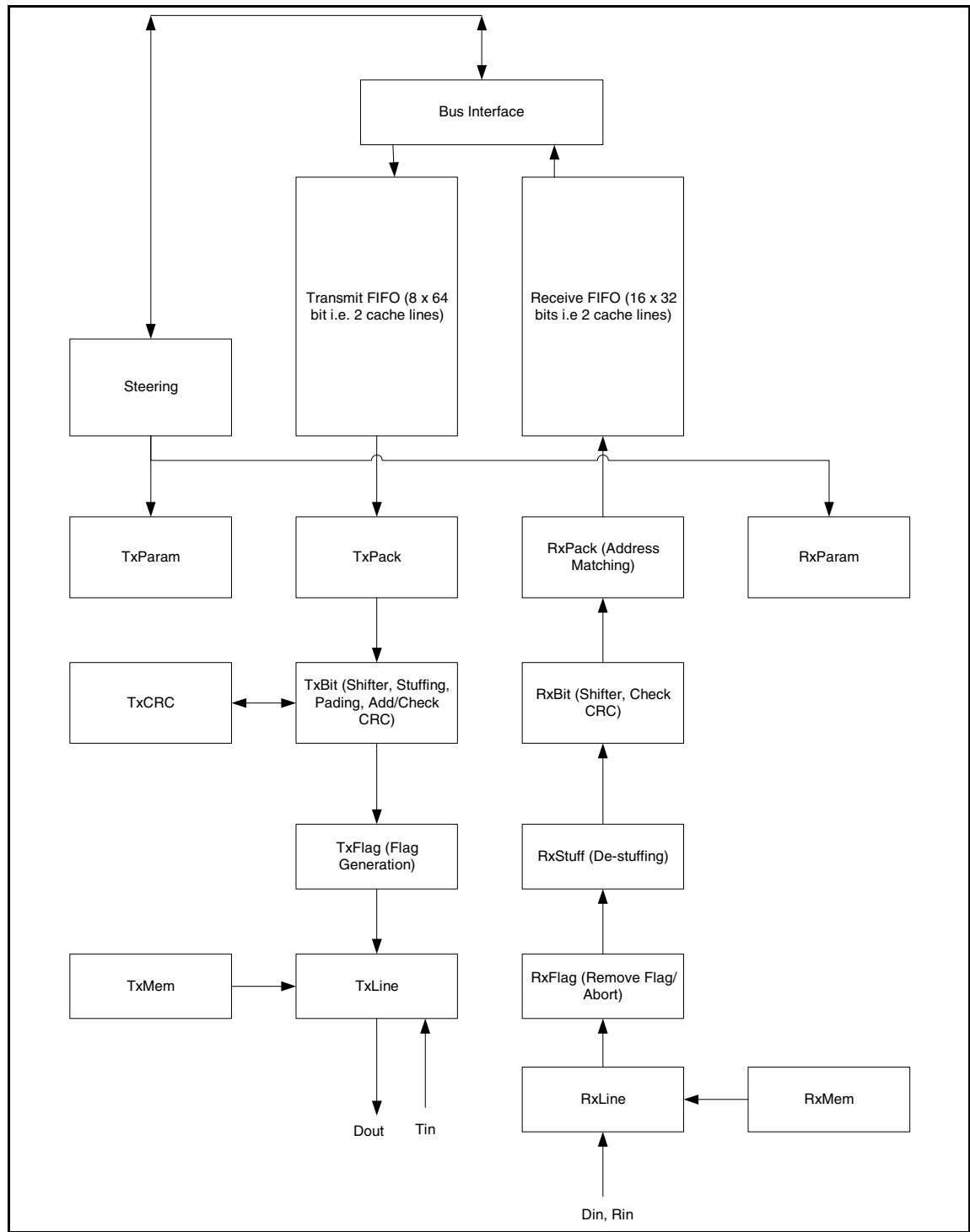


Figure 67: Synchronous Interface Block Diagram

## LINE INTERFACE

The Line Interface controls the flow of bits between the protocol engine and the external pins. In the receive direction it assembles a bit stream and clock for the protocol engine. In the transmit direction it fetches bits from the protocol engine and formats them for the line.

The serial channels share pins with the DUARTs. Each port can independently be configured for synchronous serial operation or for UART operation using reset time configuration resistors or by software (see [Section: "Reset" on page 26](#)). When port 0 is configured for synchronous mode, it is connected to channel 0; similarly, port 1 to channel 1.

The 8 pins for each interface are used as follows in synchronous mode.

**Table 223: Synchronous Serial Interface Signal Names**

<b>Pin</b>	<b>Direction</b>	<b>Synchronous serial port Function</b>
DOUT	Output	Transmit data output
CTS_TCLKIN	Input	Transmit clock input
TIN	Input	Transmit enable or frame synch input
RTS_TSTROBE	Output	Transmit strobe output
DIN	Input	Receive data input
CIN_RCLKIN	Input	Receive clock input
RIN	Input	Receive enable or frame synch input
COUT	Output	Baud rate generator output Note that the configuration must be set in the UART control block, in order for the baud clock to be output on this pin.

In addition, the following GPIO pins can be driven from the serial port when the appropriate reset time configuration resistors select this use (see [Section: "Reset" on page 26](#)).

**Table 224: Synchronous Serial Interface GPIO Pins**

<b>Pin</b>	<b>Direction</b>	<b>Channel 0</b>	<b>Channel 1</b>	<b>Function</b>
RSTROBE	Output	GPIO[0]	GPIO[1]	Receive Strobe Out



## INPUT LINE INTERFACE

In the receive direction, data on pin DIN is sampled either by an external clock signal, supplied on the CIN\_RCLKIN pin, or by the internal baud rate generator. The internal baud clock can be made available externally on the COUT pin by setting the appropriate bit in the **duart\_opcr** register (Table 210 on page 330) in the UART control block. Regardless of source, the polarity of the sampling clock edge is programmable. The clock configuration is done in the **ser\_clk** register.

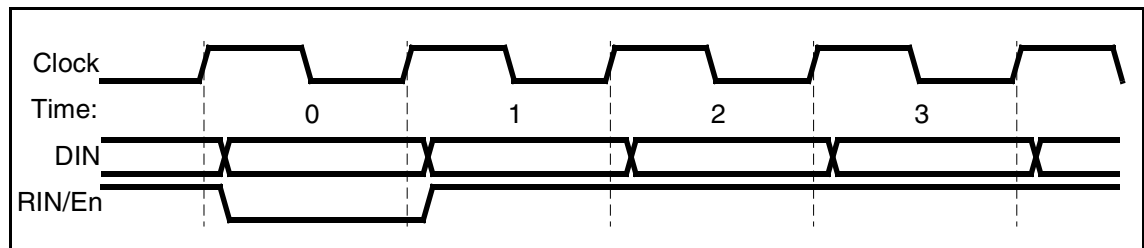
There are three ways that data bits are qualified to determine if they should be accepted and sent as part of the bit stream to the protocol engine.

- 1 **Gapped Clock:** If the external device is supplying the clock on CIN\_RCLKIN, it can omit clock pulses. Since the BCM1250 never receives a clock pulse this method can always be used to suppress the reception of bits.
- 2 **External Enable:** Regardless of clock source, the RIN pin can be supplied with an externally generated enable signal. This can qualify the current data bit or may be delayed by 1, 2 or 3 clocks.
- 3 **Internal Sequencer:** Regardless of clock source, but exclusive with (2), the enable signal can be generated by an internal sequencer, which is itself synchronized to the data stream by a pulse on RIN. The sequencer can also provide a strobe on the RSTROBE output.

The bit stream delivered to the protocol engine consists of the bits sampled during the enabled bit times; clock edges occurring during disabled bit times are suppressed and not seen by the protocol engine.

### Input Using an External Enable

An external enable signal can be provided on the RIN pin. This can be configured to be active high or low, and indicates valid data when active. The Data (on DIN) and enable (on RIN) are latched on the same edge of the clock. In the simplest case the enable qualifies the data bit that is latched at the same time, but the enable can also be delayed to qualify the data 1, 2 or 3 clocks later.



**Figure 68: Example Reception Using RIN as Active High Enable (sampling on the falling clock edge)**

The example in Figure 68 shows data and an active high enable being latched on the falling edge of the clock. The enable becomes inactive during cycle 0. If the enable delay is set to 0 then the data bit at time 0 will be ignored and the others will be accepted. If the enable delay is 3, the data bit at time 3 is skipped, while data bits at times 4, 5 and 6 are accepted (the example does not show the falling edges of cycles -3, -2, and -1 so it is not possible to determine from this figure if the data at bit times 0, 1, or 2 will be accepted).

The enable signal is level sensitive when used to enable the data. In transparent mode (see [Section: "Operation in Transparent Mode" on page 348](#)) the enable signal is used to frame the data, this can be selected as being edge (inactive to active) or level based, but only the edge based framing is likely to be useful.

RSTROBE is not used when the interface is used for an external enable. It is best to configure the pin as a GPIO. If the pin is configured as RSTROBE it will remain deasserted.

### Input Using the Internal Sequencer

An internal sequencer can be used to qualify received data bits. A user-configured table is used to generate the internal enable signal and an optional external strobe signal. An external framing pulse provided on the RIN pin is used to synchronize traversal of the table with the data stream. The table consists of up to 16 entries, each with the format shown in [Table 225](#).

**Table 225: Sequencer Table Entries**

Bits	Name	Description
0	Last	Indicates current entry is the last entry of the table.
1	Bit/Byte	0: Bit 1: Byte.
5:2	Count	One less than the number of bits/bytes controlled by this entry
6	Enable	Enables reception of the current data bit. 0: Disable 1: Enable
7	Strobe	Selects the value put on the external strobe pin while this entry is in use. The <b>ser_mode</b> register is used to select if the strobe is active high or low. 0: Deassert 1: Assert

Each entry controls the behavior of the line interface for a number of bit times equal to Count+1 if Bit/Byte is 0, or to 8\*(Count+1) if Bit/Byte is 1. During those bit times, DIN is sampled and processed on each clock edge if Enable is 1; DIN is ignored and no data is sent to the protocol engine if Enable is 0. If the RSTROBE pin is enabled then it will be driven with the value of Strobe bit in the entry.

The synchronization pulse on RIN is latched on the same clock edge as the data on DIN. The synchronization pulse is delayed by 0, 1, 2 or 3 clocks. The edge\_det bit in the **ser\_mode** register selects either the active level or the inactive to active edge of the delayed pulse as the start signal for the sequencer. If the sequencer is currently idle it will reset to map table entry zero when started, and the enable bit in that entry becomes effective immediately. Table entries are thereafter processed in order until encountering an entry with the Last indicator set.

After the last entry of the table is processed, the line interface unit waits for the next assertion of delayed RIN before it restarts with entry 0. During any interval between the end of the table and reassertion of RIN, DIN is not sampled.

Once the table scan has started the start signal is ignored. If an active level or edge is detected on the delayed RIN signal it will be flagged as an rx\_sync\_error in the **ser\_status** register and it will not affect the sequencer. (This may not be an error on some interfaces, for example if the sync is marked as level sensitive but lasts more than one cycle.)





The example in Figure 69 shows an active high sync pulse. If the delay is 0, map entry 0 determines whether data bit 0 is accepted or rejected. If that entry spans a single bit time, map entry 1 controls the disposition of bit 1. Alternatively if the delay is configured as one and entry 0 spans 14 byte times, it determines the treatment of bits 1 through 112.

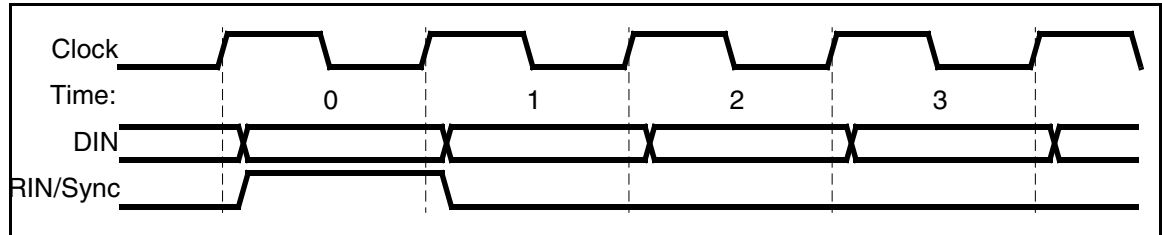


Figure 69: Example Reception Using RIN as Active High Sync (sampling on the falling clock edge)

## OUTPUT LINE INTERFACE

In the transmit direction, data is driven on pin DOUT either by an external clock signal, supplied on the CTS\_TCLKIN pin, or by the internal baud rate generator. The internal clock can be made available externally on the COUT pin if enabled by setting bit 1 (channel 0) or bit 3 (channel 1) in the UART output port configuration register (`duart_opcr`). The polarity of the transitioning clock edge is programmable. The DOUT pin will be high impedance while the interface is disabled, so a pull-up may be required.

The output bits are optionally gated by means similar to the input:

- 1 **Gapped Clock:** If the external device is supplying the clock on CTS\_TCLKIN, it can omit clock pulses. Since the BCM1250 never receives a clock pulse this method can always be used to suppress the transmission of bits.
- 2 **External Enable:** Regardless of clock source, the TIN pin can be supplied with an externally generated enable signal. This is latched on the active edge of the clock and disables output 0, 1, 2 or 3 clocks later. The DOUT pin is high impedance (undriven) when disabled.
- 3 **Internal Sequencer:** Regardless of clock source, but exclusive with (2), the enable signal can be generated by an internal sequencer, which is itself synchronized to the data stream by an edge or level on TIN. The sequencer can also provide a strobe on the TSTROBE output.

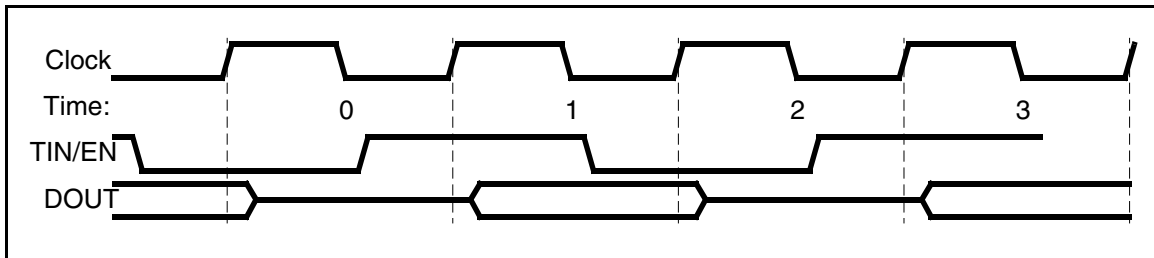
The bit stream from the protocol engine is output on the DOUT pin on every enabled clock edge. During cycles where the output is disabled DOUT is tri-stated following the clock edge and will be driven again following an enabled clock edge.

### Output Using an External Enable

The external enable on the TIN pin can be configured to be active high or low. The signal is delayed by 0 to 3 clock edges. If the delay is zero then the output is immediately enabled and will drive the current output data, provided the enable signal is still asserted on the next active edge of the clock the data will change and new data will be output. If the delay is nonzero the enable is sampled on the same edge of the clock that is used to drive DOUT. TIN must be stable for at least a set-up time prior to that clock edge (in the usual case it will transition on the opposite edge). When output is disabled, DOUT is tri-stated, and the bit stream supplied by the protocol engine does not advance.

The TSTROBE pin is used as a data valid indicator when an external enable is used. The strobe\_active bit in the **ser\_mode** register sets the active level that TSTROBE will have when DOUT is valid. If the TIN enable signal is always active (or is only changed between packets) then TSTROBE will frame the packet. The protocol engine will ensure there is at least one bit time of gap between packets.

In the example in [Figure 70](#) TIN is sampled on the rising edge of the clock and DOUT changes as a result of the rising edge of the clock. The delay is set to 1 so the enable is sampled on the rising clock edge. Since the enable is removed at the start of cycle 0 and 2, DOUT is tri-stated during bit times 0 and 2 and driven during bit times 1 and 3. (A similar figure would result if the delay is 3, removal of the enable during cycle 0 and 2 would result in DOUT being tri-stated during bit times 2 and 4 and driven at times 3 and 5).



**Figure 70: Example Transmission Using TIN as Active High Enable (Driving/Sampling on Rising Clock Edge)**

### Output Using the Internal Sequencer

The transmitter has its own serial sequencer and its own table for generating the enable signal. The table is used to generate the internal enable signal and an external strobe signal. An external framing pulse provided on the TIN pin is used to synchronize traversal of the table with the data stream. The table consists of up to 16 entries, each with the same format as the receive table shown in [Table 225 on page 340](#).

Each entry controls the behavior of the line interface for a number of bit times equal to Count+1 if Bit/Byte is 0, or to 8\*(Count+1) if Bit/Byte is 1. During those bit times, data is accepted from the protocol engine and sent on DOUT on each clock edge if Enable is 1; DOUT is set high impedance and no data is extracted from the protocol engine if Enable is 0. The TSTROBE pin is driven with the value of Strobe bit in the entry.

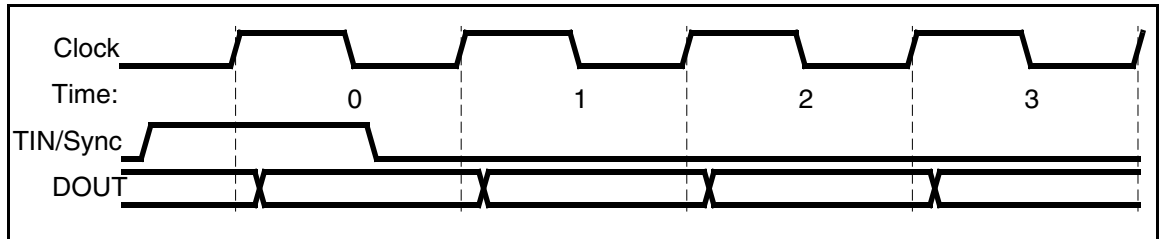
The synchronization pulse on TIN is delayed by between 0 and three active edges of the clock. The edge\_det bit in the **ser\_mode** register selects either the active level or the inactive to active edge of the delayed pulse as the start signal for the sequencer. If the delay is zero the sequencer is immediately signalled to start (and the data and enable outputs will transition accordingly) the sequencer will start synchronously to the first, second or third clock edge after the synchronization event. If the sequencer is currently idle it will reset to map table entry zero when started, and the enable bit in that entry becomes effective immediately. Table entries are thereafter processed in order until encountering an entry with the Last indicator set.

After the last entry of the table is processed, the line interface unit waits for the next start signal from the delayed TIN before it restarts with entry 0. During any interval between the end of the table and reassertion of TIN, DOUT is high impedance.

Once the table scan has started the start signal is ignored. If an active level or edge is detected on the delayed TIN signal it will be flagged as a tx\_sync\_error in the **ser\_status** register and it will not affect the sequencer. (This may not be an error on some interfaces, for example if the sync is marked as level sensitive but lasts more than one cycle.)



The example in [Figure 71](#) has the sync pulse set at time 0. The map table will be reset to entry 0 after the configured delay and then determine when DOUT is driven. In the example the delay is one (sync is sampled on the clock edge) and the data is driven for at least the first four bit times.



**Figure 71: Example Transmission Using TIN as Active High Sync (transition/sampling on rising clock edge)**

## SYNCHRONOUS SERIAL PROTOCOL ENGINE

The protocol engine converts between the bit streams used by the line interface and the packets transferred by the DMA engines. It has two modes, configured in the `ser_mode` register. In HDLC mode frames are encoded on the bit stream using the HDLC protocol. In transparent mode the bit stream is packed into bytes and the framing is based on the line interface Enable signal.

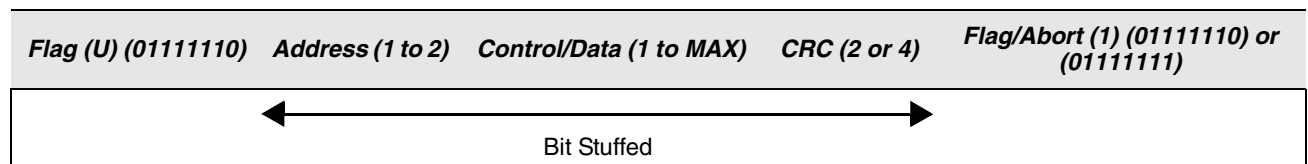
### OPERATION IN HDLC MODE

In the HDLC mode the frame structure used by the DMA engines is converted into the HDLC form on the line. The protocol engine can perform the following functions:

- insertion and deletion of HDLC flags
- bit stuffing and de-stuffing
- CRC calculation and checking
- address filtering (optional)
- frame length checking - padding of short frames to a configured minimum frame size (optional)

[Table 226](#) shows the HDLC frame structure (lengths in bytes).

**Table 226: HDLC Frame Structure**



The address, control and data fields are of variable length. The lengths and formats of addresses vary among the various link-level protocols that use bit synchronous HDLC-like framing, but lengths of 0, 1 or 2 bytes are typical. A CRC is appended on transmission and is checked on reception. The CRC can be either 16 bits or 32 bits. The CCITT CRC algorithms are used. The generator polynomials are:

$$\text{CRC-CCITT: } x^{16} + x^{12} + x^5 + 1$$

$$\text{CRC-32: } x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

CRC-32 is also known as CRC-CCITT-32. Both alternatives use a CRC preset of all 1's and send the resulting CRC in complemented form.

On the physical link, HDLC flags (8'b01111110) delimit frames. A flag must precede and follow the concatenation of the Address, Control, Data and CRC fields of a frame. Bit-stuffing prevents any occurrence of the flag pattern within those fields; the transmitter inserts a 0-bit after any sequence of 5 consecutive 1's, and the receiver performs corresponding 0 deletion.

When the same flag terminates one frame and begins the next, it is called a shared flag. A configuration option controls the minimum number of additional flags between frames. During idle periods, the transmitter sends either consecutive Flags or will send an idle as zero followed by fifteen 1s and then the output will go high impedance. Consecutive flags are logically equivalent to a single flag and do not delimit frames of length 0. Frames can also be terminated by HDLC Abort or Idle patterns (a zero bit followed by at least 7 ones). Abort and Idle termination are not distinguished by the receiver but are reported as aborted frames.

Bytes are sent and received in order of increasing address according to the system endian mode. Within each byte, except for the CRC, the least significant bit is sent or received first.

### Framing Parameters

The parameters controlling the format of frames are set in various configuration registers. The **ser\_mode** register selects the CRC used, the minimum number of flags between frames, and the pattern sent on an idle line. The **ser\_minfrm\_sz** and **ser\_maxfrm\_sz** registers set the minimum and maximum frame size respectively. The minimum should be set to the smallest size permitted for the Address, Control and Data fields prior to bit-stuffing. The maximum should be set to the largest size permitted for the total sizes in bytes of the Address, Control, Data and CRC fields prior to bit-stuffing.

### HDLC Transmitter

The user defines a frame to be transmitted by constructing the Address, Control and Data fields in a DMA buffer or chain of such buffers. For details of DMA buffers and descriptors, see [Section7: "DMA" on page 147](#).

For transmit, the DMA option flags shown in [Table 227](#) are supported.

**Table 227: Option Flags for Synchronous Serial Transmit Channel**

Transmit Commands			
Bits	Name	Default	Description
0	reserved	1'b0	Reserved
1	append CRC	1'b0	If this bit is set the computed CRC will be appended to the packet.
2	append PAD	1'b0	If this bit is set the packet will be padded to the minimum packet size.
3	abort	1'b0	If this bit is set the packet will be ended with an abort instead of a standard flag.



In most applications, the protocol engine will compute and append a CRC. If DMA option `append_CRC` is not set, the user can supply a CRC as part of the DMA buffer.

When the number of empty entries in the TxFIFO exceeds a configurable threshold (`ser_tx_wr_thres`), the DMA engine will begin to write frame data into the TxFIFO. The start and end of the frame are specially marked for the protocol engine.

Once the number of bytes transferred to the TxFIFO exceeds another configurable threshold (derived from `ser_tx_rd_thres`) or the end of frame has been written to the TxFIFO, processing of the new frame by the protocol engine can begin.

First the protocol engine ensures that a user-defined number of flags, set in the `flag_num` field of the `ser_mode` register, have been sent prior to the opening flag of the new frame. A `flag_num` of 4'b0 indicates that the closing Flag of one frame can be reused as the opening flag for the next.

The user frame is then read from the TxFIFO into the protocol engine. An opening Flag is automatically prepended and bit-stuffing is performed on the bytes supplied by the DMA.

If the transmit module empties the TxFIFO but the user frame has not completed, an underrun error is reported in the `tx_underrun` bit in the `ser_status` register. When this occurs, the transmit module terminates the current frame with an Abort sequence. It then begins to send Flag or Idle depending on the setting of `flag_en`. The value of `ser_tx_rd_thres` should be adjusted to minimize the likelihood of such underruns.

During frame transmission, the protocol engine always calculates a CRC, either the CRC-CCITT or CRC-32 as determined by the `crc_mode` bit in the `ser_mode` configuration register. The bit-stuffed CRC is inserted before the closing flag if the `append_CRC` option is set in the DMA descriptor. The transmitted CRC is always compared to the calculated CRC. If the CRC was automatically generated, the two necessarily match. If the CRC was supplied by the user and it does not match the calculated one, a transmit CRC error is reported in the `tx_crcerr` bit of the `ser_status` register.

In addition, the protocol engine keeps track of the number of bytes that are sent for the current frame. If the number of bytes provided in the user frame is smaller than `ser_min_frm_sz` and the `append_PAD` option is set in the transmit descriptor, the protocol engine automatically adds zero padding to the user frame. When padding occurs, the protocol engine also appends its calculated CRC.

Finally, a trailing Flag is automatically appended to the bit-stuffed frame. If DMA option `abort` was set, an Abort is sent instead of the final Flag.

The serialized bit stream is passed to the line interface as allowed by the transmit gating signals. When output to that interface is disabled, the serial stream does not advance; no bits are discarded. The output bit stream is not otherwise aligned or correlated with transitions on pin TIN, whether used directly as the enable signal or as synchronization for the serial sequencer.

No status is returned upon completion of DMA, but a serial DMA interrupt may be requested by setting the `interrupt` or `pkt_int` option. CRC and underrun errors set the flags `tx_crcerr` and `tx_underrun_error` in the `ser_status` register respectively. These flags are cumulative. If the corresponding bit in the `ser_err_mask` register is set, they also trigger a serial device interrupt. Reading the `ser_status` register resets these flags to 0 and clears the interrupt condition.

The transmitter can be temporarily paused (for example as the result of a flow control request) by writing the tx\_pause bit in the **ser\_cmd** register. This causes the transmitter to complete sending the current packet and then suspend operation until re-enabled. The next packet will wait in the TxFIFO while the transmitter is paused. The tx\_pause\_complete flag in the **ser\_status** register will be set to acknowledge completion of a tx\_pause command. The flag is set immediately if there is no packet in flight when the command is issued, otherwise it will be set when the end of the current packet is moved from the TxFIFO into the transmit module.

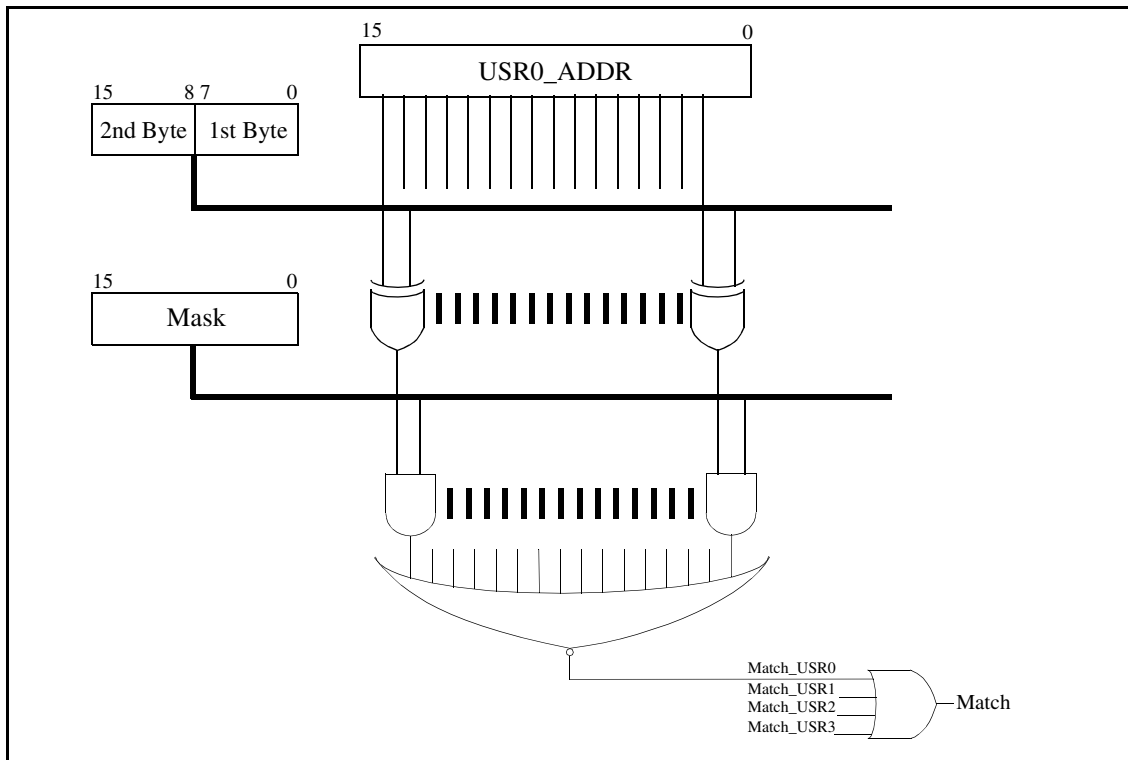
When the transmit module is idle, either Flag (octet synchronous) or Idle (bit synchronous) can be sent onto the channel as configured by flag\_en in the **ser\_mode** register.

**HDLC Receiver**

In HDLC mode, frames within the bit stream are self-identifying. The Protocol Engine monitors the input bit stream supplied by the line interface. Frame recognition begins with a Flag not followed by another Flag, an Abort or an Idle.

The receive module first removes any bit stuffing to extract the bytes of the frames delimited by the opening and closing flags.

The first step in processing is frame filtering based on the HDLC address. The bit mask in the ser\_addr\_mask configuration register selects address bits from within the first two bytes following the opening Flag of each frame. A frame is accepted if its masked address bits match any of the four address registers, ser\_usr0\_addr through ser\_usr3\_addr, under the same mask. Otherwise the frame is rejected and not sent to the RxFIFO. [Figure 72](#) shows the address matching logic.



**Figure 72: Frame Address Matching**

Setting **ser\_addr\_mask** to all zeros disables address filtering and allows reception of all frames. Link protocols with single-byte addresses should specify masks with all zeros in the most significant byte (i.e. 16'h00ff). When the number of acceptable addresses is less than four, the remaining address registers should be filled with copies of a valid address.

The recovered bytes of the Address, Control, Data and CRC fields of the frame are written, with bit-stuffing removed, into RxFIFO. If the RxFIFO becomes full in the middle of receiving a frame, the receive module discards all except the last word received when the full condition occurs. When space in the RxFIFO subsequently becomes available, this last word held is written into the FIFO with status overrun bit set. Any frames for which an opening Flag is received while the RxFIFO is full are ignored and not reflected in any status bits.

Received frames are moved from the RxFIFO to DMA buffers in memory by the serial DMA engine. The beginning and end of frame are specially marked by the protocol engine. DMA transfer begins when the number of entries in the RxFIFO exceeds a configurable threshold (**ser\_rx\_rd\_thres**) or when an End of Frame is available in the RxFIFO.

The packet is checked for errors during reception. Indications of any errors are passed to the RxFIFO and eventually written back into the DMA descriptor status field.

For each frame, the CRC is computed using CRC-CCITT or CRC-32, as selected by the **crc\_mode** in the **ser\_mode** register. The computed CRC is compared with the value in the packet. If the two do not match, **crc\_error** is signaled to the RxFIFO.

A frame that terminates with an Abort or Idle sequence is forwarded to the RxFIFO marked with an abort error flag.

The receive module also checks for a frame with a length (excluding opening and closing Flags) that exceeds the maximum length programmed in the **ser\_maxfrm\_sz** register. For such a frame, data up to the maximum length allowed is forwarded to the RxFIFO with an indication of a long frame error.

Similarly, the receive module checks for a frame that is shorter than the minimum length programmed in **ser\_minfrm\_sz** and sends a short frame error when forwarding the data to the RxFIFO. Note that frames of zero length are not recognized as such; a sequence of consecutive flags is equivalent to a single flag for the purpose of delimiting frames.

The receive module also checks for frames that are not-octet aligned, i.e., frames in which the number of bits between flags is not a multiple of 8 after any bit-stuffing is removed. These frames are forwarded to the RxFIFO with **octet\_error** set. The dribble bits are recorded in a final byte that is transferred and included in the reported frame length.

Upon completion of the DMA transfer, any error indications forwarded through the RxFIFO are reported as status bits in the first descriptor for the corresponding frame. Table 228 lists the DMA status bits that are used by the receiver's serial DMA channel.

**Table 228: Status Flags for Synchronous Serial Receive Channel**

Receive Status Flags			
Bits	Name	Default	Description
55:50	reserved	6'b0	Reserved
56	crc_error	1'b0	This bit is set if the received packet has a bad CRC.
57	abort	1'b0	This bit is set if the received packet ended with the abort flag.
58	octet_error	1'b0	This bit is set if the size of the received packet was not a multiple of 8 bits.
59	longframe_error	1'b0	This bit is set if the size of the received packet is bigger than the maximum frame size.
60	shortframe_error	1'b0	This bit is set if the size of the received packet is shorter than the minimum frame size.
61	overrun_error	1'b0	This bit is set if the received packet overran the FIFO and is therefore invalid.
62	good	1'b0	This bit is set if the received packet has no errors.
63	SOP	1'b0	This bit is set to indicate the start of the packet and the other bits are valid. Software should ensure this bit is clear when it sets up the descriptor, the DMA controller will only set it when packet reception has been completed.

In addition, a cumulative error summary is maintained in the **ser\_status** register. The receive errors are rx\_crc\_error, rx\_abort, rx\_octet\_error, rx\_longframe\_error, rx\_shortframe\_error and rx\_overrun error. When enabled by the corresponding bit in ser\_err\_mask, any set bit in **ser\_status** generates a request for a serial device interrupt. Reading the **ser\_status** register sets all bits to zero and clears the interrupt condition.

## OPERATION IN TRANSPARENT MODE

In transparent mode, bit streams are sent and received without modification. Frames are not self-identifying, but a frame structure can be imposed on the bit stream by external synchronization signals. For both transmit and receive, serialization of each byte can be either least significant bit first or most significant bit first, according to the setting of msb\_first in the **ser\_mode** configuration register. Bytes are transmitted and received in order of increasing address according to the system endian mode.

In transparent mode, the following functions may be performed:

- address matching (optional)
- padding of short frames to a configured minimum frame size (optional)
- CRC calculation and checking (optional)

In transparent mode, the data is framed by implicit start and stop indications at the bit level. The details depend upon configuration of the line interface.

**Note** The last time-slot specified in the user-defined table should always be enabled.





### Transmitter in Transparent Mode

In Transparent mode, the protocol engine does not perform bit-stuffing or Flag/Abort insertion. Thus the DMA abort option is ignored. The `append_CRC` and `append_PAD` options remain available.

The TIN signal is used as described in [Section: "Output Line Interface" on page 341](#) either as a level sensitive enable or to synchronize the table driven output sequencer. The transmit module will detect the start of packet indicator from the DMA engine as data comes out of the TxFIFO and align it to the line. If TIN is used as an enable, the `edge_det` bit in the `ser_mode` register selects if the start of a packet will be aligned with the next inactive to active edge of enable or the next active level (allowing packets to go our back-to-back).

If TIN is used as a synchronization pulse the start of a packet will be aligned with the first entry of the sequencer table being selected, which can be based on either the edge of the pulse or its level. These two rules ensure that packet boundaries in the DMA stream get aligned with the frame boundaries used on the line. If there are any idle cycles while a new frame is being aligned the DOUT pin will be set high impedance.

In other respects, operation is identical to operation of the transmitter in HDLC mode.

### Receiver in Transparent Mode

Transparent mode operates similarly to HDLC mode except that neither Flag detection/deletion nor removal of bit stuffing is performed. CRC checking and address filtering remain available. Note that since there is no way to disable the CRC check, device drivers for protocols that do not have CRCs must ignore the CRC error flag (and therefore cannot use the good packet bit) in the DMA descriptor status information.

Frames are delimited by transitions of the gating signals in each direction. If there are no transitions, transfers make no progress.

If an external enable signal is used, it serves as an envelope delimiting the frame. The initial bit in the frame is the one in the first bit time after transition of the (optionally delayed) enable signal to its active level. The final bit in the frame is the bit preceding the opposite transition.

If the serial sequencer is used, the table is traversed exactly once to delimit the frame. The synchronization pulse (optionally delayed) will start processing bits from entry zero of the table and mark the start of a new frame. The first bit in the received frame will therefore be the first bit that is enabled in the table. The final bit in the frame will be the last bit in the last entry of the table with the Enable bit set, following reception of this bit the sequencer will suspend until the next sync pulse and the packet will be complete. A sync pulse that occurs during traversal of the table will be logged as an `rx_sync_error` in the `ser_error` register but is otherwise ignored.

In other respects, operation is identical to operation of the receiver in HDLC mode.

## SYNCHRONOUS INTERFACE CONFIGURATION

There are four sections that must be configured before a serial channel can be used. These are: General Control, FIFO control, Protocol Engine/Line Interface Configuration and Address Filtering. In addition, the serial DMA channels must be configured.

At system reset, the transmit and receive modules in the Serial Link Interface are both disabled. They can be individually enabled by writing to the **ser\_cmd** register. Note that the **ser\_cmd** register is write-only.

Resetting the transmit/receive module will put it into disabled state and flushes the TxFIFO/RxFIFO. However, configuration registers are not restored to their default values. Configuration registers should be written only when the corresponding modules are disabled.

These modules need to be enabled after a reset by writing ones to the Receive/Transmit Enable fields in the **ser\_cmd** registers.

### DMA CONFIGURATION

Refer to [Section7: “DMA” on page 147](#) for configuring and initializing the serial DMA channels.

### FIFO CONFIGURATION

TxFIFO is a 64 bit wide FIFO with 8 entries. The **ser\_tx\_wr\_thres** register sets the number of empty 64 bit entries that must be in the TxFIFO before it will request DMA data. Since the DMA engine fetches in blocks of 32 bytes, this value must be set to 4 entries.

To reduce the likelihood of TxFIFO becoming empty during transmission of a frame, **ser\_tx\_rd\_thres** should be set to ensure a certain number of entries (8 bytes each) have been written before the Protocol Engine starts.

RxFIFO is a 32 bit wide FIFO with 16 entries. The **ser\_rx\_rd\_thres** register sets the number of valid entries that must be in the FIFO to request emptying by DMA. Since the DMA Engine transfers in blocks of 32 bytes, this value should be set to 8 entries.

TxFIFO and RxFIFO can be enabled or reset using the **ser\_cmd** configuration register controls. Threshold values should only be changed when the corresponding FIFO is reset.

### PROTOCOL ENGINE CONFIGURATION

The protocol engine is configured in the **ser\_mode** register. In addition the receive address mask and filter must be set in the **ser\_addr\_mask** and **ser\_usrN\_addr** (N=0,1,2,3) registers. If address filtering is not required the mask should be set to zero and all frames will be received.

## LINE INTERFACE CONFIGURATION

The line interface is configured in the **ser\_mode** and **ser\_line\_mode** registers. On reset the clock source defaults to use the internal baud rate clock, ensuring that state is correctly cleared. Since changes to the line mode register can change the clocking of the interface block care must be taken when it is written: all previous configuration requests must have completed before the **ser\_line\_mode** register is written, and no additional registers should be written until the **ser\_line\_mode** register write has completed. Any requests that are in progress close in time to when the **ser\_line\_mode** register is written will have UNPREDICTABLE results. The recommended way to safely change the **ser\_line\_mode** register is to first read the **ser\_mode** register and use the result (or use a SYNC) then write the **ser\_line\_mode** register and read it back and again use the result.

## SYNCHRONOUS SERIAL INTERRUPTS

The DMA interrupts for each channel are combined with status interrupts, and made available in the **ser\_status** register. Bits in this register are masked by the **ser\_int\_mask** register and combined to generate the system interrupts 10 (for channel 0) and 11 (for channel 1). Reading the **ser\_status** register will clear all bits in it. For a description of DMA interrupts associated with the serial channels, please refer to [Section 7: "DMA" on page 147](#).

To allow debuggers non-intrusive access to the status register, it is also made available through the **ser\_status\_debug** register, which does not clear the status on read.

## SYNCHRONOUS SERIAL LOOPBACK

There are two loopback configurations in the synchronous serial port. In both cases the transmitter output is connected internally to the receiver input. The first configuration relies on external timing pulses and uses TIN and RIN as normal. The second configuration has TIN always active and internally connects RIN to the RTST\_STROBE output. These are shown in [Figure 73](#). When the second loopback mode is used the line interface configuration for both tx and rx can be set to the values in [Table 229 on page 352](#) to allow a table driven loopback.

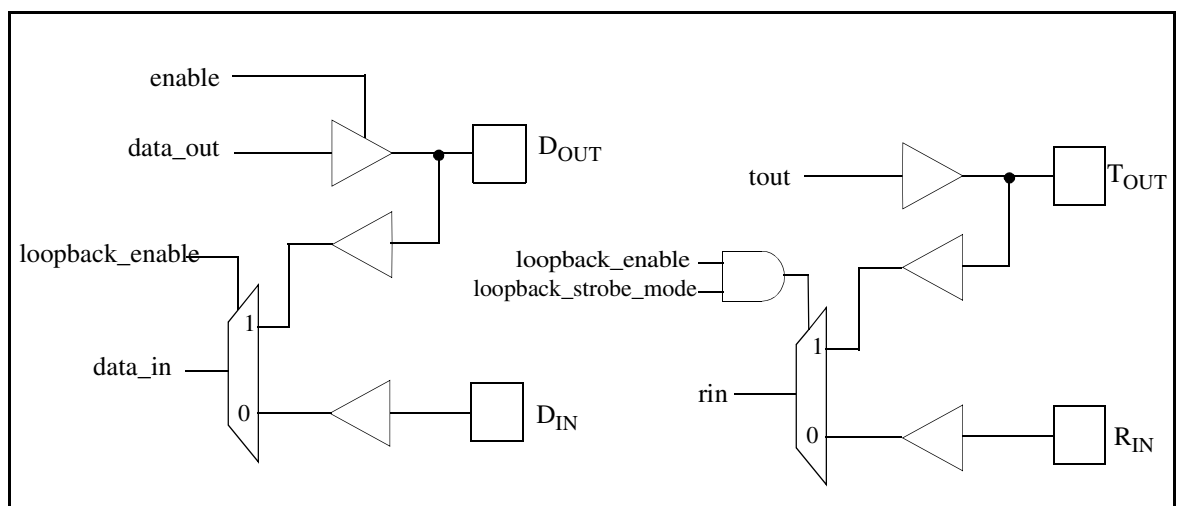


Figure 73: Synchronous Serial Loopback Connections

**Table 229: Recommended Line Interface Settings for Loopback Mode 2**

Field	Value
sync_active	1'b0
sync_delay	2'b00
strobe_active	1'b0
edge_det	1'b0
table_en	1'b1

## RMON COUNTERS

The serial interface maintains some counters that are useful for gathering RMON statistics. The counters are all 16 bits wide apart from the byte counters which are 32 bits wide (but occupy two register slots). The registers may be cleared by software writing to them. If any counter overflows it will wrap to zero and continue to count.

**Table 230: RMON Counters**

Number (offset from 00_1006_0000)	Counter	Description
0 _0 - 05C0 _1 - 09C0	Tx Byte (low)	Total number of bytes transmitted. (Low 16 bits).
1 _0 - 05C8 _1 - 09C8	Tx Byte (high)	Total number of bytes transmitted. (High 16 bits).
2 _0 - 05D0 _1 - 09D0	Rx Byte (low)	Total number of bytes received. (Low 16 bits).
3 _0 - 05D8 _1 - 09D8	Rx Byte (high)	Total number of bytes received. (High 16 bits).
4 _0 - 05E0 _1 - 09E0	Tx Underflow	Total number of transmit packets aborted due to FIFO underflow.
5 _0 - 05E8 _1 - 09E8	Rx Overflow	Total number of receive packets dropped due to overflow.
6 _0 - 05F0 _1 - 09F0	Rx Error	Total number of packets received with errors (CRC error, abort flag, or Octet error).
7 _0 - 05F8 _1 - 09F8	Rx Addr Fail	Total number of receive packets dropped due the address not matching.

## SYNCHRONOUS SERIAL REGISTER SUMMARY

All configuration and control registers are 16 bits.

**Table 231: Serial Mode Configuration Register**

ser_mode_0 - 00_1006_0500 ser_mode_1 - 00_1006_0900			
Bits	Name	Default	Descriptions
<b>Data Format Related</b>			
0	crc_mode	1'b0	0: CRC-CCITT. 1: CRC-32.
1	msb_first	1'b0	0: Transmit/Receive LSB in each byte first. 1: Transmit/Receive MSB in each byte first. This bit is only used in transparent mode.
<b>HDLC Related</b>			
5:2	flag_num	3'b1	Minimum number of Flags sent between Back-to-Back frames if HDLC is enabled. A value of 0 indicates the trailing Flag of the previous frame is used as the Open Flag of the next frame.
6	flag_en	1'b0	0: Idle is sent when Transmit module is Idle. 1: Flag is sent when Transmit module is Idle.
7	hdlc_en	1'b1	0: Transparent mode. 1: HDLC encoding.
<b>Loopback</b>			
8	loop_strobe	1'b0	Loopback Strobe mode. This configures the Sync/Enable inputs during loopback. 0: TIN and RIN are taken from the external source. 1: TIN is always active and RIN is internally connected to RtsTstrobe.
9	loop_data	1'b0	Loopback enable. If this bit is set the DIN input be internally connected to the DOUT output to allow loopback testing.
15:10	reserved	6'bx	Reserved
63:16	notimp	48'bx	Not Implemented.

**Table 232: Synchronous Serial Clock Source and Line Interface Mode Register**

ser_line_mode_0 - 00_1006_0578 ser_line_mode_1 - 00_1006_0978			
Bits	Name	Default	Description
0	rx_clk_inv	1'b0	This bit controls the inverter on the receive clock. 0: Not inverted (clock on rising edge). 1: Inverted (clock on falling edge).
1	rx_clk_src	1'b0	Receiver clock source. 0: Internal clock, generated by the baud rate generator. 1: External clock from CIN_RCLKIN pin.
3:2	rx_sync_delay	2'b0	Number of clock delays between the Frame Sync/Enable and the first bit of the frame.
4	rx_sync_active	1'b0	0: Sync/Enable is active high. 1: Sync/Enable is active low.
5	rx_strobe_active	1'b0	0: Strobe in Line Interface is active high. 1: Strobe in Line Interface is active low.

**Table 232: Synchronous Serial Clock Source and Line Interface Mode Register (Cont.)**

ser_line_mode_0 - 00_1006_0578 ser_line_mode_1 - 00_1006_0978			
Bits	Name	Default	Description
6	rx_edge_det	1'b0	[7:6]These two bits set the interface synchronization method. 00: External Enable is used using level as frame start. 01: External Enable is used, using edge as frame start.
7	rx_table_en	1'b0	10: Internal table drives enable, sync is level sensitive. 11: Internal table drives enable, sync is edge sensitive.
8	tx_clk_inv	1'b0	This bit contols the inverter on the transmit clock. 0: Not inverted (clock on rising edge). 1: Inverted (clock on falling edge).
9	tx_clk_src	1'b0	Transmitter clock source. 0: Internal clock, generated by the baud rate generator. 1: External clock from RCLKIN pin.
11:10	tx_sync_delay	2'b0	Number of clock delays between the Frame Sync/Enable and the first bit of the frame.
12	tx_sync_active	1'b0	0: Sync/Enable is active high. 1: Sync/Enable is active low.
13	tx_strobe_active	1'b0	0: Strobe in Line Interface is active high. 1: Strobe in Line Interface is active low.
14	tx_edge_det	1'b0	[15:14]These two bits set the interface synchronization method. 00: External Enable is used using level as frame start. 01: External Enable is used, using edge as frame start.
15	tx_table_en	1'b0	10: Internal table drives enable, sync is level sensitive. 11: Internal table drives enable, sync is edge sensitive.
63:16	notimp	48'bx	Not implemented.

**Table 233: Synchronous Serial Command Register (Write-only)**

ser_cmd_0 - 00_1006_0540 ser_cmd_1 - 00_1006_0940			
Bits	Name	Description	
0	rx_en	Receive enable. When the receiver is reset, writing a one enables the receiver.	
1	tx_en	Transmit enable. When the transmitter is reset or paused, writing a one enables the transmitter.	
2	rx_reset	Receive reset. Writing a one resets the receiver. This should only be done when the line is idle or being restarted, resetting an active interface can result in partial packet reception. Writing this bit will only reset the state machines; configuration registers are not returned to their default values.	
3	tx_reset	Transmit reset. Writing a one resets the transmitter. This should only be done when the line is idle or being restarted, resetting an active interface can result in UNPREDICTABLE data and strobe signals being generated. Writing this bit will only reset the state machines; configuration registers are not returned to their default values.	
4	reserved	Must be written as 0.	
5	tx_pause	Transmit pause. Writing a one causes the transmitter to pause at the end of the next packet. If the transmitter has no data it will stop immediately, otherwise it will continue sending until the next end of packet has been sent. The transmitter leaves any subsequent packet in the Tx FIFO. When the transmitter is re-enabled (by writing a 1 to the tx_en bit) the next packet will be fetched from the Tx FIFO and normal operation will continue.	
15:4	reserved	Must be written as 0.	
63:16	notimp	Not implemented.	



**Table 234: Serial Write Threshold Register**

ser_tx_wr_thres_0 - 00_1006_0568 ser_tx_wr_thres_1 - 00_1006_0968			
Bits	Name	Default	Description
3:0	thrsh	4'b100	Number of free 64 bit entries the TxFIFO must have before it signals the DMA that space is free. This must be set to 4 for normal operation. Writing other values in this register is for Broadcom Use Only.
15:4	reserved	12'b0	Reserved
63:16	notimp	48'bx	Not implemented.

**Table 235: Serial Transmit Read Threshold Register**

ser_tx_rd_thres_0 - 00_1006_0560 ser_tx_rd_thres_1 - 00_1006_0960			
Bits	Name	Default	Description
3:0	thrsh	4'b100	Number of filled 64 bit entries the TxFIFO must have before the protocol engine will start transmitting the frame. The FIFO is only 8 entries, so setting bit [3] will result in UNPREDICTABLE behaviour.
15:4	reserved	12'b0	Reserved
63:16	notimp	48'bx	Not implemented.

**Table 236: Serial Receive Read Threshold Register**

ser_rx_rd_thres_0 - 00_1006_0570 ser_rx_rd_thres_1 - 00_1006_0970			
Bits	Name	Default	Description
3:0	thrsh	4'b1000	Number of valid 32 bit entries the RxFIFO must have before it signals the DMA to read data. This must be set to 8 for normal operation.
15:4	reserved	12'b0	Reserved
63:16	notimp	48'bx	Not implemented.

**Table 237: Serial Minimum Frame Size Register**

ser_minfrm_sz_0 - 00_1006_0508 ser_minfrm_sz_1 - 00_1006_0908			
Bits	Name	Default	Description
15:0	size	16'b0	Minimum frame size in bytes.
63:16	notimp	48'bx	Not implemented.

**Table 238: Serial Maximum Frame Size Register**

ser_maxfrm_sz_0 - 00_1006_0510 ser_maxfrm_sz_1 - 00_1006_0910			
Bits	Name	Default	Description
15:0	size	16'hfff	Maximum frame size in bytes.
63:16	notimp	48'bx	Not implemented.

**Table 239: Serial DMA Enable Registers**

ser_dma_enable_0 - 00_1006_0580 ser_dma_enable_1 - 00_1006_0980			
Bits	Name	Default	Description
0	rxdma_en	1'b0	Receive DMA channel enable. Must be set to enable the channel.
3:1	reserved	3'b0	Reserved
4	txdma_en	1'b0	Transmit DMA channel enable. Must be set to enable the channel.
7:5	reserved	3'b0	Reserved
63:8	notimp	56'bx	Not implemented.

**Table 240: Synchronous Serial Status Register**

ser_status_0 - 00_1006_0588 ser_status_1 - 00_1006_0988 READ ONLY, Read Clears			
Bits	Name	Description	
0	rx_crcerr	Received frame with CRC error.	
1	rx_abort	Received frame terminated by Abort or Idle.	
2	rx_octet_error	Received frame length not a multiple of 8 bits.	
3	rx_longframe_error	Received frame longer than maximum size.	
4	rx_shortframe_error	Received frame shorter than minimum size.	
5	rx_overrun_error	RxFIFO overrun.	
6	rx_sync_error	Received a sync before the end of the receive table.	
7	reserved	Reserved	
8	tx_crcerr	Sent frame with bad user-supplied CRC.	
9	tx_underrun_error	TxFIFO underrun.	
10	tx_sync_error	Received a sync before the end of the transmit table.	
11	tx_pause_complete	Set when emptying of the Tx FIFO has stopped because of a TX pause command (sets immediately if no packet is in flight when the command is given, otherwise set after the end of packet has been moved from the FIFO to the transmitter).	
15:12	reserved	Reserved	
16	rx_eop_count	Set if the EOP interrupt was raised as a result of the packet count being reached.	
17	rx_eop_timer	Set if the EOP interrupt was raised as a result of the packet timer triggered.	





**Table 240: Synchronous Serial Status Register (Cont.)**

<b>ser_status_0 - 00_1006_0588</b> <b>ser_status_1 - 00_1006_0988</b> <b>READ ONLY, Read Clears</b>		
Bits	Name	Description
18	rx_eop_seen	Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).
19	rx_hwm	Set if the high watermark interrupt is raised.
20	rx_lwm	Set if the low watermark interrupt is raised.
21	rx_dscr	Set if the interrupt is triggered by a descriptor with the interrupt on packet end command.
22	rx_derr	Set if the controller ran out of descriptors during a packet reception. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
23	reserved	Reserved
24	tx_eop_count	Set if the EOP interrupt was raised as a result of the packet count being reached.
25	tx_eop_timer	Set if the EOP interrupt was raised as a result of the packet timer triggered.
26	tx_eop_seen	Set at the end of any packet transfer. It can be used during polling to determine if any packets have been transferred since the register was read (regardless of the setting of the int_pktcnt field).
27	tx_hwm	Set if the high watermark interrupt is raised.
28	tx_lwm	Set if the low watermark interrupt is raised.
29	tx_dscr	Set if the interrupt is triggered by a descriptor with the interrupt on packet end command.
30	tx_derr	Set if the controller ran out of descriptors during a packet transmission. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
31	tx_dzero	Set if a descriptor has a packet length of zero. The channel will be stopped. Software must disable and re-enable the channel to clear this fault.
63:32	notimp	Not implemented.

**Table 241: Serial Status Debug Register**

<b>ser_status_debug_0 - 00_1006_05A8</b> <b>ser_status_debug_1 - 00_1006_09A8</b>			
Bits	Name	Default	Description
31:0	status	32'b0	Reading this register gives the same value as reading the <b>ser_status</b> register, but does not have the side effect of clearing the register.
63:32	notimp	32'bx	Not implemented.

**Table 242: Serial Interrupt Mask Register**

<b>ser_int_mask_0 - 00_1006_0590</b> <b>ser_int_mask_1 - 00_1006_0990</b>			
Bits	Name	Default	Description
31:0	mask	32'b0	Setting a bit in this register enables generation of an interrupt when the corresponding status bit is set in the <b>ser_status</b> register.
63:32	notimp	32'bx	Not implemented.

**Table 243: Serial Address Mask Register**

ser_addr_mask_0 - 00_1006_0518 ser_addr_mask_1 - 00_1006_0918			
Bits	Name	Default	Description
15:0	mask	16'b0	Each one bit selects the address bits in the received frame to be matched with corresponding bits in the address match registers. Bits 7:0 correspond to the first byte of the frame; bits 15:8 to the second. Set this to zero for no address filtering; set bits 15:8 to zero for single-byte address filtering.
63:16	notimp	48'bx	Not implemented.

**Table 244: Serial Address Match Register**

ser_usr0_addr_0 - 00_1006_0520 ser_usr0_addr_1 - 00_1006_0920 ser_usr1_addr_0 - 00_1006_0528 ser_usr1_addr_1 - 00_1006_0928 ser_usr2_addr_0 - 00_1006_0530 ser_usr2_addr_1 - 00_1006_0930 ser_usr3_addr_0 - 00_1006_0538 ser_usr3_addr_1 - 00_1006_0938			
Bits	Name	Default	Description
15:0	addr	16'b0	The destination address to be matched in all bit positions selected by the ser_addr_mask register. Bits 7:0 correspond to the first byte of the frame; bits 15:8 to the second.
63:16	notimp	48'bx	Not implemented.

**Table 245: Sequencer Table Entries**

ser_rx_table_0 [0..15] - 00_1006_0600..0678 ser_rx_table_1 [0..15] - 00_1006_0A00..0A78 ser_tx_table_0 [0..15] - 00_1006_0700..0778 ser_tx_table_1 [0..15] - 00_1006_0B00..0B78			
Bits	Name	Description	
0	Last	Indicates current entry is the last entry of the table.	
1	Bit/Byte	Selects the multiplier for Count to give the number of bit times described by this table entry. 0: Bit 1: Byte	
5:2	Count	Number minus 1 of bits/bytes controlled by this entry	
6	Enable	Enable the serial line interface for the number of bit times described by this entry. For receive, ignore DIN during disabled bit times. For transmit, tristate DOUT during disabled bit times. 0: Disable 1: Enable	
7	Strobe	Enable the output strobe for the number of bit times described in this entry. 0: Deassert 1: Assert	
63:8	notimp	Not implemented.	



**Table 246: Serial RMON Counters**

Bits	Name	Default	Description
15:0	count	16'b0	Counter. Any write will zero the count.
63:16	notimp	48'bx	Not implemented.

---

This Page is left blank for notes



# Section 11: Generic/Boot Bus

## INTRODUCTION

The generic bus is used to attach the boot ROM and a variety of simple peripherals. Eight regions of memory are defined, each has its own chip select line, data width and set of timing parameters. All accesses to the generic bus range are accepted by the bus controller, a secondary decode determines which chip select region is used or raises an error. At reset time the generic bus address/data lines are read by the part to set static configuration options, so weak pull-up or pull-down resistors should be put on the board (see [Section: "Reset" on page 26](#) for details of the configuration bits).

## OVERVIEW

The generic bus is allocated a 767 MB block of the address space, from 00\_1009\_0000 to 00\_3FFF\_FFFF. Accesses to this address space cause transfers to take place on the generic bus pins. Eight regions are defined within the range by setting the base address and size of each region. Each region has an external active low chip select line, and a set of registers that set the transfer width and timing.

Six of the eight chip select regions are entirely free for individual devices. One (IO\_CS\_L[0]) is used for the boot Flash/ROM, and one (IO\_CS\_L[6]) is used for the PCMCIA device when it is enabled but is otherwise free. Each region can be configured to use either a 32 bit multiplexed address/data bus or an 8 bit wide data bus and 24 bit wide address bus.

After reset, region 0 is configured to map 4MB starting at the physical address location of the MIPS processor reset exception routine, 00\_1FC0\_0000 and ending at 00\_1FFF\_FFFF. Selection of a 32 bit multiplexed or 8 bit non-multiplexed bus for the boot region is made using the reset-time configuration options. An additional reset option diverts accesses made to region 0 to the SMBus interface 0 to allow boot code to be fetched from an SMBus EEPROM (see [Section: "Boot Using an SMBus EEPROM" on page 412](#)). The address range initially assigned to region 0 also covers the physical address range (00\_1FD0\_0000 to 00\_1FD0\_FFFF) that external PCI devices can access through the Expansion ROM BAR. The same ROM could therefore hold the boot code and PCI Expansion code.

The data width of each region that is configured for multiplexed address/data can be specified as 8-bit, 16-bit or 32-bit. When an access is made to a narrower region the generic bus interface logic will automatically perform repeated reads or writes and can transfer up to a cache line of data as a result of a single request.

The generic interface is big endian, thus an 8-bit device connects to the upper byte IO\_AD[31:24], a 16-bit device connects to the upper word IO\_AD[31:16], and a 32-bit device connects to the entire IO\_AD[31:0]. In non-multiplexed mode the data pins connect to the upper byte IO\_AD[31:24] and the address is provided on the lower three bytes IO\_AD[23:0]. The PCMCIA bus and other little endian devices should have D[7:0] connected to IO\_AD[31:24] and D[15:8] connected to IO\_AD[23:16] to keep the even addressed byte lane lined up correctly (see [Section 12: "PCMCIA Control Interface" on page 383](#) for full details on connecting PCMCIA cards). An alternative way of putting this is that the upper byte IO\_AD[31:24] is always the byte at the lowest memory address. This is true regardless of the internal system endian; the generic bus interface logic takes care of any swapping that is required. In alternate 8 bit multiplexed mode the data is transferred on IO\_AD[7:0], allowing direct connection to peripherals that use an 8 bit multiplexed connection.

Table 247 shows the byte lane correspondences for the generic bus. These are discussed in more detail in the sections that follow.

**Table 247: Byte Lanes for the Generic Bus**

Mode	IO_AD[31:24]	IO_AD[23:16]	IO_AD[15:8]	IO_AD[7:0]
Non Multiplex	D[7:0]	A[23:16]	A[15:8]	A[7:0]
Multiplex: ALE active	BE[3:0] A[27:24]	A[23:16]	A[15:8]	A[7:0]
Multiplex: ALE inactive / 8 bit device	D[7:0]			
Multiplex: ALE inactive / 16 bit big endian device	D[15:8]	D[7:0]		
Multiplex: ALE inactive / 32 bit big endian device	D[31:24]	D[23:16]	D[15:8]	D[7:0]
Multiplex: ALE inactive / 8 bit PCMCIA	D[7:0] CE1#			
Multiplex: ALE inactive / 16 bit PCMCIA	D[7:0] CE1#	D[15:8] CE2#		
Multiplex: ALE inactive / 16 bit little endian devices	D[7:0]	D[15:8]		
Multiplex: ALE inactive / 32 bit little endian devices	D[7:0]	D[15:8]	D[23:16]	D[31:24]
Multiplex: ALE inactive / Alternate 8 bit device				D[7:0]
Parity for byte lane (only on data, not address)	IO_ADP[3]	IO_ADP[2]	IO_ADP[1]	IO_ADP[0]
Multiplex: Offsets i.e. A[1:0]	+0	+1	+2	+3
Byte enable: Active low while ALE active (Valid for both reads and writes)	IO_AD[31]	IO_AD[30]	IO_AD[29]	IO_AD[28]

## CONFIGURING A CHIP SELECT REGION

The properties of a chip select region on the generic bus are configured by writing a set of registers. There are eight sets, denoted by having **\_0** to **\_7** appended to the register names used in this discussion.

### ADDRESS RANGE

The address range allocated to the generic bus is partitioned by configuring the base address and size of each region. The base address of the region always has bits [39:30] and [15:0] as zero, bits [29:16] are set in the **io\_ext\_start\_addr** register. This allows the start address to be set on any 64KB boundary below 00\_4000\_0000. If the start address is set below the base address of the generic bus range (00\_1009\_0000) the region is disabled. The region start address is subtracted from addresses as they pass through the bus interface, so each device sees a contiguous block starting from zero.

The size of the region is set in the **io\_ext\_mult\_size** register. The size of the region is 64KB \* (**io\_ext\_mult\_size**+1). Since only 12 bits of the register are valid, the minimum region size is 64KB, and the maximum size is 256MB. An alternative way of thinking about this is that the last address in the region has bits [39:30] all zero, bits [29:16] equal to **io\_ext\_start\_addr + io\_ext\_mult\_size** and bits [15:0] all ones. If the last address of the region is set above 00\_3FFF\_FFFF some of the region will not be accessible.



## CACHEABLE ACCESS BLOCKING

To provide protection against programming errors and prevent side effects from accidental speculative accesses through kseg0 and xkphys (see section [Section: “CPU Speculative Execution” on page 16](#)) cacheable reads may be blocked from being driven on the generic bus. This is done by setting the blk\_cache bit in the **io\_ext\_start\_addr** register. If this bit is set and a cacheable access is done to the region then the access will be blocked. This bit is clear by default and should be left clear for memory areas like the boot ROM that are likely to be cached for performance. To avoid causing the CPU to take an imprecise error as the result of a bad speculative access, a data error is not raised when a read is blocked. The access is just logged.

When a cacheable access is blocked:

- 1 No cycle is run on the generic bus.
- 2 For a read UNPREDICTABLE data is returned, marked with the valid data return code.
- 3 The **io\_cacheable\_blk** bit in the **io\_interrupt\_status** register is set (but no interrupt is raised). This bit is cleared by a read of the register.
- 4 If the log is not locked (i.e. the **io\_error** interrupt is deasserted) the address of the access gets logged in **io\_interrupt\_addr0** and **io\_interrupt\_addr1**.

## GENERIC BUS PARITY

A reset time system configuration option enables parity on the data portion of a transfer on the generic bus (not the address). This is provided on the IO\_ADP[3:0] pins, which are available as GPIO pins if generic bus parity is disabled. If parity is enabled for the system then each region can be configured in the **io\_ext\_cfg** register to have even, odd or no parity check.

On read transactions from parity generating devices, the parity bit of each byte of the incoming data is latched at the same time as the data, and checked against the parity computed for the byte. If an error is detected, then the address and region of the error are logged, the **io\_rd\_par\_int** bit is set in the **io\_interrupt\_status** register, the generic bus error interrupt is raised and the data is passed to the system bus marked with a bus error. The data must be passed to the system bus to terminate the read transaction, the error flag will prevent its use. The Bus Watcher in the SCD (see section [Section: “Bus Watcher” on page 64](#)) will note the error flag and increment the **bus\_io\_error** count. If the data is returning to one of the CPUs then the bus error exception will be raised (thus the CPU could be told about the error three times: the bus error exception, the bus watcher error interrupt and the interrupt from the generic bus controller).

When parity is enabled for a region it will be generated for the data in writes and will be driven on the parity pins with the same timing that the data has. If the device sees a parity error on writes it is device dependent how it responds and how it reports the error. [Table 247 on page 362](#) shows how the IO\_ADP pins match with the byte lanes.

Data with an uncorrectable error from the system bus will not be seen by the generic bus interface. The Bus Watcher will log and report the error.



## BUS WIDTH

The data width for a region is set in the **io\_ext\_cfg** register. If a region is in multiplexed address/data mode it can be set to have a data bus 8 or 16 or 32 bits wide. Regions using non-multiplexed mode must always be configured for 8 bit widths. The interface will pack or unpack data as required. The generic bus is always ordered in a big endian manner, so the lowest memory location is on IO\_AD[31:24]. If the region is configured as 16 bit wide the odd addresses will be on IO\_AD[23:16], for 32 bit regions IO\_AD[15:8] and IO\_AD[7:0] are added to carry the bytes with a[1:0] equal to 2 and 3.

Writes to the generic bus can be 1-8 bytes direct from the CPU, 1-32 bytes from an uncached-accelerated merge from the CPU or Data Mover, or 32 bytes when a cache line (which must be cacheable non-coherent) is moved. The interface will order these and align them so that they are written on the generic bus in ascending address order (this is useful when writing to FIFOs on generic bus devices).

Reads from the generic bus will similarly be broken down into requests that match the width of the region. Again the requests for a single read will be made in ascending address order.

If a multiplexed address/data bus is used then the upper four bits IO\_AD[31:28] are used during the address phase as byte lane enables. These active low signals are used when doing byte accesses to 16 or 32 bit regions, and for half-word accesses to 32 bit regions. The mapping from the byte enables to the byte lanes is shown in [Table 247 on page 362](#).

## GENERIC BUS TIMING

The timing of the generic bus is configurable for each of the chip select regions. When multiple devices are connected, care must be taken to ensure that the timing for one device does not confuse other devices. The chip select signals and idle time between cycles will normally be sufficient to ensure this.

Internally all the timing is done with reference to the 100 MHz clock (this clock is also used by the SMBus and serial interfaces). This clock will be output on the IO\_CLK100 pin if enabled by the reset configuration resistor on IO\_AD[23]. In most cases the clock is not needed externally and the interface is run as an asynchronous one using the computed timings given in the Hardware Data Sheet.

There are two modes for timing cycles. In fixed cycle mode the timing of the access is entirely controlled by the generic bus interface based on the parameters in the configuration registers. In acknowledgement based mode a ready/busy signal from the device controls the length of the access. The mode and active sense of the ready/busy signal is programmed in the **io\_ext\_cfg** register.

[Table 248 on page 365](#) shows the parameters that can be set to control the cycle time. The next four sections have timing diagrams that show how these parameters control the cycle. Each timing diagram includes the IO\_AD timing for both multiplexed and non-multiplexed operation. On reset the parameters for the IO\_CS\_L[0] region are suitable for a slow EPROM or flash memory (the parameters for the other regions do not get useful defaults). The parameters are programmed into the **io\_ext\_time\_cfg0** and **io\_ext\_time\_cfg\_1** registers and are expressed in cycles of the 100 MHz reference clock (i.e. multiples of 10ns).



**Table 248: Generic Bus Timing Parameters**

Name	Range (cycle = 10ns)	IO_CS[0] Reset	Description
ale_width	1 - 7 cycles	4	Width of the address latch enable pulse. This signal is asserted for this number of cycles to indicate the start of the cycle, the address and byte enables are output at the same time this signal asserts and remain stable after IO_ALE deasserts (i.e. IO_ALE is suitable for use as the latch input of a flow through latch, or the clock input of a falling-edge clocked flipflop).
ale_to_cs	1 - 3 cycles	2	Delay from IO_ALE being deasserted until assertion of the IO_CS_L line for the region addressed.
cs_width	1 - 31 cycles	24	The width of the chip select assertion in fixed cycle mode. In acknowledgement mode the IO_RDY line will not be sampled until this number of cycles have passed, but the chip select remains asserted until the IO_RDY signal has been asserted and rdy_smple+oe_to_cs cycles have passed.
rdy_smple	0 - 7 cycles	1	In acknowledgement mode this is the number of cycles from IO_RDY asserting to IO_WR_L or IO_OE_L deasserting (and data being latched in a read access). The chip select remains asserted oe_to_cs additional cycles.
idle_cycle	1 - 15 cycles	6	The number of cycles that the bus should be idle before the next IO_ALE. Note that the cycle does not end until one cycle after the IO_CS_L line has deasserted.
ale_to_wr	1 - 7 cycles	7	The number of cycles from IO_ALE deassertion to IO_WR_L assertion in a write cycle.
wr_width	1 - 15 cycles	7	In fixed mode this is the number of cycles that IO_WR_L is asserted in a write cycle. In acknowledgement mode this value is ignored and the IO_WR_L strobe deasserts rdy_smple cycles after the acknowledgement.
cs_to_oe	0 - 3 cycles	0	The number of cycles from IO_CS_L assertion to IO_OE_L assertion is a read cycle.
oe_to_cs	0 - 3 cycles	0	In fixed timing mode this sets the number of cycles before IO_CS_L deassertion that IO_OE_L deasserts in a read cycle.  In acknowledgement mode this sets the number of cycles between the strobe (IO_WR_L or IO_OE_L) deasserting and IO_CS_L deasserting.
io_timeout (in io_ext_cfg)	1 - 255 us	8	The number of microseconds to wait for the device to assert IO_RDY in acknowledgment mode before terminating the access and signalling a timeout bus error. If this parameter is set to zero the access will never be terminated with a timeout, so will hang if an acknowledgement is never provided.
burst_width	0 - 3 cycles	2'b0	Sets the number of cycles gap between a transfer in a burst transaction. The gap is two cycles larger than the value set in this field.

In the fixed mode, the cycle time is:

$$\text{ale\_width} + \text{ale\_to\_cs} + \text{cs\_width} + 1 + \text{idle\_cycle}$$

In the acknowledgement mode the minimum cycle time is:

$$\text{ale\_width} + \text{ale\_to\_cs} + \text{cs\_width} + \text{rdy\_smple} + 1 + \text{idle\_cycle}$$



The reference point for the IO\_WR\_L write strobe is the deassertion of IO\_ALE, it is the responsibility of software to ensure that the assertion timing (and width in fixed cycles) places the strobe within the cycle timing. It is required that

$$\text{ale\_to\_wr} \geq \text{ale\_to\_cs}$$

$$(\text{cs\_width} + \text{ale\_to\_cs}) \geq (\text{wr\_width} + \text{ale\_to\_wr})$$

Similarly, in acknowledgement mode the IO\_OE\_L and IO\_WR\_L timing is expected to assert the pulses before the IO\_RDY line is checked. It is required that

$$\text{cs\_width} \geq \text{cs\_to\_oe}$$

$$\text{cs\_width} \geq (\text{ale\_to\_wr} - \text{ale\_to\_cs})$$

Failure to meet these assumptions will result in UNDEFINED behavior.

### FIXED CYCLE READ ACCESS

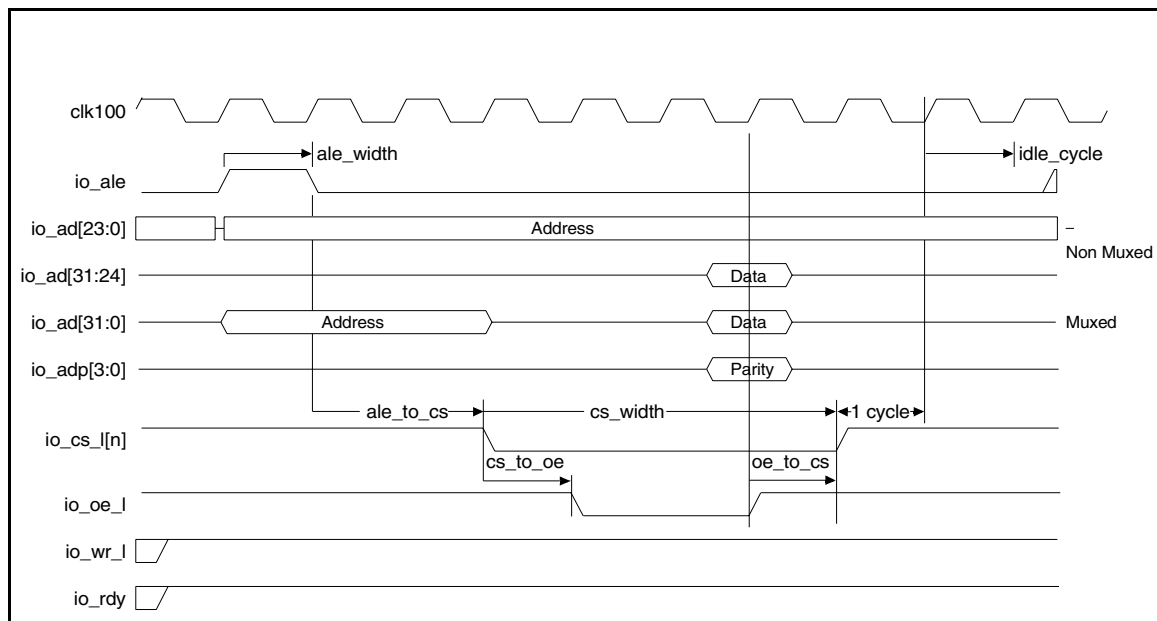


Figure 74: Fixed Cycle Read Access

A fixed read cycle has the simplest timing. The address is put out at the start of the cycle along with IO\_ALE. In the non-multiplexed version the address remains stable until the end of the cycle and the data lines are high impedance until the device drives them. In the multiplexed form the address stays valid until IO\_CS\_L asserts and the bus is turned around for data transfer. The read strobe IO\_OE\_L is asserted with a delay (which may be zero) from IO\_CS\_L and will deassert a fixed number of cycles (which may be zero) before IO\_CS\_L. The write strobe IO\_WR\_L remains deasserted for the entire cycle.

## FIXED CYCLE WRITE ACCESS

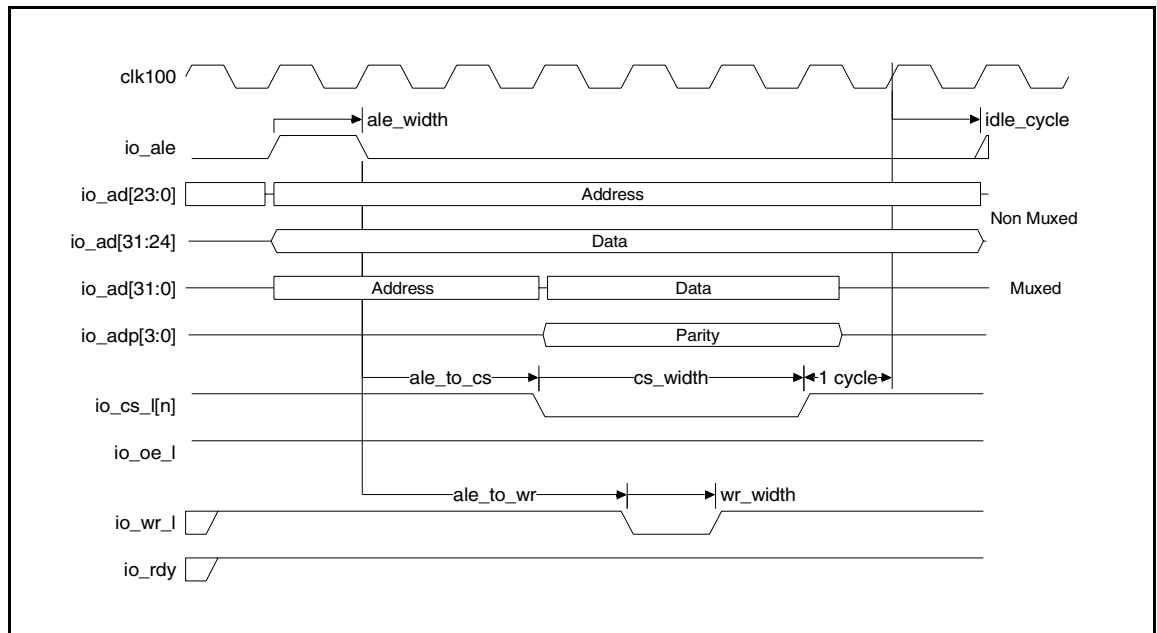
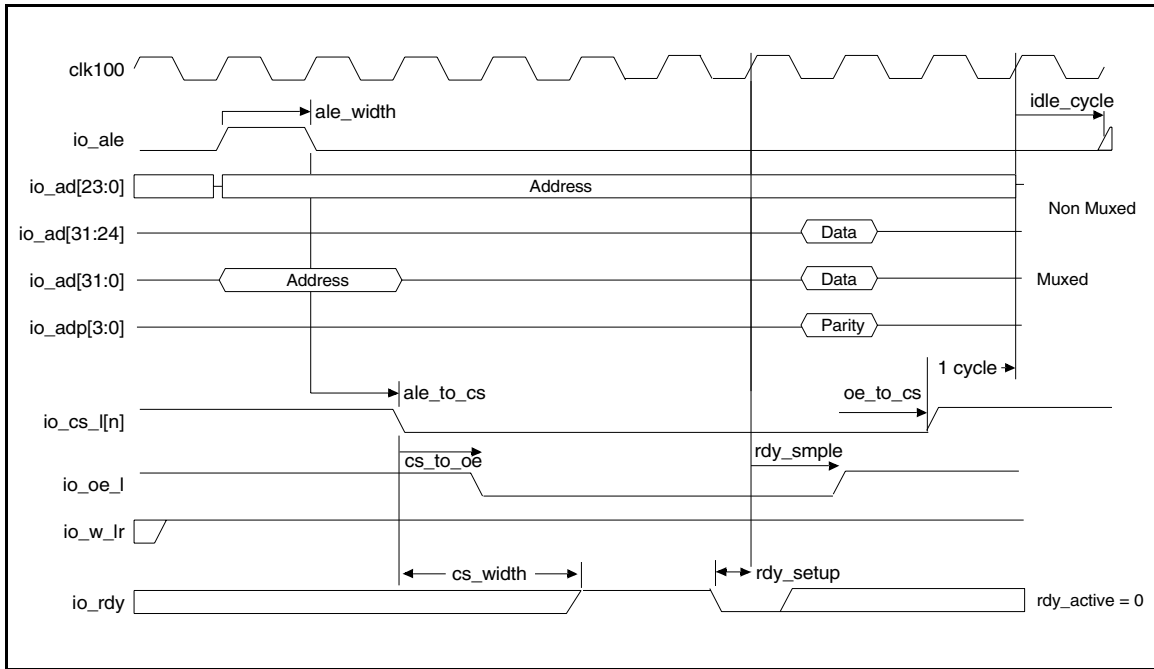


Figure 75: Fixed Cycle Write Access

The fixed cycle write is the same as the fixed cycle read, except IO\_OE\_L remains deasserted and the ale\_to\_wr and wr\_width parameters are used to set the assertion of the IO\_WR\_L signal. The data is output on the IO\_AD lines with the assertion of chip select.

**ACKNOWLEDGEMENT READ ACCESS**



**Figure 76: Acknowledge Read Access**

The start of an acknowledgement access is the same as a fixed cycle, however after IO\_CS\_L is asserted and the cs\_width interval has passed the IO\_RDY line is monitored. The IO\_RDY line is treated as an asynchronous signal, and therefore synchronized internally to the 100 MHz reference clock (if it fails to meet the rdy\_setup time it may not be sampled until the next cycle) so it should be asserted for a minimum of 20 ns. Once IO\_RDY has been asserted (to match the rdy\_active control bit) and the rdy\_smple delay has passed the data will be sampled and IO\_OE\_L deasserted. Zero to three cycles later, set by the oe\_to\_cs parameter, IO\_CS\_L will be deasserted. If the device does not signal that it is ready within the timeout period then the read will be aborted, the timeout error status set, the io\_error\_int interrupt will be raised and UNPREDICTABLE data flagged with a bus error will be returned.

In the case where the peripheral is using the IO\_CLK100 and can meet the setup and hold time for IO\_RDY the synchronizer may be bypassed. There will therefore be no delay in the acknowledgement reaching the generic bus state machine and it will take effect immediately. For a read if the rdy\_smpl parameter is set to zero and the synchronizer is bypassed the data is latched by the same IO\_CLK100 edge that the IO\_RDY is setup to and the IO\_OE\_L will deassert following that clock edge (by the clock-to-out delay).



## ACKNOWLEDGEMENT WRITE ACCESS

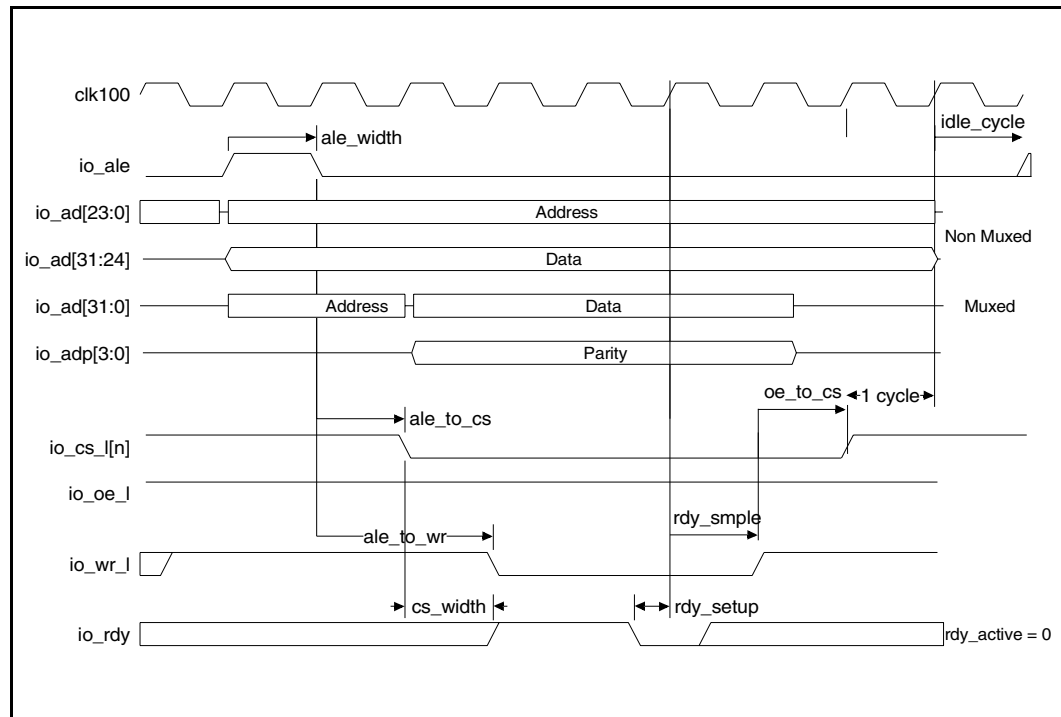


Figure 77: Acknowledge Write Access

The acknowledgement cycle write is the same as the read, except IO\_OE\_L remains deasserted and the ale\_to\_wr parameter is used to set the assertion of the IO\_WR\_L signal. The IO\_WR\_L signal is deasserted rdy\_smple cycles after the ready signal is sampled. The oe\_to\_cs parameter is used to set the number of cycles IO\_CS\_L remains asserted after IO\_WR\_L is deasserted. If the device does not signal that it is ready within the timeout period then the write will be aborted, the timeout error status set and the io\_error\_int interrupt will be raised.

### BURST MODE

A chip select region on the generic bus can be set in a burst mode. This allows faster throughput by only doing a single address cycle at the start of the burst. However, there are some restrictions when in burst mode.

The burst operation is based on the standard cycle outlined in the previous sections. The start of the transaction is the same with the address asserted with IO\_ALE. The chip select and strobe (IO\_OE\_L or IO\_WR\_L) are asserted and data transferred as normal. When the strobe deasserts (again with the normal timing) either there are no more transfers in the transaction and the access ends in the usual way, or the chip select remains asserted through an idle period of 2 or more cycles before the strobe reasserts for the next data.

This is shown in Figure 78 and Figure 79 and summarized in Table 249.

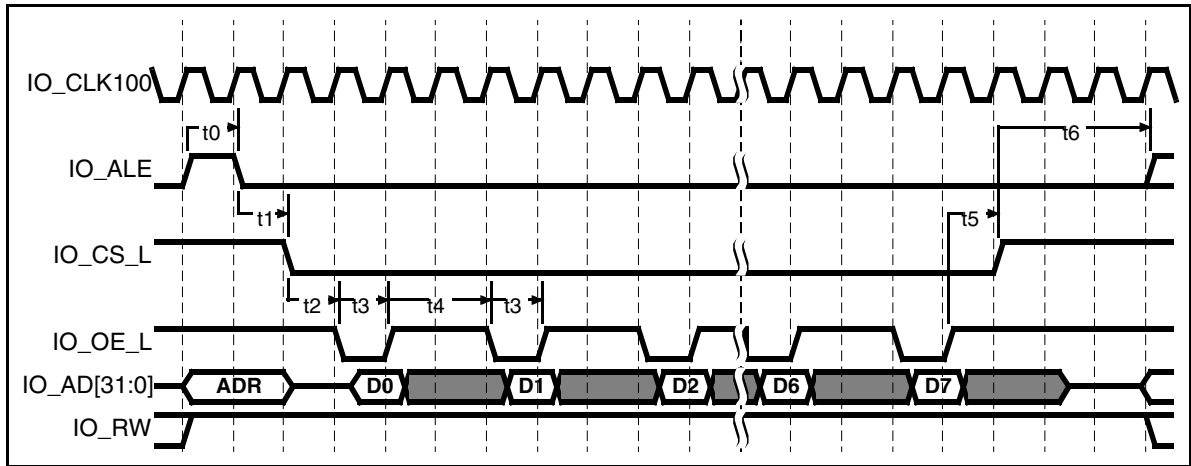


Figure 78: Generic Bus Burst Read

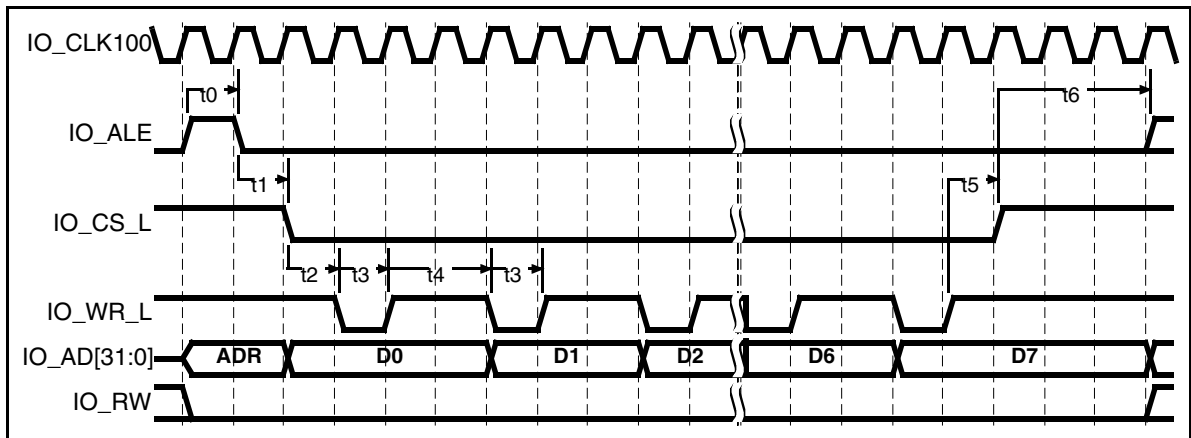


Figure 79: Generic Bus Burst Write



**Table 249: Burst Cycle Summary**

Ref	Duration (cycles)	Min Duration (cycles)	Action
t0	ale_width	1	ALE asserted and address driven on IO_AD
t1	ale_to_cs	1	Gap
t2	<b>Rd:</b> cs_to_oe <b>Wr:</b> ale_to_wr - ale_to_cs	0	IO_CS_L asserted, Strobe remains deasserted. On write the data becomes valid based on chip select assertion.
t3	<b>Rd:</b> cs_width - cs_to_oe - oe_to_cs <b>Wr:</b> wr_width	1	IO_CS_L and strobe asserted, first data transfers.  In acknowledgement mode this width is a minimum of cs_width + 1 + rdy_smpl, and will be more if the IO_RDY is not asserted after the cs_width holdoff period.
<b>Repeat next two lines while there is more data</b>			
t4	2 + burst_width	2	IO_CS_L asserted, Strobe deasserted. On write the new data becomes valid after 2 of the gap cycles.
t3	<b>Rd:</b> cs_width - cs_to_oe - oe_to_cs <b>Wr:</b> wr_width	1	IO_CS_L and strobe asserted, next data transfers. In acknowledgement mode this width is a minimum of 1 + rdy_smpl.
<b>End the cycle when there is no more data</b>			
t5	<b>Rd:</b> oe_to_cs <b>Wr:</b> ((cs_width - wr_width) - (ale_to_wr - ale_to_cs))	0	IO_CS_L asserted, Strobe deasserted. (optional gap).  In acknowledgement mode oe_to_cs is used for both reads and writes.
t6	1 + idle_cycles	2	IO_CS_L and strobe deasserted, bus idle time passes and cycle ends

The additional burst\_width parameter is used to specify the duration of the gap between transfers in a burst, to this is added 2 cycles. Thus the fastest sequence for a cache line burst to a region configured for 32 bit bus width and with burst\_width=0 and the other parameters at their minimum values is 5-3-3-3-3-3-3 (the initial 5 being ale\_width, ale\_to\_cs and the first 3 cycle access), giving 32 bytes in 26 cycles. Factoring in request and response buffering in the I/O bridge and generic interface, for multiple cache line transfers both reads and writes should achieve a little over 100 MBytes/s (reads will be slightly slower on parts with system revision < PERIPH\_REV3).

The burst mode is enabled by setting the io\_burst\_en bit in the io\_ext\_cfg register. If this is done then all transactions to the region must be smaller than or a multiple of the data width of the region and any uncached accelerated transactions (merged writes from the CPU) must have contiguous byte enables. If this is not done then extra locations will be accessed during the transfer so the result could be UNPREDICTABLE. The byte enables put out with the address are valid only for the first data transfer on the generic bus.

In burst mode the acknowledgement mode may be used. For the fastest burst the rdy\_smpl must be set to 0 and the cs\_width equal to cs\_to\_oe. In addition the IO\_RDY signal should be made synchronous. This allows the interface to detect the IO\_RDY signal (and the data for reads) during the first cycle the strobe is asserted. In the read case the ready and data are sampled and the IO\_OE\_L will be deasserted after one cycle. The ready signal should be held asserted for the remainder of the cycle to ensure the fastest burst (or can be deasserted to delay the remaining accesses). The chip select width can still be used to ignore the IO\_RDY signal, but this only applies to the first transfer in the burst (i.e. ready is ignored from the chip select assertion for the cs\_width but then must be valid for the rest of the cycle). If the IO\_RDY detection is asynchronous the internal detection can be as much as two cycles behind the external signal; this does not pose a problem if the ready indication is only used to delay the first transfer in the burst and remains asserted through the rest of the burst, but if the IO\_RDY is going to be used for each transfer in the burst then burst\_width will need to be at

least 1 (or more if the ready deasserts longer after the strobe). The rdy\_smpl value may be used to widen the strobe if required.

If the burst mode is selected with a non-multiplexed bus the address will not increment through the burst, it will remain at the start address of the burst for the entire access.

### EARLY CHIP SELECT

The chip select normally asserts ale\_to\_cs cycles after the IO\_ALE deasserts. Some peripherals will only take note of the IO\_ALE when their chip select is asserted. The early\_cs bit in the io\_ext\_time\_cfg0 register can be set to support them. It causes the external chip select signal to be asserted at the same time as IO\_ALE and remain asserted through the ale\_to\_cs delay and the normal chip select assertion time. Internally the normal chip select timing is used for control of the cycle.

## BOOT ROM SUPPORT

After reset, chip select region 0 is configured to map 4MB starting at the physical address location of the MIPS processor reset exception routine, 00\_1FC0\_0000. The timing parameters for the region are set for a slow (40 ns) address latch connected to a slow ROM with 240 ns access time from chip select and output enable. The other configuration parameters are set based on the boot\_mode configuration resistors as described in [Table 250](#).

**Table 250: Generic Bus Configuration for Each Boot Mode**

Boot Mode	Generic Configuration	Comments
00	Width = 32 bit / Fixed timing / Multiplexed A/D / No Parity	This gives the highest performance ROM connection.
01	Width = 8 bit / Fixed timing / NonMultiplexed A/D / No Parity	If address bits [27:24] and the byte enables (on bits [31:28]) are not needed, this will work with an 8 bit ROM connected for a multiplexed A/D bus.
10	Width = 32 bit / Acknowledgement mode / Timeout disabled / Access small EEPROM using SMBus	These two boot modes use the SMBus boot path described in section <a href="#">Section: "Booting Using an SMBus EEPROM"</a> on page 412.
11	Width = 32 bit / Acknowledgement mode / Timeout disabled / Access large EEPROM using SMBus	

The SMBus boot modes still use the generic bus module. The SMBus interface 0 protocol engine is placed under the control of the generic bus controller. Rather than run a generic bus cycle to fetch 32 bits of data, a request is made to the SMBus controller. The SMBus interface can be returned to normal software control by clearing the boot\_mode[1] bit (bit[18] in the **system\_cfg** register), once this is done the behavior of chip select region 0 becomes UNDEFINED until the io\_ext\_cfg\_0, io\_ext\_time\_cfg0\_0 and io\_ext\_time\_cfg1\_0 registers have been written.





## GENERIC BUS ERRORS

There are five error conditions that will raise the `io_bus_int` interrupt. When the interrupt signal is raised the associated address, data and parity are stored and the interrupt cause recorded. The causes are:

- 1 A data parity error is detected on an access to a generic bus region.
- 2 An access is performed to an address in the generic bus range that does not match any of the configured regions.
- 3 An access is performed to an address that matches multiple regions.
- 4 A region is configured in acknowledgement mode and the external device does not signal ready within the time limit set in the timeout register.
- 5 A cacheable coherent access is made to the generic bus and in the response phase of the access an agent asserts both `R_SHD` and `R_EXC` to indicate it has a tag error and cannot resolve the ownership.

Error cases (2), (3), and (5) are detected early so that no external access is made.

After an error has been signalled accesses to the generic bus may continue. However, if another error occurs before the previous interrupt was serviced by the CPU, it will be ignored. The interrupt will be cleared only when the `io_interrupt_status` register is read (a cacheable read of the register will not clear the interrupt). To ensure consistent data, software should therefore read the address, data and parity registers before this status register.

In the acknowledgement mode, if the access times out because the device does not signal ready the current transaction will be aborted (in the case of burst mode, the rest of the burst will not be completed). If the access was a read then data marked with a bus error will be returned. The next request will then be serviced. When the timeout occurs, the latency from the timeout being detected on the external bus to the bus error being returned is UNPREDICTABLE, and could be as high as 64K cycles of the `IO_CLK100`.

## DRIVE STRENGTH CONTROL

The output drive strength and internal slew rate may be set for all the general 3.3V outputs. This covers the generic bus, the serial ports, the SMBus ports, the Ethernet/packet FIFO interfaces and the GPIO pins. The drive strength effectively sets the number of transistors that will be used to switch the output. The drivers can be varied in 2mA steps from a nominal source/sink current of 2mA up to 8mA for low strength drivers and from 6mA to 12mA for high strength drivers. The slew rate can be varied from fast (recommended default) to slow in four steps. The drivers are grouped by function for setting the drive strength, and into larger groups for setting the slew rate. The parameters are described in more detail in the Hardware Data Sheet.

The drive strength registers for all of the general 3.3V outputs are included in the generic bus configuration register region of the address space.

## GENERIC BUS REGISTERS

**Table 251: Generic Bus Region Configuration Registers**

io_ext_cfg_0 - 00_1006_1000 io_ext_cfg_1 - 00_1006_1008 io_ext_cfg_2 - 00_1006_1010 io_ext_cfg_3 - 00_1006_1018 io_ext_cfg_4 - 00_1006_1020 io_ext_cfg_5 - 00_1006_1028 io_ext_cfg_6 - 00_1006_1030 io_ext_cfg_7 - 00_1006_1038 <b>A write to any bit causes all bits to be written.</b>			
Bits	Name	Default	Description
0	rdy_active	1'b0	When low the IO_RDY is active low (i.e. the device lets the line go low when it is ready), when high IO_RDY is active high.
1	io_ena_rdy	1'b0	When high, the interface uses acknowledgement based access. (See also bit 0)
3:2	io_width_sel	2'h0	Select the data width size. 2'b00: 1 byte width (default for _0 bootmode 01) 2'b01: 2 byte width 2'b10: Alternate 8 bit width (mux on D[7:0]) 2'b11: 4 byte width (default for cs0 bootmode 00, 10, 11)
4	io_parity_ena	1'b0	When high, enable parity check.
5	io_burst_en	1'b0	When high the interface will use the burst mode to run each transaction.
6	io_parity_type	1'b0	When low even parity is used, when high odd parity is used.
7	io_nonmux	1'b0	When high the bus is used in non-multiplexed mode with AD[31:24] being 8 bits of data and AD[23:0] being 24 bits of address. If the io_width_sel is not set to 2'b00 (for 8 bits) then bus operation is UNPREDICTABLE. For _0, the default is 1 in bootmode 01 and 0 in other bootmodes.
15:8	io_timeout	8'h8	Time out value. Unit is 1 us. If this field is zero the access will never timeout.
63:16	notimp	48'bx	Not Implemented.

**Table 252: Generic Bus Region Size Registers**

io_ext_mult_size_0 - 00_1006_1100 io_ext_mult_size_1 - 00_1006_1108 io_ext_mult_size_2 - 00_1006_1110 io_ext_mult_size_3 - 00_1006_1118 io_ext_mult_size_4 - 00_1006_1120 io_ext_mult_size_5 - 00_1006_1128 io_ext_mult_size_6 - 00_1006_1130 io_ext_mult_size_7 - 00_1006_1138 <b>A write to any bit causes all bits to be written.</b>			
Bits	Name	Default	Description
11:0	io_mult_size	12'h0	Size of the memory region in the multiple of 64KB. The region size is (io_mult_size + 1)*64KB
15:12	reserved	4'h0	Reserved
63:16	notimp	48'bx	Not implemented.

**Table 253: Generic Bus Region Base Address Registers**

Bits	Name	Default	Description
<b>io_ext_start_addr_0 - 00_1006_1200</b> <b>io_ext_start_addr_1 - 00_1006_1208</b> <b>io_ext_start_addr_2 - 00_1006_1210</b> <b>io_ext_start_addr_3 - 00_1006_1218</b> <b>io_ext_start_addr_4 - 00_1006_1220</b> <b>io_ext_start_addr_5 - 00_1006_1228</b> <b>io_ext_start_addr_6 - 00_1006_1230</b> <b>io_ext_start_addr_7 - 00_1006_1238</b> <b>A write to any bit causes all bits to be written.</b>			
13:0	io_start_addr	14'h0	Bits [29:16] of start address of segment. If this address is outside the generic bus range then the region is disabled.
14	reserved	1'b0	Reserved
15	io_blk_cache	1'b0	If this bit is set then cacheable accesses to this region will be blocked.
63:16	notimp	48'bx	Not implemented.

**Table 254: Generic Bus Region Timing 0 Registers**

Bits	Name	Default	Description
<b>io_ext_time_cfg0_0 - 00_1006_1600 (defaults for boot ROM)</b> <b>io_ext_time_cfg0_1 - 00_1006_1608</b> <b>io_ext_time_cfg0_2 - 00_1006_1610</b> <b>io_ext_time_cfg0_3 - 00_1006_1618</b> <b>io_ext_time_cfg0_4 - 00_1006_1620</b> <b>io_ext_time_cfg0_5 - 00_1006_1628</b> <b>io_ext_time_cfg0_6 - 00_1006_1630</b> <b>io_ext_time_cfg0_7 - 00_1006_1638</b> <b>A write to any bit causes all bits to be written.</b>			
2:0	io_ale_width	<b>_0 3'h4</b> 3'h1	Width of IO_ALE.
3	early_cs	1'b0	If this bit is set the external chip select is asserted during the ale_width and ale_to_cs gap as well as at the normal time. Note that this bit only affects the external signal, the timing follows the internal chip select asserting in the normal cycle.
5:4	io_ale_to_cs	<b>_0 2'h2</b> 2'h1	Chip select assertion after deassertion of IO_ALE.
7:6	burst_width	2'b0	Sets the gap between data transfers during a burst. The actual gap is burst_width + 2.
12:8	io_cs_width	<b>_0 5'h18</b> 5'h1	Width of the chip select asserted. If the interface is in acknowledgement mode this is the number of cycles from IO_CS_L being asserted before IO_RDY will start to be checked.
15:13	io_rdy_smple	3'h1	Number of clock cycles after the assertion of the acknowledgement is returned that the data is sampled into the interface. This is ignored if the interface is in fixed cycle mode.
63:16	notimp	48'bx	Not implemented

**Table 255: Generic Bus Region Timing 1 Registers**

<b>io_ext_time_cfg1_0 - 00_1006_1700 (defaults for boot ROM)</b>			
<b>io_ext_time_cfg1_1 - 00_1006_1708</b>			
<b>io_ext_time_cfg1_2 - 00_1006_1710</b>			
<b>io_ext_time_cfg1_3 - 00_1006_1718</b>			
<b>io_ext_time_cfg1_4 - 00_1006_1720</b>			
<b>io_ext_time_cfg1_5 - 00_1006_1728</b>			
<b>io_ext_time_cfg1_6 - 00_1006_1730</b>			
<b>io_ext_time_cfg1_7 - 00_1006_1738</b>			
<b>A write to any bit causes all bits to be written.</b>			
Bits	Name	Default	Description
2:0	io_ale_to_write	<b>_0</b> 3'h7 3'h1	Assertion of write strobe after the deassertion of ALE.
3	rdy_sync	1'b0	If this bit is clear the IO_RDY input is an asynchronous input and will be internally synchronized so there is no need for the peripheral to meet the setup and hold specification on the signal. If this bit is set the synchronizer is bypassed, so the acknowledgement is detected earlier by the generic bus state machine, but the peripheral must meet the setup and hold time specification.
7:4	io_write_width	<b>_0</b> 4'h7 4'h1	Width of the write strobe
11:8	io_idle_cycle	<b>_0</b> 4'h6 4'h1	Number of idle cycles between back_to_back operations.
13:12	io_oe_to_cs	2'h0	Number of cycles IO_OE_L deasserts before IO_CS_L deasserts. In acknowledgement mode this parameter is also the number of cycles between IO_WR_L deasserts and IO_CS_L deasserts.
15:14	io_cs_to_oe	2'h0	Number of cycles between IO_CS_L assertion and IO_OE_L assertion. This parameter must be less than io_cs_width.
63:16	notimp	48'bx	Not implemented.

**Table 256: Generic Bus Interrupt Status Register**

<b>io_interrupt_status - 00_1006_1A00 READ ONLY (Read clears interrupt)</b>			
Bits	Name	Default	Description
7:0	io_cs_err_int	8'h0	Indicates which of the 8 chip select regions have an error resulting in an interrupt. This field is only valid when one or more of bits [9], [10], [13], or [14] are set.
8	reserved	1'b0	Not used, reads as zero.
9	io_rd_par_int	1'b0	When high, indicates parity error on read data from a parity enabled device. The address of the 32 bit word accessed will be put in the address log, the data in the appropriate part of the data log, and the parity in the appropriate part of the parity log.
10	io_timeout_int	1'b0	When high, indicates timeout has occurred on one of the IO blocks. The address that was being accessed is put in the address log.
11	io_ill_addr_int	1'b0	When high, indicates an address referenced did not match any region. The address that was being accessed is put in the address log.
12	io_mult_cs_int	1'b0	When high, indicates multiple chip selects selected based on the address accessed. The address that was being accessed is put in the address log.



**Table 256: Generic Bus Interrupt Status Register (Cont.)**

<b>io_interrupt_status - 00_1006_1A00 READ ONLY (Read clears interrupt)</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
13	io_cacheable_blk	1'b0	When high, indicates that a cacheable access was blocked from a region that has the io_blk_cache bit set in its io_ext_start_addr register. The log contains the address of the blocked access.
14	io_coh_err	1'b0	When high indicates that a cacheable coherent access was made to the generic bus address space, but an agent indicated that it has a tag error and was unable to resolve ownership. The log contains the address of the failing access.
15	reserved	1'b0	Reserved
63:16	notimp	48'bx	Not implemented.

**Table 257: Generic Bus Error Data Register 0**

<b>io_interrupt_data0 - 00_1006_1A10 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	io_int_data0	16'h0	This register contains bits 15:0 of the data containing the parity error or captured when the timeout expired.
63:16	notimp	48'bx	Not implemented.

**Table 258: Generic Bus Error Data Register 1**

<b>io_interrupt_data1 - 00_1006_1A18 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	io_int_data1	16'h0	This register contains bits 31:16 of the data containing the parity error or captured when the timeout expired.
63:16	notimp	48'bx	Not implemented.

**Table 259: Generic Bus Error Data Register 2**

<b>io_interrupt_data2 - 00_1006_1A20 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	io_int_data2	16'h0	This register contains bits 47:32 of the data containing the parity error or captured when the timeout expired.
63:16	notimp	48'bx	Not implemented.

**Table 260: Generic Bus Error Data Register 3**

<b>io_interrupt_data3 - 00_1006_1A28 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	io_int_data3	16'h0	This register contains bits 63:48 of the data containing the parity error or captured when the timeout expired.

**Table 260: Generic Bus Error Data Register 3 (Cont.)**

<b>io_interrupt_data3 - 00_1006_1A28 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
63:16	notimp	48'bx	Not implemented.

**Table 261: Generic Bus Error Address Register 0**

<b>io_interrupt_addr0 - 00_1006_1A30 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	io_int_addr0	16'h0	This register contains bits 15:0 of the address causing the interrupt.
63:16	notimp	48'bx	Not implemented.

**Table 262: Generic Bus Error Address Register 1**

<b>io_interrupt_addr1 - 00_1006_1A40 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	io_int_addr1	16'h0	This register contains bits 31:16 of the address causing the interrupt.
63:16	notimp	48'bx	Not implemented.

**Table 263: Generic Bus Error Parity Register**

<b>io_interrupt_parity - 00_1006_1A50 READ ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
3:0	io_int_parity	4'h0	This register contains the parity of the data that generated the error.
63:4	notimp	60'bx	Not implemented.

**Table 264: Output Drive Control Register 0**

<b>io_drive_0 - 00_1006_1300</b> <b>A write to any bit causes all bits to be written.</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
1:0	io_slew0	2'b11	Slew0 control for I/O groups A, B, C, D, E, F, G, H, J, K.
3:2	io_drv_A	2'b01 8mA	Group A drive strength control. High drive 6/8/10/12 mA. Uses Slew0. IO_AD[31:0].
5:4	reserved	2'b11	Reserved
7:6	io_drv_B	2'b01 8mA	Group B drive strength control. High drive 6/8/10/12 mA. Uses Slew0. IO_CS[7:0].
9:8	reserved	2'b11	Reserved
11:10	io_drv_C	2'b01 8mA	Group C drive strength control. High drive 6/8/10/12 mA. Uses Slew0. IO_WR_L, IO_OE_L, IO_ALE, IO_CLK100.



**Table 264: Output Drive Control Register 0 (Cont.)**

<b>io_drive_0 - 00_1006_1300</b>			
<b>A write to any bit causes all bits to be written.</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
13:12	reserved	2'b11	Reserved
15:14	io_drv_D	2'b01 8mA	Group D drive strength control. High drive 6/8/10/12 mA. Uses Slew0. PC_EN3V PC_EV5V PC_ENVPP GPIO[15:12].
63:16	notimp	48'bx	Not implemented.

**Table 265: Output Drive Control Register 1**

<b>io_drive_1 - 00_1006_1308</b>			
<b>A write to any bit causes all bits to be written.</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
1:0	reserved	2'b11	Reserved
3:2	io_drv_E	2'b01 8 mA	Group E drive strength control. High drive 6/8/10/12 mA. Uses Slew0. GPIO[11:6], GPIO[1:0].
5:4	reserved	2'b11	Reserved
7:6	io_drv_F	2'b01 8 mA	Group F drive strength control. High drive 6/8/10/12 mA. Uses Slew0. GPIO[5:2]/IO_ADP[3:0].
9:8	reserved	2'b11	Reserved
11:10	io_drv_G	2'b01 4 mA	Group G drive strength control. Low drive 2/4/6/8mA. Uses Slew0. SDA0 SCL0.
13:12	reserved	2'b11	Reserved
15:14	io_drv_H	2'b01 4 mA	Group H drive strength control. Low drive 2/4/6/8mA. Uses Slew0. SDA1 SCL1.
63:16	notimp	48'bx	Not implemented.

**Table 266: Output Drive Control Register 2**

<b>io_drive_2 - 00_1006_1310</b>			
<b>A write to any bit causes all bits to be written.</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
1:0	reserved	2'b11	Reserved
3:2	io_drv_J	2'b01 4 mA	Group J drive strength control. Low drive 2/4/6/8mA. Uses Slew0. S0_DOUT S0_TOUT S0_COUT.
5:4	reserved	2'b11	Reserved
7:6	io_drv_K	2'b01 4 mA	Group K drive strength control. Low drive 2/4/6/8mA. Uses Slew0. S1_DOUT S1_TOUT S1_COUT.
9:8	io_slew1	2'b11	Slew1 control for I/O groups L and Q.
11:10	io_drv_L	2'b11 12 mA	Group L drive strength control. High drive 6/8/10/12mA. Uses Slew1. E0_MDIO (F0_RXFC) E0_MDC (F1_TXD0) E0_GENO (F0_TXC2) E1_MDIO (F1_TXD2) E1_MDC (F1_TXD1) E1_GENO (F1_TXD6) E2_MDIO (F1_RXFC) E2_MDC (F1_TXD7) E2_GENO (F1_TXC2).
13:12	io_slew2	2'b11	Slew2 control for I/O groups M and P.
15:14	io_drv_M	2'b11 12 mA	Group M drive strength control. High drive 6/8/10/12mA. Uses Slew2. E0_TXEN (F0_TXC0) E0_TXER (F0_TXC1) E0_TXD[7:0] (F0_TXD[7:0]) E1_TXEN (F1_TXD4) E1_TXER (F1_TXD5) E1_TXD[7:0] (F0_TXD[15:8]).
63:16	notimp	48'bx	Not implemented.

**Table 267: Output Drive Control Register 3**

<b>io_drive_3 - 00_1006_1318</b>			
<b>A write to any bit causes all bits to be written.</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
1:0	io_slew3	2'b11	Slew3 control for I/O groups N and R.
3:2	io_drv_N	2'b11 12 mA	Group N drive strength control. High drive 6/8/10/12mA. Uses Slew3. E0_TCLKO (F0_TCLKO).
5:4	reserved	2'b11	Reserved
7:6	io_drv_P	2'b11 12 mA	Group P drive strength control. High drive 6/8/10/12mA. Uses Slew2. E1_TCLKO (F1_TXD3).
9:8	reserved	2'b11	Reserved
11:10	io_drv_Q	2'b11 12 mA	Group Q drive strength control. High drive 6/8/10/12mA. Uses Slew1. E2_TXEN (F1_TXC0) E2_TXER (F1_TXC1) E2_TXD[7:0] (F1_TXD[15:8]).
13:12	reserved	2'b11	Reserved
15:14	io_drv_R	2'b11 12 mA	Group R drive strength control. High drive 6/8/10/12mA. Uses Slew3. E2_TCLKO (F2_TCLKO).
63:16	notimp	48'bx	Not implemented.





This Page is left blank for notes



---

This Page is left blank for notes



## Section 12: PCMCIA Control Interface

### INTRODUCTION

The part provides limited support for using PCMCIA memory cards attached to the generic bus. The interface allows card insertion and removal to be detected, provides and removes power from the card, and allows reading and writing of memory and attribute space on the card. The interface does not directly support I/O cards.

The standard was renamed in 1995, so PCMCIA cards are more correctly known as 16-bit PC Cards. Since the interface discussed in this section refers only to the cards defined in the PCMCIA Standard Release 2.1 the more popular name PCMCIA is used in this document.

The PCMCIA interface connects as chip select region 6 on the generic bus (see [Section: “Configuring a Chip Select Region” on page 362](#)). The generic bus configuration registers are used to set the address mapping and access timing for the card. The additional control logic provides:

- The two card enable signals (CE1# and CE2#) required by a card with a 16 bit data path.
- Detection of card insertion and removal (CD1# and CD2#), the VCC voltage requested by the card (VS1# and VS2#), card READY, and write protect state (WP).
- Software control of the PCMCIA REG# signal to select between accessing regular memory and attribute memory, and the card RESET signal.
- Outputs to control a supply of VCC and VPP to the card. VCC can either be controlled from software or automatically. Power is automatically removed when the card is removed.
- Interrupts raised to report card status changes.

### CONNECTING A PCMCIA SLOT

The PCMCIA interface uses ten of the GPIO pins to provide the additional support signals. In addition there are three dedicated pins that provide the card power enables (see [Section: “Using the Power Outputs” on page 392](#) for information on using these pins as general outputs if PCMCIA is not used). The reset-time configuration resistor on generic address bit IO\_AD[16] must pull high to convert the GPIO[15:6] pins to PCMCIA use. For proper operation the **gpio\_input\_invert** register bits [15:6] must be left as zeros (their default state) to disable the input inverter on the PCMCIA pins. Operation of the PCMCIA card is UNDEFINED if any of these bits are set, and if the voltage sense inputs are inverted the wrong VCC voltage could be supplied to the card.



### DIRECT CONNECTION OF A MEMORY ONLY CARD

Figure 80 shows the connections between the interface and the memory only PCMCIA slot. Chip select region 6 of the generic bus is configured in multiplexed mode and set for a 16 bit wide device. To allow hot-swap the card signals must be isolated from the generic bus. For the 26 address bits the buffering can be done by the address latch. For the data, a bidirectional buffer is needed.

The databus connections for the PCMCIA card reflect the little endian nature of the PCMCIA bus and the big endian byte lane assignment on the generic bus. So data[7:0] of the PCMCIA slot should connect via an isolation buffer to IO\_AD[31:24] and data[15:8] of the PCMCIA slot should connect through a buffer to IO\_AD[23:16]. This allows use of both 8 bit and 16 bit cards.

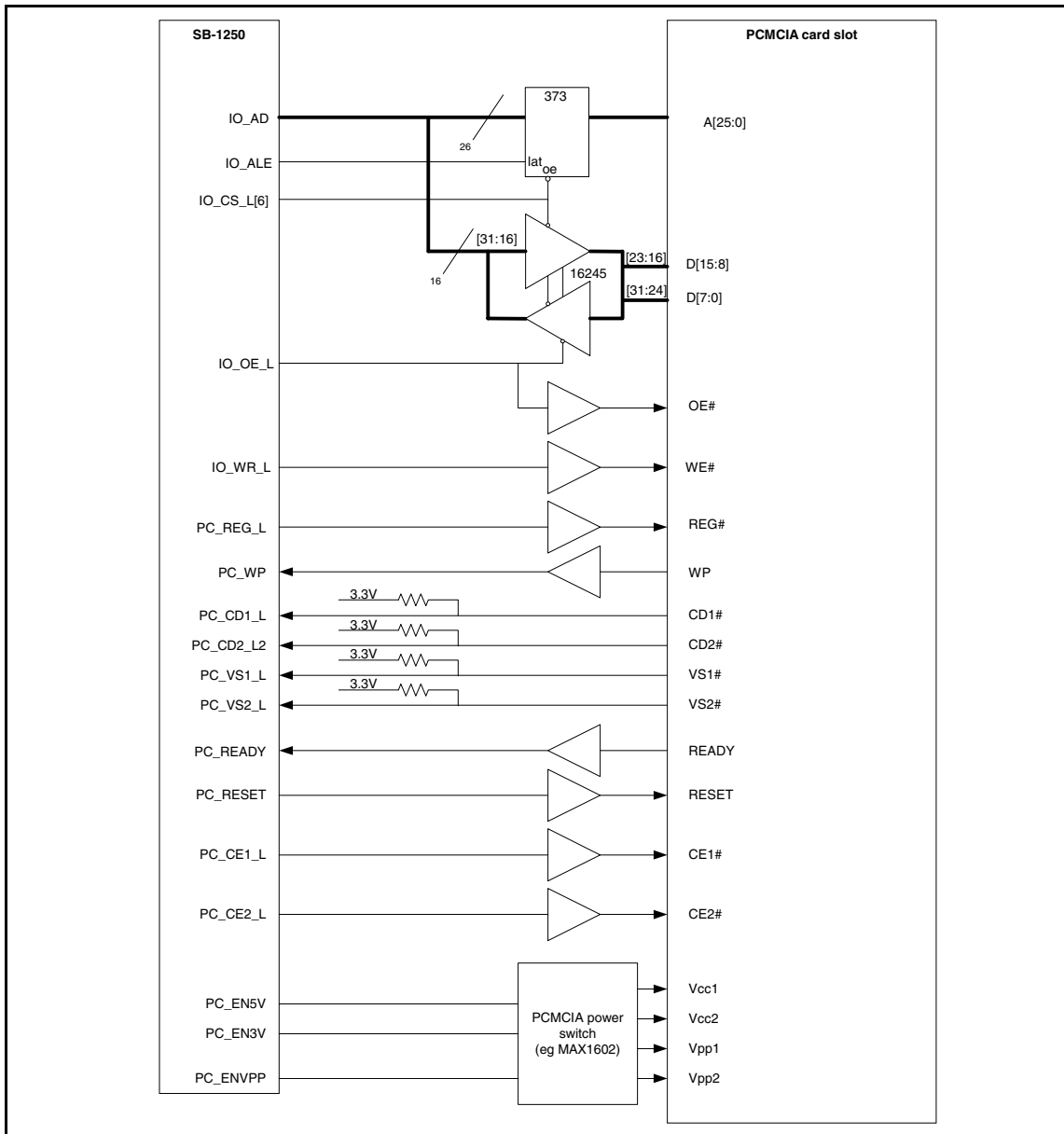


Figure 80: Example PCMCIA Slot Connection



Since the interface inputs are 3.3V only, care must be taken to buffer inputs from the card if 5V cards need to be supported. A FET switch or translating buffer can be used to provide the voltage conversion from card outputs and to provide power-down isolation for card inputs. The card detect and voltage sense lines are only statically pulled down or not connected on the card, they do not need buffers but should be pulled up to 3.3V on the main board.

The PCMCIA card uses two card enable signals (CE1# and CE2#) to indicate which byte lane is being used in a transfer. The PCMCIA control logic generates these from IO\_CE\_L[6] based on the size of the transfer requested by the CPU and the data width that the cs6 region has been configured for. If the interface is 2 bytes wide then the bottom address bit is not used by the card and the card enable lines select which byte lane is used during byte accesses. If the interface is only 1 byte wide then PC\_CE1\_L is always used with the lowest address bit, and PC\_CE2\_L is never asserted.

**Table 268: Source for PCMCIA Card Enable Signals**

io_width_sel configuration	Access Size	PC_CE1_L (data on IO_AD[31:24])	PC_CE2_L (data on IO_AD[23:16])
00 - 1 bytes	any	IO_CS_L[6]	1
01 - 2 bytes	even byte	IO_CS_L[6]	1
01 - 2 bytes	odd byte	1	IO_CS_L[6]
01 - 2 bytes	Half -word	IO_CS_L[6]	IO_CS_L[6]

The PC\_REG\_L signal is used to select between accesses to attribute memory (REG# low on the card) and regular memory. It is entirely under software control. The PC\_RESET signal is asserted to reset the card, while the card is in place it is controlled by software but the signal (and corresponding bit in the **pcmcia\_cfg** configuration register) will be set high when there is no card detected.

The WP (write protect) and READY status signals from the card are monitored and made available to software. Each of these signals has a transition detector and can raise the **pcmcia\_interrupt** when they change. These signals have a 60ns glitch filter.

The VS1# and VS2# voltage sense signals are monitored and made available to software, they have a 60ns glitch filter. These lines are static signals from the card, and are either not connected or passively pulled low on the card. They indicate the VCC that the socket requests when it is first powered up.

The PCMCIA card detect lines (CD1# and CD2#) are monitored by the hardware. A card is only considered inserted when these two signals are both low (physically these are near the two ends of the card socket and have shorter pins so they are the last pins to make contact). These signals have a 1 ms glitch filter. The (internal) card detect signal is the NOR of the filtered version of these signals. The **pcmcia\_interrupt** can be raised when the card detect status changes.

There are three dedicated outputs used to enable power to the card. Two of these (PC\_EN3V and PC\_EN5V) control VCC, and the third (PC\_ENVPP) controls the VPP programming voltage. The power enable signals are forced to be deasserted when the two card detect signals indicate a card is removed or not present. This provides the auto power down feature when a card is extracted.

Software can directly control the card power using three bits in the **pcmcia\_cfg** register, each bit corresponds to one output. These signals should be connected to an external power switch that provides the card VCC. When either CD1# or CD2# is set these bits are forced to zero and writes are ignored (i.e. software is unable to supply power if there is no card, the **pwr\_ctrl** bit in the **pcmcia\_cfg** register can be set to keep this behaviour when PCMCIA mode is disabled). When a card is inserted the card detect status change interrupt is used to alert the software. The state of the VS1# and VS2# signals is then read and software sets the appropriate configuration bits to power up the card. If the card is subsequently removed the power is automatically turned off (as described above) and the bits in the **pcmcia\_cfg** register are again cleared to zero to prevent immediate application of power when a new card is inserted (this must be done to avoid providing the wrong voltage to the new card).

Alternatively, VCC power management can be done entirely in hardware. This is configured by setting the **apron\_en** bit in the **pcmcia\_cfg** register. When this bit is set, the hardware detects the VS1# and VS2# signals and will enable the requested power to the card.

The following table summarizes the VCC power control options. This table includes the options for controlling the power enable output bits when the PCMCIA mode is not selected at reset time (this is discussed below in [Section: "Using the Power Outputs" on page 392](#)).

**Table 269: PCMCIA 3.3V and 5V VCC Power Enable Truth Table**

Reset AD[16] PCMCIA mode enable	Mode Control Bits pcmcia_cfg pwr_ctl bit	Input signals from PCMCIA card					Software Control Bits in pcmcia_cfg			Card State	Power Outputs	
		CD1# GPIO[12]	CD2# GPIO[13]	VS1# GPIO[14]	VS2# GPIO[15]	apron	cfg3v	cfg5V	PC_EN3V		PC_EN5V	
1	X	1	X	X	X	X	X	X	Not inserted	0	0	
1	X	X	1	X	X	X	X	X	Not inserted	0	0	
1	X	0	0	X	X	0	0	0	Inserted off	0	0	
1	X	0	0	X	X	0	0	1	Inserted sw set 5V	0	1	
1	X	0	0	X	X	0	1	0	Inserted sw set 3.3V	1	0	
1	X	0	0	X	X	0	1	1	Inserted sw set both power matrix decodes	1	1	
1	X	0	0	1	1	1	X	X	5V requested and supplied	0	1	
1	X	0	0	0	1	1	X	X	3.3V requested and supplied	1	0	
1	X	0	0	1	0	1	X	X	low-voltage X.XV requested none supplied	0	0	
1	X	0	0	0	0	1	X	X	X.X/3.3/5V requested 3.3V supplied	1	0	
0	1	1	X	X	X	X	X	X	Non-PCMCIA cd used forced off	0	0	
0	1	X	1	X	X	X	X	X	Non-PCMCIA cd used forced off	0	0	
0	1	0	0	X	X	X	0	0	Non-PCMCIA cd used off	0	0	
0	1	0	0	X	X	X	0	1	Non-PCMCIA cd used sw set	0	1	

Table 269: PCMCIA 3.3V and 5V VCC Power Enable Truth Table (Cont.)

Reset AD[16] PCMCIA mode enable	Mode Control Bits pcmcia_cfg pwr_ctl bit	Input signals from PCMCIA card				Software Control Bits in pcmcia_cfg			Card State	Power Outputs	
		CD1# GPIO[12]	CD2# GPIO[13]	VS1# GPIO[14]	VS2# GPIO[15]	apron	cfg3v	cfg5V		PC_EN3V	PC_EN5V
0	1	0	0	X	X	X	1	0	Non-PCMCIA cd used sw set	1	0
0	1	0	0	X	X	X	1	1	Non-PCMCIA cd used sw set	1	1
0	0	X	X	X	X	X	0	0	Non-PCMCIA off	0	0
0	0	X	X	X	X	X	0	1	Non-PCMCIA sw set	0	1
0	0	X	X	X	X	X	1	0	Non-PCMCIA sw set	1	0
0	0	X	X	X	X	X	1	1	Non-PCMCIA sw set	1	1

**Note** In the case of both PC\_EN5V and PC\_EN3V being asserted (which can only be done under software control) the PCMCIA power switch will determine what voltage is actually applied.



The VPP power is supplied under software control, but is always turned off when no card is inserted. Note that the hardware does not check the write protect state, and will supply VPP to a write protected card. This is summarized in the table below, which again includes details of use of the output pin when PCMCIA mode is not selected.

Table 270: PCMCIA VPP Power Enable Truth Table

Reset AD[16] PCMCIA mode enable	Mode Control Bits pcmcia_cfg pwr_ctl bit	Input signals from PCMCIA card			Software Control pcmcia_cfg cfgvpp bit	Card State	Power Output PC_VPPEN
		CD1# GPIO[12]	CD2# GPIO[13]	WP GPIO[11]			
1	X	1	X	X	X	Not Inserted	0
1	X	X	1	X	X	Not Inserted	0
1	X	0	0	1	0	Write Protected, vpp disabled	0
1	X	0	0	1	1	Write Protected, vpp enabled	1
1	X	0	0	0	0	Writable, vpp disabled	0
1	X	0	0	0	1	Writable, vpp enabled	1
0	1	1	X	X	X	Non-PCMCIA, cd used, forced off	0

Table 270: PCMCIA VPP Power Enable Truth Table (Cont.)

Mode Control Bits		Input signals from PCMCIA card			Software Control	Card State	Power Output
Reset AD[16] PCMCIA mode enable	pcmcia_cfg pwr_cti_bit	CD1# GPIO[12]	CD2# GPIO[13]	WP GPIO[11]	pcmcia_cfg cfgvpp_bit		PC_VPPEN
0	1	X	1	X	X	Non-PCMCIA, cd used, forced off	0
0	1	0	0	X	0	Non-PCMCIA, cd used, sw turned off	0
0	1	0	0	X	1	Non-PCMCIA, cd used, sw turned on	1
0	0	X	X	X	0	Non-PCMCIA, sw turned off	0
0	0	X	X	X	1	Non-PCMCIA, sw turned on	1

### OTHER PCMCIA SIGNALS

The Memory Only PCMCIA slot includes two signals for indicating the status of a battery on the card (BVD1 and BVD2). If these are required they can be connected via a level translating buffer to GPIO pins configured as inputs.

An additional signal WAIT# is used on some cards to indicate that it needs to extend the access time to a transaction. This can be connected through a buffer (to protect the interface input from 5V PCMCIA cards) to the generic bus IO\_RDY input. Additional logic may be required to combine the signal with the ready or busy signals from other devices if other generic bus chip select regions use the acknowledgement based mode. If the WAIT# line is connected in this way and a card that does not support WAIT# is inserted, the interface can be switched to fixed timing mode to ignore the IO\_RDY signal and allow the card to be used.

The I/O and Memory PCMCIA interface changes the use of some of the signals and adds some others.

- The READY pin becomes IREQ#, an active low interrupt signal. This will work with the standard connection to PC\_READY, and the PCMCIA interrupt can be raised whenever the pin changes state regardless of it indicating the card is ready, or is interrupting.
- The WP pin becomes IOIS16#. This is used by the card to inform the host for a particular address if a 16 bit access is possible. The generic bus does not support this behavior, so for cards which need this (indicated in the TPCE\_IO field of the CISTPL\_CFTABLE\_ENTRY configuration tuple) either the interface must be configured for 8 bit only operations or a software restriction made to ensure only 8 bit accesses are done to registers that are 8 bits. The PCMCIA specification indicates that on any card that has a mixture of 8 and 16 bit registers byte registers that are at an odd address can be read on the D[15:8] byte lane when CE2# is asserted, matching [Table 270 on page 387](#).
- The IORD# and IOWR# signals have been added. These match OE# and WE# but for accesses to I/O addresses. One method of generating these is with external logic that drives IO\_OE\_L and IO\_WE\_L to either (OE#, WE#) or (IORD#, IOWR#) based on an address bit.
- The PCMCIA card does not share the generic bus region with other peripherals, so the input port acknowledgement signal INPACK# is not needed.
- BVD2 is replaced by SPKR#, which can be used to drive a beep speaker. This would go to some other block than the BCM1250 or BCM1125/H.
- BVD1 is replaced by STSCHNG# which is used to indicate the card status has changed and the card pin replacement register should be read to get the values that would otherwise be reported on the READY, WP, BVD1 or BVD2 pins.





CompactFlash and CF+ cards have a similar interface to the PCMCIA cards. They also support a TrueIDE mode, which cannot be directly connected, but they are required to be able to work in the other modes so this should not be a problem. If only the memory mode of CF/CF+ cards is required then the interface is the same as the memory only PCMCIA interface presented above. However, CF/CF+ cards must be able to operate at 3.3V so the voltage level translators can be avoided (in this case software will have to reject any cards that require 5V for full operation).

## USING THE PCMCIA CARD

The generic bus address window scheme is used to map between the system address space and the PCMCIA flash memory address space. The PCMCIA cards are accessed through region 6 of the generic bus address space. The usual **io\_start\_addr** and **io\_multi\_size** registers are configured to set the base address that the card appears in the system address space and its size. The base address of the region will be mapped to address zero in the PCMCIA space, so PCMCIA card addresses can be computed by adding this offset. In addition the **pcmcia\_reg** bit within the **pcmcia\_cfg** register must be set according to whether the flash card's attribute or common memory is being accessed. This bit defaults to the low state which selects the common memory.

The card timing is configured using the generic bus timing registers. Since these default to being appropriate for the boot ROM (see [Section: "Generic Bus Timing" on page 364](#)) software should configure the timing in region 6 for PCMCIA access at system startup.

The first access to a card after power on should be an attribute memory read. The **pcmcia\_cfg\_reg** bit should be set to allow this. The transfer is 16 bits but only the even byte contains valid data because attributes are stored in even byte locations only. The initial attribute memory accesses should be used to extract details from the Card Information Structure (CIS). These details include the card identification and the timing parameters for the card (which will generally have a faster access time than the default). Software should update the parameters for the generic bus region 6. Once the CIS has been read the **pcmcia\_cfg\_reg** bit can be cleared to allow access to the memory portion of the card.

The PCMCIA controller monitors status change events and generates an interrupt. The interrupt is cleared by reading the **pcmcia\_status** register. An interrupt mask is provided so specific status change events can be masked out. The PCMCIA controller supports 3 card conditions or events:

- Card insertion or removal.
- Card READY status change.
- Card WP write protect state change.

Software can perform a card reset by setting the reset bit high inside the **pcmcia\_cfg** control register. The flash card remains in reset until the reset bit is set low. The reset signal is set when the system powers up and when no card is detected.

## EXAMPLE PCMCIA TIMINGS

A flash card might have the access timings given in [Table 271](#).

**Table 271: Example Flash Card AC Specs**

Parameter	Name	Min (ns)	Max (ns)
Read Cycle Time	tRC	150	-
Address Access Time	tAcca	-	150
Card Enable Access Time	tAccce	-	150
Output Enable Access Time	tAccoe	-	80
Output Disable Time	tDis	-	50
Write Cycle Time	tWC	150	-
Write Pulse Width	tWP	75	-
Address Setup Time to WE	tSUa	20	-
Card Enable Setup Time to WE	tSUcewe	0	-
Address Setup Time to WE end	tSUaweh	140	-
Data Setup Time to WE end	tSUd	50	-
Data Hold time from WE end	tHd	20	-
Card Enable to WE end	tSUceweh	100	-
Card Enable hold from WE end	tHce	0	-
Write Recover time	Trec	20	-

If such a card were connected as shown in [Figure 80 on page 384](#) possible generic bus timing parameters for this card are given in [Table 272 on page 392](#). The timing diagram that results for read and write accesses is shown in [Figure 81 on page 391](#). Note that the values used for the card timing and the propagation delays of the buffers are chosen for illustration only and do not necessarily match any real devices.



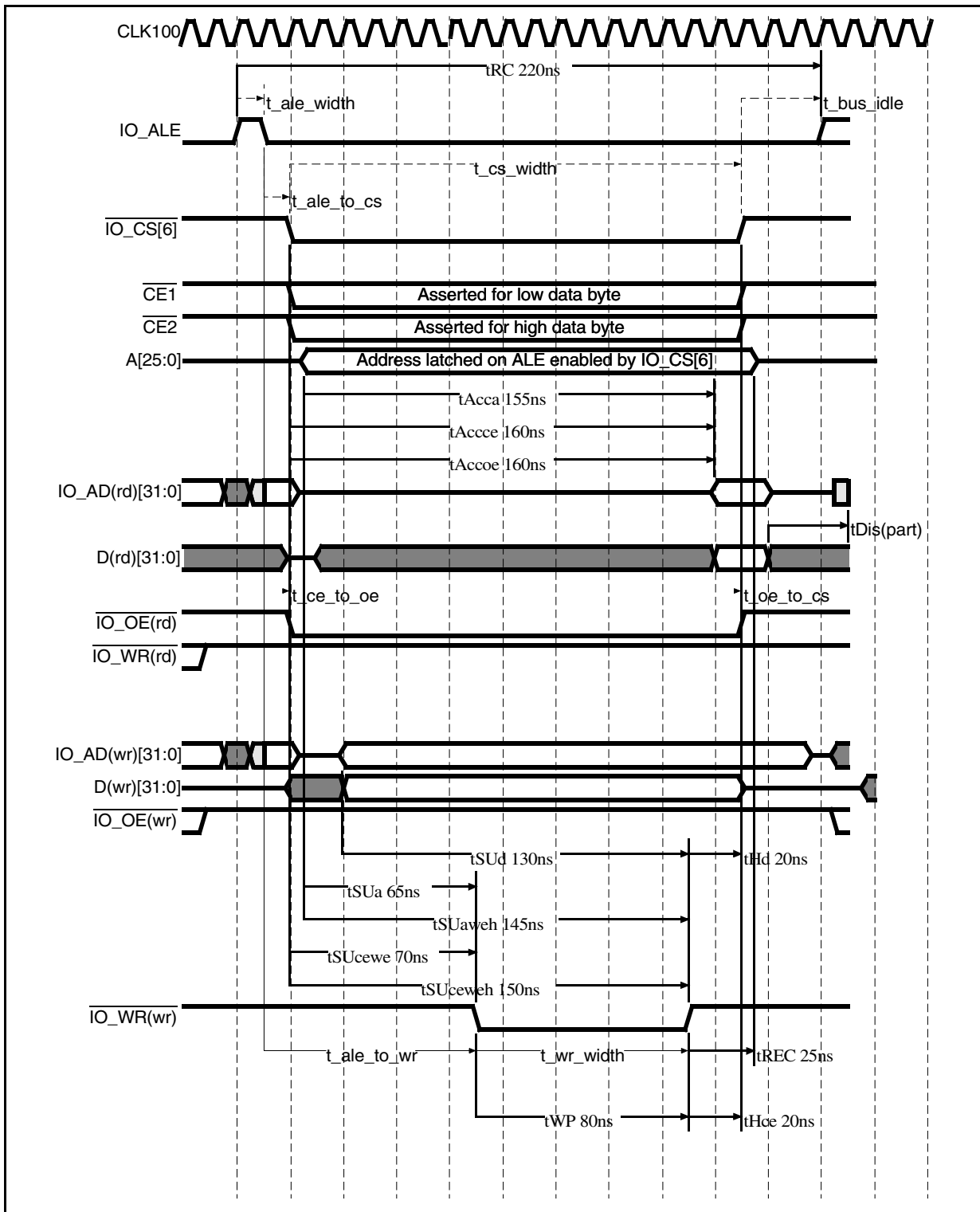


Figure 81: Example Flash Card Timing Diagram



Selection of most of the parameters is straight forward. For reads the chip select width controls the cycle and must be selected to be larger than the access time. In this case it is the  $t_{Acca(max)}$  that controls this. The chip select enables the address buffer, so the address only becomes valid after the output enable delay of the buffer (5ns in this case).

The output disable time ( $t_{Dis}$ ) sets the number of idle cycles. Assuming the next cycle is a write (i.e. the data buffer will drive on the next  $IO\_CS[6]$ ) there are two cycles or 20ns belonging to the next cycle that can be part of  $t_{Dis}$  (the cycle with  $IO\_ALE$  asserted and the  $ale\_to\_cs$  delay), thus 30 ns or three cycles are needed in the current access ( $t_{Dis(part)}$  on the diagram). There is always one idle cycle so  $idle\_cycle$  must be set to 2 to give the extra two idle cycles.

There are many parameters related to the write pulse. The controlling ones are the write width ( $t_{WP}$ ) which directly requires the  $wr\_width$  to be 8, and the address to write deassertion setup time ( $t_{SUaweh}$ ). Fortunately once these are set the data hold time is satisfied with the same chip select width as for the read.

**Table 272: Example Generic Bus Timing parameters**

Parameter	Value
ale_width	1
ale_to_cs	1
cs_width	17
cs_to_oe	0
oe_to_cs	0
ale_to_wr	8
wr_width	8
idle_cycel	2

## USING THE POWER OUTPUTS

If the PCMCIA mode is not enabled by the reset time configuration resistor then the three power control lines can be controlled by software directly through the **pcmcia\_cfg** register.

If the **pcmcia\_cfg\_pwr\_ctl** bit is set then the state of the GPIO[12] and GPIO[13] pins are used in the same way they would be (as CD1# and CD2#) in PCMCIA mode. They must be low to allow software to set the power control output lines, and the bits in the **pcmcia\_cfg** bits will be cleared if either goes high. This may be used in systems other than PCMCIA that need to implement hot-swap logic. The pins remain GPIO pins, if they are configured as outputs then the value being driven will be read back from the pin and used to control the power logic.

[Table 269 on page 386](#) and [Table 270 on page 387](#) show the logic of the power control lines when PCMCIA mode is disabled.



## PCMCIA CONTROLLER REGISTERS

The PCMCIA controller is enabled at reset time by a selection resistor on the generic address bus. The state can be read from the system configuration register. The other PCMCIA control registers are:

**Table 273: PCMCIA Configuration Register**

<b>pcmcia_cfg - 00_1006_1A60</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
0	pcmcia_cfg_reg	1'b0	When high, selects attribute memory. The PC_REG_L pin is the inverse of this bit.
1	pcmcia_cfg_3v_en	1'b0	When high, 3 volt supply enabled. In PCMCIA mode or if bit 9 is set this bit is set low whenever no card is detected. In Non-PCMCIA mode this pin allows software control of the EN3V output.
2	pcmcia_cfg_5v_en	1'b0	When high, 5 volt supply enabled. In PCMCIA mode or if bit 9 is set this bit is set low whenever no card is detected. In Non-PCMCIA mode this pin allows software control of the EN5V output.
3	pcmcia_cfg_vpp_en	1'b0	When high, programming voltage enabled. In PCMCIA mode or if bit 9 is set this bit is set low whenever no card is detected. In Non-PCMCIA mode this pin allows software control of the ENVPP output.
4	pcmcia_cfg_reset	1'b1	When high, apply reset to the pcmcia card. This bit is set when the system is reset or whenever no card is detected.
5	pcmcia_cfg_apron_en	1'b0	When high, enable auto power on.
6	pcmcia_cfg_cd_mask	1'b1	When high, disable Card detect interrupt. When low an interrupt is raised whenever the cd1_L and cd2_L (gpio[13:12]) lines either become both zero or stop being both zero. (The PCMCIA mode setting has no effect on this interrupt).
7	pcmcia_cfg_wp_mask	1'b1	When high, disable Write protect interrupt. When low an interrupt is raised whenever a change is detected on the WP (gpio[11])line (The PCMCIA mode setting has no effect on this interrupt).
8	pcmcia_cfg_rdy_mask	1'b1	When high, disable Card ready interrupt. When low an interrupt is raised whenever a change is detected on the READY (gpio[9])line (The PCMCIA mode setting has no effect on this interrupt).
9	pcmcia_cfg_pwr_ctl	1'b0	This bit is only used when the PCMCIA function is not selected at reset time. When high, the PCMCIA card detect inputs are used in driving the power select pins when PCMCIA mode is not selected..
15:10	reserved	7'h0	Reserved
63:16	notimp	48'bx	Not implemented.

**Table 274: PCMCIA Status Register**

<b>pcmcia_status - 00 1006 1A70</b>			
<b>READ ONLY, Read clears interrupt</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
0	pcmcia_status_cd1	ext	This bit reflects the state of the card detect 1 signal (PC_CD1_L) from the PCMCIA card. When both PC_CD1_L and PC_CD2_L are low the card is fully inserted.
1	pcmcia_status_cd2	ext	This bit reflects the state of the card detect 2 signal (PC_CD2_L) from the PCMCIA card. When both PC_CD1_L and PC_CD2_L are low the card is fully inserted.
2	pcmcia_status_vs1	ext	This bit reflects the state of the voltage sense 1 signal (PC_VS1_L) from the PCMCIA card.
3	pcmcia_status_vs2	ext	This bit reflects the state of the voltage sense 2 signal (PC_VS2_L) from the PCMCIA card.
4	pcmcia_status_wp	ext	This bit reflects the state of the write protect signal (PC_WP) from the PCMCIA card. When high, write protect is enabled.
5	pcmcia_status_rdy	ext	This bit reflects the state of the ready signal (PC_READY) from the PCMCIA card. When high, indicates the card is ready for access.
6	pcmcia_status_3v_en	1'b0	When high, indicates 3 volt VCC is enabled. This bit reflects the value driven on the PC_EN3V signal.
7	pcmcia_status_5v_en	1'b0	When high, indicates 5 volt VCC is enabled. This bit reflects the value driven on the PC_EN5V signal.
8	pcmcia_cd_change	1'b0	This bit is set when either of the card detect signals change. It is cleared by a read. If this bit is set an interrupt will be raised if it is not masked in the <b>pcmcia_cfg</b> register.
9	pcmcia_wp_change	1'b0	This bit is set when the write protect signal changes. It is cleared by a read. If this bit is set an interrupt will be raised if it is not masked in the <b>pcmcia_cfg</b> register.
10	pcmcia_rdy_change	1'b0	This bit is set when card ready signal changes. It is cleared by a read. If this bit is set an interrupt will be raised if it is not masked in the <b>pcmcia_cfg</b> register.
15:11	reserved	6'h0	Reserved
63:16	notimp	48'bx	Not implemented.



This Page is left blank for notes



## Section 13: GPIO

### INTRODUCTION

The part has a number of pins that are available for general use as inputs, outputs or interrupt inputs. These pins are controlled entirely by software. In addition, there are a number of pins allocated to other peripherals that may be used as general pins if the peripheral is not required.

### THE GPIO PINS

The GPIO pins can be configured for use as either inputs or outputs, and can be set to raise an interrupt. A single GPIO pin is shown in [Figure 82](#).

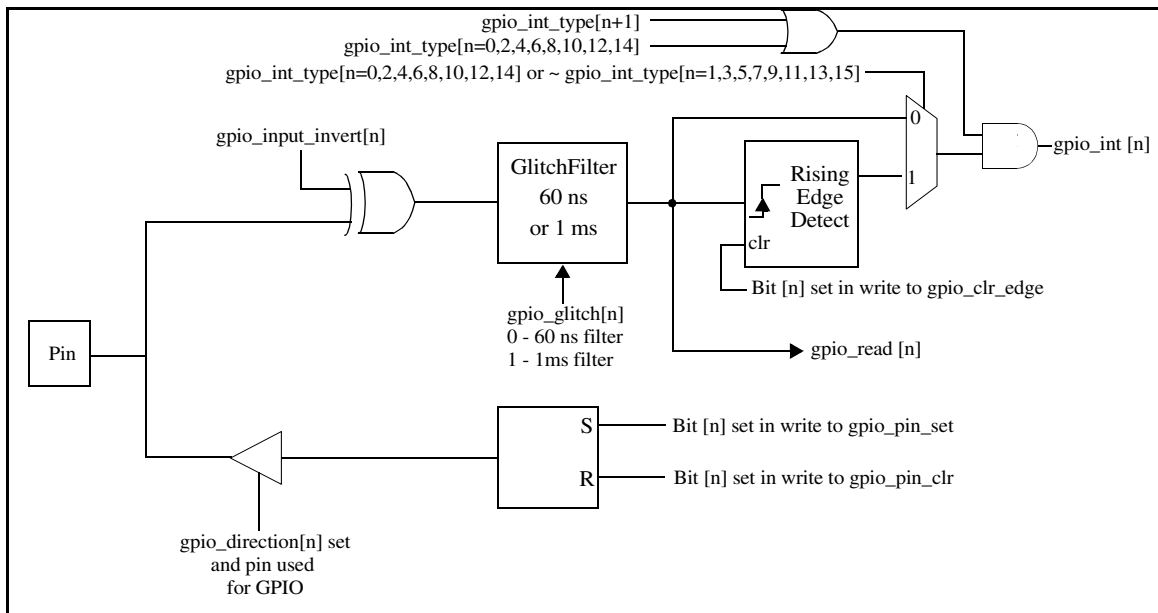


Figure 82: Single GPIO Pin Diagram

The **gpio\_direction** register sets the direction of each pin individually, bits in it should be set to enable the output buffer. When used as outputs (shown in the bottom of the figure) the CPU sets and clears bits in the output latch by writing 1s to the **gpio\_pin\_set** or **gpio\_pin\_clr** registers, and the pin will change state. The output register retains its state even when the line is configured as an input, so if an "open collector" output is required the output register only needs to be set low once and the direction of the pin can be changed from output (to pull low) to input (to float).

The input path is always active. An optional inverter is enabled by setting the corresponding bit in the **gpio\_input\_invert** register. The 100MHz reference clock is used to synchronize the signal to the internal logic and to provide filtering against glitches. By default the line has to change state for about 60ns before being recognized (a 60ns glitch filter), but the filter may be increased to provide a 1 ms glitch filter. The state of the gpio pin (after inversion and the glitch filtering) is readable from the **gpio\_read** register.





GPIO lines can be enabled as interrupts in pairs. For each pair, both can have their interrupts disabled, both can be level interrupts, both can be edge sensitive interrupts or one can be level and the other an edge interrupt. This is selected in the **gpio\_int\_type** register. When a level interrupt is selected the input after inversion and glitch filtering is directly passed to the interrupt mapper in the SCD (see [Section: "The Full Interrupt Mapper" on page 50](#)) where it must be high to signal an interrupt. If the edge detector is enabled, the signal to the SCD is asserted when a rising edge is detected on the output of the glitch filter, the signal is cleared when the corresponding bit is set in a write to the **gpio\_clr\_edge** register. A negative edge can be detected by inverting the input. Masking of the GPIO interrupt lines is also done in the interrupt mapper. The interrupt function is not affected by the direction of the line, if the GPIO pin is set as an output then the interrupt will be raised whenever the output is set appropriately (after the glitch filtering delay).

Some of the GPIO pins are used by the PCMCIA controller and some to provide parity on the generic bus. These functions are set by reset time configuration resistors on the generic bus AD lines. When used for PCMCIA operation the input inverters must be disabled by leaving **gpio\_input\_invert** bits [15:6] in their default state as zeros. The GPIO signals are summarized in the table below.

**Table 275: GPIO Pins and Alternate Uses**

GPIO Bit	Alternate Name	Alternate Use
1:0	S1:0_RSTROBE	Synchronous serial output strobes.
5:2	IO_ADP[3:0]	Generic bus parity.
6	PC_CE1_L	PCMCIA Card enable output.
7	PC_CE2_L	PCMCIA Card enable output.
8	PC_RESET	PCMCIA card reset output.
9	PC_READY	PCMCIA ready input.
10	PC_REG_L	PCMCIA regular/attribute select output.
11	PC_WP	PCMCIA write protected input.
12	PC_CD1_L	PCMCIA card detect input (the 1ms glitch filter is always used).
13	PC_CD2_L	PCMCIA card detect input (the 1ms glitch filter is always used).
14	PC_VS1_L	PCMCIA voltage sense input (the 60 ns glitch filter is always used).
15	PC_VS2_L	PCMCIA voltage sense input (the 60 ns glitch filter is always used).

## GPIO REGISTERS

There are eight GPIO registers.

**Table 276: GPIO Edge Clear Register**

<b>gpio_clr_edge - 00_1006_1A80</b> <b>WRITE ONLY</b>			
Bits	Name	Default	Description
15:0	edge_clr	16'h0	Writing a 1 in a bit position in this register will clear the corresponding edge detector.
63:16	notimp	48'bx	Not Implemented.

**Table 277: GPIO Interrupt Type Register**

<b>gpio_int_type - 00_1006_1A88</b>			
Bits	Name	Default	Description
1:0	int_type0	2'b0	The two bit fields set the interrupt type for each pair of pins. 00: Disable both lines. The interrupts will never be raised. 01: Both GPIO pins are edge sensitive interrupts. 10: Both GPIO pins are level sensitive interrupts. 11: The even numbered pin is an edge sensitive interrupt The odd numbered pin is a level sensitive interrupt.
3:2	int_type2	2'b0	
5:4	int_type4	2'b0	
7:6	int_type6	2'b0	
9:8	int_type8	2'b0	
11:10	int_type10	2'b0	
13:12	int_type12	2'b0	
15:14	int_type14	2'b0	
63:16	notimp	48'bx	Not Implemented.

**Table 278: GPIO Read Register**

<b>gpio_read - 00_1006_1AA0</b> <b>READ ONLY</b>			
Bits	Name	Default	Description
15:0	gpio	16'h0	Reads give value on each pin, after the optional inverter and glitch filtering.
63:16	notimp	48'bx	Not implemented.

**Table 279: GPIO Input Invert Control Register**

<b>gpio_input_invert - 00_1006_1A90</b>			
Bits	Name	Default	Description
15:0	invert	16'b0	Each bit controls the input inverter for the corresponding GPIO pin. If a bit is set then the value used internally and read in the <b>gpio_read</b> register will be the inverse of the value on the pin.
63:16	notimp	48'bx	Not implemented.

**Table 280: GPIO Glitch Filter Select Register**

<b>gpio_glitch - 00_1006_1A98</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
1:0	gpio_glitch	2'h0	When high the input from the pin has a 1 ms glitch filter, when low a 60 ns glitch filter.
5:2	gpio_glitch	4'h0	When high the input from the pin has a 1 ms glitch filter, when low a 60 ns glitch filter. If generic bus parity is enabled these bits are ignored.
15:6	gpio_glitch	10'h0	When high the input from the pin has a 1 ms glitch filter, when low a 60 ns glitch filter. In PCMCIA mode these bits are ignored.
63:16	notimp	48'bx	Not implemented.

**Table 281: GPIO Direction Register**

<b>gpio_direction - 00_1006_1AA8</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
1:0	gpio_dds	2'h0	High for output driven from the output latch, low for input. If the synchronous serial interface RstrobeOut output is enabled this register is ignored and the pin will be an output.
5:2	gpio_dds	4'h0	High for output, low for input. When generic bus parity is enabled these bits are ignored.
15:6	gpio_dds	10'h0	High for output, low for input. When PCMCIA is enabled these bits are ignored.
63:16	notimp	48'bx	Not implemented.

**Table 282: GPIO Pin Clear Register**

<b>gpio_pin_clr - 00_1006_1AB0 WRITE ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	pin_clr	16'h0	Writing a 1 to a bit position will clear that bit in the output latch.
63:16	notimp	48'bx	Not implemented.

**Table 283: GPIO Pin Set Register**

<b>gpio_pin_set - 00_1006_1AB8 WRITE ONLY</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
15:0	pin_set	16'h0	Writing a 1 to a bit position will set that bit in the output latch
63:16	notimp	48'bx	Not Implemented.

## OTHER PINS THAT CAN BE USED

The part has many built in peripherals, and in any particular application it is possible that some of them are not used. Pins on some of the peripherals can be used to provide additional software controlled input, output or interrupt lines. This section describes some of these.

### SERIAL PORTS

If a serial port is unused most of its pins can be reused. The port should be configured in asynchronous mode. All of the input pins (except for the data input pin) can be read through the **duart\_in\_port** register. The output handshake pins can be used by configuring them for software control. In an extreme case it would be possible to use the data out pin by enabling the transmitter and either letting the line idle (by sending nothing) to give a high output, or sending a break to give a low output.

The following table summarizes the pins that can be directly used.

**Table 284: Other Pins that can be Used as General Inputs or Outputs**

Pin Name	Direction	Channel A access (n=0)	Channel B access (n=1)
Sn_RTS_TSTROBE	Output	op[0]	op[1]
Sn_COUT	Output	op[2]	op[3]
Sn_CTS_TCLKIN	Input transition detect	ip[0]	ip[1]
Sn_CIN_RCLKIN	Input transition detect	ip[2]	ip[3]
Sn_TIN	Input	ip[4]	ip[5]
Sn_RIN	Input	ip[6]	ip[7]

See [Section: "Operation" on page 322](#) for how to configure the UART and control the I/O lines.

### PCI

If the PCI bus is not used the PCI interrupt lines can be used as active low level interrupt inputs, as can the SERR and PERR signals. If the PCI is in device mode the P\_INTB\_L, P\_INTC\_L and P\_INTD\_L lines can be used as active low level interrupt lines.

### MACs

The MAC management interface provides the only pins that can easily be used from an unused MAC interface. The MDIO line provides input and output, and the MDC line is output only. The GENO pin from a MAC is available as a general output unless the interface is in 16 bit bypass mode. The FIFO mode of the MAC may be used for bulk data DMA transfers, but there is no easy way to use it for programmed I/O.

### PCMCIA POWER CONTROL PINS

The PCMCIA power control pins are three dedicated outputs that are forced low when the part is reset. If PCMCIA support is not required then these lines can be used as general outputs by setting the PCMCIA power logic for software power control and writing the power control bits. This is described in [Section: "Using the Power Outputs" on page 392](#).



---

This Page is left blank for notes



---

This Page is left blank for notes



## Section 14: Serial Configuration Interface

### INTRODUCTION

There are two serial configuration interfaces based on the SMBus. The interfaces provide hardware assistance for simple read and write of slave devices with the part as the bus master. The hardware assistance provides a subset of the SMBus version 1.1 specification published by the Smart Battery System Implementers Forum. The bus clock is programmable, it should be set in the 10-100kHz range for conformance operation, but can go higher. In particular many devices can use a 400kHz clock.

### SMBUS OVERVIEW

There are a large number of devices that can be configured using a two wire serial interface with one clock wire and one data wire. The interface to these devices is basically the same, but there are differences in timing and electrical specifications. The serial configuration interface is based on version 1.1 of the System Management Bus (SMBus) standard from the Smart Battery System Implementers Forum (SBS-IF), but has been generalized to support other devices. This interface only implements a subset, so is not fully compliant with the SMBus 1.1 specification. This section provides a brief introduction to the interface, for full details see the SMBus specification that may be found on the SMBus web site at <http://www.smbus.org> and the related documents.

Devices on the serial configuration bus may be masters or slaves (or both). The interface is always a master. Masters can issue commands, provide the clock and control the transfer. Slaves respond to commands from masters and use the supplied clock. Each slave has a unique 7 bit address (or a few addresses) that the master uses to select it. Transactions always start with the master sending a byte containing the address of the slave and a single bit that indicates if the master is reading from the slave or writing to it. Following this one or more bytes of data are transferred. The SMBus adds a protocol layer above this basic transport.

### TRANSPORT PROTOCOL

Devices connected to the serial configuration bus use open drain outputs on both the clock and data lines. Pull-up resistors restore the lines high when no device is driving. The inputs should not be pulled above 3.3V.

Both the data (SDA) and clock (SCL) lines are high when idle. In normal operation SDA will only change while SCL is low. SDA changing with SCL high is used to signal control flags. [Figure 83](#) shows the three conditions for bit transfer on the SMBus:

#### Start

A Start condition is used to indicate the beginning of a transaction. It consists of SDA changing from high (idle) to low while SCL remains high (idle).

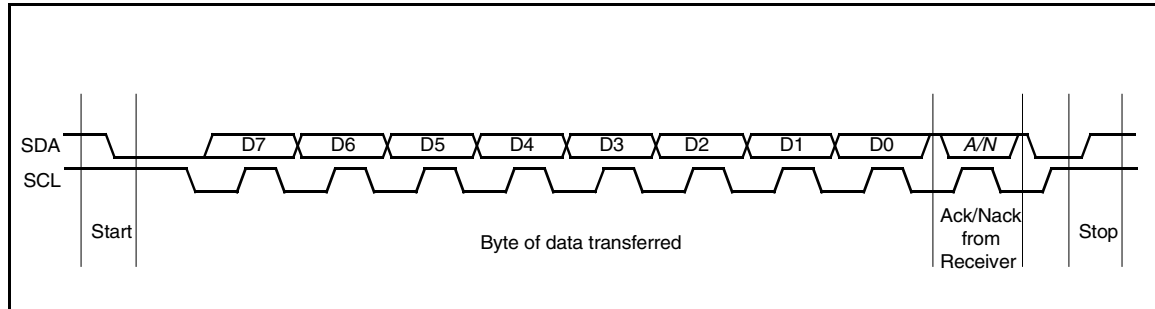
#### Data bit

A data bit is transferred by putting it on the SDA line with the SCL clock low and pulsing the clock high and then low with SDA stable. Data bytes are sent with the most significant bit first.



**Stop**

A stop condition signals the end of a transaction. It consists of SDA making a transition from low to high (idle) while SCL is high (idle).



**Figure 83: SMBus Signaling Start, Data Transfer and Stop**

A master will initiate a transfer by sending a start condition. This causes all slaves to listen to the next byte which includes the 7 bit address and a single read/write bit. Following this the master sets SDA idle and pulses SCL once more. If a slave matches the address it will acknowledge by holding SDA low. Bytes are then transferred in the direction marked by the read/write bit, with the receiver (slave during a write, master during a read) acknowledging after each. Bytes continue to be transferred until the slave does not acknowledge a byte or the master decides all bytes have been transferred. The transaction is then ended by the master signalling a stop, which returns both lines to idle.

There may be multiple masters on the bus. Most of the time they will see the start and stop framing of other masters to avoid collision, however if two masters start at the same time they may collide. In this case the first time their SDA signals differ the master that drives the line to zero wins arbitration and the other must back off. When forced to back off the interface will retry after it sees a stop condition, or if SCL and SDA are both high for 50 us (this is defined as an idle condition).

The timing specification for slave devices tends to vary with frequency. To comply with both the standard timings at 100 kHz and typical timings at 400 kHz the interface produces a SCL clock signal with a 9:7 ratio of time low:time high.

The slave device is permitted to slow the clock by stretching the low period. If a slave has stretched the clock for 25 ms then an error has occurred and the master will abort the transfer.





## TRANSPORT PROTOCOL RESET

A system reset during an SMBus transaction can leave the bus in an unuseable state. The reset will cause the master to release the clock line and SCL will therefore go high. But many slaves do not detect the system reset, and if a slave is being read it is not permitted to change the state of the data line while the clock is high. Following a reset it is possible for a slave to be driving the SDA low while SCL is high. All masters will backoff from using the bus since it looks as though it is in use. A START or STOP cannot be sent because the slave is forcing the SDA low.

The interface will attempt to clear this bad state. If SDA is detected low when the interface is released from reset it will issue clock pulses on SCL until SDA goes high. In the absence of protocol errors, at most 9 clock pulses are needed to ensure that any slave that believes a read is in progress will complete transfer of a byte (of 0s) and see a NACK, causing it to end the transaction. If SDA goes high before the slave believes the transaction is finished the interface will stop providing clock pulses but the bus will be left in a state where a new master can initiate a new transaction. The START of the next transaction will ensure all slave state machines are resynchronized to the command stream. If a genuine transaction is in progress under the control of another master the part's attempt to clear state will at worst result in clock cycle stretching and should not interfere with the transaction.

During the reset procedure accesses to the control registers are held off. This will at most last for the first 90us after reset so it is unlikely to be noticed in practice.

## SMBUS PROTOCOL

The SMBus protocol defines a number of transfer types between the host and peripherals. The interface implements all of these except for process call and block transfers. The latest revision of the SMBus includes Packet Error Code (PEC) checking, this appends an 8-bit CRC (using the polynomial  $x^8 + x^2 + x + 1$ ) at the end of the message. The CRC is calculated across all bytes in the transfer (address, mode and data). The receiver can verify correctness of the CRC and issues a Not Acknowledge if it is incorrect.

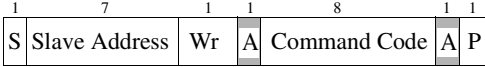
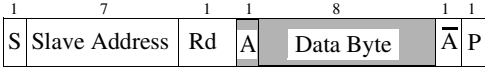
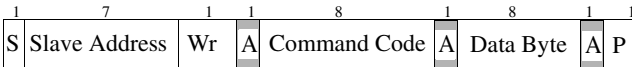
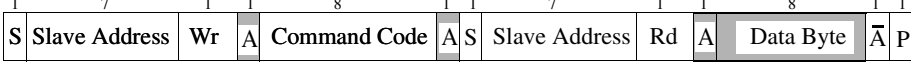
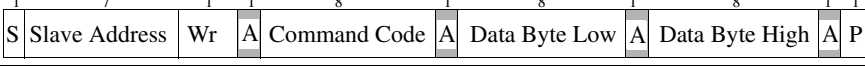
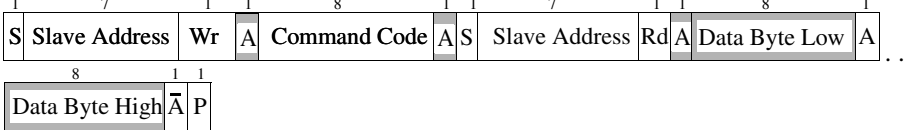
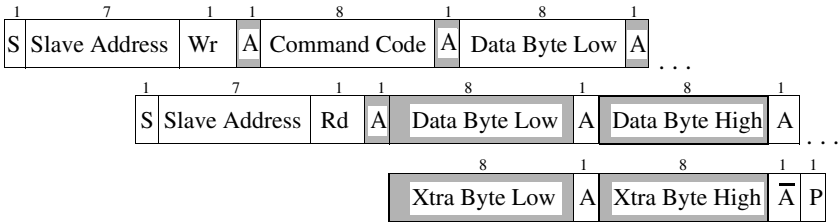
The interface allows use of PEC by optionally adding a byte at the end of a transfer, but the hardware will neither generate nor check the PEC byte. Software should calculate the correct value to be sent with writes and must load this into the **smb\_pec** register before starting the transfer. If the CRC check fails at the slave device it will NACK the PEC byte and an error will be reported in the **smb\_status** register. When the interface receives PEC it is always acting as the transaction master, so the hardware will always NACK the byte and signal a stop to terminate the transfer. Software should then check the **smb\_pec** register and can retry the transfer if it detects an error.

The supported SMBus commands are listed in [Table 285](#), along with the parameters they use.

**Table 285: Supported SMBus Transfer Types**

Transfer Type (smb_tt)	Command (smb_cmd)	1st Byte (smb_data[7:0])	2nd Byte (smb_data[15:8])	Description
In the figures below S = start, P = stop, A = ACK, $\bar{A}$ = NACK, Wr (write) = 0, Rd (read) = 1 Bits driven by the slave are shaded				
Quick Command (110)	No	No	No	Send a single bit (in the R/W bit position) to the slave.

**Table 285: Supported SMBus Transfer Types (Cont.)**

Transfer Type (smb_tt)	Command (smb_cmd)	1st Byte (smb_data[7:0])	2nd Byte (smb_data[15:8])	Description
Send Byte (000)	Yes	No	No	Send a single byte to the slave.
				
Receive Byte (101)	No	No	Data	Read a single byte from the slave using a simple read with no associated command.
				
Write Byte (001)	Yes	Data	No	Send a command byte and a single data byte to the slave.
				
Read Byte (011)	Yes	Data	No	Send a command to the slave and read back a byte.
				
Write Word (010)	Yes	LSB	MSB	Send a command and two bytes to the slave.
				
Read Word (100)	Yes	LSB	MSB	Send a command to the slave and read two bytes.
				
EEPROM Read (111)	Yes	LSB	MSB	Send a command and one byte to the slave and read four bytes back. This matches a 32 bit read from a >16 kbit EEPROM.
				

If the device that is addressed is not an SMBus device, then it will most likely use a different protocol. In this case a mapping will need to be done from the SMBus commands to those that the device supports. The Send Byte and Receive byte commands perform the fundamental transfers, but in many cases the other commands can be used (for example many memory devices can use the Read Word command to set the address pointer and read back two bytes). Since it is not interpreted by the hardware the PEC byte can be used as an additional byte in the transfer.

## EXTENDED PROTOCOL

The Extended protocol mode of the interface supports a much wider range of devices by allowing transactions other than those specified in SMBus. Up to seven bytes can be written to a device, and up to seven bytes read.

An extended mode transaction consists of a command/address being written to the device followed by either data being written to the device or a repeated start and data being read from the device.

A write transaction consists of writing a command/address, with the format selected from [Table 286](#), followed by write data, selected from [Table 287](#). If PEC is enabled then an additional byte from the **smb\_pec** register will be sent before the STOP. (The interface places no interpretation on the bytes, the names command/address, data and PEC just refer to conventional use of these bytes.)

A read transaction consists of writing the command/address, as shown in [Table 286](#), followed by a repeated START and reading data, as shown in [Table 288](#). If PEC is enabled then an additional byte will be read into the **smb\_pec** register before the NACK and STOP. The command/address may be null, in which case the START shown in [Table 288](#) will be a standard START not a repeated one.

The extended mode is invoked by setting the extend bit in the **smb\_start** register, this changes the interpretation of some of the other fields. Thus SMBus and extended transactions can be freely mixed.

**Table 286: Command/Address Options**


smb_start[12:11]	Address Type	Operation
00	Just Read	This is for reads that do not have an address phase. The Start (and slave address) that is part of the read becomes a regular Start rather than a Repeated Start.
01	Slave Address	
10	Byte Address	
11	16 bit Address	

**Table 287: Write Data Options**

smb_start [10:8]	Data Format	Operation
000	1 Byte	
001	2 Bytes	
010	3 Bytes	
011	4 Bytes	



**Table 287: Write Data Options (Cont.)**

smb_start [10:8]	Data Format	Operation
100	No Data	
101 110 111	Reserved	Reserved encodings. UNPREDICTABLE operation.

**Table 288: Read Data Options**

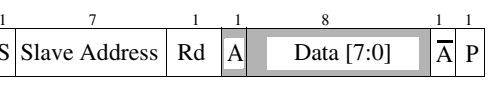
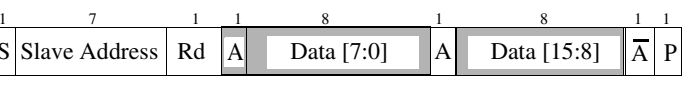
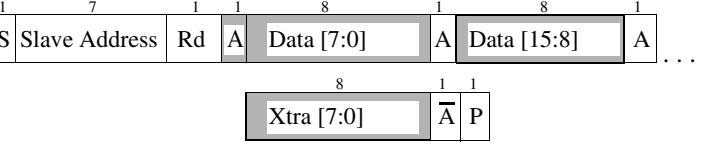
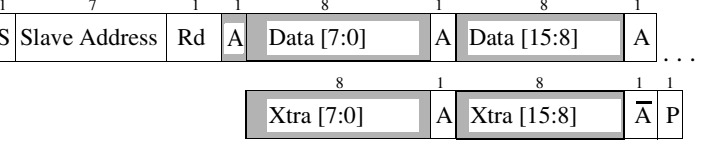
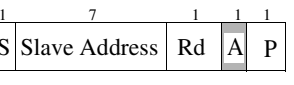
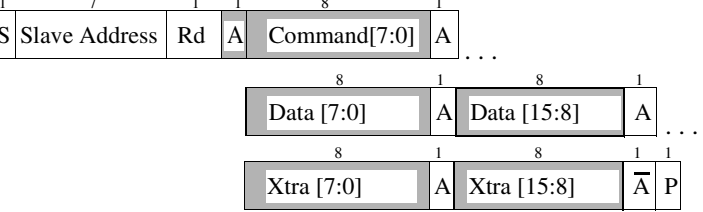
smb_start [10:8]	Data Format	Operation
000	1 Byte	
001	2 Bytes	
010	3 Bytes	
011	4 Bytes	
100	No Data	
101	5 Bytes	

Table 288: Read Data Options (Cont.)

smb_start [10:8]	Data Format	Operation
110	6 Bytes	<p>The diagram illustrates the bit-level structure of the SMBus transaction for Read Data Options (Cont.). It shows three rows of data fields, each with a 1-bit start/acknowledge bit (S, Rd, A) and an 8-bit data field. The first row contains Slave Address (7 bits), Rd (1 bit), Command[7:0] (8 bits), A (1 bit), Command[15:8] (8 bits), and A (1 bit). The second row contains Data [7:0] (8 bits), A (1 bit), Data [15:8] (8 bits), and A (1 bit). The third row contains Xtra [7:0] (8 bits), A (1 bit), Xtra [15:8] (8 bits), and A (1 bit) followed by a Parity bit (P) with a bar over it. Ellipses indicate that the transaction continues with more data bytes.</p>
111	Reserved	Reserved encoding. UNPREDICTABLE operation.

## PROGRAMMING MODEL

### USING SMBUS PROTOCOLS

A transfer is configured by setting data values in the SMBus registers and writing the transfer type to the **smb\_start** register. This starts the operation. If the remote device fails to acknowledge when expected to then the transfer is abandoned with an error. If the interface loses arbitration or detects over-long slave stretching then the transfer is aborted. The interface will retry an aborted transfer 15 times before abandoning it with an arbitration error.

Reading from a slave on the serial bus involves these steps:

- 1 Wait until the busy bit in the **smb\_status** register is clear. The interface will ignore accesses while the bit is set.
- 2 For a Read Byte (but not a Receive Byte) transfer specify the command to be sent by writing to the **smb\_cmd** register.
- 3 Specify the device address and the transfer type by writing to the **smb\_start** register. This will cause the transfer to begin. The busy bit in the **smb\_status** register will be set to a one as long as the transfer is in progress.
- 4 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the **smb\_status** or by enabling the **smb\_finish** interrupt. The error bit indicates that an acknowledgement was not received from the slave device when the master attempted to start the transaction.
- 5 Read the data from the **smb\_data** register. If one byte was requested, only the lsb field will be valid.
- 6 If PEC is enabled the **smb\_pec** register should be read and checked.

Writing to a slave on the serial bus involves these steps:

- 1 Wait until the busy bit in the **smb\_status** register is clear. The interface will ignore accesses while the bit is set.
- 2 Specify the command to be sent as the first byte by writing it in the **smb\_cmd** register.
- 3 If data is being sent write it in the **smb\_data** register.
- 4 If PEC is being used the CRC must be calculated by the software and written to the **smb\_pec** register.
- 5 Specify the device address and the transfer type by writing to the **smb\_start** register. This will cause the transfer to begin. The busy bit in the **smb\_status** register will be set to a one as long as the transfer is in progress.

- 6 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the **smb\_status** register or by enabling the `smb_finish` interrupt. The error bit indicates that an acknowledgement was not received from the slave device at some point during the transaction. (It is important to check the data sheet for the slave device, some will always terminate a transaction by not sending an acknowledgement).

A quick command can be sent with these steps:

- 1 Wait until the busy bit in the **smb\_status** register is clear. The interface will ignore accesses while the bit is set.
- 2 Specify the device address, the command bit (which is sent as the R/W bit in a Quick Command) and the transfer type by writing to the **smb\_start** register. This will cause the transfer to begin. The busy bit in the **smb\_status** register will to be set to a one as long as the transfer is in progress.
- 3 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the **smb\_status** register or by enabling the `smb_finish` interrupt. The error bit indicates that an acknowledgement was not received from the slave device.

An EEPROM read is a write with a command and one data byte followed (without a stop) by a read of four data bytes (and optional PEC). Typically the command and data byte will contain the address to be read. The command sequence reflects this:

- 1 Wait until the busy bit in the **smb\_status** register is clear. The interface will ignore accesses while the bit is set.
- 2 Specify the command (high address bits) to be sent as the first byte by writing it in the **smb\_cmd** register.
- 3 Specify the data (low address bits) to be sent by writing it in the **smb\_data** register.
- 4 Specify the device address (on some EEPROMs the low three bits of the device address provide an additional three address bits) and the transfer type by writing to the **smb\_start** register. This will cause the transfer to begin. The busy bit in the **smb\_status** register will to be set to a one as long as the transfer is in progress.
- 5 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the **smb\_status** register or by enabling the `smb_finish` interrupt. The error bit indicates that an acknowledgement was not received from the slave device when the master attempted to start the transaction or as the address was written.
- 6 Read the data from the **smb\_data** (from address and address+1) and the **smb\_xtra** (from address+2 and address+3) registers.
- 7 Most EEPROMs do not support PEC. Using standard parts if PEC is enabled the **smb\_pec** register will contain a fifth byte of data (from address+4).

The SMBus specification includes a Process Call command and Block Read and Write commands for transferring up to 32 bytes of data. The interface does not support these operations.

## USING EXTENDED PROTOCOLS

A transfer is configured by setting data values in the SMBus registers and writing the transfer type with the extend bit set to the **smb\_start** register. This starts the operation. If the remote device fails to acknowledge when expected, then the transfer is abandoned with an error. If the part loses arbitration or detects over-long slave stretching then the transfer is aborted. The interface will retry an aborted transfer 15 times before abandoning it with an arbitration error.

Reading from a slave on the serial bus involves these steps:

- 1 Wait until the busy bit in the **smb\_status** register is clear. The interface will ignore accesses while the bit is set.
- 2 If the read is preceded by an address/command (other than just a Slave Address) write the **smb\_cmd** register with the 8 or 16 bit data to be sent.
- 3 Specify the device address and the transfer type by writing to the **smb\_start** register. This will cause the transfer to begin. The busy bit in the **smb\_status** register will be set to a one as long as the transfer is in progress.
- 4 When the transaction finishes the busy bit will be cleared. This can either be detected by polling the **smb\_status** or by enabling the **smb\_finish** interrupt. The error bit indicates that an acknowledgement was not received from the slave device when the master attempted to start the transaction.
- 5 Read the data from the **smb\_cmd**, **smb\_data** and **smb\_xtra** registers.
- 6 If PEC is enabled the **smb\_pec** register should be read.

Writing to a slave on the serial bus involves these steps:

- 1 Wait until the busy bit in the **smb\_status** register is clear. The interface will ignore accesses while the bit is set.
- 2 If an address/command is needed write 8 or 16 bits to the **smb\_cmd** register.
- 3 Write the data to the **smb\_data** and **smb\_xtra** registers.
- 4 If PEC is enabled an extra data byte should be written to the **smb\_pec** register.
- 5 Specify the device address and the transfer type by writing to the **smb\_start** register. This will cause the transfer to begin. The busy bit in the **smb\_status** register will be set to a one as long as the transfer is in progress.

The SMBus interface is accessed through a number of registers in the I/O portion of the memory map. The two interfaces are identical, register names in this discussion should have **\_0** or **\_1** appended to indicate the interface. There are two interrupts generated from each interface. Both interrupts are maskable through the control register. The **smb\_finish** interrupt is generated when the interface completes an operation and the **smb\_busy** status bit changes from one to zero, the interrupt is cleared by a read from the status register. The **smb\_error** interrupt is generated when an error condition occurs, both the interrupt and status bit are cleared by writing a 1 to the error bit in the status register.

Prior to using the configuration interface the clock speed should be set. This is derived from the 100 MHz reference clock as configured in the **smb\_freq** register. The two standard frequencies are 100 kHz and 400 kHz, but a much wider range can be programmed. Since the setup and hold time parameters vary between different slave devices it may be necessary to run the interface slightly slow to ensure reliable operation.



## DIRECT ACCESS

The interface allows the hardware assist to be disabled giving direct ("bit-banged") access to the SDA and SCL lines. The CPU executes the whole protocol in software. This is useful for connecting to devices that deviate from the standard.

Direct access is enabled through the **smb\_control** register. Two bits in the **smb\_control** register are used to set the SDA and SCL lines to drive low or go high impedance, and two bits in the **smb\_status** register reflect the current state of the lines. The output from the SMB bit clock is provided in the status register since it may be useful for timing transfers.

## BOOTING USING AN SMBUS EEPROM

The part can be configured to boot using an EEPROM connected to SMBus interface 0. When configured this way accesses to the generic bus chip select 0 space are converted into SMBus accesses using either the EEPROM Read or Read Word protocols. Since there is a large protocol overhead and the SMBus will be running at the default 100 kHz, accessing the bootstrap code in this way is slow. Typically, a small primary bootstrap program would be run from the EEPROM (which can also store the ethernet ids for the part), this would do sufficient initialization to load the system code from a master device.

The boot\_type is configured using resistors on the generic bus pins IO\_AD[18:17] as described in [Section: "Reset" on page 26](#). The value on these is readable in the System Configuration register ([Table 15 on page 43](#)). While bit 18 is set in this register the SMBus 0 interface will only be used for servicing accesses to generic bus chip select 0.

While the SMBus interface is being used for booting any access to its control registers have UNPREDICTABLE results. Once booting is complete software can clear bit 18 in the System Configuration register to allow normal SMBus use. This will return the chip select 0 space to using the generic bus to service accesses. Following SMBus use software must set the chip select 0 timing parameters (even if it is setting the default values) before any access is made to the chip select 0 space or the results will be UNPREDICTABLE. When the system is reset by any method that causes the configuration bits to be sampled (See [Table 8 on page 27](#)), bit 18 in the System Configuration register will be set again, and the SMBus will be used for booting. If the watchdog timer is set to reset only one (or both) of the CPUs then the SCD will not be reset and the SMBus will not be reselected. In this configuration either the SMBus interface must be dedicated to be used by the generic chip select 0, or a ROM must be put on the generic bus to service the watchdog resets.

The boot mode works with EEPROMs that have the standard interface and use SMBus address 1010xxx. The low bits from address that the CPU issues to the generic bus (that would normally be put out on IO\_AD) is sent to the SMBus interface, which runs an addressed cycle to the EEPROM and returns 32 bits of data that would normally be placed in the **smb\_data** and **smb\_xtra** registers. Two protocols are in use for these EEPROMS:

### EEPROMS <= 16k bit

If the boot\_type is configured as 2'b10 the part will boot from a small EEPROM (up to 2K bytes) accessed using a modified Read Word protocol that fetches four bytes:

The upper 4 bits of the SMBus device address are set to 4'b1010.

The low three bits of the SMBus device address are set to bits [10:8] of the address being accessed.

The Command byte is set to bits [7:0] of the address being accessed.



### EEPROMS > 16k bit (boot type = 2'b11)

If the boot\_type is configured as 2'b11 the part will boot from a large EEPROM (bigger than 2K bytes) accessed using the EEPROM Read protocol.

The upper 4 bits of the SMBus device address are set to 4'b1010.

The low three bits of the SMBus device address are set to bits [18:16] of the address being accessed.

The Command byte is set to bits [15:8] of the address being accessed.

The Data byte is set to bits [7:0] of the address being accessed.

The data that returns from the SMBus interface is aligned and assembled to satisfy the request by the generic bus logic as if it had been returned by a 32 bit wide ROM attached to the generic bus. Note that if the SMBus transaction fails (for example the interface loses arbitration 15 times) then there is no way to recover and the boot process will hang (in this situation the CPU graduation timer will eventually trigger a Machine Check exception, but since the exception vector is also directed to the boot rom it will also be unable to make progress).

## SWITCHING FROM SMBUS MODE

Care must be taken when switching out of the SMBbus boot mode. There must be no accesses in progress to the chip select 0 region when the switch is made or the system will behave in UNDEFINED ways (for example a response could be lost). Instruction fetches are a particular concern and careful location of instructions is required to avoid them causing a problem. Two cases will be described as examples, in both cases the system is initially running from SMBus boot space in the generic chip select 0 region. (These examples are selected to provide simple illustration of the problem and are not representative of real systems.)

The first case is when the SMBus boot bit is cleared and the system continues running from a device in the chip select 0 generic bus region. This could be done when the SMBus code is used to configure the region with a different timing from the default.

```
// Fetching from cs0 region from SMBus
// t1 points to generic CS0 config registers
// t3 points to SCD registers
addr: // this is on a cache block boundary
      sh    t5, io_ext_time_cfg0(t1)//set timing registers
      sh    t6, io_ext_time_cfg1(t1)
      sync
      sync
addr+16:
      sh    t4, io_ext_cfg(t1)//set config
      sd    t7, system_cfg(t3)//remove SMBus boot
      sync
      sync
addr+32:
      // Fetched from cs0 on the generic bus
```

In this example the chip select 0 timing registers are configured in one instruction bundle. The `sync` instructions serve to pad out the bundle and prevent the next four instructions being fetched until the stores have been issued (the SB-1 will only issue an uncacheable instruction fetch when the previous instructions have graduated). The second group of four instructions write to the configuration to complete the setup and to the `system_cfg` register to disable SMBus booting. Again the `sync` instructions are used to ensure these writes have issued. This is sufficient to ensure the mode has changed and the next instruction fetch will be serviced from the generic bus.

The second example is switching to the chip select 1 region at the same address. This can be done to move to a generic bus device while leaving the SMBus mode enabled to allow a soft reset to revert back to SMBus mode. Again care must be taken during the switch-over.

```
// Fetching from cs0 region from SMBus
// t0 points to generic CS0 config registers
// t1 points to generic CS1 config registers
// t2-7 have configuration values
addr: // this is on a cache block boundary
      sh      t2, io_ext_time_cfg0(t1)//set CS1 timing registers
      sh      t3, io_ext_time_cfg1(t1)
      sh      t4, io_ext_cfg(t1)//set CS1 config
      sync
addr+16:
      sh      t5, io_ext_size(t1)//set CS1 size (to 0x3f = 4MB)
      sh      t6, io_ext_base(t0)//move CS0 base (to 00_2000_000)
      sh      t7, io_ext_base(t1)//set CS1 base (to 00_1FC0_0000)
      sync
addr+32:
      // Fetched from cs1 on the generic bus
```

Here the timing is set for the chip select 1 region in the first instruction bundle. The second group sets the size of the chip select 1 region, changes the base of chip select 0 to move it out of the way and configures the base of chip select 1 to `00_1FC0_0000`. It is important that the second group of instructions are together since they change the target of the next instruction fetch. Again the `sync` instruction is used to ensure that the stores have completed before the CPU will issue the next instruction fetch.

If the chip select region 1 were located at a different address then there would be no need to move the chip select region 0 base address. In this case the two CS1 stores (of `t5` and `t7`) would be the first two instructions in the bundle, the third instruction would be the branch to the new address range and the `sync` would be in the branch delay slot. Having the branch third and the `sync` in the delay slot will ensure that the stores are completed (due to the `sync` and the SB-1 not fetching more instructions until the current ones graduate) and the next instruction fetch comes from the chip select 1 space (because of the jump and the fact that its delay slot is satisfied from the current instruction bundle).

## SMBUS REGISTERS

The SMBus registers are mapped into the internal I/O section of the memory space. They all occupy a 64 bit wide slot, although in most cases only one or two bytes are implemented. When a register is written its entire contents must be written, writes that are smaller than the implemented width will result in the register value becoming UNPREDICTABLE.

**Table 289: SMBus Clock Frequency Registers**

smb_freq_0 - 00_1006_0010 smb_freq_1 - 00_1006_0018			
Bits	Name	Default	Description
12:0	smb_freq_div	13'h7d	The 100 MHz sys_clock is divided by this x 8 to determine the frequency of the serial clock. The default value is for 100 kHz. The minimum SMBus clock is 10 kHz which is generated using a value of 1250. For 400 kHz the value for this field is 31.
63:13	notimp	51'bx	Not implemented.

**Table 290: SMBus Command Registers**

smb_cmd_0 - 00_1006_0030 smb_cmd_1 - 00_1006_0038 Read returns value from previous smbus read command. Write sets value for next smbus write command.			
Bits	Name	Default	Description
7:0	smb_cmd	8'h0	<b>Write:</b> Low byte of command to be sent following address. <b>Read:</b> First byte received in type 5 or 6 extended mode read.
15:8	smb_cmdh	8'h0	<b>Write:</b> High byte of command to send in extended mode. <b>Read:</b> 2nd Byte received in type 6 extended mode read.
63:15	notimp	56'bx	Not Implemented.

**Table 291: SMBus Control Registers**

smb_control_0 - 00_1006_0060 smb_control_1 - 00_1006_0068			
Bits	Name	Default	Description
0	smb_mk	1'b0	When high, enables interrupt if error is high.
1	smb_finish_en	1'b0	When high, an interrupt will be generated when busy transitions from high to low. The interrupt is cleared by reading the <b>smb_status</b> register.
3:2	reserved	6'h0	Reserved
4	smb_data_out	1'b0	Driven to data pin in direct mode if smb_data_dir is set.
5	smb_data_dir	1'b0	Data direction in direct mode, set for output clear to turn driver off.
6	smb_clk_out	1'b0	Driven to clock pin in direct mode
7	smb_direct	1'b0	Set to enable direct control of clock and data lines (from bits 6:4).
63:8	notimp	56'bx	Not implemented.

**Table 292: SMBus Status Registers**

smb_status_0 - 00_1006_0020 smb_status_1 - 00_1006_0028 Read clears finish interrupt			
Bits	Name	Default	Description
0	smb_busy	1'b0	When high, indicates the serial interface is busy; when low, it is not busy.
1	smb_error	1'b0	When high, indicates an error has occurred. If the smb_mk bit in the <b>smb_control</b> register has a value of 1'b1, an interrupt will be generated. The interrupt and this status bit are cleared by writing this bit with a 1.
2	error_type	1'h0	When bit 1 is high this bit indicates the error that occurred. This bit is cleared if an expected acknowledgement is not seen and set if the transfer has failed after 15 retries.
5	smb_scl_in	1'bx	Reserved. Will be used to provide value on SCL pin in later revisions of the part.
4:3	reserved	2'b0	Reserved
6	smb_ref	1'b0	In direct mode (smb_direct set in the control register) this bit shows the output of the serial clock generator (that would be driven to the clock pin in normal mode). Software may use this as a reference clock. If direct mode is not enabled this bit is zero.
7	smb_data_in	1'b0	In direct mode (smb_direct set in the control register) this bit shows the value on the data pin, otherwise it is zero.
63:8	notimp	56'bx	Not implemented.

**Table 293: SMBus Data Registers**

smb_data_0 - 00_1006_0050 smb_data_1 - 00_1006_0058 Read returns value from previous smbus read command. Write sets value for next smbus write command.			
Bits	Name	Default	Description
7:0	smb_lb	8'h0	Least significant data byte that was read ( or will be written) across the serial bus.
15:8	smb_mb	8'h0	Most significant data byte that was read (or will be written) across the serial bus.
63:16	notimp	48'bx	Not implemented.

**Table 294: SMBus Extra Data Registers**

smb_xtra_0 - 00_1006_0000 smb_xtra_1 - 00_1006_0008 Read returns value from previous smbus read command. Write sets value for next smbus write command.			
Bits	Name	Default	Description
7:0	smb_lb	8'h0	Third data byte that was read in an EEPROM read transfer.
15:8	smb_mb	8'h0	Fourth data byte that was read in an EEPROM read transfer.
63:16	notimp	48'bx	Not implemented.

**Table 295: SMBus Packet Error Check Registers**

smb_pec_0 - 00_1006_0070 smb_pec_1 - 00_1006_0078			
Read returns value from previous smb bus read command. Write sets value for next smb bus write command.			
Bits	Name	Default	Description
7:0	pec	8'h0	This register holds the PEC information that is sent or received for a command that has the smb_pec bit set. The hardware does not generate or check the CRC, software must do that.
63:8	notimp	56'bx	Not implemented.

**Table 296: SMBus Start and Command Registers SMBus Mode**

smb_start_0 - 00_1006_0040 smb_start_1 - 00_1006_0048																					
Bits	Name	Default	Description																		
6:0	smb_addr	7'h0	The serial interface address. These 7 bits are used to form the address byte that is sent across the serial interface.																		
7	smb_qdata	1'b0	Bit of data to send as R/W bit in a Quick Command. Ignored, write as zero for all other commands.																		
10:8	smb_tt	3'h0	<table border="0"> <tr> <td>Value</td> <td>Transfer Type</td> </tr> <tr> <td>000</td> <td>1 byte write (address, command).</td> </tr> <tr> <td>001</td> <td>2 byte write (address, command, write least significant field of smb_data).</td> </tr> <tr> <td>010</td> <td>3 byte write (address, command, write both fields of smb_data).</td> </tr> <tr> <td>011</td> <td>Command and 1 byte read (address, command, address, read byte into LB field of smb_data).</td> </tr> <tr> <td>100</td> <td>Command and 2 byte read (address, command, address, read byte into LB field of smb_data, read byte into HB field of smb_data).</td> </tr> <tr> <td>101</td> <td>1 byte read (address, read byte into LB field of smb_data).</td> </tr> <tr> <td>110</td> <td>Quick Command (address with one bit of data).</td> </tr> <tr> <td>111</td> <td>EEPROM Read. Reads 32 bits into <b>smb_data</b> and <b>smb_xtra</b> from a standard EEPROM &gt;16 kbit that needs 2 address bytes.</td> </tr> </table>	Value	Transfer Type	000	1 byte write (address, command).	001	2 byte write (address, command, write least significant field of smb_data).	010	3 byte write (address, command, write both fields of smb_data).	011	Command and 1 byte read (address, command, address, read byte into LB field of smb_data).	100	Command and 2 byte read (address, command, address, read byte into LB field of smb_data, read byte into HB field of smb_data).	101	1 byte read (address, read byte into LB field of smb_data).	110	Quick Command (address with one bit of data).	111	EEPROM Read. Reads 32 bits into <b>smb_data</b> and <b>smb_xtra</b> from a standard EEPROM >16 kbit that needs 2 address bytes.
Value	Transfer Type																				
000	1 byte write (address, command).																				
001	2 byte write (address, command, write least significant field of smb_data).																				
010	3 byte write (address, command, write both fields of smb_data).																				
011	Command and 1 byte read (address, command, address, read byte into LB field of smb_data).																				
100	Command and 2 byte read (address, command, address, read byte into LB field of smb_data, read byte into HB field of smb_data).																				
101	1 byte read (address, read byte into LB field of smb_data).																				
110	Quick Command (address with one bit of data).																				
111	EEPROM Read. Reads 32 bits into <b>smb_data</b> and <b>smb_xtra</b> from a standard EEPROM >16 kbit that needs 2 address bytes.																				
13:11	reserved	3'h0	reserved, set to zero																		
14	extend	1'b0	This bit should be set to zero for SMBus protocol mode.																		
15	smb_pec	1'b0	Set to modify the transfer type to use Packet Error Checking as defined in the SMBus 1.1 specification. Zero for regular operation.																		
63:16	notimp	48'bx	Not Implemented.																		

**Table 297: SMBus Start and Command Registers Extended Mode**

<b>smb_start_0 - 00_1006_0040</b> <b>smb_start_1 - 00_1006_0048</b>			
Bits	Name	Default	Description
6:0	smb_addr	7'h0	The serial interface address. These 7 bits are used to form the address byte that is sent across the serial interface.
7	smb_qdata	1'b0	Bit of data to send as R/W bit in a Quick Command. Ignored, write as zero for all other commands.
10:8	smb_datafmt	3'h0	Data Format Value    Data Transfer 000    Data[7:0]. 001    Data[7:0], Data[15:8] 010    Data[7:0], Data[15:8], Xtra[7:0] 011    Data[7:0], Data[15:8], Xtra[7:0], Xtra[15:8] 100    No data. 101    Only valid for read transactions Cmd[7:0], Data[7:0], Data[15:8], Xtra[7:0], Xtra[15:8] 110    Only valid for read transactions Cmd[7:0], Cmd[15:8], Data[7:0], Data[15:8], Xtra[7:0], Xtra[15:8] 111    Reserved
12:11	smb_afmt	2'h0	Address/Command Format Value    Address/Command Transfer 00    Only valid for read transactions: No Address. 01    Slave Address Only 10    Slave Address then Cmd[7:0] 11    Slave Address then Cmd[7:0], Cmd[15:0]
13	smb_dir	1'b0	Set to 0 for a write transaction, 1 for a read transaction.
14	extend	1'b1	This bit should be set to one for extended protocol mode.
15	smb_pec	1'b0	Set to modify the transfer type to use Packet Error Checking as defined in the SMBus 1.1 specification. Zero for regular operation.
63:16	notimp	48'bx	Not Implemented.



This Page is left blank for notes

---

This Page is left blank for notes





## Section 15: JTAG and Debug

### INTRODUCTION

The part includes extensive debugging and performance monitoring features. These can be accessed by the CPU(s) or through the JTAG port. The JTAG interface supports the required subset of MIPS EJTAG 2.5 and additions to allow both processors to be debugged. The debug and bus trace unit in the SCD can also be controlled through the JTAG port.

A JTAG probe may be connected to the part for accessing both the boundary scan and debug functions. There is a standard connector for the MIPS EJTAG interface. The application note 'BCM1250 EJTAG Connector' #00-117 from Corelis Inc. ([www.corelis.com](http://www.corelis.com)) includes a recommendation for the connector and terminations for the BCM1250 and BCM1125/H.

### TAP CONTROLLER

The SB1250 implements a IEEE 1149.1 compliant Test Access Port (TAP) controller. The TMS pin is used to control the tap state machine. If TRST\_L is asserted then the TAP controller state machine enters the Test-Logic-Reset state.

**Table 298: JTAG Signals**

JTAG Pin	Description
TDI	Test data in. Data is shifted serially in to the part through this input.
TDO	Test data out. Data is shifted serially out of the part through this output.
TMS	Test Mode select. TAP state machine control signal. The value captured on the rising edge of TCK determines the state transition.
TCK	Test clock. The JTAG interface can be clocked at up to 10 MHz.
TRST_L	Test Reset. Asserting this signal forces the TAP state machine in to the Test-Logic-Reset state.

[Figure 84 on page 422](#) shows the state machine used by the JTAG interface. This is the standard JTAG state machine. TRST\_L can be used to reset the state machine to the Test-Logic-Reset state, which will also be entered if TMS is asserted for five TCK rising edges. The state machine advances each TCK rising edge as directed by the TMS level. The states are as defined in the JTAG specification. The IR scan branch is used to scan instructions into the Instruction register described in [Section: "Instruction Register" on page 423](#). The DR branch scans bits into and out of whatever data scan chain has been selected by an instruction.

When TRST\_L is asserted or the TAP enters the Test-Logic-Reset state, the instruction register is loaded with the IDCODE instruction, any EJTAGBOOT indication is removed and the EJTAG Control register is cleared.

The COLDRES\_L input is also used to reset the TAP, this ensures that the scan chains and boundary scan drivers remain in a safe state as the power comes up. A side effect is that the TAP is held in reset and none of the JTAG machinery can be used while COLDRES\_L is asserted. If testing is to be done using the JTAG interface with the BCM1250 held in reset then RESET\_L should be used to keep the system in reset while allowing the JTAG port to be active. The standard EJTAG connector allows the probe to control RESET\_L.



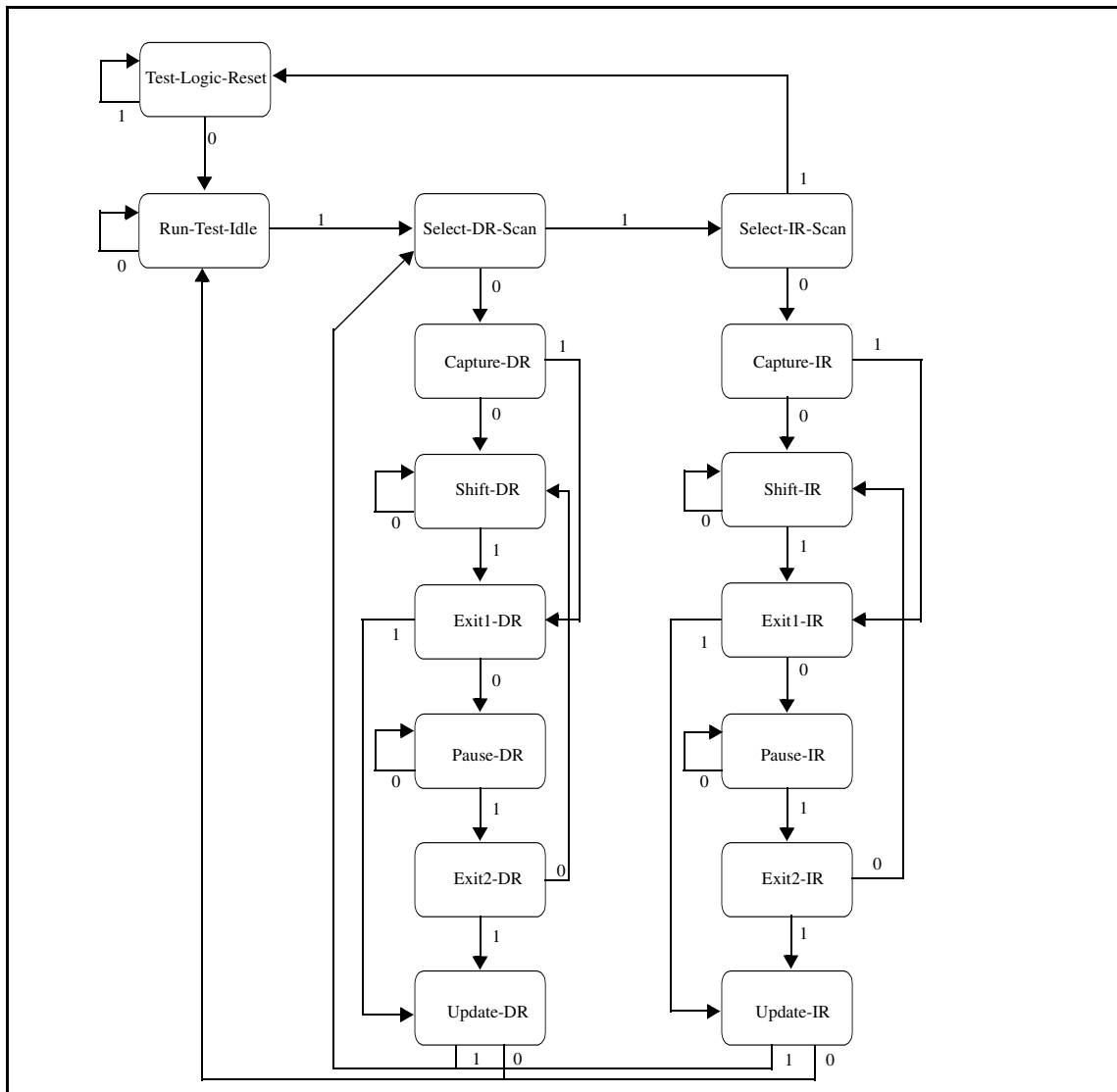


Figure 84: JTAG TAP State Machine

## INSTRUCTION REGISTER

The TAP implements a 6 bit instruction register. The instructions are shown in [Table 299](#). The required BYPASS, SAMPLE/PRELOAD, and EXTEST instructions are implemented, along with the optional IDCODE, INTEST and CLAMP instructions. 25 additional BCM1250 specific instructions have been implemented. The instruction register is scanned in and out LSB first. The value scanned out of the instruction register has a 1 in the LSB and zeros in all other bits.

**Table 299: JTAG Instructions**

IR Value	Instruction	Function
0x00	EXTEST	Selects the boundary scan register for EXTEST. This is used to drive the external pins for testing board connectivity.
0x01	IDCODE	Selects Chip Identification Data Register.
0x02	reserved	
0x03	IMPCODE	Selects the Implementation Code Register.
0x04-0x07	reserved	
0x08	ADDRESS	Selects the EJTAG Address Register.
0x09	DATA	Selects the EJTAG Data Register.
0x0A	CONTROL	Selects the EJTAG Control Register.
0x0B	EJTAGALL	Selects the EJTAG Address, Data and Control Registers.
0x0C	EJTAGBOOT	Force the CPUs to boot in debug mode.
0x0D	NORMALBOOT	Force the CPUs to boot in the normal mode.
0x0E-0x1F	EJTAG reserved	Reserved for EJTAG.
0x20	SYSCTRL	Selects System Control and Status Register.
0x21	TRACE	Selects Trace Register for Scan Out.
0x22	PERF	Selects Performance counters for Scan Out.
0x23	TRACECTRL	Selects the Trace Control Registers for Scan Out/in.
0x24	WAFERID	Selects the Wafer ID for Scan Out.
0x25	PROCESSMON	Selects the Ring Oscillator Register for Scan In/Out.
0x26	CPU0OSC	Broadcom Use Only. Selects CPU0 Observability Chain.
0x27	CPU0DSC	Broadcom Use Only. Selects CPU0 Debug Chain.
0x28	CPU0TSC	Broadcom Use Only. Selects CPU0 Test Chain.
0x29	reserved	
0x2A	CPU1OSC	Broadcom Use Only. Selects CPU1 Observability Chain.
0x2B	CPU1DSC	Broadcom Use Only. Selects CPU1 Debug Chain.
0x2C	CPU1TSC	Broadcom Use Only. Selects CPU1 Test Chain.
0x2D	reserved	
0x2E	SCANIOB0	Broadcom Use Only. Selects IOB0 Chain.
0x2F	reserved	
0x30	SCANIOB1	Broadcom Use Only. Selects IOB1 Chain.
0x31	reserved	
0x32	SCANL2C	Broadcom Use Only. Selects L2C Chain.







The DBBOOT signal will force the CPU to use the alternate debug vector when it enters debug mode. When the CPUs are next reset, they will sample their DBBOOT signals and if it is asserted they will enter Debug Mode and start execution from the alternative debug vector (physical 00\_1000\_0480). The CPUs will therefore enter debug mode and execute code from the JTAG probe, without fetching or executing any instructions from the normal boot vector. This can be used to download code into a system which has no code in ROM.

The EJTAGBOOT indication is effective until the NORMALBOOT instruction is given, TRST\_L asserted or a rising edge of TCK occurs when TAP controller is in the Test-Logic-Reset state. The PrTrap0 and PrTrap1 bits may be cleared by scanning in to the EJTAG Control Register and will read back as zeros, but this is not sufficient to clear the EJTAGBOOT state. The Bypass register is selected when the EJTAGBOOT instruction is given.

### **NORMALBOOT Instruction**

When the NORMALBOOT instruction is given and Update-IR state is left, then the DBBOOT signal is deasserted and the PrTrap0, PrTrap1 and ProbEn bits in the EJTAG Control register are set to 0. Since DBBOOT is deasserted the CPUs will start at the standard Reset vector the next time they are reset. The Bypass register is selected when the NORMALBOOT instruction is given.

### **SCAN Instructions (0x26 - 0x38)**

When these instructions are given the corresponding scan chain is selected. When the SCANALL instruction is given then all of the scan chains are selected in series in order TDI to MC to SCD to L2C to IOB1 to IOB0 to CPU1 to CPU0 to TDO. These scan chains are for Broadcom Use Only. Scanning inappropriate values in to the chains can put the BCM1250 in an UNDEFINED state that requires both a system and a JTAG reset to clear.

### **SYSCTRL Instruction**

The system control and status register is used to control the part for testing and debug purposes. This register is defined by [Table 15 on page 43](#). The System Control Sanchain, summarized in [Table 15](#), includes this register and additional control bits.

**Table 302: System Control Scan Chain**

<b>system_cfg - 00_1002_0008</b>			
<b>Bits</b>	<b>Name</b>	<b>Default</b>	<b>Description</b>
0	reserved	ext	Configuration bit for IO_AD[0].
1	reserved	ext	Configuration bit for IO_AD[1]. Indicates source for IO_CLK100. This must not be changed while the BCM1250 is operating.
2	ldt_minrstcnt	ext	Configuration bit for IO_AD[2]. Broadcom Use Only. This must be zero for normal operation.
3	ldt_bypass_pll	ext	Configuration bit for IO_AD[3]. Broadcom Use Only. This must be zero for normal operation.
4	pci_test	ext	Configuration bit for IO_AD[4]. Broadcom Use Only. This must be zero for normal operation.
5	iob0_div	ext	Configuration bit for IO_AD[5] that controls the clock divider for I/O Bridge 0. 0: IOB0 runs at CPU clock/4, for use with fast CPU clocks. 1: IOB0 runs at CPU clock/3, for use with slow CPU clocks.
6	iob1_div	ext	Configuration bit for IO_AD[6] that controls the clock divider for I/O Bridge 1. 0: IOB1 runs at CPU clock/3, for use with fast CPU clocks. 1: IOB1 runs at CPU clock/2, for use with slow CPU clocks.
11:7	pll_div	ext	Configuration bits for IO_AD[11:7] that select the PLL Divide ratio. These bits must not be changed while the BCM1250 is operating.
12	ser0_enable	ext	Configuration bit for IO_AD[12]. 0: Serial interface 0 is in asynchronous (uart) mode. 1: Serial interface 0 is in synchronous mode.
13	ser0_rstb_en	ext	Configuration bit for IO_AD[13] that allocates GPIO[0] pin to the synchronous serial interface.
14	ser1_enable	ext	Configuration bit for IO_AD[14]. 0: Serial interface 1 is in asynchronous (uart) mode. 1: Serial interface 1 is in synchronous mode.
15	ser1_rstb_en	ext	Configuration bit for IO_AD[15] that allocates GPIO[1] pin to the synchronous serial interface.
16	pcmcia_enable	ext	Configuration bit for IO_AD[16] that configures the PCMCIA mode.
18:17	boot_mode	ext	Configuration bit for IO_AD[18:17] that configures the boot mode. 00: 32 bit generic bus ROM (multiplexed). 01: 8 bit generic bus ROM (non-multiplexed). 10: SMBus EEPROM <= 16 kbit (read word protocol). 11: SMBus EEPROM > 16kbit (eeprom read word protocol).
19	pci_host	ext	Configuration bit for IO_AD[19], that configures the PCI interface to be host or device mode.
20	pci_arbiter	ext	Configuration bit for IO_AD[20], that configures the PCI interface to use an internal or external arbiter. (If the PCI is set in device mode the resistor must be set for an external arbiter)
21	southOnLDT	ext	Configuration bit for IO_AD[21], that configures the southbridge to be on the HyperTransport fabric or PCI bus.
22	big_endian	ext	Configuration bit for IO_AD[22], that configures the system to be big or little endian.
23	genclk_en	ext	Configuration bit for IO_AD[23], that enables output of the generic bus clock on IO_CLK100. If this bit is zero then the IO_CLK100 will be held in a high impedance state.
24	ldt_test_en	ext	Configuration bit for IO_AD[24]. Broadcom Use Only. This must be zero for normal operation.

**Table 302: System Control Scan Chain (Cont.)**

system_cfg - 00_1002_0008			
Bits	Name	Default	Description
25	gen_parity_en	ext	Configuration bit for IO_AD[25] that configured the generic bus parity.
31:26	config	ext	Configuration bit for IO_AD[31:26]. These configuration bits are available for interpretation by software.
32	clkstop	1'b0	Writable via JTAG only. Broadcom use only. Takes effect on UpdateDR.
33	clkstep	1'b0	Writable via JTAG only. Broadcom use only.
41:34	clkcount	8'b0	Writable via JTAG only. Broadcom use only.
42	pllbyypass	1'b0	Writable via JTAG only. Broadcom use only.
44:43	pll_iref	2'b0	Writable via JTAG only. Broadcom use only.
46:45	pll_vco	2'b0	Writable via JTAG only. Broadcom use only.
48:47	pll_vreg	2'b0	Writable via JTAG only. Broadcom use only.
49	mem_reset	1'b0	Writable via JTAG only. When set the memory controller is held in reset.
50	l2c_reset	1'b0	Writable via JTAG only. When set the level 2 cache is held in reset.
51	io_reset_0	1'b0	Writable via JTAG only. When set the I/O bridge to the PCI and HyperTransport fabric is held in reset.
52	io_reset_1	1'b0	Writable via JTAG only. When set the I/O bridge to the slow speed devices and generic bus is held in reset.
53	scd_reset	1'b0	Writable via JTAG only. When set the SCD is held in reset.
54	cpu_reset_0	1'b0	When set CPU 0 will be held in reset.
55	cpu_reset_1	1'b1	When set CPU 1 will be held in reset. This bit is set on a BCM1250 reset, causing the processor to remain in reset until released under software control.
56	unicpu0	1'b0	Set to indicate uniprocessor using physical processor 0. (This bit will always be set on the BCM1125/H.)
57	unicpu1	1'b0	Set to indicate uniprocessor using physical processor 1.
58	sb_softres	1'b0	When a write changes this bit from a 0 to a 1 a soft reset will be performed. This will reset of whole chip except for this register. Note that once it is set the bit must be cleared before writing a 1 will again cause a soft reset.
59	ext_reset	1'b0	When set the RESETOUT_L pin will be asserted.
60	system_reset	1'b0	When written with a 1 a full system reset will be performed (thus setting this bit back to zero).
61	misr_mode	1'b0	Broadcom Use Only. Set MISR mode (1=misr 0=capture).
62	scd_misr_reset	1'b0	Broadcom Use Only. Reset SCD MISRs on 0->1 transition.
63	reserved	1'b0	Reserved
95:64	stp_cnt	32'b0	Broadcom Use Only. Stop counter set to zero for normal operation.
96	pllstop	1'b0	Broadcom Use Only. Set to zero for normal operation.
97	stop_stre	1'b0	Broadcom Use Only. Set to zero for normal operation.
98	start_trc	1'b0	Broadcom Use Only. Set for trace, clear for reset.
99	stopping	1'b0	Broadcom Use Only. Read Only.
100	ss_done	1'b0	Broadcom Use Only. Read Only.
101	zbser_ard	1'b0	Broadcom Use Only. Serialize A-R-D. Set to zero for normal operation.
102	zbser_ar	1'b0	Broadcom Use Only. Serialize A-R. Set to zero for normal operation.





**Table 302: System Control Scan Chain (Cont.)**

system_cfg - 00_1002_0008			
Bits	Name	Default	Description
103:104	str_mode	2'b0	Broadcom Use Only, sets stretch mode for PLL. Set to zero for normal operation.

On the BCM1250 Bits 104:0 of this register are selected in between TDI and TDO, on the BCM1125/H there are two additional Broadcom Use Only bits so bits 106:0 are selected. The scan order is from LSB to MSB. The configuration bits may be written by the JTAG probe while the part is held in reset using RESET\_L provided the cold reset delay (Tcr) described in the Hardware Data Sheet has elapsed since COLDRES\_L was released. Most of the other bits in this schain are for Broadcom Use Only, incorrect settings will cause the part to behave in UNDEFINED ways and could require a full cold reset to restore deterministic behavior. The CPU uniprocessor bits (in the resets field) are normally set by only on uniprocessor parts. However if one of these bits is set in the control register and then the Broadcom-soft-reset bit is set, following the reset the BCM1250 will behave as a uniprocessor. This can be used to disable one of the processors, it will not be clocked and will enter a low power state.

Broadcom soft reset occurs only when a 1 is scanned into this register, and the TAP controller enters the Update-DR state. This reset is just like a normal reset except that the contents of the system control register are not reset. This allows testing of various modes.

### TRACE Instruction

When the TRACE instruction is set, the trace\_read register is selected between TDI and TDO. This is a read only register. It is scanned out LSB first. The trace buffer should be frozen and the **startread** bit set prior to reading out. This is a 64 bit register, and gets the next 64 bits of trace data every time the Capture-DR state is entered. It performs just as if the data had been read out using address 00\_1002\_0a08. Following a capture the first scan of the register returns UNPREDICTABLE data and the next will return the first valid bits. After the entire contents of the trace buffer have been read, it will return all zeros.

### PERF Instruction

The PERF instruction selects the performance registers for scan in and out. Note that the value scanned in will overwrite the counters. The performance chain is scanned out LSB first and is made up of the performance registers as shown in [Table 303](#).

**Table 303: Performance Counter Scan Chain**

Bits	Register	Description
33:0	perf_cnt_cfg[33:0]	Performance counter configuration (See <a href="#">Table 31 on page 61</a> ).
74:34	perf_cnt_0[40:0]	Performance counters (See <a href="#">Table 32 on page 62</a> ).
115:75	perf_cnt_1[40:0]	
156:116	perf_cnt_2[40:0]	
197:157	perf_cnt_3[40:0]	



**TRACECTRL and TRACECURCNT Instructions**

The trace control registers are selected for both scan in and out using the TRACECTRL instruction. The registers are connected between TDI and TDO and are scanned out LSB first. The registers are scanned out in the order shown in [Table 304](#), starting with trace\_event\_0[0] and ending with trace\_sequence\_7[24].

Writing values in the trace control scan chain has the same effect as writing the registers from the CPU. In particular when the event count field is written both the current and maximum count get written.

The TRACECURCNT instruction allows the current event counts to be read without forcing an update. The read-only Trace Current Count Scan Chain has the order shown in [Table 304](#).

**Table 304: Trace Control Scan Chain**

Bits	Register	Description
31:0	trace_event_0[31:0]	Trace Event Registers (See <a href="#">Table 46 on page 72</a> ).
63:32	trace_event_1[31:0]	
95:64	trace_event_2[31:0]	
127:96	trace_event_3[31:0]	
152:128	trace_sequence_0[24:0]	Trace sequence Registers (See <a href="#">Table 47 on page 74</a> ).
177:153	trace_sequence_1[24:0]	
202:178	trace_sequence_2[24:0]	
227:203	trace_sequence_3[24:0]	
259:228	trace_event_4[31:0]	Trace Event Registers (See <a href="#">Table 46 on page 72</a> ).
291:260	trace_event_5[31:0]	
323:292	trace_event_6[31:0]	
355:324	trace_event_7[31:0]	
380:356	trace_sequence_4[24:0]	Trace sequence Registers (See <a href="#">Table 47 on page 74</a> ).
405:381	trace_sequence_5[24:0]	
430:406	trace_sequence_6[24:0]	
455:431	trace_sequence_7[24:0]	

**Table 305: Trace Current Count Scan Chain**

Bits	Register	Description
7:0	trace_event_cnt_0[7:0]	Current count for Trace Event Registers (See <a href="#">Table 46 on page 72</a> ).
15:8	trace_event_cnt_1[7:0]	
23:16	trace_event_cnt_2[7:0]	
31:24	trace_event_cnt_3[7:0]	
39:32	trace_event_cnt_4[7:0]	
47:40	trace_event_cnt_5[7:0]	
55:48	trace_event_cnt_6[7:0]	
63:56	trace_event_cnt_7[7:0]	



## PROCESSMON Instruction

The PROCESSMON instruction selects the ring oscillator register for scan in and out. When the instruction is selected, it enables the ring oscillator pointed to by 9:8 of the data register. When this instruction is selected the counter only counts while DEBUG\_L pin is asserted. Ring oscillators will run at less than 1 GHz. This is for Broadcom Use Only.

**Table 306: Ring Oscillator Scan Chain**

Bits	Name	Description
15:0	count	Selected counter value.
18:16	sel	Oscillator select.
19	enable	High to enable counter

## BOUNDARY SCAN REGISTER

Each pin on the part with the exception of COLDRES\_L, RESET\_L, TMS, TDO, TDI, TRST\_L and TCK, can be controlled by the boundary scan register. The boundary scan register is selected between TDI and TDO when INTEST or EXTEST instructions are set in the instruction register. When the TAP controller traverses the Capture-DR state, the inputs are sampled into the shadow register.

During the Shift-DR state of the TAP controller the contents of the boundary scan shadow register are shifted in between TDI and TDO. When the Update-DR state is traversed, the contents of the BSR gets the contents of the shadow register. This allows the user to scan in new contents while the old contents of the BSR are still being held at the outputs. During EXTEST, outputs are driven by the BSR instead of the normal internal signals. During INTEST the inputs to the part are driven by the BSR instead of the pins.

Figure 85 shows an example of the boundary scan register and the shadow register for most of the pins on the part. All the pins apart from HyperTransport use this input/output pin building block. It consists of three BC\_1 scan cells, one for the output, one for the output enable and one for the input. The Mode signal causes the JTAG to drive both the INTEST and EXTEST data.

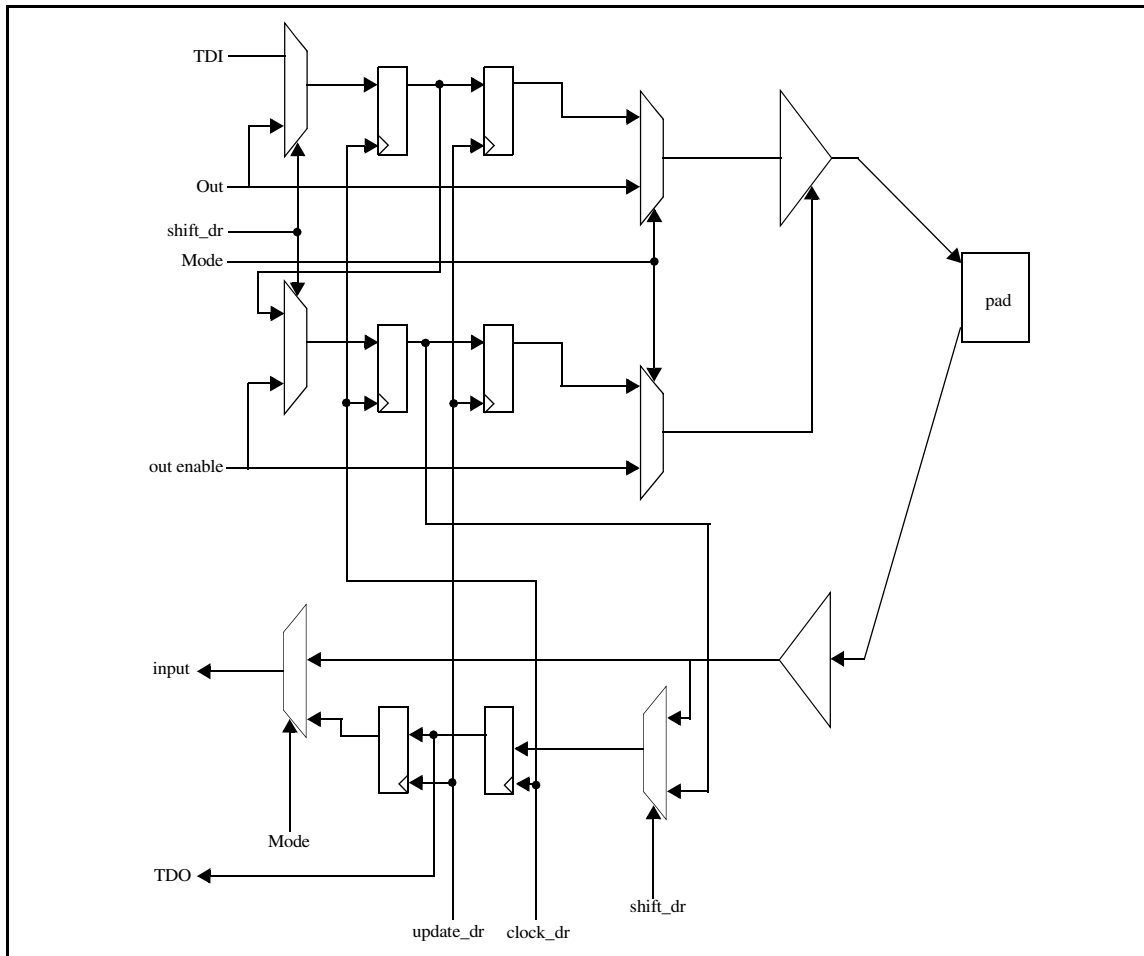


Figure 85: JTAG Boundary Scan Register Block

To use the boundary scan and EXTEST to test board connectivity the following steps should be performed:

- 1 The TAP controllers of both parts under test are set to the EXTEST instruction.
- 2 The desired pattern on the output pins is shifted into the BSR shadow register.
- 3 The Update-DR state is traversed.
- 4 The Capture-DR state of the target part is traversed.
- 5 The contents of the shadow register in the target part is scanned out and then compared to expected results.

To use boundary scan and INTEST to run test vectors against the part the following steps should be performed:

- 1 The instruction register for the part is set to the INTEST instruction.
- 2 The desired test pattern is scanned into the shadow register.
- 3 The Update-DR state is traversed.
- 4 The Capture-DR state is traversed.
- 5 The results from the vector are scanned out from the shadow register. At the same time the next test pattern can be scanned in.

The differential signals on the HyperTransport interface are scanned a little differently. All of these are unidirectional, but the signals are also differential and double data rate. The data rate is doubled and the differential is formed in the output pads, the differential is received and the data rate halved in the input pads.

Figure 86 shows the output cell. In normal operation two bits of data are presented, one for transmission when the clock is high the other when the clock is low. The inverted version of these bits drives the negative output of the differential pair. The scan function uses two BC\_1 cells, one drives each of the pins, but rather than the pin data they monitor the single data rate version of the data. Thus when performing EXTEST the two pins are separate (although they should be driven as inverses to guarantee correct reception), but when scanning out vectors the data for both edges of the clock can be captured.

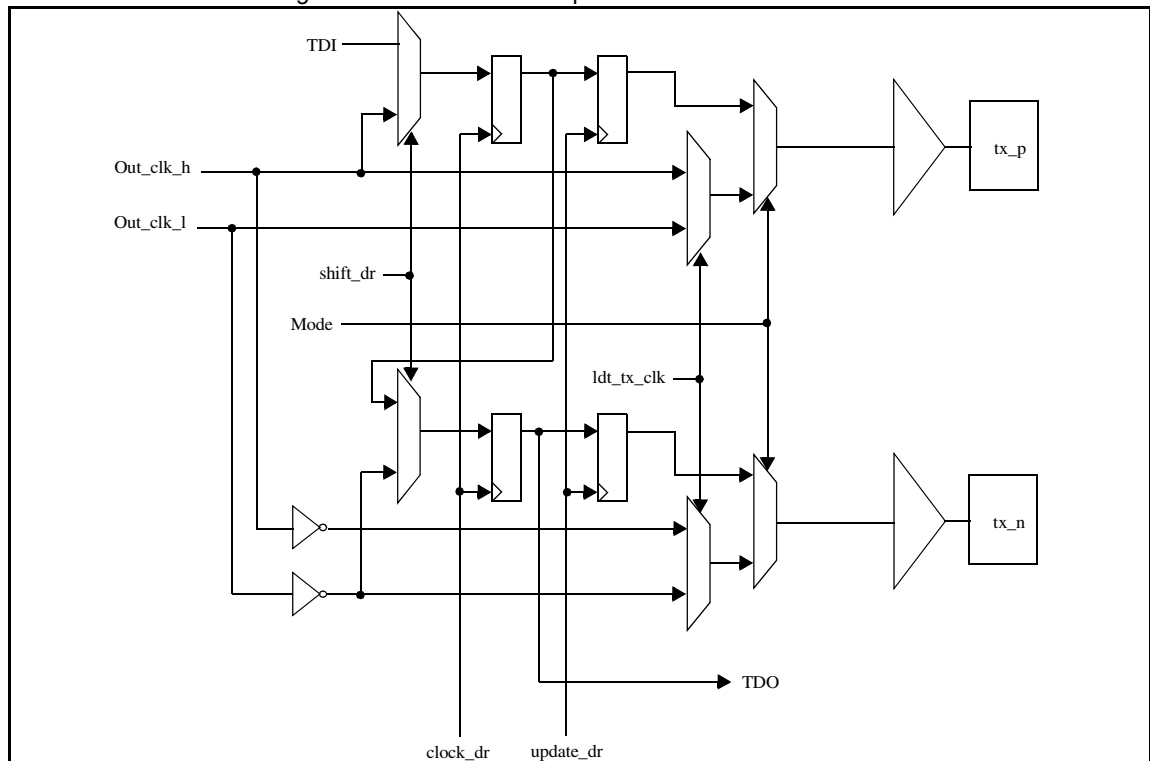


Figure 86: JTAG HyperTransport Output Boundary Scan Block

The HyperTransport input block, shown in Figure 87, is similar. It uses two BC\_1 cells to monitor the two pins. However, these will always be inverses when a correct differential signal is received. The outputs of the BC\_1 cells can be used to drive the single data rate version of the data into the part simplifying INTEST vector injection.

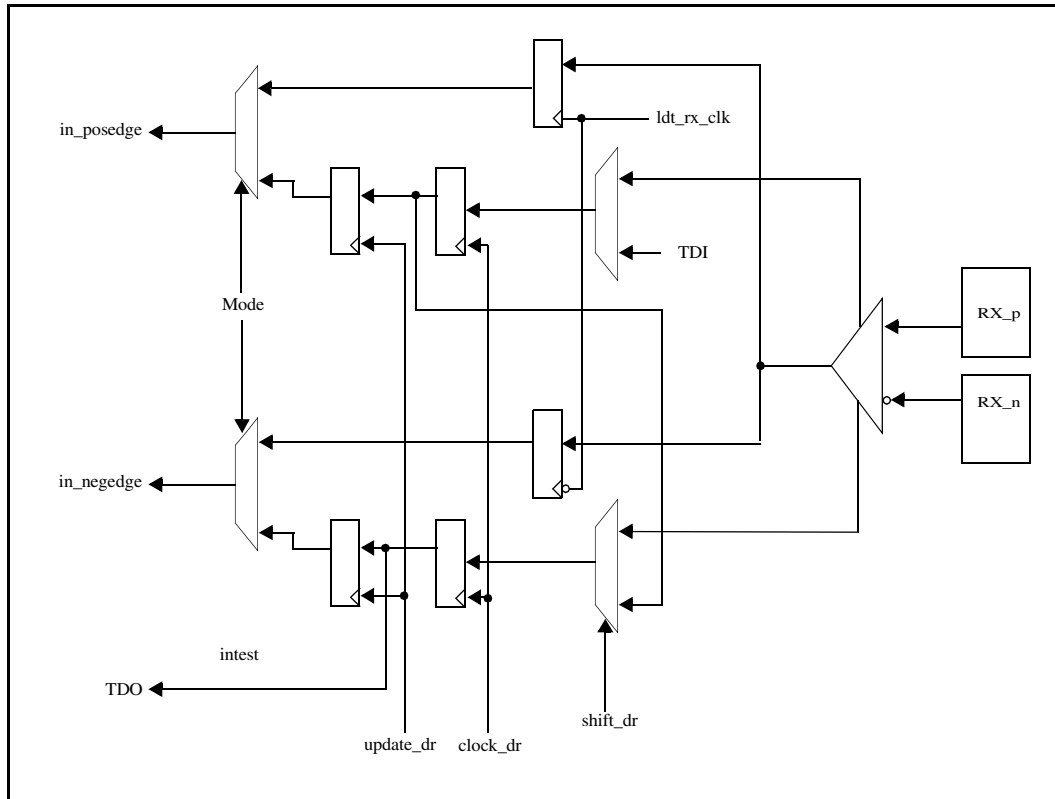


Figure 87: JTAG HyperTransport Input Boundary Scan Block

The order of the input/output pins are shown in the BSDL file for the part.

### BSRMODE - HOLDING BOUNDARY SCAN ACTIVE

Using the normal JTAG definition it is not possible to keep the boundary scan register active (driving pins or into the part) and scan other internal registers, because the boundary scan must be made inactive when a non-BSR instruction is scanned in to the instruction register. The large amount of internal state in the part makes it useful to be able to do this so the BSRMODE instruction has been added. If the BSRMODE instruction is used after EXTEST, CLAMP or INTEST then the BSR mode is maintained (keeping the signals driven from the BSR) even if another JTAG instruction is used. This allows any number of other chains to be scanned while the BSR values are held. If EXTEST, CLAMP or INTEST is followed by any instruction other than BSRMODE the BSR mode is removed (per the standard JTAG behaviour). After BSRMODE has been used its state should be cleared by performing one of the BSR instructions followed by a non-BSR instruction.



## PROCESSOR AND PROBE ACCESS

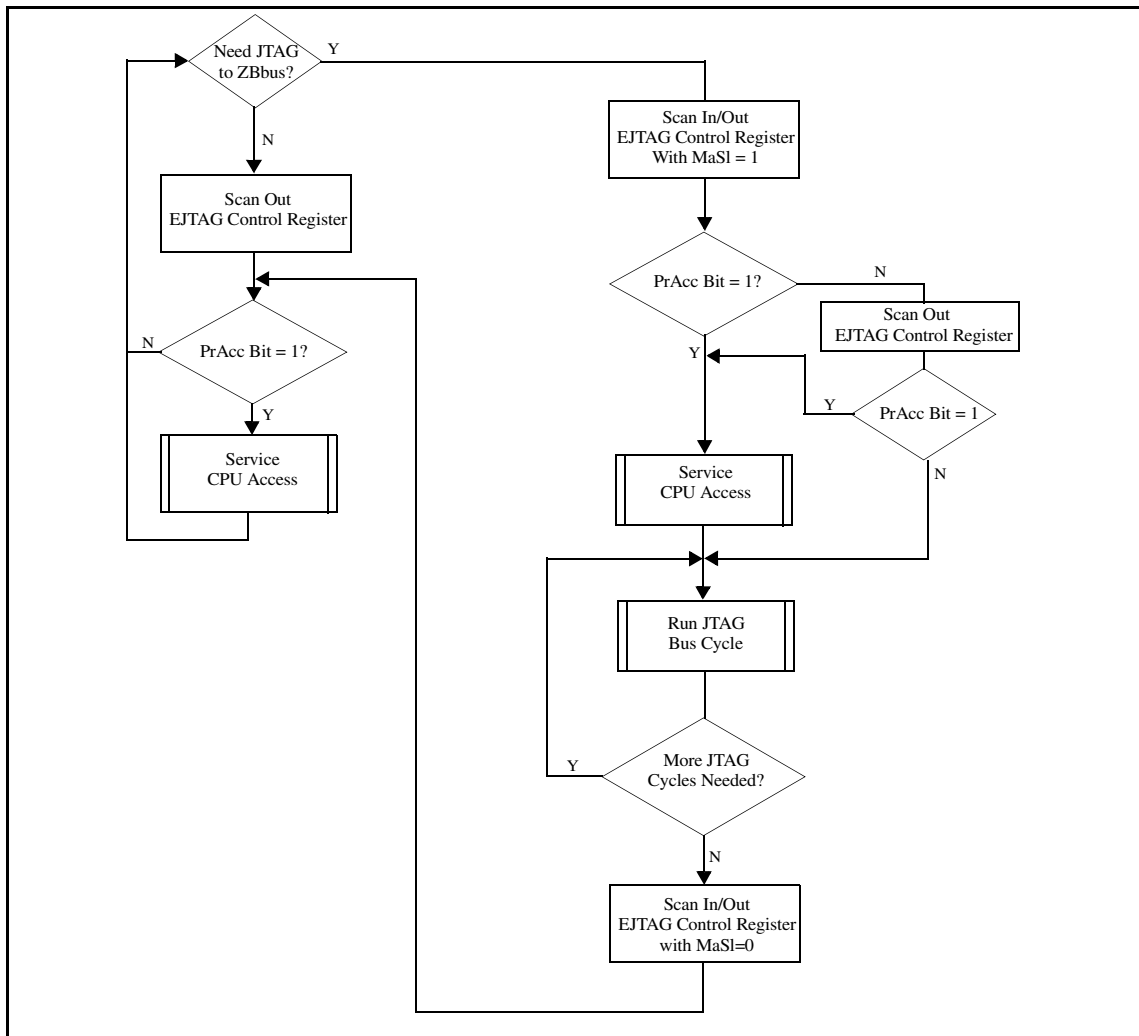
The JTAG probe can act as both a master and a slave on the ZBbus. As a master, the probe can initiate any bus transaction. As a slave, the probe responds to accesses in the EJTAG memory range 00\_1000\_0000 to 00\_1001\_FFFF. (This address range will be accessed by the CPU using the alternative debug vector at VA FFFF\_FFFF\_B000\_0480.) At any time the probe can either be a master or a slave, however it is possible for an external debugger to switch between the two modes to support both processor accesses to the probe and probe accesses to memory.

Probe accesses are controlled by three JTAG registers: the Address Register, the Data Register and the Control Register. These are described in the subsequent sections. JTAG commands are provided for scanning these registers individually or as a concatenated chain.

There are two Control Register bits that set the behavior of the probe agent. The ProbEn bit enables accesses to the EJTAG range and the MaSI bit selects if the probe is a master or a slave. [Table 307](#) shows the behavior for the possible cases.

**Table 307: CPU and Probe Accesses**

ProbEn	MaSI	CPU Access to 00_1000_0000 - 00_1001_FFFF	Probe Access to ZBbus
0	0	Probe memory disabled. Read returns zeros, writes are discarded.	Master mode disabled.
0	1	Probe memory disabled. Read returns zeros, writes are discarded.	Master mode active. The probe can run ZBbus cycles.
1	0	Probe memory active. Reads and writes are serviced by the probe. The EJTAG control register should be scanned to detect CPU requests.	Master mode disabled.
1	1	Probe accesses delayed. Reads and writes are blocked and will be serviced when MaSI is set to zero.	Master mode active. The probe can run ZBbus cycles.



**Figure 88: Example JTAG Probe Flowchart**

An external debugger connected to the JTAG probe can use both master and slave accesses to communicate with the part. An example flow is shown in [Figure 88](#). Normally the debugger will scan out the Control Register and service CPU requests. When it needs to run an access (for example to read memory) it will scan in the Control Register with the MaSI bit set. A check needs to be done for CPU accesses that may have arrived while the mode was being changed, once they are serviced the interface is ready for the master access.



## PROCESSOR ACCESSES TO THE JTAG SPACE

When the JTAG probe is configured for slave accesses, an access to the JTAG range (00\_1000\_0000 to 00\_1001\_FFFF) is responded to by the debugger. If the CPU is set to use the Alternate Debug Vector then a debug exception will cause it to fetch instructions from 00\_1000\_0480.

When the CPU does a read from JTAG memory, the JTAG unit latches the address, bus command, cache attributes and the byte enables from the request into the scannable Address Register. The JTAG unit then sets the PrAcc bit to 1. The debugger software will be scanning out the EJTAG Control Register, and will see the PrAcc bit set. It will then scan out the Address Register, decode A\_CMD as a read and fetch the data in the debugger memory, and then scan the data, properly aligned, into the Data Register. It will then scan in the control register with the PrAcc bit cleared. The JTAG unit will then terminate the bus transaction with a data cycle on the bus, containing the data, dcode and modified flag from the Data Register.

When the CPU does a write to JTAG memory, the JTAG unit latches the address, bus command, cache attributes and the byte enables from the request into the scannable Address Register and the data, dcode, responder id and modified flag into the Data Register. The JTAG unit then sets the PrAcc bit to 1. The debugger software will be scanning out the EJTAG Control Register, and will see the PrAcc bit set. It will then scan out the Address Register, decode A\_CMD as a write, scan the Data Register, and write the data in the debugger memory. It will then scan in the Control Register with the PrAcc bit cleared causing the JTAG unit to free up the buffers and allow more transactions.

The CPU can map the JTAG space cacheable non-coherent or uncacheable. It must not be mapped cacheable coherent, or the behavior of the system becomes UNDEFINED.

## PROBE ACCESSES TO THE ZBBUS

When the MaSI bit is set in the EJTAG control register, the probe can master the ZBbus and access any area of system memory. A write to system memory is done by scanning into the Address Register the address, byte enables, command and cache attribute bits. The data, dcode and modified flag are scanned in to the Data Register. The four responder id bits that are scanned into the Data Register are ignored, the SCD agent ID is always used for a request from the JTAG unit. When the PbAcc and PW bits are set in the EJTAG Control Register the write will be initiated, and the PbAcc bit will be cleared when it has completed.

A read from system memory is done by scanning into the Address Register the address, byte enables, command and cache attribute bits. The access is started by scanning a 1 into the PbAcc bit, and a 0 into the PW bit. The EJTAG unit then makes a read request on the bus and when the transaction has finished, the data, dcode, responder id and modified flag will be returned in the Data Register, and the PbAcc bit will be cleared. The probe should keep scanning out the Control Register until it sees that the PbAcc bit has been cleared. It then can scan out the Data Register.

Note that if ProbeEn is 1, a master access should not be made to an address serviced by the SCD, since these will deadlock.

## ADDRESS REGISTER

The Address Register is 77 bits long and contains the address bits, byte enables, bus command and cache attributes. To maintain coherency the cache attributes must be set correctly for the address so snoop responses are correct (in some situations the debugger may want to use different attributes but it must do so with care, for example a cacheable non-coherent read to a memory address can be used to force data to come from the L2 cache or memory, never a processor L1 cache, and an uncacheable read from a memory address will force the data to come from memory). The address is for a 32 byte cache block, the byte enables complete selection within the block. [Table 308](#) shows the Address Register, which is scanned starting with the LSB.

**Table 308: JTAG Address Register Scan Chain**

Bits	ZBbus signal (see <a href="#">Table 2 on page 20</a> )	Description
34:0	A_AD[39:5]	Address
66:35	A_BE[31:0]	Byte Enables. Indicate which byte lanes are valid in the Data Register, see <a href="#">Table 5 on page 24</a> for the mapping from byte enable to byte address.
69:67	A_CMD[2:0]	Command. See <a href="#">Table 3 on page 22</a> .
71:70	A_L1CA[1:0]	L1 Cacheability attribute. See <a href="#">Table 4 on page 23</a> .
72	A_L2CA	L2 Cache Allocate request.
76:73	A_ID[9:6]	Requester agent ID (see <a href="#">Table 1 on page 20</a> ). These bits are only valid when the probe is the slave, in master mode the SCD agent ID will be used for the transaction.

## DATA REGISTER

The Data Register contains 256 data bits, the status/error code, the ID of the agent that put the data on the ZBbus, the modified flag, and (on reads) the response phase information for the transaction. The register is shown in [Table 309](#), and is scanned from LSB to MSB. The mapping from the byte enables in the address register to bits in this register is fixed; however, the byte address associated with each byte lane depends on the system endian mode as shown in [Table 5 on page 24](#).

**Table 309: Data Register Scan Chain**

Bits	ZBbus signal (see <a href="#">Table 2 on page 20</a> )	Description
255:0	D_DA[255:0]	Data. Only the byte lanes that had byte enables set contain valid data.
258:256	D_CODE[2:0]	Data status/error code (see <a href="#">Table 6 on page 24</a> ).
259	D_MOD	Set to indicate the data is modified (dirty).
263:260	D_RSP[3:0]	Id of Agent putting the data on the ZBbus. This is a read only field, bits scanned in will be ignored.
264	R_L2HIT	Response indicating block is in the L2 cache (read only, only valid for probe master reads).
270:265	R_EXC[5:0]	Response indicating block is exclusive in an agent (read only, only valid for probe master reads).
276:271	R_SHD[5:0]	Response indicating block is shared by an agent (read only, only valid for probe master reads).



## EJTAG CONTROL REGISTER

This is a 12 bit register to control the various operations of the debug support modules. This register is selected by shifting in the CONTROL instruction. Bits in the EJTAG Control Register can be set/cleared by shifting in data; status is read by shifting out this register. This EJTAG Control Register can only be accessed by the TAP interface.

**Table 310: EJTAG Control Register**

Bit	Name	Description	Use	Reset Value
0	DM0	Debug Mode 0. This bit indicates the value of the EDEN signal from CPU0. When this bit is set, it indicates that CPU0 is signalling a debug event (see SB-1 CPU User Manual). This bit is read only, and any writes to it are ignored.	R	X
1	DM1	Debug Mode 1. This bit indicates the value of the EDEN signal from CPU1. When this bit is set, it indicates that CPU1 is signalling a debug event (see SB-1 CPU User Manual). This bit is read only, and any writes to it are ignored. On the BCM1125/H this bit is not used and reads are UNPREDICTABLE.	R	X
2	EJTAG Break0	Setting this bit to 1 causes the DINT pin to CPU0 to be set. This allows the probe to signal a debug interrupt to CPU0. The bit must be cleared to remove the interrupt.	R/W	0
3	EJTAG Break1	Setting this bit to 1 causes the DINT pin to CPU1 to be set. This allows the probe to signal a debug interrupt to CPU1. The bit must be cleared to remove the interrupt. On the BCM1125/H this bit is not used.	R/W	0
4	PrTrap0	Probe Trap0. Setting this bit to 1 forces the DBBOOT signal to CPU0 to 1. This allows the probe to force CPU0 into debug mode upon the next deassertion of reset. This bit is set on the Update-IR state of the EJTAGBOOT instruction. It is reset on the Update-IR state of the NORMALBOOT instruction or by asserting the TRST_L pin or by a rising edge of TCK when the TAP controller is in the Test-Logic-Reset state. This bit can also be set or cleared by scanning a 1 or 0 into this bit.	R/W	0
5	PrTrap1	Probe Trap1. This bit performs the same way as PrTrap0, except it does so for CPU1. On the BCM1125/H this bit is not used.	R/W	0
6	ProbEn	Probe Enable. Setting this bit to 1 will indicate that EJTAG memory is handled by the probe, so processor accesses are answered. 0: The probe does not handle EJTAG memory transactions. Accesses to the EJTAG memory space always returns 0. 1: The probe does handle EJTAG memory transactions. When this bit is set the MaSI bit is used to determine if the probe is acting as bus master or slave. See discussion in <a href="#">Section: "Processor and Probe Access" on page 435</a> . Note that setting the PrTrap bits without this bit set causes an invalid state. The EJTAGBOOT instruction will set this bit to 1. The NORMALBOOT instruction will clear this bit. This bit is also cleared by asserting TRST_L or by a rising edge of TCK while in the Test-Logic-Reset State.	R/W	0

**Table 310: EJTAG Control Register (Cont.)**

Bit	Name	Description	Use	Reset Value
7	PrAcc	Processor Access. Read value of this bit indicates if a Processor Access (PA) to the EJTAG memory is pending: 0: No pending processor access. 1: Pending processor access. The probe control software must clear this bit to 0 to indicate the end of the PA. Write of 1 is ignored.	R/W	0
8	PW	Processor Access Write. This bit is not used.	R/W	X
9	PbAcc	Probe Initiated Transaction. When set this bit indicates a probe initiated transaction is in progress. The probe sets this bit to indicate to the JTAG unit that the probe is initiating a transaction. The JTAG unit clears this bit when the transaction has been completed. This bit can only be set if the ProbEn bit is cleared. Probe initiated transactions can occur only if the MaSl bit is set. 0: No pending probe access. 1: Pending probe access. If the probe writes a zero to this bit it is ignored. The only way the bit can be set is by the probe writing a 1, the only way the bit can be cleared is by the JTAG unit clearing it.	R/W	0
10	MaSl	Probe Master/Slave. When set this bit indicates the probe can initiate a transaction, if ProbEn is set then CPU accesses will be blocked until this bit is clear. When clear CPU transactions will be accepted if ProbEn is set. 0: CPU accesses accepted. 1: Probe can initiate a bus transaction.	R/W	0
11	ClkStop	Clock stop flag. This bit is for Broadcom Use Only.	R/O	0



## DIFFERENCES FROM EJTAG 2.5 (FEB. 22, 2000) SPECIFICATION

The BCM1250 has some differences from the EJTAG 2.5 due to support for the dual processor configuration and system level debug access. A summary, with reference to the section numbers in the EJTAG 2.5 (Feb. 22, 2000) specification:

### 2.2.2

The BCM1250 has no dseg (A kseg1 address is used for probe serviced accesses)

### 2.2.2.1 N/A

### 2.2.2.2 N/A

### 2.2.3.2

The random register continues to run in debug mode

### 2.3.2

Alternate Vector location is FFFF\_FFFF\_B000\_0480

- Set by SW in EDEBUG register
- Set by DBBOOT signal from JTAG TAP (in EJTAG control register)

### 2.3.5/2.3.6

Hardware breakpoints/watchpoints can be done in extended debug mode using the regular Watch registers not special ones.

### 2.3.7

Watch register exceptions are precise for both data and instructions

### 2.6.3

Processor reset is done through System Config register not EJTAG Control

### 2.6.4

Rocc is not supported.

### 3

There is no DCR (dseg is not supported)

### 4

There are no special debug hardware breakpoints. In extended debug mode the CP0 Watch registers can be used to provide one hardware data break and one hardware instruction breakpoint.

### 5

A single TAP is shared by the two processors and the system logic.

### 5.5.3

The data register is 277 bits (contains ZBbus data + info)



**5.5.4**

The address register is 77 bits (contains ZBbus address + cmd)

**5.5.5**

Rocc - Not implemented

Psz - Not implemented, size in address register

Doze - Not implemented

Halt - Not implemented

PerRst - Use System Config register for resets

PRnW - Not used

PrAcc - Supported

PrRst - Use System Config register for resets

ProbEn - Supported (works with additional MaSI bit to set direction)

ProbTrap - Two Copies, one per CPU. Also causes EJTAG boot on reset

EjtagBrk - Two Copies, one per CPU, level sensitive probe must clear

DM - Two copies. But shows CPU EDEN signal rather than DM

**5.6.3**

Rocc is not supported

**5.6.4**

Processor access is through physical address 00\_1000\_0000 - 00\_1001\_FFFF and uses ZBbus request

There is an additional mode for allowing the probe to initiate bus requests to memory or memory mapped I/O.

**7.1.2**

External interrupt possible through DEBUG\_L pin but the driver must be open collector or open drain.

**7.1.2**

The probe RST signal can be connected to RESET\_L.

This Page is left blank for notes



---

This Page is left blank for notes





## Section 16: Reference

### INTERNAL REGISTER ADDRESSES BY FUNCTION

This section lists the registers and address assignments, per ZBbus agent. In the electronic version of the document the Table/Page column provides a hyperlink to the table that defines the register.

**Note:** The following table details the specific addresses and names of each register. Each register is 1 byte, 2 bytes, 4 bytes or 8 bytes. Registers can be read or written as double-word (8 bytes), word (4 bytes), 2 byte or signal byte access. Reads that are wider than the defined width or the register will return UNPREDICTABLE data in the bits that are not defined. The address used to access the register will need to be adjusted if the system is running in Big Endian mode. See [Section: "Internal Registers" on page 11](#) for details. Configuration registers should be mapped in uncacheable space, so transactions will never be wider than 8 bytes (and will never span more than an single 8 byte aligned range).

Reading registers marked 'Write Only' will give UNPREDICTABLE results. Writes to 'Read Only' registers will be ignored.

**Table 311: Internal Register Addresses by Function**

Name	Address	Table/ Page	Description
<b>Memory Controller</b>			
mc_config_0	00_1005_1100	<a href="#">72/135</a>	Channel 0 attributes.
mc_dramcmd_0	00_1005_1120	<a href="#">75/139</a>	Channel 0 SDRAM command.
mc_drammode_0	00_1005_1140	<a href="#">76/139</a>	Channel 0 SDRAM mode.
mc_timing1_0	00_1005_1160	<a href="#">77/140</a>	Channel 0 SDRAM timing 1.
mc_timing2_0	00_1005_1180	<a href="#">78/141</a>	Channel 0 SDRAM timing 2.
mc_cs_start_0	00_1005_11a0	<a href="#">79/141</a>	Channel 0 CS[3:0] start address.
mc_cs_end_0	00_1005_11c0	<a href="#">80/141</a>	Channel 0 CS[3:0] end+1 address.
mc_interleave_0	00_1005_11e0	<a href="#">81/142</a>	Channel 0 interleaved CS position.
mc_cs0_row_0	00_1005_1200	<a href="#">82/142</a>	Channel 0 CS0 row address bits.
mc_cs0_col_0	00_1005_1220	<a href="#">83/142</a>	Channel 0 CS0 column address bits.
mc_cs0_ba_0	00_1005_1240	<a href="#">84/143</a>	Channel 0 CS0 bank select.
mc_cs1_row_0	00_1005_1260	<a href="#">82/142</a>	Channel 0 CS1 row address bits.
mc_cs1_col_0	00_1005_1280	<a href="#">83/142</a>	Channel 0 CS1 column address bits.
mc_cs1_ba_0	00_1005_12a0	<a href="#">84/143</a>	Channel 0 CS1 DRAM bank select.
mc_cs2_row_0	00_1005_12c0	<a href="#">82/142</a>	Channel 0 CS2 row address bits.
mc_cs2_col_0	00_1005_12e0	<a href="#">83/142</a>	Channel 0 CS2 column address bits.
mc_cs2_ba_0	00_1005_1300	<a href="#">84/143</a>	Channel 0 CS2 DRAM bank select.
mc_cs3_row_0	00_1005_1320	<a href="#">82/142</a>	Channel 0 CS3 row address bits.
mc_cs3_col_0	00_1005_1340	<a href="#">83/142</a>	Channel 0 CS3 column address bits.
mc_cs3_ba_0	00_1005_1360	<a href="#">83/142</a>	Channel 0 CS3 DRAM bank select.

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
mc_cs_attr_0	00_1005_1380	85/143	Channel 0 CS[3:0]attribute.
mc_test_data_0	00_1005_1400	86/144	Channel 0 ECC error set data bits.
mc_test_ecc_0	00_1005_1420	87/144	Channel 0 ECC error set ecc bits.
mc_clock_cfg_0	00_1005_1500	74/138	Channel 0 memory clock ratio.
mc__1	00_1005_2000 ... 00_1005_2600	See_0 above	Memory controller channel 1 configuration (at memory controler 0 + 1000).
<b>L2 Controller</b>			
l2_read_address	00_1004_0018	56/100	Read only. Last address/tag in a read (for testing)
l2_ecc_address	00_1004_0038	56/100	Read only. Last address with ecc error (correctable or not).
l2_misc_value	00_1004_0058	57/100	Read only. PERIPH_REV3 and later. Value of L2 hidden registers.
l2_way_disable	00_1004_1x00	/91	Accesses made to this range of addresses will write the value x to l2_wayen[3:0] register. If l2_wayen[i] is clear Way i is removed from the L2 replacement algorithm. See Section: "Using the L2 Cache as Memory" on page 91
l2_cache_disable	00_1004_2x00	/94	Accesses made to this range of addresses will write the value x to l2_cache_disable[3:0] register. See Section: "Reduced Cache Size" on page 94.
l2_misc_config	00_1004_3x00	/99	Accesses made to this range of addresses will write the value x to l2_misc_config[3:0] register. See Section: "Cache Configuration Register" on page 99.
<b>PCI Host Bridge</b>			
Type 00 PCI header	00_FE00_0000 _00_FE00_00FF	127/236	Configuration space bus=0,dev=0.
<b>HyperTransport Host Bridge</b>			
Type 01 PCI header	00_FE00_0800 _00_FE00_08FF	140/244	Configuration space bus=0,dev=1.
<b>Ethernet DMA and MACs</b>			
mac_tx_byte_0	00_1006_4000	167/288	RMON transmit byte counter.
mac_collisions_0	00_1006_4008	167/288	RMON total collisions.
mac_late_col_0	00_1006_4010	167/288	RMON late collisions.
mac_ex_col_0	00_1006_4018	167/288	RMON excessive collisions.
mac_fcs_error_0	00_1006_4020	167/288	RMON packets Tx with bad FCS.
mac_tx_abort_0	00_1006_4028	167/288	RMON transmit packets aborted.
mac_tx_bad_0	00_1006_4038	167/288	RMON total of bad Tx packets.
mac_tx_good_0	00_1006_4040	167/288	RMON total sucessfully transmitted packets.
mac_tx_runt_0	00_1006_4048	167/288	RMON total runt packets transmitted.
mac_tx_oversize_0	00_1006_4050	167/288	RMON total oversize packets transmitted.



**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
mac_rx_bytes_0	00_1006_4080	167/288	RMON total received bytes.
mac_rx_mcast_0	00_1006_4088	167/288	RMON total received multicast packets.
mac_rx_bcast_0	00_1006_4090	167/288	RMON total received broadcast packets.
mac_rx_bad_0	00_1006_4098	167/288	RMON total received bad packets.
mac_rx_good_0	00_1006_40A0	167/288	RMON total received good packets.
mac_rx_runt_0	00_1006_40A8	167/288	RMON total runt packets received.
mac_rx_oversize_0	00_1006_40B0	167/288	RMON total oversize packets received.
mac_rx_fcs_error_0	00_1006_40B8	167/288	RMON total packets received with bad FCS.
mac_rx_length_error_0	00_1006_40C0	167/288	RMON total packets received with length error.
mac_rx_code_error_0	00_1006_40C8	167/288	RMON total packets received with code error.
mac_rx_align_error_0	00_1006_40D0	167/288	RMON total packets received with align error.
mac_cfg_0	00_1006_4100	176/301	Ethernet interface configuration register.
mac_thrsh_cfg_0	00_1006_4108	179/306	Ethernet interface FIFO threshold configuration register.
mac_vlantag_0	00_1006_4110	181/309	VLAN tag for insertion into packets on transmit
mac_frame_cfg_0	00_1006_4118	180/307	Ethernet interface MAC frame configuration.
mac_rx_fifo_ptrs_0	00_1006_4120	186/313	MAC FIFO pointers (Debug, Read Only).
mac_tx_fifo_ptrs_0	00_1006_4128	186/313	MAC FIFO pointers. (Debug, Read Only.)
mac_adfilter_cfg_0	00_1006_4200	192/315	MAC receive address filter configuration register.
mac_ethernet_addr_0	00_1006_4208	190/314	MAC source ethernet address for insertion during transmission.
mac_type_cfg_0	00_1006_4210	191/315	MAC packet type table for receive packet comparisons.
mac_hash0_0	00_1006_4240	189/314	Address filter hash map.
mac_hash1_0	00_1006_4248	189/314	Address filter hash map.
mac_hash2_0	00_1006_4250	189/314	Address filter hash map.
mac_hash3_0	00_1006_4258	189/314	Address filter hash map.
mac_hash4_0	00_1006_4260	189/314	Address filter hash map.
mac_hash5_0	00_1006_4268	189/314	Address filter hash map.
mac_hash6_0	00_1006_4270	189/314	Address filter hash map.
mac_hash7_0	00_1006_4278	189/314	Address filter hash map.
mac_addr0_0	00_1006_4280	187/313	Address filter exact match.
mac_admask0_0	00_1006_4218	188/314	Address filter exact match mask register (PERIPH_REV3).
mac_addr1_0	00_1006_4288	187/313	Address filter exact match.
mac_admask1_0	00_1006_4220	188/314	Address filter exact match mask register (PERIPH_REV3).
mac_addr2_0	00_1006_4290	187/313	Address filter exact match.
mac_addr3_0	00_1006_4298	187/313	Address filter exact match.
mac_addr4_0	00_1006_42A0	187/313	Address filter exact match.

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
mac_addr5_0	00_1006_42A8	187/313	Address filter exact match.
mac_addr6_0	00_1006_42B0	187/313	Address filter exact match.
mac_addr7_0	00_1006_42B8	187/313	Address filter exact match.
mac_chup0_0	00_1006_4300	193/317	Receive DMA channel select MSB.
mac_chup1_0	00_1006_4308	193/317	Receive DMA channel select MSB.
mac_chup2_0	00_1006_4310	193/317	Receive DMA channel select MSB.
mac_chup3_0	00_1006_4318	193/317	Receive DMA channel select MSB.
mac_chlo0_0	00_1006_4320	193/317	Receive DMA channel select LSB.
mac_chlo1_0	00_1006_4328	193/317	Receive DMA channel select LSB.
mac_chlo2_0	00_1006_4330	193/317	Receive DMA channel select LSB.
mac_chlo3_0	00_1006_4338	193/317	Receive DMA channel select LSB.
mac_enable_0	00_1006_4400	177/305	MAC enable register.
mac_status_0	00_1006_4408	182/309	MAC status/error register (Read Only, read clears).
mac_status1_0	00_1006_4430	183/312	MAC status/error register (Read Only, read clears ch 1).
mac_int_mask_0	00_1006_4410	185/313	MAC interrupt mask register.
dma_asic_addr_mac_0	00_1006_4418	94/166	ASIC mode base address.
mac_txd_ctl_0	00_1006_4420	178/305	Transmit DMA control register.
mac_mdio_0	00_1006_4428	194/317	MDIO pin control register.
mac_debug_status_0	00_1006_4448	184/312	MAC status/error register (Debug, Read Only, no side effects).
dma_config0_mac_0_rx_ch_0	00_1006_4800	91/163	DMA config 0 register.
dma_config1_mac_0_rx_ch_0	00_1006_4808	92/164	DMA config 1 register.
dma_dscr_base_mac_0_rx_ch_0	00_1006_4810	93/166	DMA descriptor base register.
dma_dscr_cnt_0_rx_ch_0	00_1006_4818	95/166	DMA descriptor count.
dma_dscr_a_mac_0_rx_ch_0	00_1006_4820	96/167	DMA current descriptor A.
dma_dscr_b_mac_0_rx_ch_0	00_1006_4828	97/167	DMA current descriptor B.
dma_cur_dscr_addr_mac_0_rx_ch_0	00_1006_4830	98/167	DMA current descriptor address.
dma_oodpktlost_mac_0_rx_ch_0	00_1006_4838	99/168	DMA packet lost counter. (PERIPH_REV3)
dma__mac_0_rx_ch_1	00_1006_4900		Rx channel 1 at Rx channel 0 + 100.
dma__mac_0_tx_ch_0	00_1006_4c00		Tx channel 0 at Rx channel 0 + 400.
dma__mac_0_tx_ch_1	00_1006_4d00		Tx channel 1 at Rx channel 0 + 500.
mac_1	00_1006_5000		MAC 1 registers (+1000 from mac_0 registers).
mac_2	00_1006_6000		MAC 2 registers (+2000 from mac_0 registers).

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
<b>Serial Ports</b>			
duart_mode_reg_1a	00_1006_0100	197/326	Mode register 1 port A MR1A.
duart_mode_reg_2a	00_1006_0110	198/326	Mode register 2 port A MR2A.
duart_status_a	00_1006_0120	200/327	Status register port A (Read Only).
duart_clk_sel_a	00_1006_0130	201/328	Clock select register port A.
duart_full_ctl_a	00_1006_0140	202/328	Full control port A.
duart_cmd_a	00_1006_0150	199/327	Command register port A.
duart_rx_hold_reg_a	00_1006_0160	203/328	RX holding register port A (Read Only, read pops character from FIFO).
duart_tx_hold_reg_a	00_1006_0170	204/329	TX holding register port A (Write Only).
duart_opcr_a	00_1006_0180	211/331	Output port control register alias that only alters port A bits.
duart_aux_ctrl_a	00_1006_0190	213/331	Aux control register alias that only alters port A bits.
duart_mode_reg_1b	00_1006_0200	197/326	Mode register 1 port B.
duart_mode_reg_2b	00_1006_0210	198/326	Mode register 2 port B.
duart_status_b	00_1006_0220	200/327	Status register port B (Read Only).
duart_clk_sel_b	00_1006_0230	201/328	Clock select register port B.
duart_full_ctl_b	00_1006_0240	202/328	Full control port B.
duart_cmd_b	00_1006_0250	199/327	Command register port B.
duart_rx_hold_reg_b	00_1006_0260	203/328	RX holding register port B (Read Only, read pops character from FIFO).
duart_tx_hold_reg_b	00_1006_0270	204/329	TX holding register port B (Write Only).
duart_opcr_b	00_1006_0280	211/331	Output port control register alias that only alters port B bits.
duart_aux_ctrl_b	00_1006_0290	213/331	Aux control register alias that only alters port B bits.
duart_inport_chng	00_1006_0300	206/329	Input port change register (Read Only, read clears).
duart_aux_cntrl	00_1006_0310	212/331	Aux control register.
duart_isr_a	00_1006_0320	215/332	Interrupt status register port A (Read Only).
duart_imr_a	00_1006_0330	218/333	Interrupt mask register port A.
duart_isr_b	00_1006_0340	216/332	Interrupt status register port B (Read Only).
duart_imr_b	00_1006_0350	219/333	Interrupt mask register port B.
duart_out_port	00_1006_0360	222/334	Output port register (Write Only).
duart_opcr	00_1006_0370	210/330	Output port control.
duart_in_port	00_1006_0380	205/329	Input port state (Read Only).
duart_isr	00_1006_0390	214/332	Full interrupt status (Read Only).
duart_imr	00_1006_03a0	217/333	Full interrupt mask.
duart_set_opr	00_1006_03b0	220/334	Output port register bit set (Write Only).
duart_clear_opr	00_1006_03c0	221/334	Output port register bit clear (Write Only).
duart_inport_chng_a	00_1006_03d0	208/330	Input port change register for channel A (Read Only, read clears channel A change state)

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
duart_inport_chng_b	00_1006_03E0	209/330	Input port change register for channel B (Read Only, read clears channel B change state)
duart_inport_chng_debug	00_1006_03F0	207/330	Alias of input port change register with no read side effects. (Read Only)
dma_config0_ser_0_rx	00_1006_0400	91/163	Receive DMA control register.
dma_config1_ser_0_rx	00_1006_0408	92/164	Receive DMA control register.
dma_dscr_base_ser_0_rx	00_1006_0410	93/166	Receive DMA descriptor base address.
dma_dscr_cnt_ser_0_rx	00_1006_0418	95/166	Receive DMA descriptor count .
dma_cur_dscr_a_ser_0_rx	00_1006_0420	96/167	Receive DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_0_rx	00_1006_0428	97/167	Receive DMA current descriptor B (Read Only).
dma_cur_daddr_ser_0_rx	00_1006_0430	98/167	Receive DMA current descriptor address (Read Only).
dma_config0_ser_0_tx	00_1006_0480	91/163	Transmit DMA control register.
dma_config1_ser_0_tx	00_1006_0488	92/164	Transmit DMA control register.
dma_dscr_base_ser_0_tx	00_1006_0490	93/166	Transmit DMA descriptor base address.
dma_dscr_cnt_ser_0_tx	00_1006_0498	95/166	Transmit DMA descriptor count.
dma_cur_dscr_a_ser_0_tx	00_1006_04A0	96/167	Transmit DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_0_tx	00_1006_04A8	97/167	Transmit DMA current descriptor B (Read Only).
dma_cur_daddr_ser_0_tx	00_1006_04B0	98/167	Transmit DMA current descriptor address (Read Only).
ser_mode_0	00_1006_0500	231/353	Mode select.
ser_minfrm_sz_0	00_1006_0508	237/355	Min frame size.
ser_maxfrm_sz_0	00_1006_0510	238/356	Max frame size.
ser_addr_mask_0	00_1006_0518	243/358	Address mask.
ser_usr0_addr_0	00_1006_0520	244/358	Match address 0.
ser_usr1_addr_0	00_1006_0528	244/358	Match address 1.
ser_usr2_addr_0	00_1006_0530	244/358	Match address 2.
ser_usr3_addr_0	00_1006_0538	244/358	Match address 3.
ser_cmd_0	00_1006_0540	233/354	Command
ser_tx_rd_thrsh_0	00_1006_0560	235/355	Transmit FIFO read threshold.
ser_tx_wr_thrsh_0	00_1006_0568	234/355	Transmit FIFO write threshold.
ser_rx_rd_thrsh_0	00_1006_0570	236/355	Receive FIFO read threshold.
ser_line_mode_0	00_1006_0578	232/353	Line Interface configuration register.
ser_dma_enable_0	00_1006_0580	239/356	DMA channel enable register.
ser_status_0	00_1006_0588	240/356	Serial interface and DMA status (Read Only, read clears).
ser_int_mask_0	00_1006_0590	242/357	Interrupt mask.
dma_asic_addr_ser_0	00_1006_0598	94/166	ASIC mode address.
ser_debug_status_0	00_1006_05A8	241/357	Serial interface and DMA status (Read Only, no side effects).
ser_tx_byte_lo_0	00_1006_05C0	246/359	Serial interface transmit byte count (low 16 bits).



**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
ser_tx_byte_hi_0	00_1006_05C8	246/359	Serial interface transmit byte count (high 16 bits).
ser_rx_byte_lo_0	00_1006_05D0	246/359	Serial interface receive byte count (low 16 bits).
ser_rx_byte_hi_0	00_1006_05D8	246/359	Serial interface receive byte count (high 16 bits).
ser_tx_underrun_0	00_1006_05E0	246/359	Serial interface transmit underrun count.
ser_rx_overflow_0	00_1006_05E8	246/359	Serial interface receive overflow count.
ser_rx_errors_0	00_1006_05F0	246/359	Serial interface receive error packet count.
ser_rx_badaddr_0	00_1006_05F8	246/359	Serial interface receive address mismatch count.
ser_rx_table0_0	00_1006_0600	245/358	Sequence table.
ser_rx_table1_0	00_1006_0608	245/358	Sequence table.
ser_rx_table2_0	00_1006_0610	245/358	Sequence table.
ser_rx_table3_0	00_1006_0618	245/358	Sequence table.
ser_rx_table4_0	00_1006_0620	245/358	Sequence table.
ser_rx_table5_0	00_1006_0628	245/358	Sequence table.
ser_rx_table6_0	00_1006_0630	245/358	Sequence table.
ser_rx_table7_0	00_1006_0638	245/358	Sequence table.
ser_rx_table8_0	00_1006_0640	245/358	Sequence table.
ser_rx_table9_0	00_1006_0648	245/358	Sequence table.
ser_rx_table10_0	00_1006_0650	245/358	Sequence table.
ser_rx_table11_0	00_1006_0658	245/358	Sequence table.
ser_rx_table12_0	00_1006_0660	245/358	Sequence table.
ser_rx_table13_0	00_1006_0668	245/358	Sequence table.
ser_rx_table14_0	00_1006_0670	245/358	Sequence table.
ser_rx_table15_0	00_1006_0678	245/358	Sequence table.
ser_tx_table0_0	00_1006_0700	245/358	Sequence table.
ser_tx_table1_0	00_1006_0708	245/358	Sequence table.
ser_tx_table2_0	00_1006_0710	245/358	Sequence table.
ser_tx_table3_0	00_1006_0718	245/358	Sequence table.
ser_tx_table4_0	00_1006_0720	245/358	Sequence table.
ser_tx_table5_0	00_1006_0728	245/358	Sequence table.
ser_tx_table6_0	00_1006_0730	245/358	Sequence table.
ser_tx_table7_0	00_1006_0738	245/358	Sequence table.
ser_tx_table8_0	00_1006_0740	245/358	Sequence table.
ser_tx_table9_0	00_1006_0748	245/358	Sequence table.
ser_tx_table10_0	00_1006_0750	245/358	Sequence table.
ser_tx_table11_0	00_1006_0758	245/358	Sequence table.
ser_tx_table12_0	00_1006_0760	245/358	Sequence table.
ser_tx_table13_0	00_1006_0768	245/358	Sequence table.



**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
ser_tx_table14_0	00_1006_0770	245/358	Sequence table.
ser_tx_table15_0	00_1006_0778	245/358	Sequence table.
dma_config0_ser_1_rx	00_1006_0800	91/163	Receive DMA control register.
dma_config1_ser_1_rx	00_1006_0808	92/164	Receive DMA control register.
dma_dscr_base_ser_1_rx	00_1006_0810	93/166	Receive DMA descriptor base address.
dma_dscr_count_ser_1_rx	00_1006_0818	95/166	Receive DMA descriptor count.
dma_cur_dscr_a_ser_1_rx	00_1006_0820	96/167	Receive DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_1_rx	00_1006_0828	97/167	Receive DMA current descriptor B (Read Only).
dma_cur_daddr_ser_1_rx	00_1006_0830	98/167	Receive DMA current descriptor address (Read Only).
dma_config0_ser_1_tx	00_1006_0880	91/163	Transmit DMA control register.
dma_config1_ser_1_tx	00_1006_0888	92/164	Transmit DMA control register.
dma_dscr_base_ser_1_tx	00_1006_0890	93/166	Transmit DMA descriptor base address.
dma_dscr_count_ser_1_tx	00_1006_0898	95/166	Transmit DMA descriptor count.
dma_cur_dscr_a_ser_1_tx	00_1006_08A0	96/167	Transmit DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_1_tx	00_1006_08A8	97/167	Transmit DMA current descriptor B (Read Only).
dma_cur_daddr_ser_1_tx	00_1006_08B0	98/167	Transmit DMA current descriptor address (Read Only).
ser_mode_1	00_1006_0900	231/353	Mode select.
ser_minfrm_sz_1	00_1006_0908	237/355	Min frame size.
ser_maxfrm_sz_1	00_1006_0910	238/356	Max frame size.
ser_addr_mask_1	00_1006_0918	243/358	Address mask.
ser_usr0_addr_1	00_1006_0920	244/358	Match address 0.
ser_usr1_addr_1	00_1006_0928	244/358	Match address 1.
ser_usr2_addr_1	00_1006_0930	244/358	Match address 2.
ser_usr3_addr_1	00_1006_0938	244/358	Match address 3.
ser_cmd_1	00_1006_0940	233/354	Command
ser_tx_rd_thrsh_1	00_1006_0960	235/355	Transmit FIFO read threshold.
ser_tx_wr_thrsh_1	00_1006_0968	234/355	Transmit FIFO write threshold.
ser_rx_rd_thrsh_1	00_1006_0970	236/355	Receive FIFO read threshold.
ser_line_mode_1	00_1006_0978	232/353	Line Interface configuration register.
ser_dma_enable_1	00_1006_0980	239/356	DMA channel enable register.
ser_status_1	00_1006_0988	240/356	Serial interface and DMA status (Read Only, read clears).
ser_int_mask_1	00_1006_0990	242/357	Interrupt mask.
dma_asic_addr_ser_1	00_1006_0998	94/166	ASIC mode address.
ser_debug_status_1	00_1006_09A8	241/357	Serial interface and DMA status (Read Only, no side effects).
ser_tx_byte_lo_1	00_1006_09C0	246/359	Serial interface transmit byte count (low 16 bits).
ser_tx_byte_hi_1	00_1006_09C8	246/359	Serial interface transmit byte count (high 16 bits).



**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
ser_rx_byte_lo_1	00_1006_09D0	246/359	Serial interface receive byte count (low 16 bits).
ser_rx_byte_hi_1	00_1006_09D8	246/359	Serial interface receive byte count (high 16 bits).
ser_tx_underrun_1	00_1006_09E0	246/359	Serial interface transmit underrun count.
ser_rx_overflow_1	00_1006_09E8	246/359	Serial interface receive overflow count.
ser_rx_errors_1	00_1006_09F0	246/359	Serial interface receive error packet count.
ser_rx_badaddr_1	00_1006_09F8	246/359	Serial interface receive address mismatch count.
ser_rx_table0_1	00_1006_0A00	245/358	Sequence table.
ser_rx_table1_1	00_1006_0A08	245/358	Sequence table.
ser_rx_table2_1	00_1006_0A10	245/358	Sequence table.
ser_rx_table3_1	00_1006_0A18	245/358	Sequence table.
ser_rx_table4_1	00_1006_0A20	245/358	Sequence table.
ser_rx_table5_1	00_1006_0A28	245/358	Sequence table.
ser_rx_table6_1	00_1006_0A30	245/358	Sequence table.
ser_rx_table7_1	00_1006_0A38	245/358	Sequence table.
ser_rx_table8_1	00_1006_0A40	245/358	Sequence table.
ser_rx_table9_1	00_1006_0A48	245/358	Sequence table.
ser_rx_table10_1	00_1006_0A50	245/358	Sequence table.
ser_rx_table11_1	00_1006_0A58	245/358	Sequence table.
ser_rx_table12_1	00_1006_0A60	245/358	Sequence table.
ser_rx_table13_1	00_1006_0A68	245/358	Sequence table.
ser_rx_table14_1	00_1006_0A70	245/358	Sequence table.
ser_rx_table15_1	00_1006_0A78	245/358	Sequence table.
ser_tx_table0_1	00_1006_0B00	245/358	Sequence table.
ser_tx_table1_1	00_1006_0B08	245/358	Sequence table.
ser_tx_table2_1	00_1006_0B10	245/358	Sequence table.
ser_tx_table3_1	00_1006_0B18	245/358	Sequence table.
ser_tx_table4_1	00_1006_0B20	245/358	Sequence table.
ser_tx_table5_1	00_1006_0B28	245/358	Sequence table.
ser_tx_table6_1	00_1006_0B30	245/358	Sequence table.
ser_tx_table7_1	00_1006_0B38	245/358	Sequence table.
ser_tx_table8_1	00_1006_0B40	245/358	Sequence table.
ser_tx_table9_1	00_1006_0B48	245/358	Sequence table.
ser_tx_table10_1	00_1006_0B50	245/358	Sequence table.
ser_tx_table11_1	00_1006_0B58	245/358	Sequence table.
ser_tx_table12_1	00_1006_0B60	245/358	Sequence table.
ser_tx_table13_1	00_1006_0B68	245/358	Sequence table.
ser_tx_table14_1	00_1006_0B70	245/358	Sequence table.

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
ser_tx_table15_1	00_1006_0B78	245/358	Sequence table.
<b>Generic Bus</b>			
io_ext_cfg_0	00_1006_1000	251/374	cs0 interface configuration.
io_ext_cfg_1	00_1006_1008	251/374	cs1 interface configuration.
io_ext_cfg_2	00_1006_1010	251/374	cs2 interface configuration.
io_ext_cfg_3	00_1006_1018	251/374	cs3 interface configuration.
io_ext_cfg_4	00_1006_1020	251/374	cs4 interface configuration.
io_ext_cfg_5	00_1006_1028	251/374	cs5 interface configuration.
io_ext_cfg_6	00_1006_1030	251/374	cs6 interface configuration.
io_ext_cfg_7	00_1006_1038	251/374	cs7 interface configuration.
io_ext_mult_size_0	00_1006_1100	252/374	cs0 region size in 64KB units.
io_ext_mult_size_1	00_1006_1108	252/374	cs1 region size in 64KB units.
io_ext_mult_size_2	00_1006_1110	252/374	cs2 region size in 64KB units.
io_ext_mult_size_3	00_1006_1118	252/374	cs3 region size in 64KB units.
io_ext_mult_size_4	00_1006_1120	252/374	cs4 region size in 64KB units.
io_ext_mult_size_5	00_1006_1128	252/374	cs5 region size in 64KB units.
io_ext_mult_size_6	00_1006_1130	252/374	cs6 region size in 64KB units.
io_ext_mult_size_7	00_1006_1138	252/374	cs7 region size in 64KB units.
io_ext_start_addr_0	00_1006_1200	253/375	cs0 region start address (put outside generic bus range to disable).
io_ext_start_addr_1	00_1006_1208	253/375	cs1 region start address (put outside generic bus range to disable).
io_ext_start_addr_2	00_1006_1210	253/375	cs2 region start address (put outside generic bus range to disable).
io_ext_start_addr_3	00_1006_1218	253/375	cs3 region start address (put outside generic bus range to disable).
io_ext_start_addr_4	00_1006_1220	253/375	cs4 region start address (put outside generic bus range to disable).
io_ext_start_addr_5	00_1006_1228	253/375	cs5 region start address (put outside generic bus range to disable).
io_ext_start_addr_6	00_1006_1230	253/375	cs6 region start address (put outside generic bus range to disable).
io_ext_start_addr_7	00_1006_1238	253/375	cs7 region start address (put outside generic bus range to disable).
io_ext_time_cfg0_0	00_1006_1600	254/375	cs0 Timing parameter configuration 0.
io_ext_time_cfg0_1	00_1006_1608	254/375	cs1 Timing parameter configuration 0.
io_ext_time_cfg0_2	00_1006_1610	254/375	cs2 Timing parameter configuration 0.
io_ext_time_cfg0_3	00_1006_1618	254/375	cs3 Timing parameter configuration 0.
io_ext_time_cfg0_4	00_1006_1620	254/375	cs4 Timing parameter configuration 0.
io_ext_time_cfg0_5	00_1006_1628	254/375	cs5 Timing parameter configuration 0.

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
io_ext_time_cfg0_6	00_1006_1630	<a href="#">254/375</a>	cs6 Timing parameter configuration 0.
io_ext_time_cfg0_7	00_1006_1638	<a href="#">254/375</a>	cs7 Timing parameter configuration 0.
io_ext_time_cfg1_0	00_1006_1700	<a href="#">255/376</a>	cs0 Timing parameter configuration 1.
io_ext_time_cfg1_1	00_1006_1708	<a href="#">255/376</a>	cs1 Timing parameter configuration 1.
io_ext_time_cfg1_2	00_1006_1710	<a href="#">255/376</a>	cs2 Timing parameter configuration 1.
io_ext_time_cfg1_3	00_1006_1718	<a href="#">255/376</a>	cs3 Timing parameter configuration 1.
io_ext_time_cfg1_4	00_1006_1720	<a href="#">255/376</a>	cs4 Timing parameter configuration 1.
io_ext_time_cfg1_5	00_1006_1728	<a href="#">255/376</a>	cs5 Timing parameter configuration 1.
io_ext_time_cfg1_6	00_1006_1730	<a href="#">255/376</a>	cs6 Timing parameter configuration 1.
io_ext_time_cfg1_7	00_1006_1738	<a href="#">255/376</a>	cs7 Timing parameter configuration 1.
io_interrupt_status	00_1006_1a00	<a href="#">256/376</a>	Interrupt status for generic bus sources (Read Only, read clears).
io_interrupt_data0	00_1006_1a10	<a href="#">257/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_data1	00_1006_1a18	<a href="#">258/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_data2	00_1006_1a20	<a href="#">259/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_data3	00_1006_1a28	<a href="#">260/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_addr0	00_1006_1a30	<a href="#">261/378</a>	Address latched on generic interrupt assertion (Read Only).
io_interrupt_addr1	00_1006_1a40	<a href="#">262/378</a>	Address latched on generic interrupt assertion (Read Only).
io_interrupt_parity	00_1006_1a50	<a href="#">263/378</a>	Parity latched on generic interrupt assertion (Read Only).
pcmcia_cfg	00_1006_1a60	<a href="#">273/393</a>	PCICMA controller configuration.
pcmcia_status	00_1006_1a70	<a href="#">274/394</a>	PCMCIA controller status (Read Only, read clears interrupt).
<b>GPIO</b>			
gpio_clr_edge	00_1006_1a80	<a href="#">276/398</a>	Write a 1 to clear the edge detector for that bit (Write Only).
gpio_int_type	00_1006_1a88	<a href="#">277/398</a>	Select interrupt type for pairs of pins.
gpio_input_invert	00_1006_1a90	<a href="#">279/398</a>	Set to invert the input.
gpio_glitch	00_1006_1a98	<a href="#">280/399</a>	Set for 1ms glitch filter, clear for 60 ns.
gpio_read	00_1006_1aa0	<a href="#">278/398</a>	Shows current value of signal after possible invert and glitch (Read Only).
gpio_direction	00_1006_1aa8	<a href="#">281/399</a>	GPIO pin data direction. 1 for output, 0 for input.
gpio_pin_clr	00_1006_1ab0	<a href="#">282/399</a>	Write 1 to set that bit to zero (Write Only).
gpio_pin_set	00_1006_1ab8	<a href="#">283/399</a>	Write 1 to set that bit to one (Write Only).

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
<b>Output Drive Control</b>			
io_drive_0	00_1006_1300	264/378	3.3V output drive strength and slew rate control registers. See Section: "Drive Strength Control" on page 373.
io_drive_1	00_1006_1308	265/379	
io_drive_2	00_1006_1310	266/380	
io_drive_3	00_1006_1318	267/380	
<b>SMBus</b>			
smb_serial_xtra_0	00_1006_0000	294/416	SMB port extra data register.
smb_serial_xtra_1	00_1006_0008	294/416	SMB port extra data register.
smb_serial_freq_0	00_1006_0010	289/415	Frequency for SMB port.
smb_serial_freq_1	00_1006_0018	289/415	Frequency for SMB port.
smb_serial_status_0	00_1006_0020	292/416	SMB port status (Read clears finish interrupt).
smb_serial_status_1	00_1006_0028	292/416	SMB port status (Read clears finish interrupt).
smb_serial_cmd_0	00_1006_0030	290/415	SMB port command.
smb_serial_cmd_1	00_1006_0038	290/415	SMB port command.
smb_serial_start_0	00_1006_0040	296/417	SMB port start register.
smb_serial_start_1	00_1006_0048	296/417	SMB port start register.
smb_serial_data_0	00_1006_0050	293/416	SMB port data register.
smb_serial_data_1	00_1006_0058	293/416	SMB port data register.
smb_serial_control_0	00_1006_0060	291/415	SMB port control register.
smb_serial_control_1	00_1006_0068	291/415	SMB port control register.
smb_serial_pec_0	00_1006_0070	295/417	SMB port pec register.
smb_serial_pec_1	00_1006_0078	295/417	SMB port pec register.
<b>SCD: Timers</b>			
watchdog_timer_init_cnt_0	00_1002_0050	23/59	Watchdog initial count.
watchdog_timer_cnt_0	00_1002_0058	24/59	Watchdog current count (Read Only).
watchdog_timer_cfg_0	00_1002_0060	25/59	Watchdog configuration (Write clears interrupt).
general_timer_init_cnt_0	00_1002_0070	26/60	General timer initial count.
general_timer_init_cnt_1	00_1002_0078	26/60	General timer initial count.
general_timer_cnt_0	00_1002_0080	27/60	General timer current count (Read Only).
general_timer_cnt_1	00_1002_0088	27/60	General timer current count (Read Only).
general_timer_cfg_0	00_1002_0090	28/60	General timer configuration and enable (Write clears interrupt).
general_timer_cfg_1	00_1002_0098	28/60	General timer configuration and enable (Write clears interrupt).
watchdog_timer_init_cnt_1	00_1002_0150	23/59	Watchdog initial count.
watchdog_timer_cnt_1	00_1002_0158	24/59	Watchdog current count (Read Only).
watchdog_timer_cfg_1	00_1002_0160	25/59	Watchdog configuration (Write clears interrupt).
general_timer_init_cnt_2	00_1002_0170	26/60	General timer initial count.



**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
general_timer_init_cnt_3	00_1002_0178	26/60	General timer initial count.
general_timer_cnt_2	00_1002_0180	27/60	General timer current count (Read Only).
general_timer_cnt_3	00_1002_0188	27/60	General timer current count (Read Only).
general_timer_cfg_2	00_1002_0190	28/60	General timer configuration and enable (Write clears interrupt).
general_timer_cfg_3	00_1002_0198	28/60	General timer configuration and enable (Write clears interrupt).
zbbus_cycle_cnt	00_1003_0000	29/61	ZBbus cycle count.
zbbus_cycle_cp0	00_1002_0c00	30/61	ZBbus cycle count compare register zero.
zbbus_cycle_cp1	00_1002_0c08	30/61	ZBbus cycle count compare register one.
<b>SCD: System Control</b>			
system_revision	00_1002_0000	14/43	System Id and revision (Read Only).
system_cfg	00_1002_0008	15/43	System configuration register.
system_scratch	00_1002_0c10	16/45	Scratch register for software use
system_manuf	00_1003_8000	14/43	Read Only. Manufacturing Information Register.
<b>SCD: Address Trap</b>			
addr_trap_index	00_1002_00b0	40/68	Index of interrupting trap (Read Only).
addr_trap_reg	00_1002_00b8	42/68	Address of interrupting trap (Read Only, read clears interrupt).
addr_trap_reg_debug	00_1002_0460	41/68	Address of interrupting trap (Read Only, No side effects).
addr_trap_up_cfg_0	00_1002_0400	43/68	Top of address trap range.
addr_trap_up_cfg_1	00_1002_0408	43/68	Top of address trap range.
addr_trap_up_cfg_2	00_1002_0410	43/68	Top of address trap range.
addr_trap_up_cfg_3	00_1002_0418	43/68	Top of address trap range.
addr_trap_down_cfg_0	00_1002_0420	44/69	Bottom of address trap range.
addr_trap_down_cfg_1	00_1002_0428	44/69	Bottom of address trap range.
addr_trap_down_cfg_2	00_1002_0430	44/69	Bottom of address trap range.
addr_trap_down_cfg_3	00_1002_0438	44/69	Bottom of address trap range.
addr_trap_cfg_0	00_1002_0440	45/69	Address trap configuration.
addr_trap_cfg_1	00_1002_0448	45/69	Address trap configuration.
addr_trap_cfg_2	00_1002_0450	45/69	Address trap configuration.
addr_trap_cfg_3	00_1002_0458	45/69	Address trap configuration.

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
<b>SCD: Interrupt Mapper</b>			
interrupt_diag_0	00_1002_0010	21/52	Force interrupt diagnostic register.
interrupt_mask_0	00_1002_0028	21/52	Interrupt mask.
interrupt_source_status_0	00_1002_0040	21/52	Status of system interrupt sources (Read Only).
ldt_interrupt_0	00_1002_0018	21/52	Status of HyperTransport interrupt sources (Read Only).
ldt_interrupt_clr_0	00_1002_0020	21/52	Clear ldt_interrupt_0 by writing 1s to this location.
interrupt_trace_0	00_1002_0038	21/52	Set bits in this register to cause the corresponding interrupt to be passed to the trace trigger logic.
ldt_interrupt_set	00_1002_0048	19/49	Write HyperTransport interrupt message to this address to raise a HyperTransport interrupt (Write Only).
mailbox_cpu_0	00_1002_00c0	17/46	Status of mailbox (Read Only, has alias).
alias_mailbox_cpu_0	00_1002_1xx0	17/46	Alias of status of mailbox (has incomplete decode, so access from pci won't hang). (Read Only).
mailbox_set_cpu_0	00_1002_00c8	17/46	Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only).
alias_mailbox_set_cpu_0	00_1002_1xx8	17/46	Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only, has incomplete decode, so access from pci won't hang).
mailbox_clr_cpu_0	00_1002_00d0	17/46	Clear bits in mailbox_cpu_0 by writing 1s to this location (Write Only).
interrupt_status0_0	00_1002_0100	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status1_0	00_1002_0108	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status2_0	00_1002_0110	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status3_0	00_1002_0118	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status4_0	00_1002_0120	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status5_0	00_1002_0128	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status6_0	00_1002_0130	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status7_0	00_1002_0138	21/52	Status of mapped interrupt sources (Read Only).
interrupt_map0_0	00_1002_0200	18/47	Source to IRQ map register.
interrupt_map1_0	00_1002_0208	18/47	Source to IRQ map register.
interrupt_map2_0	00_1002_0210	18/47	Source to IRQ map register.
interrupt_map3_0	00_1002_0218	18/47	Source to IRQ map register.
interrupt_map4_0	00_1002_0220	18/47	Source to IRQ map register.
interrupt_map5_0	00_1002_0228	18/47	Source to IRQ map register.
interrupt_map6_0	00_1002_0230	18/47	Source to IRQ map register.
interrupt_map7_0	00_1002_0238	18/47	Source to IRQ map register.
interrupt_map8_0	00_1002_0240	18/47	Source to IRQ map register.
interrupt_map9_0	00_1002_0248	18/47	Source to IRQ map register.
interrupt_map10_0	00_1002_0250	18/47	Source to IRQ map register.



**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
interrupt_map11_0	00_1002_0258	18/47	Source to IRQ map register.
interrupt_map12_0	00_1002_0260	18/47	Source to IRQ map register.
interrupt_map13_0	00_1002_0268	18/47	Source to IRQ map register.
interrupt_map14_0	00_1002_0270	18/47	Source to IRQ map register.
interrupt_map15_0	00_1002_0278	18/47	Source to IRQ map register.
interrupt_map16_0	00_1002_0280	18/47	Source to IRQ map register.
interrupt_map17_0	00_1002_0288	18/47	Source to IRQ map register.
interrupt_map18_0	00_1002_0290	18/47	Source to IRQ map register.
interrupt_map19_0	00_1002_0298	18/47	Source to IRQ map register.
interrupt_map20_0	00_1002_02a0	18/47	Source to IRQ map register.
interrupt_map21_0	00_1002_02a8	18/47	Source to IRQ map register.
interrupt_map22_0	00_1002_02b0	18/47	Source to IRQ map register.
interrupt_map23_0	00_1002_02b8	18/47	Source to IRQ map register.
interrupt_map24_0	00_1002_02c0	18/47	Source to IRQ map register.
interrupt_map25_0	00_1002_02c8	18/47	Source to IRQ map register.
interrupt_map26_0	00_1002_02d0	18/47	Source to IRQ map register.
interrupt_map27_0	00_1002_02d8	18/47	Source to IRQ map register.
interrupt_map28_0	00_1002_02e0	18/47	Source to IRQ map register.
interrupt_map29_0	00_1002_02e8	18/47	Source to IRQ map register.
interrupt_map30_0	00_1002_02f0	18/47	Source to IRQ map register.
interrupt_map31_0	00_1002_02f8	18/47	Source to IRQ map register.
interrupt_map32_0	00_1002_0300	18/47	Source to IRQ map register.
interrupt_map33_0	00_1002_0308	18/47	Source to IRQ map register.
interrupt_map34_0	00_1002_0310	18/47	Source to IRQ map register.
interrupt_map35_0	00_1002_0318	18/47	Source to IRQ map register.
interrupt_map36_0	00_1002_0320	18/47	Source to IRQ map register.
interrupt_map37_0	00_1002_0328	18/47	Source to IRQ map register.
interrupt_map38_0	00_1002_0330	18/47	Source to IRQ map register.
interrupt_map39_0	00_1002_0338	18/47	Source to IRQ map register.
interrupt_map40_0	00_1002_0340	18/47	Source to IRQ map register.
interrupt_map41_0	00_1002_0348	18/47	Source to IRQ map register.
interrupt_map42_0	00_1002_0350	18/47	Source to IRQ map register.
interrupt_map43_0	00_1002_0358	18/47	Source to IRQ map register.
interrupt_map44_0	00_1002_0360	18/47	Source to IRQ map register.
interrupt_map45_0	00_1002_0368	18/47	Source to IRQ map register.
interrupt_map46_0	00_1002_0370	18/47	Source to IRQ map register.
interrupt_map47_0	00_1002_0378	18/47	Source to IRQ map register.



**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
interrupt_map48_0	00_1002_0380	18/47	Source to IRQ map register.
interrupt_map49_0	00_1002_0388	18/47	Source to IRQ map register.
interrupt_map50_0	00_1002_0390	18/47	Source to IRQ map register.
interrupt_map51_0	00_1002_0398	18/47	Source to IRQ map register.
interrupt_map52_0	00_1002_03a0	18/47	Source to IRQ map register.
interrupt_map53_0	00_1002_03a8	18/47	Source to IRQ map register.
interrupt_map54_0	00_1002_03b0	18/47	Source to IRQ map register.
interrupt_map55_0	00_1002_03b8	18/47	Source to IRQ map register.
interrupt_map56_0	00_1002_03c0	18/47	Source to IRQ map register.
interrupt_map57_0	00_1002_03c8	18/47	Source to IRQ map register.
interrupt_map58_0	00_1002_03d0	18/47	Source to IRQ map register.
interrupt_map59_0	00_1002_03d8	18/47	Source to IRQ map register.
interrupt_map60_0	00_1002_03e0	18/47	Source to IRQ map register.
interrupt_map61_0	00_1002_03e8	18/47	Source to IRQ map register.
interrupt_map62_0	00_1002_03f0	18/47	Source to IRQ map register.
interrupt_map63_0	00_1002_03f8	18/47	Source to IRQ map register.
int_mapper_1	00_1002_2000		2000-23f8 Registers for CPU 1 interrupt mapper (+2000 from CPU 0).
<b>SCD: Performance Counters</b>			
perf_cnt_cfg	00_1002_04c0	31/61	Performance counter control.
perf_cnt_0	00_1002_04d0	32/62	Performance counter.
perf_cnt_1	00_1002_04d8	32/62	Performance counter.
perf_cnt_2	00_1002_04e0	32/62	Performance counter.
perf_cnt_3	00_1002_04e8	32/62	Performance counter.
<b>SCD: Bus Watcher</b>			
bus_err_status	00_1002_0880	35/65	Bus error status (Read Only).
bus_err_status_debug	00_1002_08D0	36/65	Bus error status (Read Only).
bus_err_data_0	00_1002_08a0	37/65	Data or address/control information latched on bus error (Read Only).
bus_err_data_1	00_1002_08a8	37/65	Data or address/control information latched on bus error (Read Only).
bus_err_data_2	00_1002_08b0	37/65	Data or address/control information latched on bus error (Read Only).
bus_err_data_3	00_1002_08b8	37/65	Data or address/control information latched on bus error (Read Only).
bus_l2_errors	00_1002_08c0	38/66	Count of L2 ECC errors (4 8-bit counters).
bus_mem_io_errors	00_1002_08c8	39/66	Count of memory ECC errors and I/O errors (3 8-bit counters).





Table 311: Internal Register Addresses by Function (Cont.)

Name	Address	Table/ Page	Description
<b>SCD: Debug Controller</b>			
jtag_space	00_1000_0000		JTAG serviced debug space 1000_0000-1001_ffff.
<b>SCD: Trace Buffer</b>			
trace_cfg	00_1002_0a00	48/76	Trace configuration.
trace_read	00_1002_0a08	50/79	Trace buffer read register (Read Only).
trace_event_0	00_1002_0a20	46/72	Trace event selector.
trace_event_1	00_1002_0a28	46/72	Trace event selector.
trace_event_2	00_1002_0a30	46/72	Trace event selector.
trace_event_3	00_1002_0a38	46/72	Trace event selector.
trace_sequence_0	00_1002_0a40	47/74	Trace sequence and action.
trace_sequence_1	00_1002_0a48	47/74	Trace sequence and action.
trace_sequence_2	00_1002_0a50	47/74	Trace sequence and action.
trace_sequence_3	00_1002_0a58	47/74	Trace sequence and action.
trace_event_4	00_1002_0a60	46/72	Trace event selector.
trace_event_5	00_1002_0a68	46/72	Trace event selector.
trace_event_6	00_1002_0a70	46/72	Trace event selector.
trace_event_7	00_1002_0a78	46/72	Trace event selector.
trace_sequence_4	00_1002_0a80	47/74	Trace sequence and action.
trace_sequence_5	00_1002_0a88	47/74	Trace sequence and action.
trace_sequence_6	00_1002_0a90	47/74	Trace sequence and action.
trace_sequence_7	00_1002_0a98	47/74	Trace sequence and action.
<b>SCD: Data Mover</b>			
dm_dscr_base_0	00_1002_0b00	114/184	Data Mover channel 0 ring base address (Read clears some bits).
dm_dscr_count_0	00_1002_0b08	116/185	Data Mover channel 0 descriptor count.
dm_dscr_addr_0	00_1002_0b10	117/185	Data Mover channel 0 current address (Read Only).
dm_debug_dscr_base_0	00_1002_0b18	115/184	Debug alias for channel 0 ring base address (Read Only, no side effects).
dm_partial_0	00_1002_0ba0	120/186	CRC/Checksum partial result register.
dm_dscr_base_1	00_1002_0b20	114/184	Data Mover channel 1 ring base address (Read clears some bits).
dm_dscr_count_1	00_1002_0b28	116/185	Data Mover channel 1 descriptor count.
dm_dscr_addr_1	00_1002_0b30	117/185	Data Mover channel 1 current address. (Read Only)
dm_debug_dscr_base_1	00_1002_0b38	115/184	Debug alias for channel 1 ring base address (Read Only, no side effects).
dm_partial_1	00_1002_0ba8	120/186	CRC/Checksum partial result register.
dm_dscr_base_2	00_1002_0b40	114/184	Data Mover channel 2 ring base address (Read clears some bits).
dm_dscr_count_2	00_1002_0b48	116/185	Data Mover channel 2 descriptor count.

**Table 311: Internal Register Addresses by Function (Cont.)**

Name	Address	Table/ Page	Description
dm_dscr_addr_2	00_1002_0b50	117/185	Data Mover channel 2 current address (Read Only).
dm_debug_dscr_base_2	00_1002_0b58	115/184	Debug alias for channel 2 ring base address (Read Only, no side effects).
dm_partial_2	00_1002_0bb0	120/186	CRC/Checksum partial result register.
dm_dscr_base_3	00_1002_0b60	114/184	Data Mover channel 3 ring base address (Read clears some bits).
dm_dscr_count_3	00_1002_0b68	116/185	Data Mover channel 3 descriptor count.
dm_dscr_addr_3	00_1002_0b70	117/185	Data Mover channel 3 current address (Read Only).
dm_debug_dscr_base_3	00_1002_0b78	115/184	Debug alias for channel 3 ring base address (Read Only, no side effects).
dm_partial_3	00_1002_0bb8	120/186	CRC/Checksum partial result register.
crc_def_0	00_1002_0b80	118/185	CRC definition register.
ctcp_def_0	00_1002_0b88	119/186	CRC and checksum definition register.
crc_def_1	00_1002_0b90	118/185	CRC definition register.
ctcp_def_1	00_1002_0b98	119/186	CRC and checksum definition register.



## BCM1250/BCM1125/H INTERNAL REGISTERS ORDERED BY ADDRESS

*Table 312: Internal Registers Ordered by Address*

Name	Address	Table/ Page	Description
jtag_space	00_1000_0000		JTAG serviced debug space 1000_0000-1001_ffff.
system_revision	00_1002_0000	14/43	System Id and revision (Read Only).
system_cfg	00_1002_0008	15/43	System configuration register.
interrupt_diag_0	00_1002_0010	21/52	Force interrupt diagnostic register.
ldt_interrupt_0	00_1002_0018	21/52	Status of HyperTransport interrupt sources (Read Only).
ldt_interrupt_clr_0	00_1002_0020	21/52	Clear ldt_interrupt_0 by writing 1s to this location.
interrupt_mask_0	00_1002_0028	21/52	Interrupt mask.
interrupt_trace_0	00_1002_0038	21/52	Set bits in this register to cause the corresponding interrupt to be passed to the trace trigger logic.
interrupt_source_status_0	00_1002_0040	21/52	Status of system interrupt sources (Read Only).
ldt_interrupt_set	00_1002_0048	19/49	Write HyperTransport interrupt message to this address to raise a HyperTransport interrupt (Write Only).
watchdog_timer_init_cnt_0	00_1002_0050	23/59	Watchdog initial count.
watchdog_timer_cnt_0	00_1002_0058	24/59	Watchdog current count (Read Only).
watchdog_timer_cfg_0	00_1002_0060	25/59	Watchdog configuration (Write clears interrupt).
general_timer_init_cnt_0	00_1002_0070	26/60	General timer initial count.
general_timer_init_cnt_1	00_1002_0078	26/60	General timer initial count.
general_timer_cnt_0	00_1002_0080	27/60	General timer current count (Read Only).
general_timer_cnt_1	00_1002_0088	27/60	General timer current count (Read Only).
general_timer_cfg_0	00_1002_0090	28/60	General timer configuration and enable (Write clears interrupt).
general_timer_cfg_1	00_1002_0098	28/60	General timer configuration and enable (Write clears interrupt).
addr_trap_index	00_1002_00b0	40/68	Index of interrupting trap (Read Only).
addr_trap_reg	00_1002_00b8	42/68	Address of interrupting trap (Read Only, read clears interrupt).
mailbox_cpu_0	00_1002_00c0	17/46	Status of mailbox (Read Only, has alias).
mailbox_set_cpu_0	00_1002_00c8	17/46	Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only).
mailbox_clr_cpu_0	00_1002_00d0	17/46	Clear bits in mailbox_cpu_0 by writing 1s to this location (Write Only).
interrupt_status0_0	00_1002_0100	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status1_0	00_1002_0108	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status2_0	00_1002_0110	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status3_0	00_1002_0118	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status4_0	00_1002_0120	21/52	Status of mapped interrupt sources (Read Only).

**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
interrupt_status5_0	00_1002_0128	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status6_0	00_1002_0130	21/52	Status of mapped interrupt sources (Read Only).
interrupt_status7_0	00_1002_0138	21/52	Status of mapped interrupt sources (Read Only).
watchdog_timer_init_cnt_1	00_1002_0150	23/59	Watchdog initial count.
watchdog_timer_cnt_1	00_1002_0158	24/59	Watchdog current count (Read Only).
watchdog_timer_cfg_1	00_1002_0160	25/59	Watchdog configuration (Write clears interrupt).
general_timer_init_cnt_2	00_1002_0170	26/60	General timer initial count.
general_timer_init_cnt_3	00_1002_0178	26/60	General timer initial count.
general_timer_cnt_2	00_1002_0180	27/60	General timer current count (Read Only).
general_timer_cnt_3	00_1002_0188	27/60	General timer current count (Read Only).
general_timer_cfg_2	00_1002_0190	28/60	General timer configuration and enable (Write clears interrupt).
general_timer_cfg_3	00_1002_0198	28/60	General timer configuration and enable (Write clears interrupt).
interrupt_map0_0	00_1002_0200	18/47	Source to IRQ map register.
interrupt_map1_0	00_1002_0208	18/47	Source to IRQ map register.
interrupt_map2_0	00_1002_0210	18/47	Source to IRQ map register.
interrupt_map3_0	00_1002_0218	18/47	Source to IRQ map register.
interrupt_map4_0	00_1002_0220	18/47	Source to IRQ map register.
interrupt_map5_0	00_1002_0228	18/47	Source to IRQ map register.
interrupt_map6_0	00_1002_0230	18/47	Source to IRQ map register.
interrupt_map7_0	00_1002_0238	18/47	Source to IRQ map register.
interrupt_map8_0	00_1002_0240	18/47	Source to IRQ map register.
interrupt_map9_0	00_1002_0248	18/47	Source to IRQ map register.
interrupt_map10_0	00_1002_0250	18/47	Source to IRQ map register.
interrupt_map11_0	00_1002_0258	18/47	Source to IRQ map register.
interrupt_map12_0	00_1002_0260	18/47	Source to IRQ map register.
interrupt_map13_0	00_1002_0268	18/47	Source to IRQ map register.
interrupt_map14_0	00_1002_0270	18/47	Source to IRQ map register.
interrupt_map15_0	00_1002_0278	18/47	Source to IRQ map register.
interrupt_map16_0	00_1002_0280	18/47	Source to IRQ map register.
interrupt_map17_0	00_1002_0288	18/47	Source to IRQ map register.
interrupt_map18_0	00_1002_0290	18/47	Source to IRQ map register.
interrupt_map19_0	00_1002_0298	18/47	Source to IRQ map register.
interrupt_map20_0	00_1002_02a0	18/47	Source to IRQ map register.
interrupt_map21_0	00_1002_02a8	18/47	Source to IRQ map register.
interrupt_map22_0	00_1002_02b0	18/47	Source to IRQ map register.
interrupt_map23_0	00_1002_02b8	18/47	Source to IRQ map register.



**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
interrupt_map24_0	00_1002_02c0	18/47	Source to IRQ map register.
interrupt_map25_0	00_1002_02c8	18/47	Source to IRQ map register.
interrupt_map26_0	00_1002_02d0	18/47	Source to IRQ map register.
interrupt_map27_0	00_1002_02d8	18/47	Source to IRQ map register.
interrupt_map28_0	00_1002_02e0	18/47	Source to IRQ map register.
interrupt_map29_0	00_1002_02e8	18/47	Source to IRQ map register.
interrupt_map30_0	00_1002_02f0	18/47	Source to IRQ map register.
interrupt_map31_0	00_1002_02f8	18/47	Source to IRQ map register.
interrupt_map32_0	00_1002_0300	18/47	Source to IRQ map register.
interrupt_map33_0	00_1002_0308	18/47	Source to IRQ map register.
interrupt_map34_0	00_1002_0310	18/47	Source to IRQ map register.
interrupt_map35_0	00_1002_0318	18/47	Source to IRQ map register.
interrupt_map36_0	00_1002_0320	18/47	Source to IRQ map register.
interrupt_map37_0	00_1002_0328	18/47	Source to IRQ map register.
interrupt_map38_0	00_1002_0330	18/47	Source to IRQ map register.
interrupt_map39_0	00_1002_0338	18/47	Source to IRQ map register.
interrupt_map40_0	00_1002_0340	18/47	Source to IRQ map register.
interrupt_map41_0	00_1002_0348	18/47	Source to IRQ map register.
interrupt_map42_0	00_1002_0350	18/47	Source to IRQ map register.
interrupt_map43_0	00_1002_0358	18/47	Source to IRQ map register.
interrupt_map44_0	00_1002_0360	18/47	Source to IRQ map register.
interrupt_map45_0	00_1002_0368	18/47	Source to IRQ map register.
interrupt_map46_0	00_1002_0370	18/47	Source to IRQ map register.
interrupt_map47_0	00_1002_0378	18/47	Source to IRQ map register.
interrupt_map48_0	00_1002_0380	18/47	Source to IRQ map register.
interrupt_map49_0	00_1002_0388	18/47	Source to IRQ map register.
interrupt_map50_0	00_1002_0390	18/47	Source to IRQ map register.
interrupt_map51_0	00_1002_0398	18/47	Source to IRQ map register.
interrupt_map52_0	00_1002_03a0	18/47	Source to IRQ map register.
interrupt_map53_0	00_1002_03a8	18/47	Source to IRQ map register.
interrupt_map54_0	00_1002_03b0	18/47	Source to IRQ map register.
interrupt_map55_0	00_1002_03b8	18/47	Source to IRQ map register.
interrupt_map56_0	00_1002_03c0	18/47	Source to IRQ map register.
interrupt_map57_0	00_1002_03c8	18/47	Source to IRQ map register.
interrupt_map58_0	00_1002_03d0	18/47	Source to IRQ map register.
interrupt_map59_0	00_1002_03d8	18/47	Source to IRQ map register.
interrupt_map60_0	00_1002_03e0	18/47	Source to IRQ map register.

**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
interrupt_map61_0	00_1002_03e8	18/47	Source to IRQ map register.
interrupt_map62_0	00_1002_03f0	18/47	Source to IRQ map register.
interrupt_map63_0	00_1002_03f8	18/47	Source to IRQ map register.
addr_trap_up_cfg_0	00_1002_0400	43/68	Top of address trap range.
addr_trap_up_cfg_1	00_1002_0408	43/68	Top of address trap range.
addr_trap_up_cfg_2	00_1002_0410	43/68	Top of address trap range.
addr_trap_up_cfg_3	00_1002_0418	43/68	Top of address trap range.
addr_trap_down_cfg_0	00_1002_0420	44/69	Bottom of address trap range.
addr_trap_down_cfg_1	00_1002_0428	44/69	Bottom of address trap range.
addr_trap_down_cfg_2	00_1002_0430	44/69	Bottom of address trap range.
addr_trap_down_cfg_3	00_1002_0438	44/69	Bottom of address trap range.
addr_trap_cfg_0	00_1002_0440	45/69	Address trap configuration.
addr_trap_cfg_1	00_1002_0448	45/69	Address trap configuration.
addr_trap_cfg_2	00_1002_0450	45/69	Address trap configuration.
addr_trap_cfg_3	00_1002_0458	45/69	Address trap configuration.
addr_trap_reg_debug	00_1002_0460	41/68	Address of interrupting trap (Read Only, No side effects).
perf_cnt_cfg	00_1002_04c0	31/61	Performance counter control.
perf_cnt_0	00_1002_04d0	32/62	Performance counter.
perf_cnt_1	00_1002_04d8	32/62	Performance counter.
perf_cnt_2	00_1002_04e0	32/62	Performance counter.
perf_cnt_3	00_1002_04e8	32/62	Performance counter.
bus_err_status	00_1002_0880	35/65	Bus error status (Read Only).
bus_err_data_0	00_1002_08a0	37/65	Data or address/control information latched on bus error (Read Only).
bus_err_data_1	00_1002_08a8	37/65	Data or address/control information latched on bus error (Read Only).
bus_err_data_2	00_1002_08b0	37/65	Data or address/control information latched on bus error (Read Only).
bus_err_data_3	00_1002_08b8	37/65	Data or address/control information latched on bus error (Read Only).
bus_l2_errors	00_1002_08c0	38/66	Count of L2 ECC errors (4 8-bit counters).
bus_mem_io_errors	00_1002_08c8	39/66	Count of memory ECC errors and I/O errors (3 8-bit counters).
bus_err_status_debug	00_1002_08D0	36/65	Bus error status (Read Only).
trace_cfg	00_1002_0a00	48/76	Trace configuration.
trace_read	00_1002_0a08	50/79	Trace buffer read register (Read Only).
trace_event_0	00_1002_0a20	46/72	Trace event selector.
trace_event_1	00_1002_0a28	46/72	Trace event selector.
trace_event_2	00_1002_0a30	46/72	Trace event selector.



**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
trace_event_3	00_1002_0a38	46/72	Trace event selector.
trace_sequence_0	00_1002_0a40	47/74	Trace sequence and action.
trace_sequence_1	00_1002_0a48	47/74	Trace sequence and action.
trace_sequence_2	00_1002_0a50	47/74	Trace sequence and action.
trace_sequence_3	00_1002_0a58	47/74	Trace sequence and action.
trace_event_4	00_1002_0a60	46/72	Trace event selector.
trace_event_5	00_1002_0a68	46/72	Trace event selector.
trace_event_6	00_1002_0a70	46/72	Trace event selector.
trace_event_7	00_1002_0a78	46/72	Trace event selector.
trace_sequence_4	00_1002_0a80	47/74	Trace sequence and action.
trace_sequence_5	00_1002_0a88	47/74	Trace sequence and action.
trace_sequence_6	00_1002_0a90	47/74	Trace sequence and action.
trace_sequence_7	00_1002_0a98	47/74	Trace sequence and action.
dm_dscr_base_0	00_1002_0b00	114/184	Data Mover channel 0 ring base address (Read clears some bits).
dm_dscr_count_0	00_1002_0b08	116/185	Data Mover channel 0 descriptor count.
dm_dscr_addr_0	00_1002_0b10	117/185	Data Mover channel 0 current address (Read Only).
dm_debug_dscr_base_0	00_1002_0b18	115/184	Debug alias for channel 0 ring base address (Read Only, no side effects).
dm_dscr_base_1	00_1002_0b20	114/184	Data Mover channel 1 ring base address (Read clears some bits).
dm_dscr_count_1	00_1002_0b28	116/185	Data Mover channel 1 descriptor count.
dm_dscr_addr_1	00_1002_0b30	117/185	Data Mover channel 1 current address. (Read Only)
dm_debug_dscr_base_1	00_1002_0b38	115/184	Debug alias for channel 1 ring base address (Read Only, no side effects).
dm_dscr_base_2	00_1002_0b40	114/184	Data Mover channel 2 ring base address (Read clears some bits).
dm_dscr_count_2	00_1002_0b48	116/185	Data Mover channel 2 descriptor count.
dm_dscr_addr_2	00_1002_0b50	117/185	Data Mover channel 2 current address (Read Only).
dm_debug_dscr_base_2	00_1002_0b58	115/184	Debug alias for channel 2 ring base address (Read Only, no side effects).
dm_dscr_base_3	00_1002_0b60	114/184	Data Mover channel 3 ring base address (Read clears some bits).
dm_dscr_count_3	00_1002_0b68	116/185	Data Mover channel 3 descriptor count.
dm_dscr_addr_3	00_1002_0b70	117/185	Data Mover channel 3 current address (Read Only).
dm_debug_dscr_base_3	00_1002_0b78	115/184	Debug alias for channel 3 ring base address (Read Only, no side effects).
crc_def_0	00_1002_0b80	118/185	CRC definition register.
ctcp_def_0	00_1002_0b88	119/186	CRC and checksum definition register.
crc_def_1	00_1002_0b90	118/185	CRC definition register.

**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
ctcp_def_1	00_1002_0b98	119/186	CRC and checksum definition register.
dm_partial_0	00_1002_0ba0	120/186	CRC/Checksum partial result register.
dm_partial_1	00_1002_0ba8	120/186	CRC/Checksum partial result register.
dm_partial_2	00_1002_0bb0	120/186	CRC/Checksum partial result register.
dm_partial_3	00_1002_0bb8	120/186	CRC/Checksum partial result register.
zbbus_cycle_cp0	00_1002_0c00	30/61	ZBbus cycle count compare register zero.
zbbus_cycle_cp1	00_1002_0c08	30/61	ZBbus cycle count compare register one.
system_scratch	00_1002_0c10	16/45	Scratch register for software use
alias_mailbox_cpu_0	00_1002_1xx0	17/46	Alias of status of mailbox (has incomplete decode, so access from pci won't hang). (Read Only).
alias_mailbox_set_cpu_0	00_1002_1xx8	17/46	Set bits in mailbox_cpu_0 by writing 1s to this location (Write Only, has incomplete decode, so access from pci won't hang).
int_mapper_1	00_1002_2000		2000-23f8 Registers for CPU 1 interrupt mapper (+2000 from CPU 0).
zbbus_cycle_cnt	00_1003_0000	29/61	ZBbus cycle count.
system_manuf	00_1003_8000	14/43	Read Only. Manufacturing Information Register.
l2_read_address	00_1004_0018	56/100	Read only. Last address/tag in a read (for testing)
l2_ecc_address	00_1004_0038	56/100	Read only. Last address with ecc error (correctable or not).
l2_misc_value	00_1004_0058	57/100	Read only. PERIPH_REV3 and later. Value of L2 hidden registers.
l2_way_disable	00_1004_1x00	/91	Accesses made to this range of addresses will write the value x to l2_wayen[3:0] register. If l2_wayen[i] is clear Way i is removed from the L2 replacement algorithm. See Section: "Using the L2 Cache as Memory" on page 91
l2_cache_disable	00_1004_2x00	/94	Accesses made to this range of addresses will write the value x to l2_cache_disable[3:0] register. See Section: "Reduced Cache Size" on page 94.
l2_misc_config	00_1004_3x00	/99	Accesses made to this range of addresses will write the value x to l2_misc_config[3:0] register. See Section: "Cache Configuration Register" on page 99.
mc_config_0	00_1005_1100	72/135	Channel 0 attributes.
mc_dramcmd_0	00_1005_1120	75/139	Channel 0 SDRAM command.
mc_drammode_0	00_1005_1140	76/139	Channel 0 SDRAM mode.
mc_timing1_0	00_1005_1160	77/140	Channel 0 SDRAM timing 1.
mc_timing2_0	00_1005_1180	78/141	Channel 0 SDRAM timing 2.
mc_cs_start_0	00_1005_11a0	79/141	Channel 0 CS[3:0] start address.
mc_cs_end_0	00_1005_11e0	80/141	Channel 0 CS[3:0] end+1 address.
mc_interleave_0	00_1005_11e0	81/142	Channel 0 interleaved CS position.
mc_cs0_row_0	00_1005_1200	82/142	Channel 0 CS0 row address bits.





Table 312: Internal Registers Ordered by Address (Cont.)

Name	Address	Table/ Page	Description
mc_cs0_col_0	00_1005_1220	83/142	Channel 0 CS0 column address bits.
mc_cs0_ba_0	00_1005_1240	84/143	Channel 0 CS0 bank select.
mc_cs1_row_0	00_1005_1260	82/142	Channel 0 CS1 row address bits.
mc_cs1_col_0	00_1005_1280	83/142	Channel 0 CS1 column address bits.
mc_cs1_ba_0	00_1005_12a0	84/143	Channel 0 CS1 DRAM bank select.
mc_cs2_row_0	00_1005_12c0	82/142	Channel 0 CS2 row address bits.
mc_cs2_col_0	00_1005_12e0	83/142	Channel 0 CS2 column address bits.
mc_cs2_ba_0	00_1005_1300	84/143	Channel 0 CS2 DRAM bank select.
mc_cs3_row_0	00_1005_1320	82/142	Channel 0 CS3 row address bits.
mc_cs3_col_0	00_1005_1340	83/142	Channel 0 CS3 column address bits.
mc_cs3_ba_0	00_1005_1360	83/142	Channel 0 CS3 DRAM bank select.
mc_cs_attr_0	00_1005_1380	85/143	Channel 0 CS[3:0]attribute.
mc_test_data_0	00_1005_1400	86/144	Channel 0 ECC error set data bits.
mc_test_ecc_0	00_1005_1420	87/144	Channel 0 ECC error set ecc bits.
mc_clock_cfg_0	00_1005_1500	74/138	Channel 0 memory clock ratio.
mc_1	00_1005_2000 ... 00_1005_2600	See _0 above	Memory controller channel 1 configuration (at memory controller 0 + 1000).
smb_serial_xtra_0	00_1006_0000	294/416	SMB port extra data register.
smb_serial_xtra_1	00_1006_0008	294/416	SMB port extra data register.
smb_serial_freq_0	00_1006_0010	289/415	Frequency for SMB port.
smb_serial_freq_1	00_1006_0018	289/415	Frequency for SMB port.
smb_serial_status_0	00_1006_0020	292/416	SMB port status (Read clears finish interrupt).
smb_serial_status_1	00_1006_0028	292/416	SMB port status (Read clears finish interrupt).
smb_serial_cmd_0	00_1006_0030	290/415	SMB port command.
smb_serial_cmd_1	00_1006_0038	290/415	SMB port command.
smb_serial_start_0	00_1006_0040	296/417	SMB port start register.
smb_serial_start_1	00_1006_0048	296/417	SMB port start register.
smb_serial_data_0	00_1006_0050	293/416	SMB port data register.
smb_serial_data_1	00_1006_0058	293/416	SMB port data register.
smb_serial_control_0	00_1006_0060	291/415	SMB port control register.
smb_serial_control_1	00_1006_0068	291/415	SMB port control register.
smb_serial_pec_0	00_1006_0070	295/417	SMB port pec register.
smb_serial_pec_1	00_1006_0078	295/417	SMB port pec register.
duart_mode_reg_1a	00_1006_0100	197/326	Mode register 1 port A MR1A.
duart_mode_reg_2a	00_1006_0110	198/326	Mode register 2 port A MR2A.
duart_status_a	00_1006_0120	200/327	Status register port A (Read Only).
duart_clk_sel_a	00_1006_0130	201/328	Clock select register port A.

**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
duart_full_ctl_a	00_1006_0140	202/328	Full control port A.
duart_cmd_a	00_1006_0150	199/327	Command register port A.
duart_rx_hold_reg_a	00_1006_0160	203/328	RX holding register port A (Read Only, read pops character from FIFO).
duart_tx_hold_reg_a	00_1006_0170	204/329	TX holding register port A (Write Only).
duart_opcr_a	00_1006_0180	211/331	Output port control register alias that only alters port A bits.
duart_aux_ctrl_a	00_1006_0190	213/331	Aux control register alias that only alters port A bits.
duart_mode_reg_1b	00_1006_0200	197/326	Mode register 1 port B.
duart_mode_reg_2b	00_1006_0210	198/326	Mode register 2 port B.
duart_status_b	00_1006_0220	200/327	Status register port B (Read Only).
duart_clk_sel_b	00_1006_0230	201/328	Clock select register port B.
duart_full_ctl_b	00_1006_0240	202/328	Full control port B.
duart_cmd_b	00_1006_0250	199/327	Command register port B.
duart_rx_hold_reg_b	00_1006_0260	203/328	RX holding register port B (Read Only, read pops character from FIFO).
duart_tx_hold_reg_b	00_1006_0270	204/329	TX holding register port B (Write Only).
duart_opcr_b	00_1006_0280	211/331	Output port control register alias that only alters port B bits.
duart_aux_ctrl_b	00_1006_0290	213/331	Aux control register alias that only alters port B bits.
duart_inport_chng	00_1006_0300	206/329	Input port change register (Read Only, read clears).
duart_aux_cntrl	00_1006_0310	212/331	Aux control register.
duart_isr_a	00_1006_0320	215/332	Interrupt status register port A (Read Only).
duart_imr_a	00_1006_0330	218/333	Interrupt mask register port A.
duart_isr_b	00_1006_0340	216/332	Interrupt status register port B (Read Only).
duart_imr_b	00_1006_0350	219/333	Interrupt mask register port B.
duart_out_port	00_1006_0360	222/334	Output port register (Write Only).
duart_opcr	00_1006_0370	210/330	Output port control.
duart_in_port	00_1006_0380	205/329	Input port state (Read Only).
duart_isr	00_1006_0390	214/332	Full interrupt status (Read Only).
duart_imr	00_1006_03a0	217/333	Full interrupt mask.
duart_set_opr	00_1006_03b0	220/334	Output port register bit set (Write Only).
duart_clear_opr	00_1006_03c0	221/334	Output port register bit clear (Write Only).
duart_inport_chng_a	00_1006_03d0	208/330	Input port change register for channel A (Read Only, read clears channel A change state)
duart_inport_chng_b	00_1006_03e0	209/330	Input port change register for channel B (Read Only, read clears channel B change state)
duart_inport_chng_debug	00_1006_03f0	207/330	Alias of input port change register with no read side effects. (Read Only)
dma_config0_ser_0_rx	00_1006_0400	91/163	Receive DMA control register.
dma_config1_ser_0_rx	00_1006_0408	92/164	Receive DMA control register.



**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
dma_dscr_base_ser_0_rx	00_1006_0410	93/166	Receive DMA descriptor base address.
dma_dscr_cnt_ser_0_rx	00_1006_0418	95/166	Receive DMA descriptor count .
dma_cur_dscr_a_ser_0_rx	00_1006_0420	96/167	Receive DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_0_rx	00_1006_0428	97/167	Receive DMA current descriptor B (Read Only).
dma_cur_daddr_ser_0_rx	00_1006_0430	98/167	Receive DMA current descriptor address (Read Only).
dma_config0_ser_0_tx	00_1006_0480	91/163	Transmit DMA control register.
dma_config1_ser_0_tx	00_1006_0488	92/164	Transmit DMA control register.
dma_dscr_base_ser_0_tx	00_1006_0490	93/166	Transmit DMA descriptor base address.
dma_dscr_cnt_ser_0_tx	00_1006_0498	95/166	Transmit DMA descriptor count.
dma_cur_dscr_a_ser_0_tx	00_1006_04A0	96/167	Transmit DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_0_tx	00_1006_04A8	97/167	Transmit DMA current descriptor B (Read Only).
dma_cur_daddr_ser_0_tx	00_1006_04B0	98/167	Transmit DMA current descriptor address (Read Only).
ser_mode_0	00_1006_0500	231/353	Mode select.
ser_minfrm_sz_0	00_1006_0508	237/355	Min frame size.
ser_maxfrm_sz_0	00_1006_0510	238/356	Max frame size.
ser_addr_mask_0	00_1006_0518	243/358	Address mask.
ser_usr0_addr_0	00_1006_0520	244/358	Match address 0.
ser_usr1_addr_0	00_1006_0528	244/358	Match address 1.
ser_usr2_addr_0	00_1006_0530	244/358	Match address 2.
ser_usr3_addr_0	00_1006_0538	244/358	Match address 3.
ser_cmd_0	00_1006_0540	233/354	Command
ser_tx_rd_thrsh_0	00_1006_0560	235/355	Transmit FIFO read threshold.
ser_tx_wr_thrsh_0	00_1006_0568	234/355	Transmit FIFO write threshold.
ser_rx_rd_thrsh_0	00_1006_0570	236/355	Receive FIFO read threshold.
ser_line_mode_0	00_1006_0578	232/353	Line Interface configuration register.
ser_dma_enable_0	00_1006_0580	239/356	DMA channel enable register.
ser_status_0	00_1006_0588	240/356	Serial interface and DMA status (Read Only, read clears).
ser_int_mask_0	00_1006_0590	242/357	Interrupt mask.
dma_asic_addr_ser_0	00_1006_0598	94/166	ASIC mode address.
ser_debug_status_0	00_1006_05A8	241/357	Serial interface and DMA status (Read Only, no side effects).
ser_tx_byte_lo_0	00_1006_05C0	246/359	Serial interface transmit byte count (low 16 bits).
ser_tx_byte_hi_0	00_1006_05C8	246/359	Serial interface transmit byte count (high 16 bits).
ser_rx_byte_lo_0	00_1006_05D0	246/359	Serial interface receive byte count (low 16 bits).
ser_rx_byte_hi_0	00_1006_05D8	246/359	Serial interface receive byte count (high 16 bits).
ser_tx_underrun_0	00_1006_05E0	246/359	Serial interface transmit underrun count.
ser_rx_overflow_0	00_1006_05E8	246/359	Serial interface receive overflow count.

**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
ser_rx_errors_0	00_1006_05F0	246/359	Serial interface receive error packet count.
ser_rx_badaddr_0	00_1006_05F8	246/359	Serial interface receive address mismatch count.
ser_rx_table0_0	00_1006_0600	245/358	Sequence table.
ser_rx_table1_0	00_1006_0608	245/358	Sequence table.
ser_rx_table2_0	00_1006_0610	245/358	Sequence table.
ser_rx_table3_0	00_1006_0618	245/358	Sequence table.
ser_rx_table4_0	00_1006_0620	245/358	Sequence table.
ser_rx_table5_0	00_1006_0628	245/358	Sequence table.
ser_rx_table6_0	00_1006_0630	245/358	Sequence table.
ser_rx_table7_0	00_1006_0638	245/358	Sequence table.
ser_rx_table8_0	00_1006_0640	245/358	Sequence table.
ser_rx_table9_0	00_1006_0648	245/358	Sequence table.
ser_rx_table10_0	00_1006_0650	245/358	Sequence table.
ser_rx_table11_0	00_1006_0658	245/358	Sequence table.
ser_rx_table12_0	00_1006_0660	245/358	Sequence table.
ser_rx_table13_0	00_1006_0668	245/358	Sequence table.
ser_rx_table14_0	00_1006_0670	245/358	Sequence table.
ser_rx_table15_0	00_1006_0678	245/358	Sequence table.
ser_tx_table0_0	00_1006_0700	245/358	Sequence table.
ser_tx_table1_0	00_1006_0708	245/358	Sequence table.
ser_tx_table2_0	00_1006_0710	245/358	Sequence table.
ser_tx_table3_0	00_1006_0718	245/358	Sequence table.
ser_tx_table4_0	00_1006_0720	245/358	Sequence table.
ser_tx_table5_0	00_1006_0728	245/358	Sequence table.
ser_tx_table6_0	00_1006_0730	245/358	Sequence table.
ser_tx_table7_0	00_1006_0738	245/358	Sequence table.
ser_tx_table8_0	00_1006_0740	245/358	Sequence table.
ser_tx_table9_0	00_1006_0748	245/358	Sequence table.
ser_tx_table10_0	00_1006_0750	245/358	Sequence table.
ser_tx_table11_0	00_1006_0758	245/358	Sequence table.
ser_tx_table12_0	00_1006_0760	245/358	Sequence table.
ser_tx_table13_0	00_1006_0768	245/358	Sequence table.
ser_tx_table14_0	00_1006_0770	245/358	Sequence table.
ser_tx_table15_0	00_1006_0778	245/358	Sequence table.
dma_config0_ser_1_rx	00_1006_0800	91/163	Receive DMA control register.
dma_config1_ser_1_rx	00_1006_0808	92/164	Receive DMA control register.
dma_dscr_base_ser_1_rx	00_1006_0810	93/166	Receive DMA descriptor base address.

Table 312: Internal Registers Ordered by Address (Cont.)

Name	Address	Table/ Page	Description
dma_dscr_count_ser_1_rx	00_1006_0818	95/166	Receive DMA descriptor count.
dma_cur_dscr_a_ser_1_rx	00_1006_0820	96/167	Receive DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_1_rx	00_1006_0828	97/167	Receive DMA current descriptor B (Read Only).
dma_cur_daddr_ser_1_rx	00_1006_0830	98/167	Receive DMA current descriptor address (Read Only).
dma_config0_ser_1_tx	00_1006_0880	91/163	Transmit DMA control register.
dma_config1_ser_1_tx	00_1006_0888	92/164	Transmit DMA control register.
dma_dscr_base_ser_1_tx	00_1006_0890	93/166	Transmit DMA descriptor base address.
dma_dscr_count_ser_1_tx	00_1006_0898	95/166	Transmit DMA descriptor count.
dma_cur_dscr_a_ser_1_tx	00_1006_08A0	96/167	Transmit DMA current descriptor A (Read Only).
dma_cur_dscr_b_ser_1_tx	00_1006_08A8	97/167	Transmit DMA current descriptor B (Read Only).
dma_cur_daddr_ser_1_tx	00_1006_08B0	98/167	Transmit DMA current descriptor address (Read Only).
ser_mode_1	00_1006_0900	231/353	Mode select.
ser_minfrm_sz_1	00_1006_0908	237/355	Min frame size.
ser_maxfrm_sz_1	00_1006_0910	238/356	Max frame size.
ser_addr_mask_1	00_1006_0918	243/358	Address mask.
ser_usr0_addr_1	00_1006_0920	244/358	Match address 0.
ser_usr1_addr_1	00_1006_0928	244/358	Match address 1.
ser_usr2_addr_1	00_1006_0930	244/358	Match address 2.
ser_usr3_addr_1	00_1006_0938	244/358	Match address 3.
ser_cmd_1	00_1006_0940	233/354	Command
ser_tx_rd_thrsh_1	00_1006_0960	235/355	Transmit FIFO read threshold.
ser_tx_wr_thrsh_1	00_1006_0968	234/355	Transmit FIFO write threshold.
ser_rx_rd_thrsh_1	00_1006_0970	236/355	Receive FIFO read threshold.
ser_line_mode_1	00_1006_0978	232/353	Line Interface configuration register.
ser_dma_enable_1	00_1006_0980	239/356	DMA channel enable register.
ser_status_1	00_1006_0988	240/356	Serial interface and DMA status (Read Only, read clears).
ser_int_mask_1	00_1006_0990	242/357	Interrupt mask.
dma_asic_addr_ser_1	00_1006_0998	94/166	ASIC mode address.
ser_debug_status_1	00_1006_09A8	241/357	Serial interface and DMA status (Read Only, no side effects).
ser_tx_byte_lo_1	00_1006_09C0	246/359	Serial interface transmit byte count (low 16 bits).
ser_tx_byte_hi_1	00_1006_09C8	246/359	Serial interface transmit byte count (high 16 bits).
ser_rx_byte_lo_1	00_1006_09D0	246/359	Serial interface receive byte count (low 16 bits).
ser_rx_byte_hi_1	00_1006_09D8	246/359	Serial interface receive byte count (high 16 bits).
ser_tx_underrun_1	00_1006_09E0	246/359	Serial interface transmit underrun count.
ser_rx_overflow_1	00_1006_09E8	246/359	Serial interface receive overflow count.
ser_rx_errors_1	00_1006_09F0	246/359	Serial interface receive error packet count.

Table 312: Internal Registers Ordered by Address (Cont.)

Name	Address	Table/ Page	Description
ser_rx_badaddr_1	00_1006_09F8	246/359	Serial interface receive address mismatch count.
ser_rx_table0_1	00_1006_0A00	245/358	Sequence table.
ser_rx_table1_1	00_1006_0A08	245/358	Sequence table.
ser_rx_table2_1	00_1006_0A10	245/358	Sequence table.
ser_rx_table3_1	00_1006_0A18	245/358	Sequence table.
ser_rx_table4_1	00_1006_0A20	245/358	Sequence table.
ser_rx_table5_1	00_1006_0A28	245/358	Sequence table.
ser_rx_table6_1	00_1006_0A30	245/358	Sequence table.
ser_rx_table7_1	00_1006_0A38	245/358	Sequence table.
ser_rx_table8_1	00_1006_0A40	245/358	Sequence table.
ser_rx_table9_1	00_1006_0A48	245/358	Sequence table.
ser_rx_table10_1	00_1006_0A50	245/358	Sequence table.
ser_rx_table11_1	00_1006_0A58	245/358	Sequence table.
ser_rx_table12_1	00_1006_0A60	245/358	Sequence table.
ser_rx_table13_1	00_1006_0A68	245/358	Sequence table.
ser_rx_table14_1	00_1006_0A70	245/358	Sequence table.
ser_rx_table15_1	00_1006_0A78	245/358	Sequence table.
ser_tx_table0_1	00_1006_0B00	245/358	Sequence table.
ser_tx_table1_1	00_1006_0B08	245/358	Sequence table.
ser_tx_table2_1	00_1006_0B10	245/358	Sequence table.
ser_tx_table3_1	00_1006_0B18	245/358	Sequence table.
ser_tx_table4_1	00_1006_0B20	245/358	Sequence table.
ser_tx_table5_1	00_1006_0B28	245/358	Sequence table.
ser_tx_table6_1	00_1006_0B30	245/358	Sequence table.
ser_tx_table7_1	00_1006_0B38	245/358	Sequence table.
ser_tx_table8_1	00_1006_0B40	245/358	Sequence table.
ser_tx_table9_1	00_1006_0B48	245/358	Sequence table.
ser_tx_table10_1	00_1006_0B50	245/358	Sequence table.
ser_tx_table11_1	00_1006_0B58	245/358	Sequence table.
ser_tx_table12_1	00_1006_0B60	245/358	Sequence table.
ser_tx_table13_1	00_1006_0B68	245/358	Sequence table.
ser_tx_table14_1	00_1006_0B70	245/358	Sequence table.
ser_tx_table15_1	00_1006_0B78	245/358	Sequence table.
io_ext_cfg_0	00_1006_1000	251/374	cs0 interface configuration.
io_ext_cfg_1	00_1006_1008	251/374	cs1 interface configuration.
io_ext_cfg_2	00_1006_1010	251/374	cs2 interface configuration.
io_ext_cfg_3	00_1006_1018	251/374	cs3 interface configuration.

**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
io_ext_cfg_4	00_1006_1020	<a href="#">251/374</a>	cs4 interface configuration.
io_ext_cfg_5	00_1006_1028	<a href="#">251/374</a>	cs5 interface configuration.
io_ext_cfg_6	00_1006_1030	<a href="#">251/374</a>	cs6 interface configuration.
io_ext_cfg_7	00_1006_1038	<a href="#">251/374</a>	cs7 interface configuration.
io_ext_mult_size_0	00_1006_1100	<a href="#">252/374</a>	cs0 region size in 64KB units.
io_ext_mult_size_1	00_1006_1108	<a href="#">252/374</a>	cs1 region size in 64KB units.
io_ext_mult_size_2	00_1006_1110	<a href="#">252/374</a>	cs2 region size in 64KB units.
io_ext_mult_size_3	00_1006_1118	<a href="#">252/374</a>	cs3 region size in 64KB units.
io_ext_mult_size_4	00_1006_1120	<a href="#">252/374</a>	cs4 region size in 64KB units.
io_ext_mult_size_5	00_1006_1128	<a href="#">252/374</a>	cs5 region size in 64KB units.
io_ext_mult_size_6	00_1006_1130	<a href="#">252/374</a>	cs6 region size in 64KB units.
io_ext_mult_size_7	00_1006_1138	<a href="#">252/374</a>	cs7 region size in 64KB units.
io_ext_start_addr_0	00_1006_1200	<a href="#">253/375</a>	cs0 region start address (put outside generic bus range to disable).
io_ext_start_addr_1	00_1006_1208	<a href="#">253/375</a>	cs1 region start address (put outside generic bus range to disable).
io_ext_start_addr_2	00_1006_1210	<a href="#">253/375</a>	cs2 region start address (put outside generic bus range to disable).
io_ext_start_addr_3	00_1006_1218	<a href="#">253/375</a>	cs3 region start address (put outside generic bus range to disable).
io_ext_start_addr_4	00_1006_1220	<a href="#">253/375</a>	cs4 region start address (put outside generic bus range to disable).
io_ext_start_addr_5	00_1006_1228	<a href="#">253/375</a>	cs5 region start address (put outside generic bus range to disable).
io_ext_start_addr_6	00_1006_1230	<a href="#">253/375</a>	cs6 region start address (put outside generic bus range to disable).
io_ext_start_addr_7	00_1006_1238	<a href="#">253/375</a>	cs7 region start address (put outside generic bus range to disable).
io_drive_0	00_1006_1300	<a href="#">264/378</a>	3.3V output drive strength and slew rate control registers. See <a href="#">Section: "Drive Strength Control" on page 373</a> .
io_drive_1	00_1006_1308	<a href="#">265/379</a>	3.3V output drive strength and slew rate control register.
io_drive_2	00_1006_1310	<a href="#">266/380</a>	3.3V output drive strength and slew rate control register.
io_drive_3	00_1006_1318	<a href="#">267/380</a>	3.3V output drive strength and slew rate control register.
io_ext_time_cfg0_0	00_1006_1600	<a href="#">254/375</a>	cs0 Timing parameter configuration 0.
io_ext_time_cfg0_1	00_1006_1608	<a href="#">254/375</a>	cs1 Timing parameter configuration 0.
io_ext_time_cfg0_2	00_1006_1610	<a href="#">254/375</a>	cs2 Timing parameter configuration 0.
io_ext_time_cfg0_3	00_1006_1618	<a href="#">254/375</a>	cs3 Timing parameter configuration 0.
io_ext_time_cfg0_4	00_1006_1620	<a href="#">254/375</a>	cs4 Timing parameter configuration 0.
io_ext_time_cfg0_5	00_1006_1628	<a href="#">254/375</a>	cs5 Timing parameter configuration 0.
io_ext_time_cfg0_6	00_1006_1630	<a href="#">254/375</a>	cs6 Timing parameter configuration 0.



Table 312: Internal Registers Ordered by Address (Cont.)

Name	Address	Table/ Page	Description
io_ext_time_cfg0_7	00_1006_1638	<a href="#">254/375</a>	cs7 Timing parameter configuration 0.
io_ext_time_cfg1_0	00_1006_1700	<a href="#">255/376</a>	cs0 Timing parameter configuration 1.
io_ext_time_cfg1_1	00_1006_1708	<a href="#">255/376</a>	cs1 Timing parameter configuration 1.
io_ext_time_cfg1_2	00_1006_1710	<a href="#">255/376</a>	cs2 Timing parameter configuration 1.
io_ext_time_cfg1_3	00_1006_1718	<a href="#">255/376</a>	cs3 Timing parameter configuration 1.
io_ext_time_cfg1_4	00_1006_1720	<a href="#">255/376</a>	cs4 Timing parameter configuration 1.
io_ext_time_cfg1_5	00_1006_1728	<a href="#">255/376</a>	cs5 Timing parameter configuration 1.
io_ext_time_cfg1_6	00_1006_1730	<a href="#">255/376</a>	cs6 Timing parameter configuration 1.
io_ext_time_cfg1_7	00_1006_1738	<a href="#">255/376</a>	cs7 Timing parameter configuration 1.
io_interrupt_status	00_1006_1a00	<a href="#">256/376</a>	Interrupt status for generic bus sources (Read Only, read clears).
io_interrupt_data0	00_1006_1a10	<a href="#">257/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_data1	00_1006_1a18	<a href="#">258/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_data2	00_1006_1a20	<a href="#">259/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_data3	00_1006_1a28	<a href="#">260/377</a>	Data latched on generic interrupt assertion (Read Only).
io_interrupt_addr0	00_1006_1a30	<a href="#">261/378</a>	Address latched on generic interrupt assertion (Read Only).
io_interrupt_addr1	00_1006_1a40	<a href="#">262/378</a>	Address latched on generic interrupt assertion (Read Only).
io_interrupt_parity	00_1006_1a50	<a href="#">263/378</a>	Parity latched on generic interrupt assertion (Read Only).
pcmcia_cfg	00_1006_1a60	<a href="#">273/393</a>	PCICMA controller configuration.
pcmcia_status	00_1006_1a70	<a href="#">274/394</a>	PCMCIA controller status (Read Only, read clears interrupt).
gpio_clr_edge	00_1006_1a80	<a href="#">276/398</a>	Write a 1 to clear the edge detector for that bit (Write Only).
gpio_int_type	00_1006_1a88	<a href="#">277/398</a>	Select interrupt type for pairs of pins.
gpio_input_invert	00_1006_1a90	<a href="#">279/398</a>	Set to invert the input.
gpio_glitch	00_1006_1a98	<a href="#">280/399</a>	Set for 1ms glitch filter, clear for 60 ns.
gpio_read	00_1006_1aa0	<a href="#">278/398</a>	Shows current value of signal after possible invert and glitch (Read Only).
gpio_direction	00_1006_1aa8	<a href="#">281/399</a>	GPIO pin data direction. 1 for output, 0 for input.
gpio_pin_clr	00_1006_1ab0	<a href="#">282/399</a>	Write 1 to set that bit to zero (Write Only).
gpio_pin_set	00_1006_1ab8	<a href="#">283/399</a>	Write 1 to set that bit to one (Write Only).
mac_tx_byte_0	00_1006_4000	<a href="#">167/288</a>	RMON transmit byte counter.
mac_collisions_0	00_1006_4008	<a href="#">167/288</a>	RMON total collisions.
mac_late_col_0	00_1006_4010	<a href="#">167/288</a>	RMON late collisions.
mac_ex_col_0	00_1006_4018	<a href="#">167/288</a>	RMON excessive collisions.
mac_fcs_error_0	00_1006_4020	<a href="#">167/288</a>	RMON packets Tx with bad FCS.
mac_tx_abort_0	00_1006_4028	<a href="#">167/288</a>	RMON transmit packets aborted.
mac_tx_bad_0	00_1006_4038	<a href="#">167/288</a>	RMON total of bad Tx packets.



Table 312: Internal Registers Ordered by Address (Cont.)

Name	Address	Table/ Page	Description
mac_tx_good_0	00_1006_4040	167/288	RMON total successfully transmitted packets.
mac_tx_runt_0	00_1006_4048	167/288	RMON total runt packets transmitted.
mac_tx_oversize_0	00_1006_4050	167/288	RMON total oversize packets transmitted.
mac_rx_bytes_0	00_1006_4080	167/288	RMON total received bytes.
mac_rx_mcast_0	00_1006_4088	167/288	RMON total received multicast packets.
mac_rx_bcast_0	00_1006_4090	167/288	RMON total received broadcast packets.
mac_rx_bad_0	00_1006_4098	167/288	RMON total received bad packets.
mac_rx_good_0	00_1006_40A0	167/288	RMON total received good packets.
mac_rx_runt_0	00_1006_40A8	167/288	RMON total runt packets received.
mac_rx_oversize_0	00_1006_40B0	167/288	RMON total oversize packets received.
mac_rx_fcs_error_0	00_1006_40B8	167/288	RMON total packets received with bad FCS.
mac_rx_length_error_0	00_1006_40C0	167/288	RMON total packets received with length error.
mac_rx_code_error_0	00_1006_40C8	167/288	RMON total packets received with code error.
mac_rx_align_error_0	00_1006_40D0	167/288	RMON total packets received with align error.
mac_cfg_0	00_1006_4100	176/301	Ethernet interface configuration register.
mac_thrsh_cfg_0	00_1006_4108	179/306	Ethernet interface FIFO threshold configuration register.
mac_vlantag_0	00_1006_4110	181/309	VLAN tag for insertion into packets on transmit
mac_frame_cfg_0	00_1006_4118	180/307	Ethernet interface MAC frame configuration.
mac_rx_fifo_ptrs_0	00_1006_4120	186/313	MAC FIFO pointers (Debug, Read Only).
mac_tx_fifo_ptrs_0	00_1006_4128	186/313	MAC FIFO pointers. (Debug, Read Only.)
mac_adfilter_cfg_0	00_1006_4200	192/315	MAC receive address filter configuration register.
mac_ethernet_addr_0	00_1006_4208	190/314	MAC source ethernet address for insertion during transmission.
mac_type_cfg_0	00_1006_4210	191/315	MAC packet type table for receive packet comparisons.
mac_admask0_0	00_1006_4218	188/314	Address filter exact match mask register (PERIPH_REV3).
mac_admask1_0	00_1006_4220	188/314	Address filter exact match mask register (PERIPH_REV3).
mac_hash0_0	00_1006_4240	189/314	Address filter hash map.
mac_hash1_0	00_1006_4248	189/314	Address filter hash map.
mac_hash2_0	00_1006_4250	189/314	Address filter hash map.
mac_hash3_0	00_1006_4258	189/314	Address filter hash map.
mac_hash4_0	00_1006_4260	189/314	Address filter hash map.
mac_hash5_0	00_1006_4268	189/314	Address filter hash map.
mac_hash6_0	00_1006_4270	189/314	Address filter hash map.
mac_hash7_0	00_1006_4278	189/314	Address filter hash map.
mac_addr0_0	00_1006_4280	187/313	Address filter exact match.
mac_addr1_0	00_1006_4288	187/313	Address filter exact match.

**Table 312: Internal Registers Ordered by Address (Cont.)**

Name	Address	Table/ Page	Description
mac_addr2_0	00_1006_4290	187/313	Address filter exact match.
mac_addr3_0	00_1006_4298	187/313	Address filter exact match.
mac_addr4_0	00_1006_42A0	187/313	Address filter exact match.
mac_addr5_0	00_1006_42A8	187/313	Address filter exact match.
mac_addr6_0	00_1006_42B0	187/313	Address filter exact match.
mac_addr7_0	00_1006_42B8	187/313	Address filter exact match.
mac_chup0_0	00_1006_4300	193/317	Receive DMA channel select MSB.
mac_chup1_0	00_1006_4308	193/317	Receive DMA channel select MSB.
mac_chup2_0	00_1006_4310	193/317	Receive DMA channel select MSB.
mac_chup3_0	00_1006_4318	193/317	Receive DMA channel select MSB.
mac_chlo0_0	00_1006_4320	193/317	Receive DMA channel select LSB.
mac_chlo1_0	00_1006_4328	193/317	Receive DMA channel select LSB.
mac_chlo2_0	00_1006_4330	193/317	Receive DMA channel select LSB.
mac_chlo3_0	00_1006_4338	193/317	Receive DMA channel select LSB.
mac_enable_0	00_1006_4400	177/305	MAC enable register.
mac_status_0	00_1006_4408	182/309	MAC status/error register (Read Only, read clears).
mac_int_mask_0	00_1006_4410	185/313	MAC interrupt mask register.
dma_asic_addr_mac_0	00_1006_4418	94/166	ASIC mode base address.
mac_txd_ctl_0	00_1006_4420	178/305	Transmit DMA control register.
mac_mdio_0	00_1006_4428	194/317	MDIO pin control register.
mac_status1_0	00_1006_4430	183/312	MAC status/error register (Read Only, read clears ch 1).
mac_debug_status_0	00_1006_4448	184/312	MAC status/error register (Debug, Read Only, no side effects).
dma_config0_mac_0_rx_ch_0	00_1006_4800	91/163	DMA config 0 register.
dma_config1_mac_0_rx_ch_0	00_1006_4808	92/164	DMA config 1 register.
dma_dscr_base_mac_0_rx_ch_0	00_1006_4810	93/166	DMA descriptor base register.
dma_dscr_cnt_0_rx_ch_0	00_1006_4818	95/166	DMA descriptor count.
dma_dscr_a_mac_0_rx_ch_0	00_1006_4820	96/167	DMA current descriptor A.
dma_dscr_b_mac_0_rx_ch_0	00_1006_4828	97/167	DMA current descriptor B.
dma_cur_dscr_addr_mac_0_rx_ch_0	00_1006_4830	98/167	DMA current descriptor address.
dma_oodpktlost_mac_0_rx_ch_0	00_1006_4838	99/168	DMA packet lost counter. (PERIPH_REV3)
dma__mac_0_rx_ch_1	00_1006_4900		Rx channel 1 at Rx channel 0 + 100.
dma__mac_0_tx_ch_0	00_1006_4C00		Tx channel 0 at Rx channel 0 + 400.
dma__mac_0_tx_ch_1	00_1006_4D00		Tx channel 1 at Rx channel 0 + 500.
mac_1	00_1006_5000		MAC 1 registers (+1000 from mac_0 registers).
mac_2	00_1006_6000		MAC 2 registers (+2000 from mac_0 registers).



*Table 312: Internal Registers Ordered by Address (Cont.)*

Name	Address	Table/ Page	Description
Type 00 PCI header	00_FE00_0000 - 00_FE00_00FF	127/236	Configuration space bus=0,dev=0.
Type 01 PCI header	00_FE00_0800 - 00_FE00_08FF	140/244	Configuration space bus=0,dev=1.



# INDEX

## Numerics

16-Bit GMII Style Packet FIFO [299](#)

### A

A Comment on the term Bank [103](#)  
 A\_CMD [22](#), [78](#)  
 A\_L1CA [20](#), [23](#), [69](#), [78](#), [439](#)  
 A\_L2CA [23](#), [26](#), [77](#), [90](#), [93](#), [164](#), [206](#), [212](#), [224](#), [230](#),  
[242](#), [439](#)  
 Accesses from the BCM1250 to the PCI or HyperTransport [214](#)  
 Accesses from the HyperTransport to the BCM1250 [216](#)  
 Accesses from the PCI to the BCM1250 [217](#)  
 Accessing the BCM1250 from an BCM1250 on a Double Hosted  
 Chain [212](#)  
 Accessing the BCM1250 from HyperTransport Devices [210](#)  
 Accessing the BCM1250 from PCI Devices [205](#)  
 Acknowledgement Read Access [369](#)  
 Acknowledgement Write Access [370](#)  
 addr\_trap\_cfg [67](#), [69](#)  
 addr\_trap\_down [67](#), [69](#)  
 addr\_trap\_index [67](#), [68](#)  
 addr\_trap\_reg [53](#), [67](#), [68](#)  
 addr\_trap\_reg\_debug [68](#)  
 addr\_trap\_up [67](#), [68](#)  
 Address Trapping [67](#)  
 alias\_mailbox\_cpu [46](#)  
 alias\_mbox\_set\_cpu [46](#)  
 ASIC mode [159](#)  
 Asynchronous Mode [322](#)  
 Audience [3](#)

### B

Bank [103](#)  
 Bank Address [117](#)  
 baud rate [322](#)  
 Baud Rate Generators [322](#)  
 BCM1125 [1](#), [3](#), [4](#), [120](#), [136](#), [190](#), [194](#), [196](#), [264](#), [292](#),  
[425](#), [430](#)  
 BCM1250 Block Diagram [2](#), [3](#)  
 Big Endian System  
     Match Bit Lanes [203](#)  
     Match Byte Lanes [202](#)  
 Block [5](#)  
 Broadcom Use Only [6](#)  
 Bus Error [53](#), [184](#), [211](#), [249](#)  
 Bus error [77](#), [136](#), [137](#)  
 bus error [18](#), [18](#), [25](#), [53](#), [64](#), [66](#), [111](#), [112](#), [215](#), [235](#),  
[240](#), [241](#), [364](#), [369](#), [374](#)  
 Bus Error Exceptions [18](#)  
 Bus Width [365](#)  
 bus\_err\_data [65](#)  
 bus\_err\_status [17](#), [18](#), [53](#), [65](#)  
 bus\_err\_status\_debug [65](#)  
 bus\_io\_error [364](#)  
 bus\_l2\_errors [17](#), [66](#)  
 bus\_mem\_io\_errors [17](#), [18](#), [66](#)  
 BusErr-DPA [18](#)

### C

Cache Block [5](#)

cache error [17](#)  
 Cache Error Exceptions [17](#)  
 Cache Line [5](#)  
 Cache Management Access [94](#)  
 cacheability attribute [68](#), [69](#), [89](#)  
 CacheErr-D [17](#)  
 CacheErr-DPA [17](#)  
 CacheErr-I [17](#)  
 CAS time check policy [122](#)  
 Cause [47](#)  
 CF+ cards [390](#)  
 Channel Select [109](#)  
 checksum [178](#)  
 Checksum Generation [178](#)  
 Chip Select [110](#)  
 Choosing Interleave Parameters [120](#)  
 Clock Ratios and Clocking Scheme [107](#)  
 Clock, Reset and Test [8](#)  
 Closed Page policy [122](#)  
 COLDRES\_L [26](#), [41](#), [260](#)  
 Column Address [117](#)  
 Comments on Using the L2 as Memory [93](#)  
 CompactFlash [390](#)  
 Configuration Header Descriptions [236](#)  
 Configuration Resistors [27](#), [41](#)  
 Configuration Space [194](#)  
 Connecting A PCMCIA Slot [384](#)  
 CPU Speculative Execution [16](#)  
 CPU to CPU Communication [17](#), [19](#)  
 CRC [178](#), [180](#), [269](#)  
 CRC Generation [179](#)  
 crc\_def [179](#), [185](#)  
 CRC32 (Ethernet) [180](#)  
 CRC32C (iSCSI) [180](#)  
 CRC-CCITT (HDLC) [180](#)  
 ctc\_def [186](#)  
 CTS [322](#), [323](#), [324](#)

### D

D\_CODE [24](#), [65](#), [77](#)  
 D\_ID [77](#)  
 D\_MOD [25](#), [77](#)  
 D\_RSP [77](#)  
 data buffers [147](#)  
 Data Mover Operation [176](#)  
 Data Phase [24](#)  
 DDR FCRAM [123](#)  
 DDR SDRAMs [123](#)  
 DEBUG\_L [71](#), [72](#), [74](#), [76](#)  
 descriptors [147](#)  
 Destination Address Filtering [278](#)  
 Device Mode [190](#)  
 DINT [47](#), [48](#), [50](#)  
 Direct Connection of a Memory Only Card [385](#)  
 DLL [128](#), [138](#)  
 dm\_cur\_dscr\_addr [185](#)  
 dm\_debug\_dscr\_base [184](#)  
 dm\_dscr\_base [177](#), [184](#)  
 dm\_dscr\_base\_0 [54](#)

dm\_dscr\_base\_1 54  
 dm\_dscr\_base\_2 54  
 dm\_dscr\_base\_3 54  
 dm\_dscr\_count 185  
 dm\_partial 186  
 DMA  
   ASIC mode 159  
   buffer 148  
   buffers 147  
   bytes before offset overwritten 155  
   Coherence 155  
   completion interrupts 158  
   descriptor 149  
   descriptor chain 153  
   descriptor interrupts 158  
   descriptor ring 151  
   descriptors 147, 151, 154, 156  
   disable 152  
   Ethernet priority 156  
   L2 allocate 155  
   packet 149  
   pause 152  
   scatter/gather 155  
   watermarks 157  
 DMA Coherence and Cache Options 155  
 DMA Configuration 351  
 DMA Configurations 156  
 DMA Controllers 147  
 dma\_asic\_addr 159, 160, 166  
 dma\_config0 148, 156, 157, 163  
 dma\_config1 152, 156, 159, 164, 285  
 dma\_cur\_daddr 167  
 dma\_cur\_dscr\_a 167  
 dma\_cur\_dscr\_addr 153  
 dma\_cur\_dscr\_b 167  
 dma\_dscr\_base 166  
 dma\_dscr\_cnt 166  
 dma\_oodpktlost 158, 168, 283  
 DRAM 103  
 dscr\_a 149, 149, 154, 160, 169, 170  
 dscr\_b 149, 149, 154, 169, 170  
 DUART 322  
 DUART operation 323  
 duart\_aux\_ctrl 332  
 duart\_clear\_opr 324, 335  
 duart\_clk\_sel 323, 329  
 duart\_cmd 323, 328  
 duart\_full\_ctl 326, 329  
 duart\_imr 325, 334  
 duart\_imr\_a 325  
 duart\_imr\_b 325  
 duart\_in\_port 324, 330  
 duart\_inport\_chng 324, 330  
 duart\_inport\_chng\_a 331  
 duart\_inport\_chng\_b 331  
 duart\_inport\_chng\_debug 331  
 duart\_isr 325, 333, 334  
 duart\_isr\_a 325  
 duart\_isr\_b 325  
 duart\_mode\_reg\_1 323, 324, 326, 327  
 duart\_mode\_reg\_2 323, 324, 327  
 duart\_opcr 322, 324, 331, 340, 342

duart\_out\_port 324, 335  
 duart\_rx\_hold 326, 329  
 duart\_set\_opr 324, 335  
 duart\_status 323, 328  
 duart\_tx\_hold 323, 330

## E

ECC 125  
 ECC Diagnostic Management Accesses (ECC diag bits nonzero) 98  
 ECC error 64, 66, 89, 91, 94, 96, 97, 104, 125  
 Endian policy  
   as an optimization 204  
   Little endian system 201  
   match bit lanes 203  
   match byte lanes 202  
 EPC 17  
 ErrCtl 17, 18  
 Ethernet 180  
   alignment error 276, 291  
   Code Error 276  
   Code error 291  
   Collision 273  
   collisions 289, 302  
   CRC 269  
   CRC Error 276  
   CRC error 274, 291  
   Dribble Error 276  
   dribble error 291  
   Excessive Collision 273  
   excessive collisions 289, 302  
   filter 278  
   flow control 284  
   frame format 268  
   inter-frame gap 308  
   interrupts 286  
   Late Collision 273  
   late collisions 289, 302  
   Length Error 276, 291  
   length error 303  
   MDC 287  
   MDIO 287  
   Overflow 273, 277  
   oversize 290  
   Oversize Packet 276  
   oversize packets 291, 303  
   packet type 282  
   Pause Frame 284  
   PHY 271  
   prepended header 270  
   receive FIFO 266, 277  
   receive fifo 275  
   RMON statistics 289  
   runt 303  
   Runt Packet 276  
   runt packets 290, 291  
   transmit FIFO 266, 272  
   Underflow 273, 277  
   underflow 302  
   underrun 290  
 Ethernet and Serial DMA Engines 157  
 Ethernet MAC 264  
 Ethernet MACs 264

10/21/02

Example Startup Code to clear the L2 Cache 99

Explicit Descriptor Interrupts 158

**F**

Feature Control 219

FIFO Configuration 351

Fixed Cycle Write Access 368

Flow Control 284

Flow control 294

flow control 152, 271

watermark 157

**G**

General Timers 58

general\_timer\_cfg 60

general\_timer\_cnt 60

general\_timer\_init\_cnt 60

Generic Bus

address range 363

address space 362

burst mode 371

chip selects 362

data width 362, 365

error conditions 374

parity 364

Timing 365

Generic Bus Errors 374

Generic Bus Registers 375

GMII 264, 267

GPIO 397

GPIO Registers 399

gpio\_clr\_edge 54, 55, 398, 399

gpio\_direction 397, 400

gpio\_glitch 400

gpio\_input\_invert 397, 398, 399

gpio\_int\_type 54, 55, 398, 399

gpio\_pin\_clr 397, 400

gpio\_pin\_set 397, 400

gpio\_read 397, 399

**H**

HDLC 180, 337, 344, 345, 347

Hint Based policy 122

Host Mode 190

HT 190

HyperTransport 190

Access to PCI 211

Additional Status Register 255

Bounce space 213

Bridge Command Register 247

Bridge Control Register 196, 249, 260

Bridge Primary (ZBbus) Status Register 248

Bridge Secondary (HT) Status Register 248

bus number 234

Command Register 249

COMPAT 195

Compatibility space 197

Configuration 194

configuration 234

configuration address 234

device number 234

Diagnostic Receive CRC Expected 255

Diagnostic Receive CRC Received 255

double-hosted chain 212

EOI 198

Error Control Register 253

Error Status Register 254

expansion space 194

function 234

generating interrupt messages 199

I/O space 195

interrupt acknowledgement 199

Isochronous BAR 252

isochronous bit 211

Isochronous Ignore Mask 253

Link Configuration Register 251

Link Control Register 250

Link Frequency Register 251, 252, 253

Little endian system 201

match bit lanes 203

match byte lanes 202

memory mapped 194

ordering rules 231

Peer to peer 211

Read restrictions 200

register 234

SRI Command Register 252

SRI Data Buffer Allocation Register 254

SRI Transmit Buffer Count Max Register 255

SRI Transmit Control Register 254

subtractive decode 195

Target Done 236

When not used 236

HyperTransport Configuration Header 245

HyperTransport Expansion Space 194

HyperTransport Fabric to PCI Bus 221

HyperTransport Resets 260

**I**

I/O Bridge 1 157

I/O Bridge Clocks 32

IFG 269

Interface Overview 265

Interface to PHY 271

interrupt\_diag 50, 52

interrupt\_idt 48, 49, 50, 52

interrupt\_idt\_clr 52

interrupt\_idt\_set 48, 52

interrupt\_map 52

interrupt\_mask 50, 52

interrupt\_source\_status 50, 52

interrupt\_status 52

interrupt\_trace 51, 52

interrupts

Arbitrated 49

Fixed 49

Non-vectored 49

IO\_CLK100 31, 43

io\_drive 379

io\_ext\_cfg 364, 365, 372, 375

io\_ext\_mult\_size 363, 375

io\_ext\_start\_addr 363, 376

io\_ext\_time\_cfg\_1 365

io\_ext\_time\_cfg0 365, 373, 376

io\_ext\_time\_cfg1 377

io\_int\_status 53

io\_interrupt\_addr 379



- [io\\_interrupt\\_addr0](#) [364](#)  
[io\\_interrupt\\_addr1](#) [364](#)  
[io\\_interrupt\\_data0](#) [378](#)  
[io\\_interrupt\\_data1](#) [378](#)  
[io\\_interrupt\\_data2](#) [378](#)  
[io\\_interrupt\\_data3](#) [378](#)  
[io\\_interrupt\\_parity](#) [379](#)  
[io\\_interrupt\\_status](#) [18](#), [364](#), [374](#), [377](#)  
[io\\_multi\\_size](#) [390](#)  
[io\\_start\\_addr](#) [390](#)  
[IPv4 Header Checksum](#) [283](#)  
[iSCSI](#) [180](#)  
[IsocBAR](#) [212](#)  
[IsocIgnMask](#) [212](#)
- J**
- [JTAG](#) [41](#), [70](#)  
[JTAG and Debug](#) [422](#)
- L**
- [L1CA](#) [23](#)  
[L2 Cache](#) [89](#)  
[L2 cache](#) [89](#)  
[l2\\_cache\\_disable](#) [94](#)  
[l2\\_ecc\\_tag](#) [89](#), [97](#), [100](#)  
[l2\\_misc\\_config](#) [99](#)  
[l2\\_misc\\_value](#) [91](#), [94](#), [99](#), [100](#)  
[l2\\_read\\_tag](#) [97](#), [100](#)  
[l2\\_way\\_enable](#) [91](#), [94](#)  
[LDT](#) [190](#)  
[ldt\\_interrupt](#) [55](#)  
[ldt\\_interrupt\\_clear](#) [50](#)  
[LDT\\_PWROK](#) [26](#), [260](#)  
[LDT\\_RESET\\_L](#) [26](#), [57](#), [260](#)  
[Line](#) [5](#)  
[Line Interface Configuration](#) [352](#)  
[Little Endian System](#)  
     [No Swaps](#) [201](#)
- M**
- [MAC Registers](#) [302](#)  
[mac\\_addr](#) [314](#)  
[mac\\_adfilter\\_cfg](#) [171](#), [172](#), [279](#), [282](#), [283](#), [285](#), [293](#), [316](#)  
[mac\\_admask](#) [315](#)  
[mac\\_cfg](#) [271](#), [272](#), [273](#), [275](#), [277](#), [281](#), [282](#), [284](#), [293](#),  
     [294](#), [299](#), [302](#)  
[mac\\_chlo](#) [281](#), [294](#), [299](#), [318](#)  
[mac\\_chup](#) [281](#), [294](#), [299](#), [318](#)  
[mac\\_debug\\_status](#) [313](#)  
[mac\\_enable](#) [301](#), [306](#)  
[mac\\_ethernet\\_addr](#) [274](#), [285](#), [315](#)  
[mac\\_frame\\_cfg](#) [271](#), [276](#), [284](#), [308](#)  
[mac\\_hash](#) [315](#)  
[mac\\_int\\_mask](#) [271](#), [286](#), [314](#)  
[mac\\_mdio](#) [287](#), [294](#), [318](#)  
[mac\\_rx\\_fifo\\_ptrs](#) [314](#)  
[mac\\_status](#) [157](#), [271](#), [273](#), [277](#), [285](#), [286](#), [310](#)  
[mac\\_status\\_0](#) [54](#)  
[mac\\_status\\_1](#) [54](#)  
[mac\\_status\\_2](#) [54](#)  
[mac\\_status\\_debug](#) [286](#)  
[mac\\_status1](#) [286](#), [313](#)  
[mac\\_status1\\_0](#) [56](#)  
[mac\\_status1\\_1](#) [56](#)
- [mac\\_status1\\_2](#) [56](#)  
[mac\\_thrsh\\_cfg](#) [272](#), [275](#), [294](#), [307](#)  
[mac\\_txd\\_ctl](#) [274](#), [306](#)  
[mac\\_type\\_cfg](#) [316](#)  
[mac\\_vlantag](#) [152](#), [164](#), [274](#), [285](#), [310](#)  
[MACs](#) [401](#)  
[mailbox](#) [48](#), [49](#)  
[mailbox\\_0](#) [46](#)  
[mailbox\\_1](#) [46](#)  
[mailbox\\_cpu](#) [46](#)  
[Management Interface to PHY](#) [287](#)  
[Mapping](#) [109](#)  
[mbox\\_clear](#) [46](#)  
[mbox\\_clr\\_cpu](#) [46](#)  
[mbox\\_set](#) [46](#)  
[mbox\\_set\\_cpu](#) [46](#)  
[mc\\_clock\\_cfg](#) [127](#), [138](#)  
[mc\\_config](#) [105](#), [106](#), [112](#), [119](#), [121](#), [126](#), [128](#), [135](#)  
[mc\\_cs\\_end](#) [111](#)  
[mc\\_cs\\_interleave](#) [111](#), [119](#), [121](#)  
[mc\\_cs\\_start](#) [111](#)  
[mc\\_csN\\_ba](#) [121](#)  
[mc\\_csN\\_col](#) [121](#)  
[mc\\_csN\\_row](#) [121](#)  
[mc\\_dramcmd](#) [126](#), [139](#), [139](#)  
[mc\\_drammode](#) [123](#), [124](#), [126](#), [139](#)  
[mc\\_test\\_data](#) [125](#)  
[mc\\_test\\_ecc](#) [107](#), [125](#)  
[mc\\_timing1](#) [125](#), [131](#), [135](#), [140](#)  
[mc\\_timing2](#) [125](#)  
[memory barrier](#) [107](#)  
[Memory clock](#) [33](#)  
[Memory Configurations](#) [109](#)  
[Memory Controller Architecture](#) [104](#)  
[Memory Locked in the L2 Cache](#) [92](#)  
[Memory Mapped Devices](#) [194](#)  
[MESI](#) [21](#)  
[MII](#) [264](#), [267](#)
- N**
- [NMI](#) [47](#), [50](#)  
[Normal Operation](#) [89](#)  
[Not Implemented](#) [5](#)
- O**
- [Open Page policy](#) [122](#)  
[Other Documentation](#) [4](#)  
[Overview of the ZBbus Protocol](#) [20](#)
- P**
- [P\\_RESET\\_L](#) [57](#)  
[P\\_RST\\_L](#) [26](#)  
[Packet FIFO](#) [292](#)  
     [16-bit encoded mode](#) [300](#)  
     [16-bit GMII style](#) [299](#)  
     [8-bit Encoded mode](#) [296](#)  
     [8-bit EOP flagged](#) [298](#)  
     [8-bit GMII style](#) [295](#)  
     [8-bit SOP flagged](#) [297](#)  
     [CRC](#) [294](#), [299](#)  
     [flow control](#) [294](#)  
[Packet FIFO interface](#) [264](#)  
[Packets Dropped by the DMA Channel](#) [283](#)  
[Page policy](#)



CAS time check 122  
 Closed 122  
 Hint Based 122  
 Open 122  
 Pause Frame 279, 315  
 Pause frame 305  
 pause frame 152, 164, 168, 267, 271, 284, 284, 303, 313, 317  
 PCI 190, 401  
     match byte lanes 202  
     access to mailbox registers 207  
     Adaptive Extend Register 243, 244  
     adaptive retry 217  
     Additional Status and Control Register 242  
     arbiter 190, 222  
     BAR 205  
     BAR0 Map Table Entry 242  
     bus number 234  
     bus zero 191  
     Bypass Control Register 244  
     Cache Line Size 241  
     ClassRevSet Register 233  
     Command Register 240  
     Compatibility space 197  
     Configuration 194  
     configuration 234  
     configuration address 234  
     Device Mode 199  
     device number 234  
     DEVSEL 206  
     endian policy (BAR0) 206  
     errors 235  
     Feature Control 242  
     function 234  
     I/O space 195  
     IDSEL generation 235  
     INTA Control Register 233, 243  
     interrupt acknowledgement 199  
     Latency Timer 241  
     Little endian system 201  
     match bit lanes 203  
     memory mapped 194  
     ordering rules 231  
     Read 217  
     Read Host Register 243  
     ReadLine 217  
     ReadMultiple 217  
     register 234  
     Status Register 240  
     SubSysSet Register 233  
     subtractive decode 195, 196, 199  
     Timeout Register 241  
     VendorIdSet 233  
 PCI and HyperTransport Address Range 192  
 PCI BAR2 46  
 PCI BAR3 46  
 PCI Bus and HyperTransport Fabric 190  
 PCI Bus To HyperTransport Fabric 219  
 PCI Configuration Header 236  
 PCI Full Access Space 199  
 PCI I/O Space 195

PCMCIA 384  
     attribute memory 386  
     status signals 386  
 PCMCIA Power Control Pins 401  
 pcmcia\_cfg 386, 387, 390, 393, 394  
 pcmcia\_status 395  
 PEC 406  
 Peer-to-Peer Accesses 219  
 perf\_cnt 62  
 perf\_cnt\_cfg 61  
 Performance Monitoring Features 134  
 Performance of the PCI and HyperTransport Interfaces 213  
 PERIPH\_REV3 6, 81, 91, 94, 99, 103, 117, 123, 126, 138, 140, 152, 154, 158, 163, 164, 168, 171, 172, 177, 178, 185, 186, 187, 270, 279, 282, 283, 285, 301, 305, 308, 310, 313, 315, 317, 372, 425, 448, 449, 478, 479  
 PHY 271  
 Physical Address 34  
 Physical Addresses 5  
 PLL 31, 41, 43  
 Preamble 269  
 Prid 29, 42  
 Protocol Engine and GMII/MII 267  
 Protocol Engine Configuration 271, 351

**R**

R\_EXC 23, 23, 77  
 R\_L2HIT 23, 77  
 R\_SHD 23, 23, 77  
 Read Only 5  
 Receive Path 277  
 Receiver Operation 275  
 Reference 446  
 Reserved 5  
 Reset 26  
 RESET\_L 26, 27, 30, 41, 232, 260, 422, 430, 432, 443  
 RESETOUT\_L 26, 27, 41, 44, 57  
 Response Phase 23  
 Rings and Chains 151  
 RMON 353  
 RMON Counters 289, 289, 353  
 RMON statistical counters 265  
 Row Address 117  
 Row, Column and Bank Configuration 112, 117  
 RTS 322, 323, 324

**S**

sb\_softres 41  
 SCD 41, 89  
 SDRAM Refresh 126  
 SDRAM Timing 125  
 ser\_addr\_mask 348, 359  
 ser\_clk 340  
 ser\_cmd 355  
 ser\_dma\_enable 357  
 ser\_err\_mask 346  
 ser\_int\_mask 358  
 ser\_line\_mode 354  
 ser\_maxfrm\_sz 345, 348, 357  
 ser\_min\_frm\_sz 346  
 ser\_minfrm\_sz 345, 356  
 ser\_mode 341, 343, 344, 345, 346, 348, 354  
 ser\_rx\_rd\_thres 356



[ser\\_rx\\_table](#) [359](#)  
[ser\\_status](#) [157](#), [341](#), [343](#), [346](#), [349](#), [357](#)  
[ser\\_status\\_debug](#) [358](#)  
[ser\\_tx\\_rd\\_thres](#) [346](#), [356](#)  
[ser\\_tx\\_table](#) [359](#)  
[ser\\_tx\\_wr\\_thres](#) [346](#), [356](#)  
[ser\\_usr0\\_addr](#) [359](#)  
[Serial Configuration Interface](#) [404](#)  
[Serial Interfaces](#) [321](#)  
[Serial Ports](#) [401](#)  
[SFD](#) [269](#)  
[Signal Description](#) [7](#)  
[smb\\_cmd](#) [410](#), [412](#), [416](#)  
[smb\\_control](#) [413](#), [416](#)  
[smb\\_data](#) [410](#), [412](#), [417](#)  
[smb\\_freq](#) [412](#), [416](#)  
[smb\\_pec](#) [406](#), [408](#), [410](#), [412](#), [418](#)  
[smb\\_start](#) [408](#), [410](#), [412](#), [418](#), [419](#)  
[smb\\_status](#) [406](#), [410](#), [412](#), [413](#), [417](#)  
[smb\\_xtra](#) [412](#), [417](#)  
[SMBus](#) [404](#)  
[SMBus Overview](#) [404](#)  
[SMBus Registers](#) [416](#)  
[SouthBridge](#) [195](#)  
[Southbridge](#) [195](#)  
[southbridge](#) [199](#)  
[southOnLDT](#) [196](#)  
[Special HyperTransport Space](#) [199](#)  
[SRI Command Register](#) [194](#), [256](#)  
[Standard Management Mode Accesses \(both ECC diag address bits zero\)](#) [97](#)  
[Standard RAM](#) [92](#)  
[Subtractive Decode](#) [195](#)  
[Supported DRAMs and DIMMs](#) [123](#)  
[Synchronous Interface Configuration](#) [351](#)  
[Synchronous Serial Loopback](#) [352](#)  
[Synchronous Serial Protocol Engine](#) [344](#)  
[system](#) [41](#)  
[System Control and Debug Unit](#) [41](#)  
[System Overview](#) [9](#)  
[System Reset Initialization of the HyperTransport Interface](#) [256](#)  
[system\\_cfg](#) [41](#), [42](#), [43](#), [321](#), [373](#)  
[system\\_manuf](#) [43](#)  
[system\\_reset](#) [41](#)  
[system\\_revision](#) [41](#), [42](#)  
[system\\_scratch](#) [42](#), [45](#)

**T**

[TCP](#) [178](#), [180](#)  
[TCP checksum](#) [283](#)  
[Terminology](#) [5](#)  
[The GPIO Pins](#) [397](#)  
[Timer Registers](#) [59](#)  
[Timer Special Cases](#) [58](#)  
[Timers](#) [57](#)  
[Timing Parameter Guidelines](#) [131](#)  
[trace buffer](#) [70](#), [76](#), [77](#), [79](#)  
[trace\\_cfg](#) [75](#), [76](#), [79](#), [84](#), [85](#), [86](#)  
[trace\\_event](#) [72](#)  
[trace\\_read](#) [76](#)  
[trace\\_sequence](#) [74](#)  
[Transaction ID](#) [21](#)  
[transaction ID](#) [67](#)  
[Transmitter Operation](#) [272](#)  
[Transport Protocol](#) [404](#)  
[trigger event](#) [70](#)  
[trigger sequence](#) [73](#)

**U**

[UDP](#) [178](#)  
[UDP checksum](#) [283](#)  
[UNDEFINED](#) [5](#)  
[UNPREDICTABLE](#) [5](#)  
[Using the L2 Cache as Memory](#) [91](#)  
[Using the PCI in Device Mode](#) [232](#)

**V**

[VGA](#) [195](#)  
[VGA display controller](#) [196](#)  
[Viewing Endian Policy as an Optimization](#) [204](#)  
[Virtual Address](#) [5](#)  
[VLAN TAG](#) [269](#)  
[VLAN tag](#) [270](#), [274](#)

**W**

[Watchdog Timers](#) [57](#)  
[watchdog\\_timer\\_cfg](#) [59](#)  
[watchdog\\_timer\\_cnt](#) [59](#)  
[watchdog\\_timer\\_init\\_cnt](#) [59](#)  
[Write Only](#) [5](#)

**Z**

[ZBbus Monitoring](#) [134](#)  
[zbbus\\_cycle\\_count](#) [58](#), [61](#)  
[zbbus\\_cycle\\_cp0](#) [54](#), [61](#)  
[zbbus\\_cycle\\_cp1](#) [54](#), [61](#)





---

***Broadcom Corporation***

16215 Alton Parkway  
P.O. Box 57013  
Irvine, California 92619-7013  
Phone: 949-450-8700  
Fax: 949-450-8710

Broadcom® Corporation reserves the right to make changes without further notice to any products or data herein to improve reliability, function, or design. Information furnished by Broadcom Corporation is believed to be accurate and reliable. However, Broadcom Corporation does not assume any liability arising out of the application or use of this information, nor the application or use of any product or circuit described herein, neither does it convey any license under its patent rights nor the rights of others.