

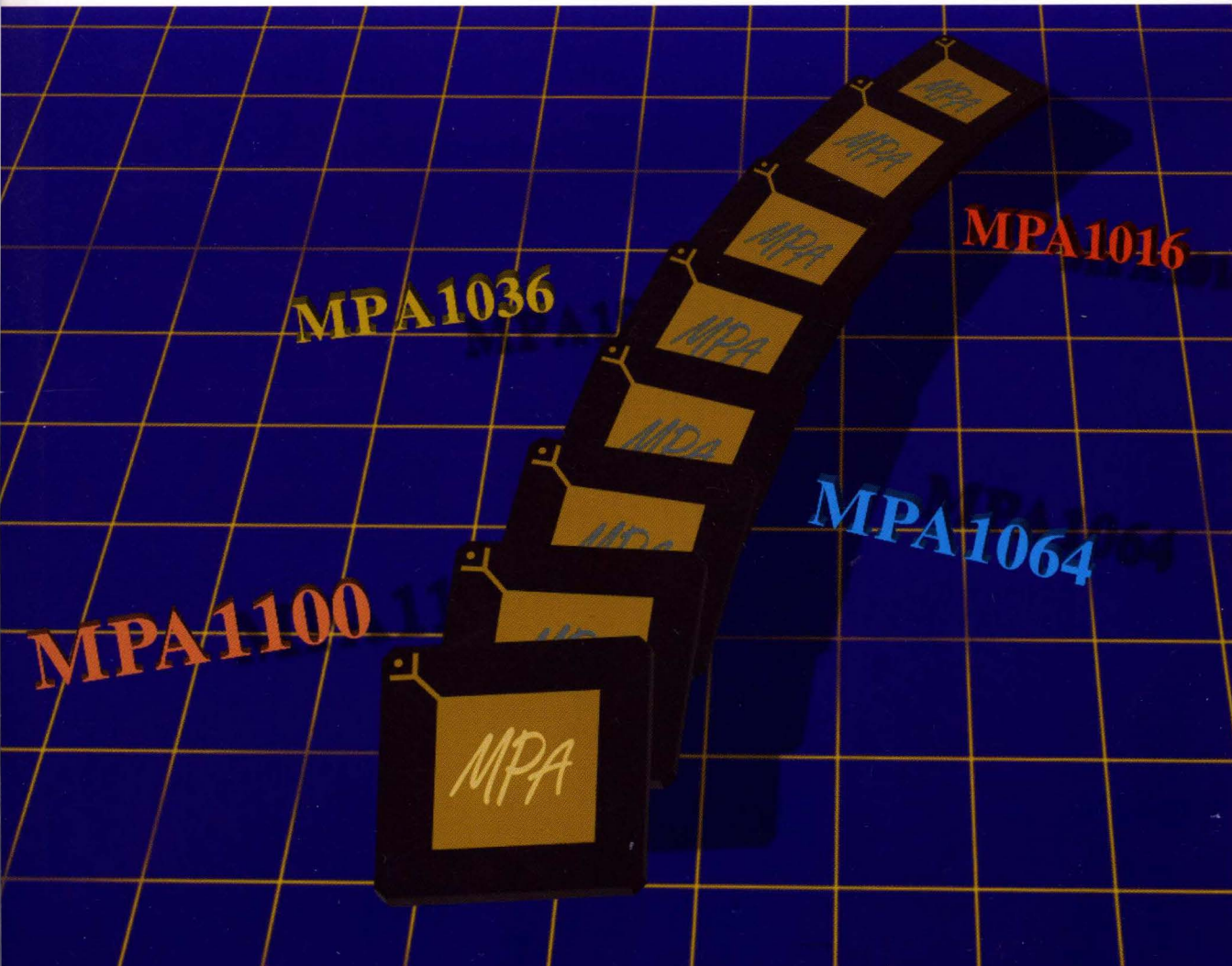


MOTOROLA

DL201/D REV 2

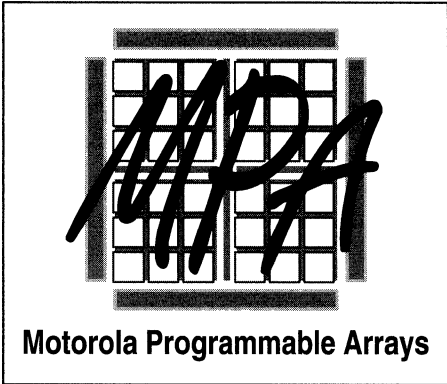
MPA **MOTOROLA PROGRAMMABLE ARRAY DATA**

MOTOROLA PROGRAMMABLE ARRAY DATA



Motorola Programmable Arrays
"Programmable Solutions That Fit"

Q3/97
DL201
REV 2



General Information **1**

Product Specifications **2**

Packaging, Quality & Reliability **3**

Application Notes **4**

How to Reach Us **5**

The brands or product names mentioned are trademarks or registered trademarks of their respective holders.

Suggested References:

Packaging Manual for ASiC Arrays, Motorola Inc., 1993. Stock Code BR916/D

HDC Series Design Reference Guide, Motorola Inc., 1991. Stock Code HDCDM/D


Reliability and Quality Handbook, Motorola Inc., 1993. Stock Code BR518/D

H4CPlus Series Design Reference Guide, Motorola Inc., 1994. Stock Code H4CPDM/D



MPA Motorola Programmable Array Data

This databook contains device specifications for Motorola's Programmable Arrays (MPAs).

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters can and do vary in different applications. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part. Motorola and  are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

© Motorola, Inc. 1997
Previous Edition © February 1996
"All Rights Reserved"

Printed in U.S.A.

CONTENTS

Ch 1. General Information

Introduction	1-2
MPA Product Briefs	
MPA1000 Description and Features	1-3
MPA Design System Description and Features	1-4
MPA17000 Series Serial EPROMs	1-5

Ch 2. MPA Product Specifications

MPA1000 Product Description	2-2
Architectural Overview	2-5
Device Configuration	2-15
Configuration Data Format	2-33
JTAG Boundary Scan	2-35
MPA1000 Pin Definitions	2-38
Pin Assignments	
MPA1016	2-39
MPA1036	2-41
MPA1064	2-44
MPA1100	2-48
Electrical Specifications	2-52
MPA17128 and MPA1765 Product Descriptions	2-54
MPA17C256 Product Description	2-64
Design System Product Description	2-76
Design System Product List	2-80

Ch 3. Packaging, Quality & Reliability

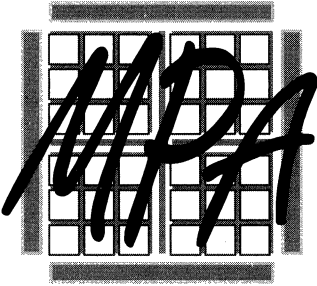
Packaging & Case Information	3-2
Motorola Quality and Reliability	3-13

Ch 4. Application Notes

68030 DRAM Controller Design Using Verilog HDL (AN1561/D)	4-2
Programming Multiple MPA1000 Devices Using Serial Peripheral Interface (SPI) (AN1562/D)	4-12
Effective Synthesis Techniques for MPA1000 Devices (AN1563/D)	4-25
Interfacing to the PowerPC™ with a Motorola Programmable Array (AN1564/D)	4-31
Using VIEWlogic's PROSeries 6.1 with the MPA Design System (AN1565/D)	4-83
In System Prototyping Using HDLs and FPGAs (AN1566/D)	4-93
Tuning the MPA Design System for Speed (AN1569/D)	4-96
Estimating Power in the MPA1000 Family (AN1587/D)	4-110
Using Mentor Graphics' Design Architect ver. A3 with the MPA Design System (AN1588/D)	4-113
Using OrCAD's Capture and Simulate with the MPA Design System (AN1589/D)	4-122
Using VIEWlogic's Workview Office 7.0 with the MPA Design System (AN1592/D)	4-133
Programming Large Configuration Files into Smaller Serial PROMs (AN1595/D)	4-145
Using Exemplar Logic's Galileo with the MPA Design System (AN1604/D)	4-158
Integrating Schematic Capture and Verilog Synthesis When Designing with the MPA (AN1613/D) ...	4-180
Optimizing VHDL/Verilog Designs for Speed for the MPA Family	
Using Exemplar Galileo Synthesis (AN1614/D)	4-198
An FPGA Primer for PLD Users (AN1615/D)	4-204
Using JTAG Boundary Scan with the Motorola MPA1000 Family of FPGAs (AN1618/D)	4-209
MPA1000 Primer for Schematic Designers (AN1619/D)	4-226
HDL Techniques for Faster Synthesized Counters (AN1623/D)	4-236

Ch 5. How to Reach Us

Three Ways to Receive Motorola Semiconductor Technical Information	5-2
Distributor and Worldwide Sales Offices	5-3



Motorola Programmable Arrays

General Information

1

INTRODUCTION TO MOTOROLA PROGRAMMABLE ARRAYS

1

Field programmable logic and in particular, field programmable arrays, have become the solution of choice for logic design implementation in applications where time to market is a critical product development factor. In addition, reconfigurable arrays have been used to enhance Customer product flexibility in ways that no other technology can match.

Microprocessors have traditionally been used to satisfy time to market and end product flexibility needs. This solution may not meet performance constraints and lacks the concurrency possible in an unconstrained hardware design. Typical design processes, therefore, reach a point where the overall design is partitioned into hardware and software components. An interface is defined and the design process continues along two parallel paths. Sometime later, the software and hardware components must be integrated. Problems usually develop at this point because of interface misinterpretation or partitioning that cannot meet design requirements. This impacts the hardware, the software and the schedule. If the hardware design is realized in programmable logic, the hardware can be manipulated as easily as the software.

Products which adapt to the end users particular requirements through self directed or end user directed reconfiguration are becoming more prevalent. As the number of modes of operation increases, mode specific hardware becomes a less cost effective solution. In the case where the end user is truly directing the adaptation, predetermined hardware solutions become untenable. Reconfigurable logic enables design solutions where dynamic hardware-software repartitioning is possible.

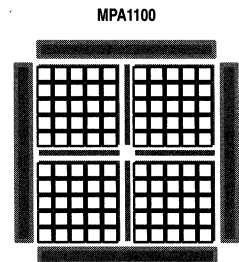
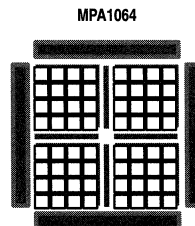
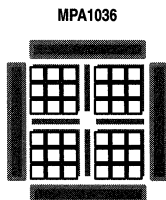
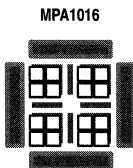
Programmable logic not only vastly improves the time necessary to implement a static design, but significant time to market and product feature benefits can be realized when hardware can be dynamically altered as easily as software.

To reduce design cycles, designers have also turned towards high level design languages and logic synthesis tools. Many programmable logic solutions are poorly suited to this design methodology, however. An incompatibility exists between logic synthesis algorithms originally developed for gate level design and the block-like structures found on many programmable logic devices. This can result in significant under utilization or degraded performance. In either case a more expensive device is required. Real gate level programmable devices are ideally suited to this design methodology.

When schematic based design methods are used, some programmable logic solutions impose significant constraints on design implementation to insure satisfactory results. This imposition tends to bind the design to a particular programmable device and requires a significant learning investment. Any design specification changes which impact design decisions made to fit this imposed structure can have disastrous effects on utilization and performance and potentially require a more expensive device or even a costly redesign. Gate level programmable devices coupled with sophisticated, timing driven, implementation tools minimize device specific optimization.

Any design process includes a significant amount of learning. Usually engineers spend most of this time learning about product requirements or prototyping critical portions of the design to prove implementation feasibility. Many programmable logic solutions are not push button; time must be spent learning programmable device architecture or implementation tool quirks. Worse yet, the design may require modification or manual component placement to meet design targets. The cost? Time to market.

The reconfigurable Motorola Programmable Array (MPA) and MPA design system maximize application flexibility and minimize time to market by delivering a gate level, push button, programmable logic solution.



MOTOROLA
SEMICONDUCTOR TECHNICAL DATA

MPA1000 Programmable Arrays

Motorola Programmable Array (MPA) products are a high density, high performance, low cost, solution for your reconfigurable logic needs. When used with our automatic high performance design tools, MPA delivers custom logic solutions in minutes rather than weeks. And the low cost keeps those solutions competitive throughout the product lifecycle.

The MPA architecture has solved the historical problems associated with fine grain devices without sacrificing re-programmability, reliability, or cost. MPA1000 devices are reprogrammable SRAM based products manufactured on a standard 0.43 μ Leff CMOS process with logic capacities from 3,500 to more than 22,000 equivalent FPGA gates. MPA Logic resources hold a single gate or storage element providing a highly efficient, adaptable, design implementation medium. Gate level logic resources, abundant hierarchical interconnection resources and automatic, timing driven, tools work together to quickly provide design implementations that meet timing constraints without sacrificing device utilization.

Staying focused on end product design rather than implementation tools or device architecture gets the design done faster and, unlike other programmable solutions, without programmable logic device specificity to impede future design migration efforts. The combination of automatic tools and gate level architecture is ideal for traditional schematic driven or high level language based design methodologies. In fact, logic synthesis tools were originally designed for and produce the most efficient results when targeting gate level devices.

High MPA1000 register count and controlled clock skew is ideal for designs employing pipelining techniques such as communications. The unique set of MPA1000 I/O programming options make these devices suitable for industrial and computer interfacing circuits.

MPA1016
MPA1036
MPA1064
MPA1100

1

PROGRAMMABLE ARRAY
3,500 to 22,000 GATES

- Multiple I/O from 80–200 I/O Pins
- Programmable 3V/5V I/O at Any Site
- Multiple Packaging Options
- Fine Grain Structure Is Optimized for Logic Synthesis
- Programmable Output Drive, 4/6mA @ 5.0V and 3.3V
- High Register Count, with 560–2,900 Flip-Flops
- IEEE 1149.1 JTAG Boundary Scan
- Eight Low-Skew (<1ns) Clocks

MPA1000 Family Members

FPGA Gates*	Part No.	Logic Cells	Internal Flip-Flops	I/O Cell Flip-Flops	Avail I/O Pins	Packages	Availability
3500	MPA1016FN	1600	400	122	61	84 PLCC	NOW
	MPA1016DD			160	80	128 PQFP	NOW
8000	MPA1036FN	3600	900	122	61	84 PLCC	NOW
	MPA1036DD			160	80	128 PQFP	NOW
	MPA1036DH			240	120	160 PQFP	NOW
	MPA1036HI			240	120	181 PGA	NOW
14200	MPA1064DH	6400	1600	240	120	160 PQFP	NOW
	MPA1064DK			320	160	208 PQFP	NOW
	MPA1064KE			320	160	224 PGA	NOW
	MPA1064BG			320	160	256 PBGA	3Q97
22000	MPA1100DK	10000	2500	320	160	208 PQFP	NOW
	MPA1100HV			400	200	299 PGA	NOW
	MPA1100BG			400	200	256 PBGA	3Q97

* Equivalent to Industry Standards, as supplied by most manufacturers.



Motorola Programmable Array Design System

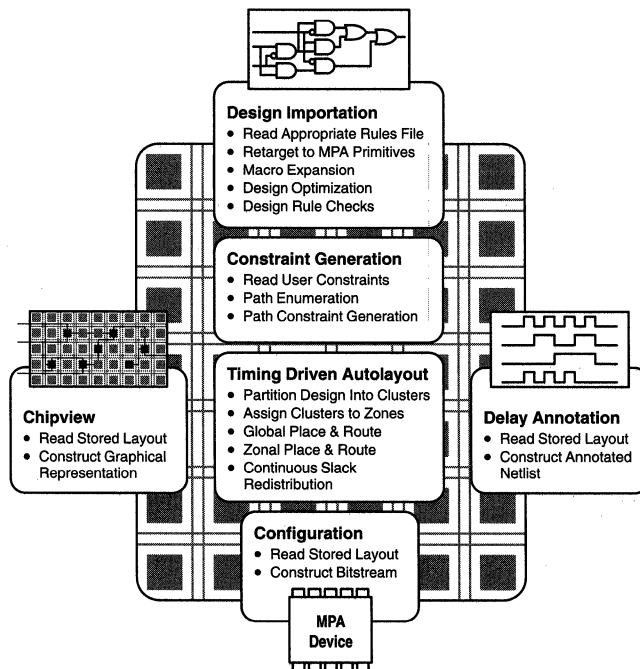
DESIGN SYSTEM

1

The Motorola Programmable Array (MPA) design system is a bridge between a design capture environment and Motorola field programmable arrays. The MPA design system automatically transforms designs into device configurations to realize a design, when loaded into an MPA device. A design is automatically analyzed, optimized, transformed into MPA cells, partitioned, placed and routed based on timing constraints for every path in the design. MPA design tools understand and optimally utilize the MPA device architecture; this eliminates the need to learn a new set of rules and makes these tools ideally suited for use with logic synthesis. Full incremental design support reduces design implementation time and powerful library retargeting capabilities allow you to reuse designs which may have been implemented on less capable devices. The MPA design system operates on existing hardware platforms and supports design capture and simulation tools from more than 10 vendors. All these features plus on-line, hypermedia, help make the MPA design system a powerful, yet extremely easy to use, design implementation engine.

Features

- Push Button Implementation
- Optimal Use of MPA Device Resources
- Optimal Results with Gate Level Design Input
- Library of Common MSI Functions
- Design Flow Manager
- Design Retargeter
- Timing Driven with Integrated Static Timing Analysis
- Layout Delay extraction for post layout simulation
- Layout viewer
- Incremental design support
- On-line, hypermedia, documentation
- Supports all popular design capture and simulation tools
- Lowest cost FPGA development systems.
- Instant access; Downloading via the internet (WWW, ftp).
- Supports multiple speed grades



MOTOROLA
SEMICONDUCTOR TECHNICAL DATA

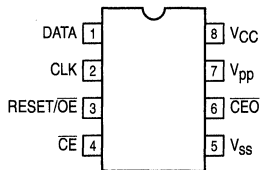
MPA17000 Serial EPROMs

The MPA17128, MPA1765 serial OTP EPROMs provide a compact, low pin count, non-volatile configuration store for MPA1000 devices.

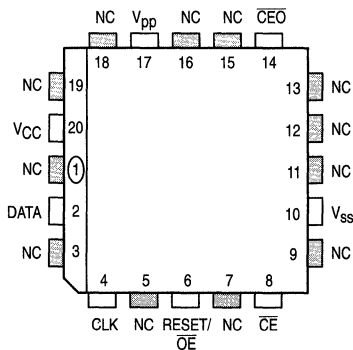
MPA17000 devices can be cascaded for increased memory capacity when needed. They are available in the standard 8-pin plastic DIP (N suffix), 8-pin SOIC (D suffix) and 20-pin PLCC (FN suffix) packages.

- Configuration EPROM for MPA1000 Devices
- Voltage Range — 4.5 to 6.0V
- Maximum Read Current of 10mA
- Standby Current of 10 μ A, Typical
- Industry Standard Synchronous Serial Interface
- Full Static Operation
- 10MHz Maximum Clock Rate at 5.0V
- Programmable Polarity on Hardware Reset
- Programs With Industry Standard Programmers
- Electrostatic Discharge Protection > 2000 Volts
- 8-Pin PDIP and SOIC; 20-Pin PLCC Packages
- Commercial (0 to +70°C) and Industrial (-40 to +85°C)

8-Lead Pinouts
(Top View)



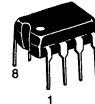
20-Lead Pinout
(Top View)



MPA17128 MPA1765

1

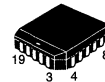
128K, 64K SERIAL EPROM



P SUFFIX
PLASTIC PACKAGE
CASE 626-05



D SUFFIX
PLASTIC PACKAGE
CASE 751-05



FN SUFFIX
PLCC PACKAGE
CASE 775-02

PIN NAMES

Pins	Function
DATA	Data I/O
CLK	Clock
RESET/OE	Reset Input and Output Enable
CE	Chip Enable Input
V _{SS}	Ground
CEO	Chip Enable Output
V _{PP}	Programming Voltage Supply
V _{CC}	+4.5 to 6.0V Power Supply
NC	Not Connected



Advance Information
MPA17000 Serial EEPROM

1

The MPA17C256 is an easy to use and cost effective serial configuration memory ideally suited for use with today's popular SRAM based FPGAs. The MPA17C256 is available in 8-pin PDIP and 20-pin SOIC and PLCC packages, adhering to industry standard pinouts. The device interfaces downstream FPGA(s) with a very simple enable, clock and data interface. The MPA17C256 is reprogrammable with no need for a higher programming "super voltage"; it may even be reprogrammed on board. The MPA17C256 also has user programmable RESET/OE polarity.

- EE Programmable 262,144 x 1 bit Serial Memories Designed to Store Configuration Programs for FPGAs
- Simple Interface to SRAM FPGAs
- Cascadable to Support Additional Configurations or Future Higher Density FPGAs
- Low Power CMOS EEPROM Process
- Programmable Reset Polarity
- Available in Space Efficient 8-Pin PDIP, 20-Pin SOIC and 20-Pin PLCC Packages
- In-System Programmable via 2-Wire Bus

Controlling the MPA17C256 Serial EEPROM

Most connections between the FPGA device and the Serial EEPROM are simple and self-explanatory:

- The DATA output of the MPA17C256 drives DIN of the FPGA devices
- The master FPGA DCLK output drives the CLK input of the MPA17C256
- The CEO output of the first MPA17C256 drives the CE input of the next MPA17C256 in a cascade chain of EEPROMs.
- SER_EN must be connected to VCC
- \overline{CE} enables the chip and is required to enable the DATA output pin
- RESET/OE is chip reset and is part of the DATA output enable structure

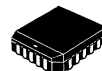
MPA17C256



P SUFFIX
 8-LEAD PLASTIC PACKAGE
 CASE 626-05



DW SUFFIX
 20-LEAD PLASTIC SOIC WIDE PACKAGE
 CASE 751D-04



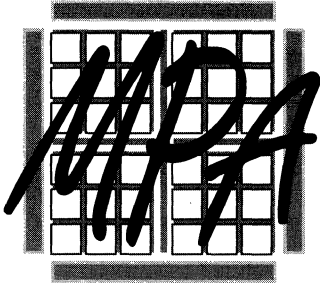
FN SUFFIX
 20-LEAD PLCC PACKAGE
 CASE 775-02

PIN NAMES

Pins	Function
DATA	Data I/O
CLK	Clock
RESET/OE	Reset Input and Output Enable
CE	Chip Enable Input
V _{SS}	Ground
CEO	Chip Enable Output
SER_EN	Programming Enable
V _{CC}	+4.5 to 6.0V Power Supply
NC	Not Connected

This document contains information on a new product. Specifications and information herein are subject to change without notice.





Motorola Programmable Arrays

Product Specifications **2**

MPA1000 Product Description

Motorola Programmable Array (MPA) products are a high density, high performance, low cost, solution for your reconfigurable logic needs. When used with our automatic high performance design tools, MPA delivers custom logic solutions in minutes rather than weeks. And the low cost keeps those solutions competitive throughout the product lifecycle.

The MPA architecture has solved the historical problems associated with fine grain devices without sacrificing re-programmability, reliability, or cost. MPA1000 devices are reprogrammable SRAM based products manufactured on a standard 0.43 μ Leff CMOS process with logic capacities from 3,500 to more than 22,000 equivalent FPGA gates. MPA logic resources hold a single gate or storage element providing a highly efficient, adaptable, design implementation medium. Gate level logic resources, abundant hierarchical interconnection resources and automatic, timing driven, tools work together to quickly provide design implementations that meet timing constraints without sacrificing device utilization.

Staying focused on end product design rather than implementation tools or device architecture gets the design done faster and, unlike other programmable solutions, without programmable logic device specificity to impede

future design migration efforts. The combination of automatic tools and gate level architecture is ideal for traditional schematic driven or high level language based design methodologies. In fact, logic synthesis tools were originally designed for and produce the most efficient results when targeting gate level devices.

High MPA1000 register count and controlled clock skew is ideal for designs employing pipelining techniques such as communications. The unique set of MPA1000 I/O programming options make these devices suitable for industrial and computer Interfacing circuits.

Features

- Multiple I/O from 80–200 I/O Pins
- Programmable 3V/5V I/O at Any Site
- Multiple Packaging Options
- Fine Grain Structure Is Optimized for Logic Synthesis
- Programmable Output Drive, 4/6mA @ 5.0V and 3.3V
- High Register Count, with 560–2,900 Flip–Flops
- IEEE 1149.1 JTAG Boundary Scan
- Eight Low–Skew (<1ns) Clocks

Table 2–1. MPA1000 Family Members

FPGA Gates*	Part No.	Logic Cells	Internal Flip–Flops	I/O Cell Flip–Flops	Avail I/O Pins	Packages	Availability
3500	MPA1016FN	1600	400	122	61	84 PLCC	NOW
	MPA1016DD			160	80	128 PQFP	NOW
8000	MPA1036FN	3600	900	122	61	84 PLCC	NOW
	MPA1036DD			160	80	128 PQFP	NOW
	MPA1036DH			240	120	160 PQFP	NOW
	MPA1036HI			240	120	181 PGA	NOW
14200	MPA1064DH	6400	1600	240	120	160 PQFP	NOW
	MPA1064DK			320	160	208 PQFP	NOW
	MPA1064KE			320	160	224 PGA	NOW
	MPA1064BG			320	160	256 PBGA	3Q97
22000	MPA1100DK	10000	2500	320	160	208 PQFP	NOW
	MPA1100HV			400	200	299 PGA	NOW
	MPA1100BG			400	200	256 PBGA	3Q97

* Equivalent to Industry Standards, as supplied by most manufacturers.



MPA1000 Serial EPROM/EEPROM Family

Capacity	MPA Companion	Part Number	Packages	Availability	Notes
64K	MPA1016	MPA1765P MPA1765D MPA1765FN	8 DIP 8 SOIC 20 PLCC	NOW	OTP
128K	MPA1036	MPA17128P MPA17128D MPA17128FN	8 DIP 8 SOIC 20 PLCC	NOW	OTP
256K	MPA1064	MPA17C256P MPA17C256DW MPA17C256FN	8 DIP 20 SOIC 20 PLCC	3Q97	Eraseable

2

MPA1000 Capacity

Programmable logic gate capacity is difficult to ascertain because it is design and design tool dependent. Programmable logic capacities can only be meaningfully compared using identical designs and automatic tools. Figure 2-1 shows that under these circumstances, the MPA1036 contains from 2.1 to 1.3 XC3190 devices.

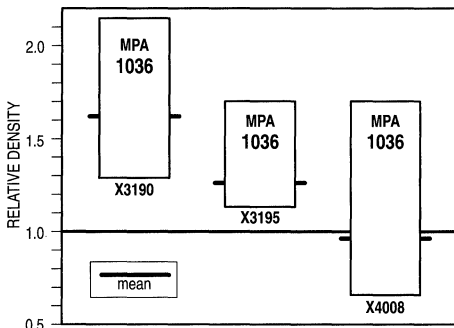


Figure 2-1. Equivalent Gate Capacity

Table 2-1 on page 2-2 shows the members of the MPA1000 family and lists the I/O, logic cell, flip flop and gate capacities for each device. To facilitate Customer device selection, Motorola rates MPA device capacity in FPGA equivalent gates. The equivalent gate counts shown were derived using identical designs and a push button implementation methodology. While this method is useful in a comparative sense, actual device capacity remains a design dependent quantity. Designs with high register gate or XOR gate to total gate ratio will pack more efficiently than the averages shown in Figure 2-1.

MPA1000 Performance

Device performance is more design and design tool dependent than device capacity. Table 2-2 shows selected cell performance figures for a typical ungraded MPA1000 device. Calculating MPA1000 DFF toggle rate from this

information yields an unrealistically high expectation for device performance. Some manufacturers publish specifications for small functional blocks like counters. While more useful than toggle rates, they are based on ideal placement and routing conditions seldom achievable without manual intervention. Industry benchmarks are useful for relative comparisons of benchmark design performance, but benchmark designs don't end up in products. In addition, the design methodology used requires, manual, architecture dependent, design optimization and expert level architectural and design tool experience. Using this design methodology for real designs means a costly learning curve, severe technology migration limitations and many hours of extra design effort for each end product. If the incentive to use a programmable solution is time to market and product flexibility, this is not the ideal approach. A push button, gate level, approach increases design flexibility and improves time to market. The MPA1000 and MPA design system have been engineered to deliver a high performance gate level solution. Gate level design is widely understood, technology independent and synthesis friendly. A library of common MSI functions with optimized gate level representations are provided to reduce design implementation time.

Table 2-2. Selected MPA1000 Performance Figures

	Typical
MEDIUM BUS DELAY	1.2ns
DFF CLK TO Q	0.6ns
DFF SETUP TIME	1.5ns
TYPICAL DFF TOGGLE RATE	256MHz

(25° C, V_{CC} = 5V)

If identical designs and timing constraints are used with automatic, timing driven, design tools, a more appropriate performance comparison can be made. Figure 2-2 compares the MPA1036 vs. the XC4008 for 7 designs. The typical MPA1036 device is 48% faster than the XC4008-6 and 28% faster than the XC4008-4 for 7 identical, complex, chip level designs. In real design situations, gate level flexibility and hierarchical routing coupled with sophisticated, timing driven, design tools results in significant performance gains and reduced time to market.



2

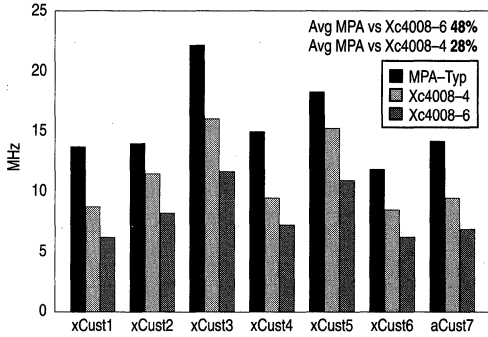


Figure 2-2. MPA1036 versus XC4008 - 7 Push Button Designs

If step and repeat style designs typical of industry benchmarks are used (Figure 2-3), MPA retains its performance edge. While the performance gap shrinks by about 10%, absolute design performance increases dramatically compared to those shown in Figure 2-2. As critical path depth decreases, design performance increases as expected. In general these benchmarks tend to have

narrowly distributed performance constraints and shallow path depths atypical of many real design implementations. In either case using benchmark information to estimate product performance for arbitrary designs is unlikely to yield reliable results. This information is intended to illustrate the range of performance enhancement possible when MPA is selected.

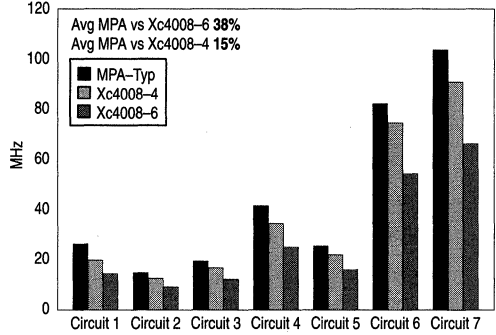


Figure 2-3. MPA1036 versus XC4008 - 7 Push Button, Step & Repeat Designs



MPA1000 Architectural Overview

MPA1000 Architecture

MPA1000 is a high density, high performance, low cost device family which maximizes application flexibility and minimizes time to market by delivering a gate level reprogrammable logic solution. Combined with automatic high performance design tools, the MPA1000 family is ideally suited to logic synthesis or gate level (gate array like) design methods.

Logic resources in the MPA1000 are fine grained – each logic cell holds a single gate or a storage element. This provides a highly efficient, adaptable, design implementation medium. Gate level logic resources, abundant hierarchical interconnection resources and automatic, timing driven, tools work together to quickly provide design implementations that meet timing constraints without sacrificing device utilization.

The MPA1000 architecture has solved the historical problems associated with fine grain architectures without sacrificing re-programmability, reliability, or cost. Previous reprogrammable fine grain architectures utilized routing architectures substantially similar to that of coarse grained products. Other fine grained architectures resorted to antifuse programming elements to address performance issues, increasing cost, while reducing reliability and abandoning reconfigurability. MPA utilizes a new routing structure which takes advantage of fine logic block granularity to achieve superior design performance.

MPA1000 devices are manufactured using a standard submicron CMOS process. SRAM cells comprise device configuration memory. MPA1000 devices can be quickly and infinitely reprogrammed.

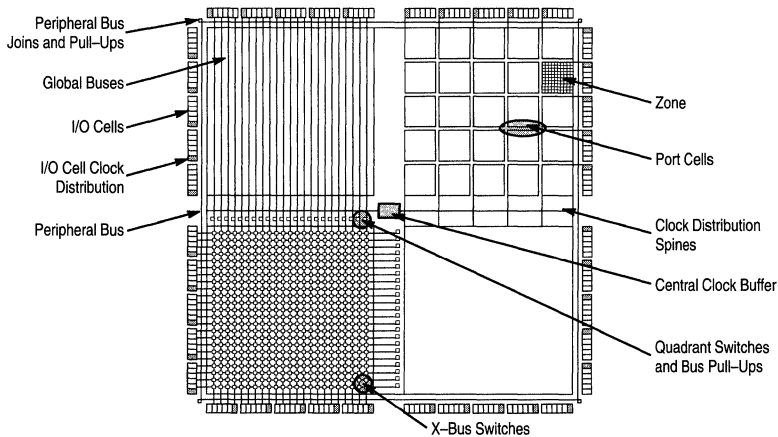


Figure 2-4. MPA Architectural Overview

Partitioned Resources

Each device is a multilevel partitioned array of cells. At the highest level of hierarchy each device is partitioned into 4 equal sized sections called quadrants. I/O cells surround the quadrants. Each quadrant is further subdivided into zones. A zone consists of a 10x10 array of core cells, 20 port cells and a clock distribution cell (Figure 2-6). Zone core cells are

organized into 2x2 groups called tiles. The number of zones per quadrant defines a particular device as shown in Figure 2-5. Partitioning the device in this manner minimizes bus loading and provides an opportunity to segment device level placement and routing. This speeds design implementation time, especially if multiple processors are used. Figure 2-4 is a synopsis of the overall MPA structure.

2



2

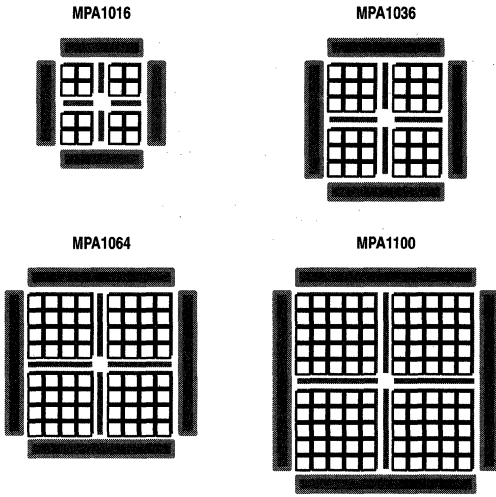


Figure 2-5. The MPA 1016, MPA1036, MPA1064 and MPA1100

Hierarchical functionality complements the robust routing resource to deliver extremely efficient design realizations. While the look up table approach of non-gate level devices can provide any function of its inputs, this flexibility is costly when simple functions are required. In contrast the simplicity, small size, and hierarchical organization of the MPA1000 delivers a more silicon efficient implementation. Logic blocks of arbitrary size and aspect ratio are automatically constructed, optimized and interconnected based on design constraints and gate level design representations. This capability complements logic synthesis technology and maximizes design migration potential. As FPGA device capacity increases, design diversity will also increase. The malleable granularity and adjustable routing resource of the MPA can accommodate this diversity with consistent silicon efficiency and performance.

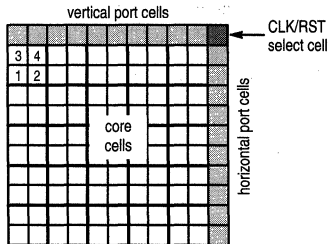


Figure 2-6. Zone Structure

Core Cells

Each core cell has 2 inputs, each input is configured to receive signals from 1 of 7 potential sources (Figure 2-7). 5 sources are from local interconnect and 2 are from zone level interconnect. Each cell output connects to 8 other cells via local interconnect and is configured to connect to up to 4 medium buses. Cells are sometimes used to provide additional routing resource. The ability to use a core cell as a routing resource or as logic provides a programmable means of adjusting routing resource to fit design specific requirements.

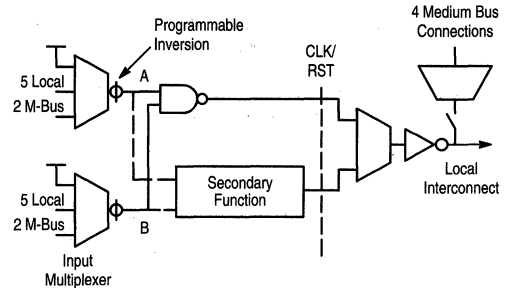


Figure 2-7. Core Cell Structure

Each cell has three states; repowering buffer, primary function, and secondary function. In addition, all cell inputs have programmable input inversion. MPA1000 core cells are organized in 2x2 groups called tiles. Within a tile, each of the 4 cells has a different secondary function (Figure 2-8). The core cell primary function is a 2 input NAND. Secondary functions include; XOR, register, and wired OR. The register element is configured as a DFF or latch with clock enable and set or reset. A special, 1ns skew, network is provided to drive register clock and reset/set pins. High performance, gate level, cells necessitate controlled clock skew to avoid negative setup time situations. The MPA cell states were chosen based on a careful analysis of macrocell utilization statistics from a large number of ASIC designs implemented in Motorola's H4C array.

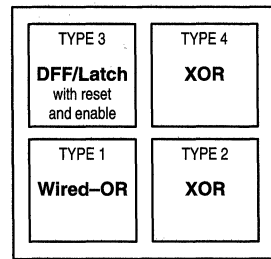


Figure 2-8. Core Cell Secondary Function



☞ One Hot State Machine Design is Preferred

Designing your state machines as one hot is usually the most efficient method for the register rich MPA.

☞ BUFF from the MICROLIB

BUFF is the only buffer available to the designer that will not get mapped out on import.

There are quite a few trivial optimizations that get made to your design during import, one of the most common is getting rid of superfluous 'buffers'. Examples of which include INV (inverters), AN2 gates with both inputs tied together, AN2 gates with one input tied high etc. (A more complete description of this re-mapping process can be found in the on-line help of the MPA Design System under: "Help on Design → Logic Optimisation → Summary of Optimisations".) Don't panic at the above statement that inverters are "gotten rid of". They are simply mapped to the correct sense on the core cell's programmable input multiplexers. No delay penalty is incurred for inserting an INV.

☞ Tri-State drivers are not available internally

Because the MPA's routing resources are fully buffered (actively driven) there are no internal tri-state buffers available. Designers accustomed to using such elements to allow multiple drivers access to a single data line, should instead consider using multiplexers.

☞ Wired-Or a.k.a. Open Drain

In some instances, it may be preferable to use a collection of open drain drivers to drive a single data line. The MPA library elements that accommodate this type of connection include: WINV, WOR2, WND2, and WBUF. It is important to remember that open drain drivers can only actively pull a signal low, a passive pull up resistor is required to pull the net high; that's the job of the WPUP library element. By default, instantiating a WPUP element results in a single pull-up resistor being attached to the net. Assigning the attribute DPLD_PUP with a value of BOTH results in two pull up resistors being added in parallel to the net. The low to high transition time is thus improved, but at the expense of more static current drain when any of the attached drivers is holding the net low.

Besides lower speed, another drawback of using open drain drivers in the MPA is the restriction that all the open drain drivers within a zone must reside on the same Wired-OR Bus, and that drivers in other zones must also be in placed in the same relative horizontal position. The autolayout tool

handles all of this automatically, but it does tend to reduce the number of valid solutions available to the autolayout tool for the remainder of your design.

☞ You Must Use All Macro Inputs, ONE and ZERO

The autolayout tool insists that all MACROLIB and MICROLIB inputs be used. If you don't need a particular input for your design, you are still required to tie it to logic or a ONE or a ZERO (from the MICROLIB). There is no routing consumed when specifying a ONE or a ZERO, the tie off is made at the cell's input selection mux. There is no fan out restriction for a ONE or a ZERO.

2

I/O cells

I/O cells are located at the device periphery surrounding the quadrants (Figure 2-4 on page 2-5). Besides direct input and output, each I/O cell can be configured to be; input, output, bidirectional, registered input, registered output, registered I/O. The two registers can be independently configured as a latch or D-type flip flop. Input register setup time is adjustable to compensate for clock network input delay. Input buffer threshold adjustment provides either TTL or CMOS levels. Output buffer drive capability is programmable to 4mA or 6mA. And each output can be independently programmed to either 3V or 5V levels with slew rate control. The output buffer can be configured as an open drain to facilitate system level wired OR applications. Figure 2-10 sums up I/O cell structure. Dedicated, fully IEEE 1149.1 compliant boundary scan is also provided.

The output buffers of unused I/O cell outputs are "turned-off" presenting a high impedance load to the external world. Similarly, input buffers of unused I/O cell inputs are also "turned-off"; there is no requirement to tie unused inputs high or low.

The MPA's output drivers are actually composed of a pair of 4mA drivers, only the second of which has controllable slew rate.

Default output configuration is the first 4mA driver. If the user attributes the I/O cell (or its formal port) with DPLD_OPDRIVE set to a value of 6mA, then the second 4mA driver will be added in parallel.

If the design calls for multiple 6mA drivers to be switched simultaneously, the designer should consider also attributing the outputs with DLPD_OPSLEW set to a value of "low". Doing so decreases the di/dt term in the familiar $V = L di/dt$ equation, thus reducing ground bounce.

```
instance outbuff attribute dpld_opdrive 6ma
instance outbuff attribute dpld_opslew low
```

Figure 2-9. Sample .PAT Entries for a 6mA, Low Slew Rate Output Called "Outbuff"



Table 2-3. Slew Rates for the MPA1000 Family (Note 1.)

Output Conditions	t _r (ns) at 5V	t _f (ns) at 5V	t _r (ns) at 3.6V	t _f (ns) at 3.6V
DPLD_OPDRIVE=6mA & DLPD_OPSLEW=high	1.7	2.0	0.9	1.2
DPLD_OPDRIVE=6mA & DLPD_OPSLEW=low	0.6	1.0	0.3	0.9
DPLD_OPDRIVE=4mA & DLPD_OPSLEW=high (Note 2.)	1.1	1.4	0.6	1.0

- Measurements taken between 10% and 90% of V_{DD} at 25°C, C_L = 50pF. Note that DPLD_OPDRIVE = 4mA with DLPD_OPSLEW = low is an illegal combination.
- Default values.

2

Start Off Easy, Begin with IPBUF, OPBUF, IPCLK, IPRST

The Complex I/O can be a space and a time saver for your more critical designs, but you may want to consider starting off slow and use the simpler I/O structures.

Enable and Reset Pins on Complex I/Os do not have to be tied

There are too many permutations possible in the I/O cell to make each available as a unique macrocell in the IOLIB. Consequently a short cut has been made available to the designer using Complex I/O, namely it is not necessary to tie reset or clock enable inputs high or low when using elements of the IOLIB. (N.B. This is not true for elements from the MICROLIB or MACROLIB. Each of these inputs must be used or otherwise tied off.) The autolayout software will make the obvious assumptions about how the unused input should be tied and make the tie off for you.

Don't fix your I/O locations unnecessarily

Fixing your I/O locations using DPLD_PAD_PLACE attribute may place an undue burden on the autolayout tool. Most designs will route to a higher performance level if the autolayout tool is given as much freedom as possible with regards to I/O pin placement.

Twinning Outputs

Two outputs can be connected in parallel to increase the the output drive current, however, to avoid contention between drivers, care must be taken to insure that output signals are synchronized. Use the following as guidelines:

- Connected outputs must reside in the same I/O zone. (The I/O pad ring is divided into zones each containing 5 I/O cells and two primary clocks. To identify a zone in a packaged product, look for groups of 5 adjacent I/Os in the pinout assignment.)
- Output signals must be gated through the I/O flip-flop registers.
- Output flip-flops must be clocked by a common primary clock signal via the clock distribution network, which is balanced and has a skew of <1ns between any two registered clock inputs. (Primary clock signals are the only way in which the I/O flip-flop clocks. Clock signals may originate external to the device via library element IPCLK or from the array by routing the signal to the primary clock bus via APCLK.)
- To reduce ground bounce, twinned outputs should be as close as possible to a VSSE pin. If ground bounce persists, alternate slew rate – fast on one, slow on the other.
- Using open drain output is a safer alternative, although, the speed will be limited by the pull-up resistor.

Hierarchical Routing Resources

The MPA interconnection structure is partitioned into 3 levels; Global, Zonal (or medium), and Local. Local interconnection is used to connect a core cell to 8 of it's perpendicular neighbors (Figure 2-11). Zonal interconnect consists of the medium buses and connects groups of cells within a zone (Figure 2-12). Global interconnect includes global buses, x buses and interquadrant switches (Figure 2-4 on page 2-5). Global buses provide quadrant and chip level inter-zone and zone to I/O cell interconnections. Special interconnection resources are also present and consist of clock distribution, wired-OR and peripheral bus. Routing specialization provides an opportunity for level specific performance optimization. Specialization also diminishes the amount of interconnection options required at each core cell, reducing cell size and boosting silicon efficiency.

Local Interconnect

Local interconnect provides the fastest path between 8 neighboring core cells. Local interconnect is continuous across the device and is not effected by zonal boundaries. Local interconnection favors frequently used connections, the cell to the immediate left and immediate right of the driving cell have 2 connections. Local connections are used for high performance intrazone connections and are also used to cross zone boundaries when necessary.



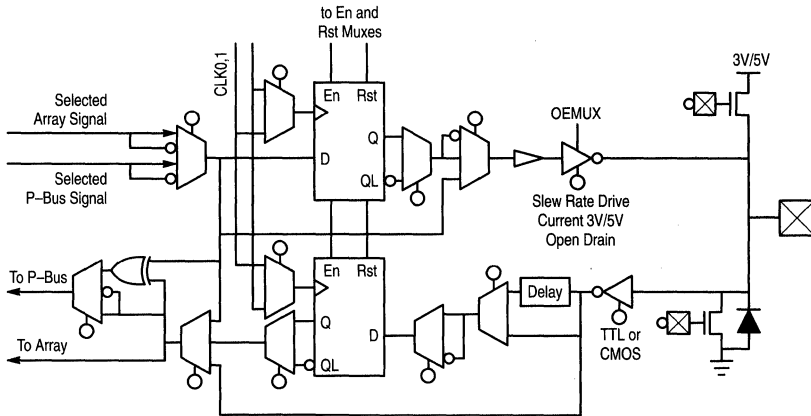


Figure 2-10. Input/Output Cell Structure

2

Medium Interconnect

Medium interconnect spans a single zone and provides intrazone connections beyond the span of local interconnect or for connection of zone cells to global signals through the port cells. There are 4 horizontal and 4 vertical medium buses per core cell. Medium bus connectivity to core cells is sparse to minimize loading and limit core cell input multiplexer size. This connectivity is arranged so that a tile can be fully connected to the 16 medium buses which cross it.

zonal and global resources (Figure 2-13). All 4 medium buses, 4 global buses and the x bus in a given row or column connect to the port cell.

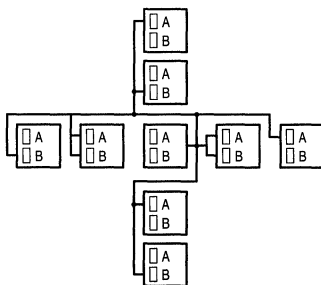


Figure 2-11. Local Interconnect

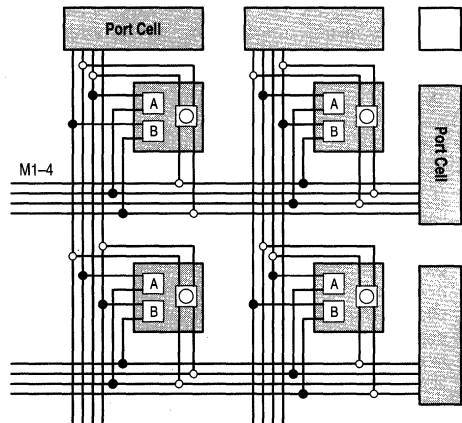


Figure 2-12. Medium Interconnect

Port Cells

At zone edges, port cells provide a bridge between global resources and zonal resources. Port cells transport signals into and out of a zone and are the only interface between

Port cells also provide connections to 4 of the 8 low skew clock distribution lines which span the device. Port cells also provide global to x bus access and serve as a pathway for zonal wired OR buses to connect to global busses.



2

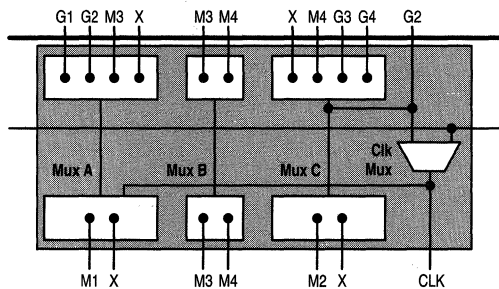


Figure 2-13. Vertical Port Cells

Global Interconnect

Global interconnect consists of global buses, x buses, interquadrant switches. There are 4 horizontal and 4 vertical global buses passing over each core cell. All Global buses only connect to the port cells, I/O cells and interquadrant switches. Global buses span a quadrant and are used to interconnect the zones within the quadrant together. Between quadrants, interquadrant switches connect two global buses together to form a device level connection.

Each core cell contains a x-bus switch (Figure 2-16) which is independent of cell logic or interconnect functions. A single vertical and a single horizontal x bus passes over each core cell and connects to this switch. Each x bus connects to all the core cells in a single zone column or row and terminates at the port cells on opposite edges of the zone. Each x bus has 10 connections inside the zone and 2 port cell connections. Port cell connections are used to make x to global, x to x and medium to x connections. Medium to x connections are used to hop over a single zone X buses are used to facilitate 90° global bus turns and provide a means for global bus fanout.

High Fan Out

As mentioned previously, the routing resources of the MPA are fully buffered. There is no reason for the designer to concern himself with loading effects of high fan out net. However, high fan out nets can have an undesirable impact on routing resource consumption. Using only local routing, a single driver could under the most ideal conditions drive only 8 local neighbors. In real world designs however, each of the destinations of a high fan out net has its own downstream circuitry associated with it; there is a vanishingly low probability that they will be placed in the 8 local adjacent locations. For fan outs greater than 8, exclusive local routing is impossible, and both medium and global routes will be used to complete the net. If the fan out is large enough, and the circuitry placed sufficiently far apart in the array, routing resource consumption may become problematic.

The primary clock and reset distribution network may be used to route high fan out signals. Driving the high fan out net internally with an ACLK or ARST buffer, or externally with an IPCLK or IPRST buffer will put the signal on one of the 8 global Clk/Rst distribution lines. The routing congestion can thus be solved, but at the expense of reducing the clock and reset routing solution space. Do not route nets to I/O (other than Clk/Rst) on the primary clock network. There is no mechanism for completing such a route on the MPA devices.

For software versions 2.4 and later, ACLK and ARST insertions for high fanout nets will be automatic.

Delays in Routing

Both PCB and older ASIC designers share the mind set that delay through a multi-level logic path is principally a function of "gate delay". In the ASIC world, routing paths are as short as possible and do not pass through multiple levels of pass gates, muxes, and buffers. Similarly, a PCB trace is a simple and hopefully short run of metal, with most of the "gate" delay happening as a function of package input and output delays. A "logical" net in an FPGA however may be a series of several different electrical nodes, each being separated by a mux or switch of some type. The consequence of this is that "routing delays" not gate delays are the first order factor determining the resultant circuit's speed.

Empirical analysis of several hundred sample designs suggests that a multiplication factor of 2.4 can be applied to the sum of a path's gate delays to come up with a very rough estimate of what the post autolayout total path delay might be. There are many factors that influence that actual number, so please consider this only as a very crude estimate.

S-R Flops, Avoid the Temptation

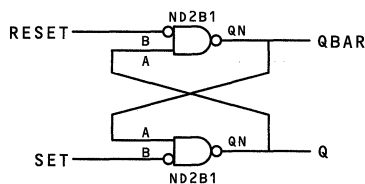


Figure 2-14. A Classic S-R Flop; An Accident Waiting to Happen

The above construction of an asynchronous S-R flip flop is familiar to all, but should be so for its unfavorable characteristics. Remember that routing delay in an FPGA is the highest order term in delay equation. In the above construction, the (active high) SET pulse width must be greater than the ND2B1 propagation delay plus Q to A routing plus another ND2B1 delay plus QBAR to A routing



delay. Without a detailed analysis of the post autolayout path delays, the pulse width specification can not be known. The same holds for the RESET pulse width. A new autolayout run on the same design may alter these path lengths considerably. Additionally this sort of asynchronous feedback loop will generally cause back annotation, simulation and timing analysis tools trouble.

Avoid asynchronous design.

Delay Lines, Avoid the Temptation

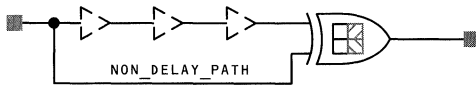


Figure 2-15. A Delay Line for Turning Edges into Pulses, a Dangerous Proposal

Remember that in an FPGA routing is not just a piece of wire. Routing is comprised of wire, muxes and pass gates. In the above example, the intent is to turn a rising or falling input edge into an output pulse. The assumption is that the "NON_DELAY_PATH" will have a shorter delay than the "delay line" formed by the series of BUFF elements. Again, the MPA design software does not guarantee minimum delays and so it is possible that the an autolayout run might result in the NON_DELAY_PATH to have a delay significantly close the delay line path. The circuit may not work.

Avoid any design habit that makes assumptions about minimum delays, even for just plain routes.

I/O Cell Connections and Peripheral Bus

I/O cells are a pathway between array and bonding pads. Global buses, x buses and adjacent zone medium buses can be connected to I/O cells at quadrant edges. Each I/O cells is directly connected to the adjacent bonding pad.

A specialized bus, called the peripheral bus, resides in the I/O cell – quadrant interface (Figure 2-4 on page 2-5). The peripheral bus comprises 8 lines which are interrupted at device corners by a peripheral bus switch similar to the interquadrant switch. This switch joins peripheral bus segments to create connections spanning more than a single device edge. Peripheral buses carry I/O control signals common to two or more I/O cells such as a latch enable or tristate control signal. The I/O cells can also drive these buses with an open drain device. When combined with programmable pullups located in the corners of the device, the peripheral bus can be used to form wide gates for address decoding (Figure 2-17).

Use the P-Bus to route enable signals

Whenever an enable signal goes to more than one I/O cell, it is recommended that the designer employ the P-Bus (by inserting and APBUF).

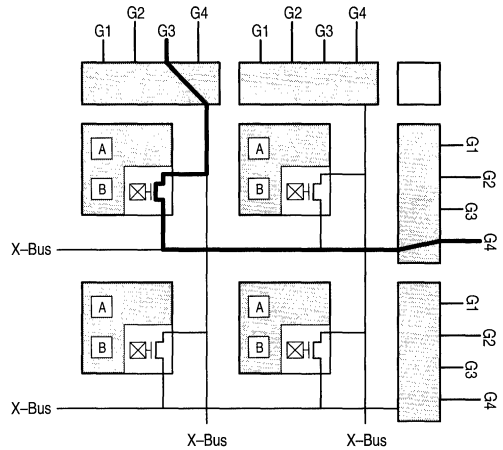


Figure 2-16. Global Bus Turn Using the X-Bus

Wired OR Nets

Wired OR nets are constructed using type 1 core cells. When the type 1 secondary function is enabled, the NAND drives an open drain device directly connected to a special bus shared by all the type 1 cells in the same zone row. This bus, the zone wired OR bus, terminates in the port cell and has a single, dedicated, pullup. When this bus is used, the port cell wired OR to global bus connection and the global bus pullup located near the interquadrant switch are enabled. These resources can be used to map 3-state buses onto the MPA1000 device.

Clock Distribution

Clock distribution is implemented through a dedicated, low skew, network consisting of; 8 dedicated clock input lines connected to 2 I/O cells on each device edge, a central clock buffer, a distribution comb structure, zone corner clock selection cells and the zone port cells along the top of each zone. The zone corner cell selects 2 of the 8 lines for zone clocks and 2 of the 8 lines for zone reset (Figure 2-18). Zone registers are connected to these clock and reset signals through the top row of port cells. The comb extends into the I/O cells via a similar clock selection cell attached to each group of 5 I/O cells. This group is called an I/O zone. All 8 clock lines can be driven from the I/O bonding pad or the array. The distribution network is balanced and has a skew of < 1ns between any two register clock inputs.



2

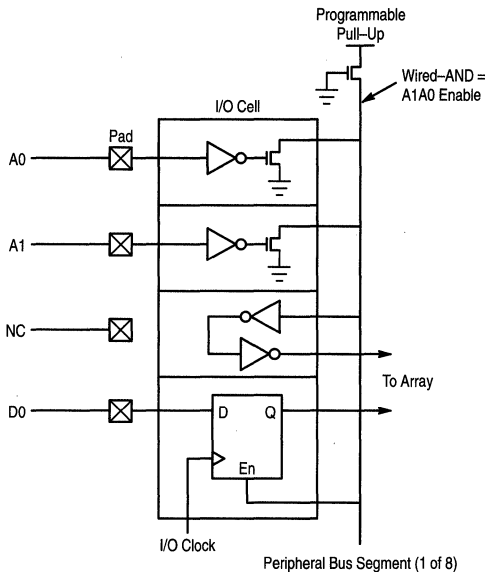


Figure 2-17. Using the Peripheral Bus for Address Decoding

Connections in the port cells allow the clock network to drive zone logic in addition to the register clock and reset inputs. Unused clock lines can be used for efficient distribution of any high fanout signal. If there are more clocks in the design than clock resources, the MPA design system automatically constructs a comb from global buses to generate a secondary clock network with a skew of < 3ns. Secondary clock construction is facilitated by a port cell

connection which provides non-clock network access to zone register clock and reset inputs.

Secondary Clock Networks Consume Routing Resources

The MPA easily handles a fair number of secondary clock networks, but networks with large numbers of Clk/Rst loads are more efficiently accommodated by moving onto the primary Clock Distribution Network using ACLK or ARST buffers mentioned previously.

Tertiary clock (reset) networks are identified by the autolayout software as any net driving four or fewer clock (reset) inputs not on the primary clock distribution network. There is no skew guarantee on these tertiary clock nets; they are routed on normal resources.

Too Many Clocks

The MPA is best suited for designs with a few primary clocks, but multiple clocks are supported. The problem with tertiary and especially secondary clock networks is that they consume a fair amount of routing resources. An otherwise easy to fit design may not be routable once multiple secondary clocks are accommodated.

Gated Clocks, Avoid When Possible

Inserting anything but an INV in a clock path will result in the clock being pulled off the primary clock network and placed either secondary or tertiary routing (depending on the number of clock loads downstream of the inserted gate). As mentioned above, this tends to spread the resulting layout out a bit more and consequently can slow things down some. If a gated clock is desired, try instead using register elements with clock enables.



2

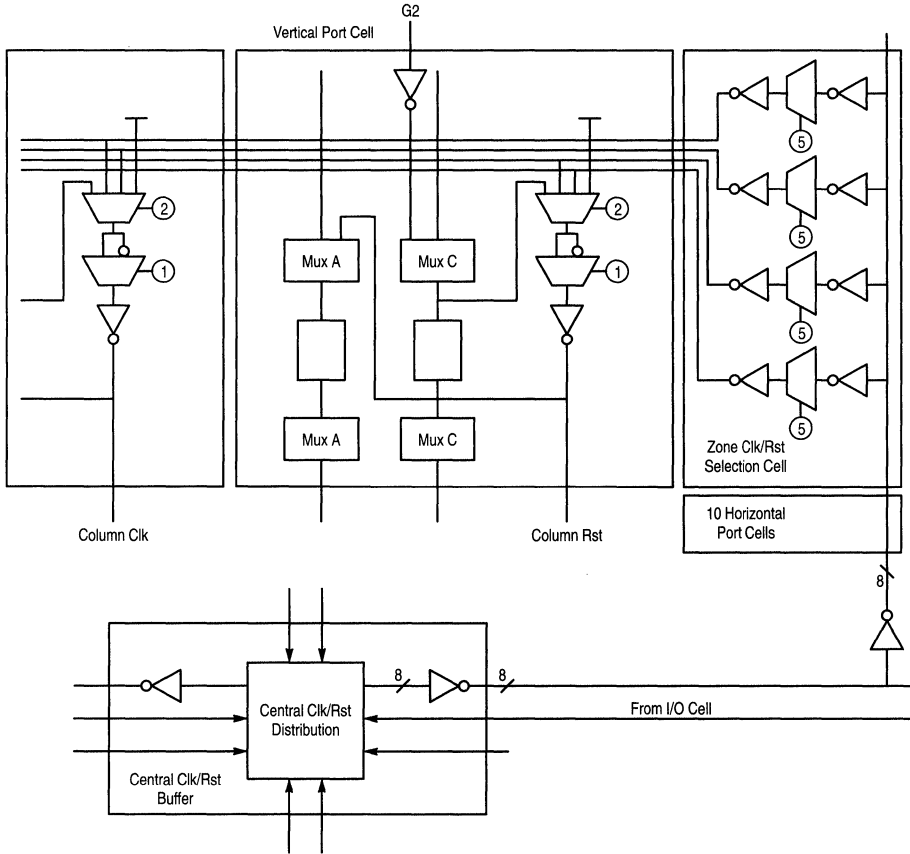


Figure 2-18. Clock Distribution Network Connectivity



ACLK & ARST Consume Clk/Rst I/O Sites

Each ACLK and ARST buffer used resides in one of the 8 clock pad locations. Using an ACLK or ARST consumes this pad location such that it is no longer available to use as an I/O site. The designer is allowed a total of 8 ACLK, ARST, IPCLK, IPRST cells in his design.

I/O Cells Can Only Be Clocked From the Primary Clock Distribution Network

Clocking I/O macros via secondary or tertiary clocks is prohibited. Reset is however permitted to be sourced from the array or Peripheral Bus (P-Bus).

Clock Sense Selection is Made in the Vertical Port Cell

All flops within a column will have the same clock and reset (or will lie unused).

Do Not Use the Primary Clk/Rst Distribution Network to Route Clock Enable Signals

Referring to Figure 2-18 on page 2-13, note that a clock is paired with a reset and brought down to all 5 of the Type 3 cells within a column. If the associated clock enable (if used) is also on the primary clock network, there would be no efficient route available to get it down to the target flops. Do not use the Primary Clock Distribution Network to route clock enables. (Do use it for "Latch Enable" signals.)

2



MPA1000 Device Configuration

Configuration Overview

MPA1000 devices have an SRAM configuration memory. Configuration memory contents completely define MPA device function. The MPA1000 design system generates configurations from completed layouts. On chip control logic loads configurations in one of four modes automatically on power up or under external control. MPA1000 devices have a very rapid configuration load cycle, infinite reload and are in system reconfigurable. The configuration modes are; Boot From ROM (BFR1:3) and microprocessor peripheral or MICRO Mode. In either mode, multiple devices can be daisy chained to form a large programmable subsystem.

In all BFR modes, the MPA device controls configuration and loads from either a byte wide or serial memory. In BFR mode 1 (Figure 2–20), the device generates 18 bits of address and reads 8 bits of configuration data. MPA devices generate 18 bits of address or 262K bytes (e.g., 17 MPA1036 devices). If a larger address range is required, BFR mode 3 (Figure 2–26 on page 2–21) can be used. In BFR mode 3 an external address generator is used to extend the address space. BFR mode 2 is a special case of BFR 3. In this case the address generator is resident in the external serial EPROM and data is presented to the device 1 bit at a time. The MPA design system download POD and the MPA17000 serial EPROMs are used with this mode.

In MICRO Mode, the MPA1000 device becomes an 8 bit peripheral slave device. A microcontroller or microprocessor controls the configuration process. MICRO Mode provides more control over configuration and user mode device behavior than other modes. For example MICRO Mode can be used to both write and read configuration memory.

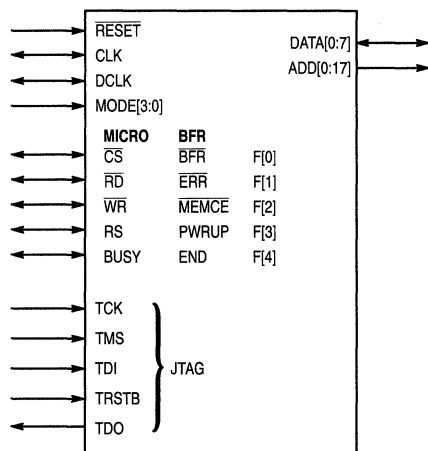


Figure 2–19. Configuration Interface Signals

Configuration information generated by the MPA Design System includes Error Check Bytes (ECBs). ECBs are used to detect configuration data corruption while configurations are loaded into the device. The configuration process halts and error status is indicated if an ECB mismatch is detected anytime during the configuration process. ECB checks insure the integrity of configuration data and protect MPA devices from possible damage.

Depending on the selected configuration mode, some user I/O pins become unavailable for post configuration use. These pins are listed as “dedicated” in Table 2–7 on page 2–25 and Table 2–5 on page 2–17. The system level interface for MPA configuration is shown in Figure 2–19. Note that the meaning of the F[4:0] pins is mode specific, refer to Table 2–7 and Table 2–5 for detailed signal descriptions.

2

Table 2–4. MODE[3:0] Pin Programming

Mode Bits				Description
[3]	[2]	[1]	[0]	
X	X	0	0	MICRO Mode — Micro-processor/controller interface circuitry with parallel (byte wide) data.
X	X	0	1	BFR Mode (1) — Boot From ROM, byte wide data. MPA generates ROM addresses.
X	X	1	0	BFR Mode (2) — Boot From ROM, serial data. (Low pin count serial EPROM generates own addresses.)
X	X	1	1	BFR Mode (3) — Boot From ROM, byte wide data. MPA does not generate ROM addresses.
X	1	X	X	Use external clock for configuration.
1	X	X	X	Enable JTAG circuitry and pins.

Configuration Clock

The MPA1000 device has an internal oscillator. The internal configuration clock is derived from the oscillator and is presented at the CLK pin when MODE[2] is low. When MODE[2] is high, the internal clock is disconnected from the oscillator and an external clock must be presented on the CLK pin to configure the device.

The configuration clock drives the configuration logic and its associated state machine. If using an external configuration clock, it is necessary to provide it always to ensure RESET, BFR and PWRUP signal transitions are detected and handled in the expected fashion by the configuration logic.

Bootstrap Voltage

Signal pathways in the MPA 1000 device are controlled with n-channel transistors. The gates of these transistors are connected to individual SRAM configuration memory cells. To pass a rail to rail signal through these transistors during user operation, the gate voltage must be elevated above V_{DD}



MPA1000 Product Description

to compensate for transistor threshold and body effect voltage drops. MPA1000 devices contain a charge pump to generate this elevated voltage, called the bootstrap voltage. The charge pump is connected to the supply line of each SRAM cell and is driven by the internal oscillator.

Since configuration memory is generally not dynamically changing during user operation, the charge pump must only supply small leakage current losses and is not designed to supply sufficient current for SRAM write operations. During configuration, the charge pump (bootstrap) is internally disabled by shunting the SRAM supply to V_{DD} through a large p-channel device. In order for the charge pump to operate properly, the internal oscillator as well as the bootstrap circuitry must be enabled. In MICRO Mode, the processor has control over these functions. In BFR modes, the on chip configuration controller insures proper sequencing of these controls.

The MPA1000 device is only guaranteed to function properly with bootstrap enabled. The internal oscillator must be running and bootstrap should be activated 100 μ s before user inputs or outputs are enabled. If dynamic configuration modification is desired, the bootstrap voltage can be supplied externally on the V_{pp} pin and MICRO Mode can be used to disable bootstrap by shutting off the on-board oscillator. The bootstrap voltage should be $V_{DD} + 1.5V$. At no time should V_{pp} exceed 6.5V.

JTAG

The MPA1000 device contains dedicated JTAG IEEE 1149.1 boundary scan circuitry. JTAG can be used on configured devices. JTAG is enabled any time $MODE[3]$ is raised. When $MODE[3]$ is high, 5 user I/O pins become JTAG controls and user mode operation of those pins is interrupted. Since the TAP controller can take control of all device pins, care must be used to prevent the TAP controller from interfering with device user mode or configuration operation.

Boot From ROM (BFR) Modes

In BFR modes, the MPA device controls device configuration and assumes a memory-processor interface to

the configuration store. The MPA device either asserts addresses directly (internal address generation) or issues address reset and increment pulses (external address generation). Data is read either serially or 8 bits at a time. Table 2-5 describes BFR interface signal operation. $ADD[17:0]$ are only used in BFR mode 1. $DATA[7:1]$ are not used in BFR mode 2 (serial data).

A BFR load sequence is initiated by: a falling edge of \overline{BFR} , device power up or a rising edge of \overline{RESET} . \overline{MEMCE} falls to indicate the start of a configuration load sequence. On subsequent alternate rising edges of CLK, the data bus value is latched. The configuration process terminates when a complete configuration is successfully loaded and \overline{END} is asserted or when a configuration error is detected and \overline{ERR} is asserted. After \overline{END} is asserted, the device will begin user mode operation 3 clocks after \overline{PWRUP} is asserted or 3 clocks after \overline{END} if \overline{PWRUP} was already high. All configuration timing is synchronous with the internal or externally supplied configuration clock. Figure 2-22 describes BFR sequence timing details.

All BFR sequences begin with an internal device reset sequence where the entire configuration memory is reset. The duration of this sequence depends on the size of the MPA device being configured. A falling \overline{MEMCE} edge indicates configuration commencement and data loads begin after 2 subsequent configuration clocks. The first positive edge of DCLK signals the external address generator to increment the byte or bit address. Prior to \overline{MEMCE} assertion, DCLK is tristated.

The duration of the configuration process is also dependent on device size. Configuration duration can be estimated for BFR 1,3 by dividing the total number of configuration bytes by 1/2 the configuration clock frequency. For example, the MPA1036 device has 139 rows of 105 bytes including the ECB or 14,595 bytes. If the configuration clock is 2MHz, configuration will take approximately 15ms. If BFR 2 is used, the configuration process will take approximately 8 times longer. See "Device Configuration Memory Organization" on page 2-33 for device specific configuration memory sizes.

2



Table 2-5. BFR Mode Configuration Control Pins

Pin Name	BFR	I/O		Description
MODE[3:0]	MODE[3:0]	I	Dedicated*	Configuration mode
RESET	RESET	I	Dedicated	Configuration reset — Clear configuration memory. Configure when released.
CLK	CLK	I/O	Dedicated	Configuration clock — If MODE[2] is low, the internal configuration clock is presented. If MODE[2] is high, an external clock must be supplied.
F0	BFR	I	Dedicated	BFR initiate — A falling edge starts a reset and configure sequence.
F1	ERR	O	Dedicated	Error — Configuration checksum (ECB) or incorrect device ID error. Open drain output
F2	MEMCE	O	Dedicated	Memory Enable — Active low during configuration sequence.
F3	PWRUP	I	Dedicated	Power up — After configuration complete; enable bootstrap, enable user inputs, enable user outputs. Often simply tied to V _{DD} .
F4	END	O	Dedicated	Configuration completed — Asserted when a configuration has been successfully loaded into the device.
DCLK	DCLK	I/O	Dedicated	Data clock — Each output pulse indicates current data bus value has been latched and data address should increment. Becomes an input after configuration completes.
DATA[7:0]	DATA[7:0]	I	User/Data	Data port
ADD[17:0]	ADD[17:0]	O	User/Address	Address output — If internal address generation is selected. (BFR Mode 1)
JTAG[4:0]		I/O	User/JTAG	JTAG pins — Active when MODE[3] is asserted.

* Dedicated — Pins used for configuration. Not available for user I/O.

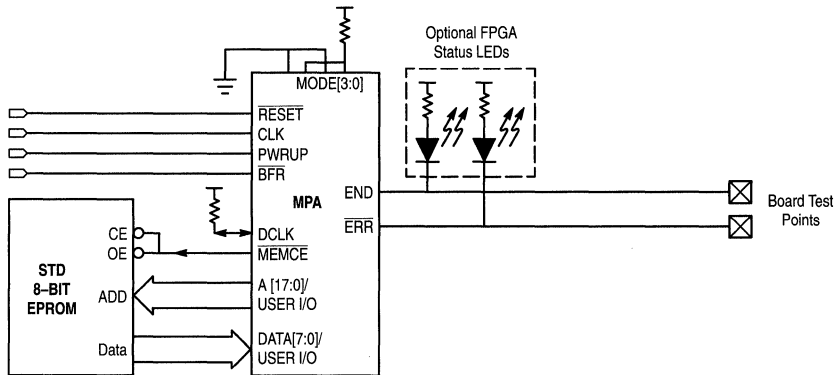


Figure 2-20. BFR Mode 1: 8-Bit Data, Internal Address, External Clock

BFR Mode 1 Operation: 8 bit data, Internal Address Generation

In BFR 1, MPA configuration logic asserts an 18 bit address and reads data 8 bits at a time as shown in Figure 2-20 on page 2-17. A paging scheme could also be used where additional upper address bits were provided by an external page register. Multiple configurations could be accessed by writing the page register, asserting $\overline{\text{BFR}}$, and self loading the referenced configuration.

ADD[17:0] are tristated during device reset, asserted during configuration and released for user mode operation. DCLK is tristated until 1 clock prior to MEMCE assertion. The

first address is asserted coincident with the falling edge of MEMCE and the data bus is latched 2 configuration clocks later. The internal address counter is incremented on each positive DCLK edge (Figure 2-21). This process proceeds until an entire row of configuration data is loaded into the internal row data register and the ECB is verified. ADD[17:0] (current address) and DCLK (=1) hold while the internal write cycle takes place. Start Access (SA) marks the beginning of the write cycle and End Access (EA) marks write completion (Figure 2-27). After the write completes, the address presentation and data latching process resumes. When the entire device configuration is loaded, END is asserted, DCLK



MPA1000 Product Description

is tri-stated and 2 clocks later user inputs are enabled and MEMCE is deasserted. One additional clock and user outputs are enabled and user mode operation commences. If the written ECB does not match the internally calculated value, ERR is asserted 2 clocks after the ECB is written. Once ERR is asserted, the configuration process halts and cannot be restarted until a new configuration process is initiated using BFR, RESET or a power down. When END is

asserted, DCLK becomes an input and the internal address counter remains active until PWRUP is asserted. Figure 2-30 shows how this can be used in a multiple device subsystem. Because DCLK becomes an input, it must be tied high with a weak pullup when used in a single device configuration (Figure 2-20) to prevent a floating input condition.

2

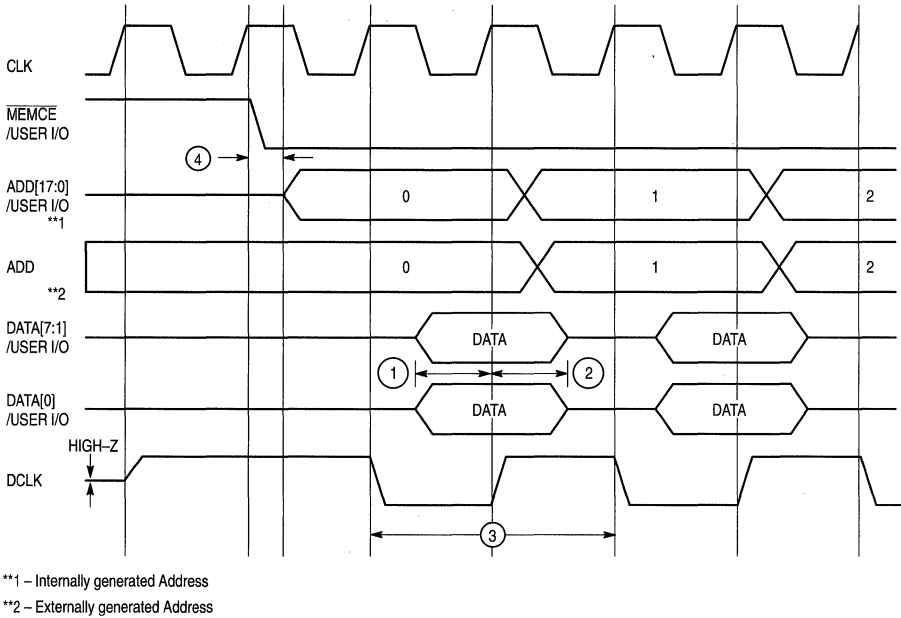
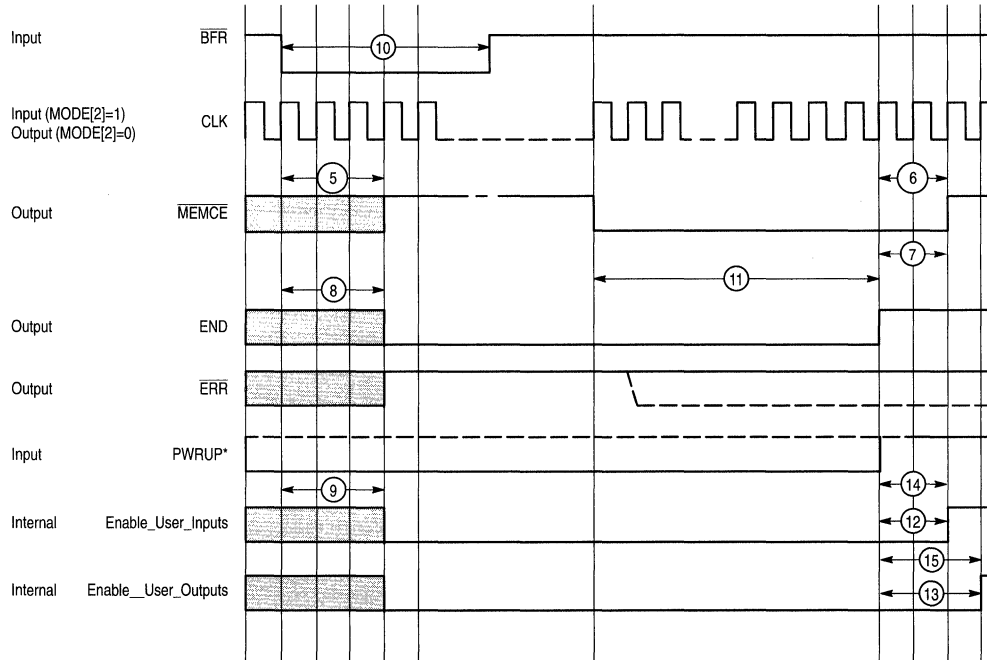


Figure 2-21. BFR Data Access Detail

Number	Characteristic	Min	Max	Unit	Notes
1	Data Setup to DCLK	20		ns	
2	Data Hold after DCLK	0		ns	
3	DCLK Period (When Active)	2	2	CLK	
4	CLK to Address Valid (Internal Generator)	15		ns	





* PWRUP can be, and usually is, tied to V_{DD} internally.

Figure 2–22. BFR Sequence

Number	Characteristic	Period	Unit	Notes
5	BFR Low to $\overline{\text{MEMCE}}$ High	3	CLK	If BFR reasserts during a boot
6	END High to $\overline{\text{MEMCE}}$ High	2	CLK	
7	PWRUP to $\overline{\text{MEMCE}}$ High	2	CLK	
8	BFR Low to END Low	3	CLK	Note 3.
9	BFR Low to Internal Disable	3	CLK	Note 3.
10	BFR Pulse Width	50	ns	Minimum
11	Configuration Sequence Duration			Configuration sequence dependent on device size
12	END to Enable User Inputs	2	CLK	If PWRUP asserted, Note 4.
13	END to Enable User Outputs	3	CLK	If PWRUP asserted, Note 4.
14	PWRUP to Enable User Inputs	2	CLK	Note 5.
15	PWRUP to Enable User Outputs	3	CLK	Note 5.

3. BFR is usually an asynchronous input, 4 CLKs assumes T_{SO_BFR} is met.

4. PWRUP can be, and usually is, tied to V_{DD} .

5. PWRUP may be an asynchronous signal, 2,3 CLK, assumes T_{SO_PWRUP} is met.



A Sample BFR Mode 2 Load Sequence

The most common boot configuration for the MPA is the BFR Mode 2, using a serial boot (E)EPROM. The timing overview for such a boot load is given in Figure 2-23 and Figure 2-24, with timing notes in Table 2-6.

In this example the CLK signal can either be sourced by the MPA or generated externally and received by the MPA (according to the state of the MODE[2] pin). BFR is usually asynchronous, Figure 2-23 assumes the falling edge of BFR meets the set-up requirement with respect to the rising edge of the CLK signal. Three CLKs later The END signal de-asserts and a reset sequence begins. The length of the

reset sequence is a function of the array type as shown in Table 2-6. As the reset sequence ends DCLK (connected to the EPROMS clock input) goes high, then MEMCE asserts (connected to the EPROM's RESET/OE pin). The first bit of configuration data will appear at the EPROM's data pin after this falling MEMCE. Data is latched into the MPA as DCLK is raised. The next rising edge of DCLK causes the EPROM to shift out the second configuration bit, and so on.

The internal configuration SRAM of the MPA is loaded up one row at a time. The number and width of the rows varies by array type. After a row's worth of data is read in to a configuration shift register, the MPA holds DCLK high for 12

2

Table 2-6. BFR Mode 1 Sequence Timing for All MPA Family Members

Number	Characteristics	CLKs	Notes
1	BFR Low to MEMCE Low MPA1016 MPA1036 MPA1064 MPA1100	971 1411 1851 2291	Internal SRAM Reset Sequence =21+(10*95), 95 SRAM Rows =21+(10*139), 139 SRAM Rows =21+(10*183), 183 SRAM Rows =21+(10*227), 227 SRAM Rows
2	Low to DCLK Hold Off MPA1016 MPA1036 MPA1064 MPA1100	1232 1760 2288 2800	Shifting in ID and first row of SRAM data =80+(2*576), ID & data type then 576 bits/row =80+(2*840), ID & data type then 840 bits/row =80+(2*1104), ID & data type then 1104 bits/row =80+(2*1360), ID & data type then 1360 bits/row
3	Internal SRAM Row Load	12	All devices, every row
4	Subsequent Row Sequence MPA1016 MPA1036 MPA1064 MPA1100	1163 1691 2219 2731	Shifting in row data =12+(2*576)-1, 576 bits / row =12+(2*840)-1, 840 bits / row =12+(2*1104)-1, 1104 bits / row =12+(2*1360)-1, 1360 bits / row
	BFR Low to User Outputs Enabled MPA1016 MPA1036 MPA1064 MPA1100	111,540 236,544 408,012 622,312	The complete BFR Sequence =971+1232+12+(1163*94)+3, reset+1st_row+rows+I/O =1411+1760+12+(1691*138)+3 =1851+2288+12+(2219*182)+3 =2291+2800+12+(2731*226)+3

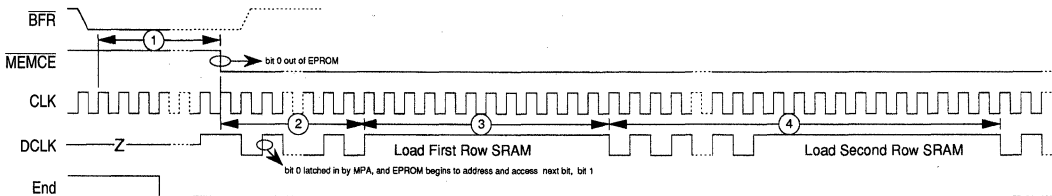


Figure 2-23. Start of a Typical Serial Boot From ROM Sequence

(Clock may be internal or external. BFR is an external asynchronous signal, T_{SU_BFR} is assumed to have been met.)

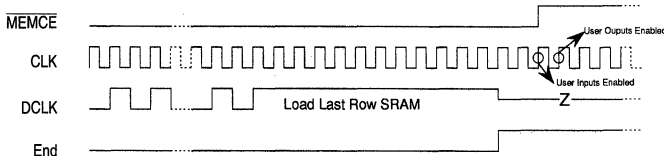


Figure 2-24. Completion of a Serial Boot From ROM Sequence



CLK cycles and transfers this data to the internal SRAM row. Provided no device ID or check-sum errors are detected, the load will continue in this row by row fashion until complete. As the last row of SRAM is written, the END signal asserts then user I/O is enabled as shown.

BFR Mode 2 Operation: 1 bit (serial) data, External Address Generation

BFR mode 2 is used for connecting MPA devices to a serial configuration memory. The MPA device provides an address increment signal (DCLK) rather than an internally generated address as in BFR mode 1. Low pin count serial memories, like the MPA17128, contain address generation logic which responds to a single increment signal. Addressing is sequential starting at zero. Multiple MPA17000 devices can be daisy chained if a larger memory is required (See MPA17128 data sheet on page 1–6). Serial memories are programmed (written) in the opposite bit order from the way they are read. The MPA Design System configuration generation program will generate a correctly formatted PROM programming file by reflecting each configuration byte before writing the file.

MEMCE is high until configuration commences. MEMCE is connected to the RST/OE pin of the MPA17128 holding its internal address counter at 0 and its outputs tristated. The falling edge of MEMCE enables the memory data pin and 2

clocks later a data bit is latched into the MPA1000 device. The first rising edge of DCLK signals the memory to index its address register and present the next locations data bit. Each time 8 bits are accumulated by the MPA1000, they are written to the internal row data register. As in BFR 1, this process proceeds until a complete row is loaded and the ECB is verified. DCLK holds while the row data register is written to the current configuration memory row. After the write completes, additional bits are loaded until the next row boundary is reached. Configuration completion and error indications are identical to BFR 1.

BFR Mode 3 Operation: 8 bit data, External Address Generation

BFR mode 3 is identical to BFR mode 2 except that 8 bits of data are loaded rather than one. An external address generator is used and responds to the MPA address increment signal (DCLK). BFR mode 3 is useful because BFR 1 requires 18 user I/O signals (ADD[17:0]) during configuration. While these are subsequently released, it does impose restrictions on surrounding circuitry complicating overall system design. Secondly in applications requiring rapid configuration of a large number of MPA devices or many alternate configurations, the MPA 18 bit address space may not be large enough and an external counter (address generator) would be required anyway.

2

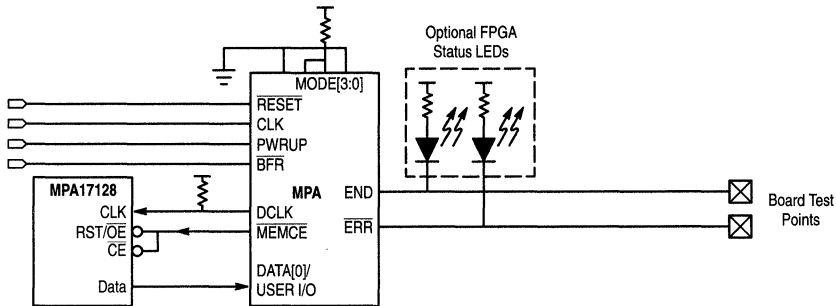


Figure 2–25. BFR Mode 2: 1–Bit (Serial) Data, External Address, External Clock

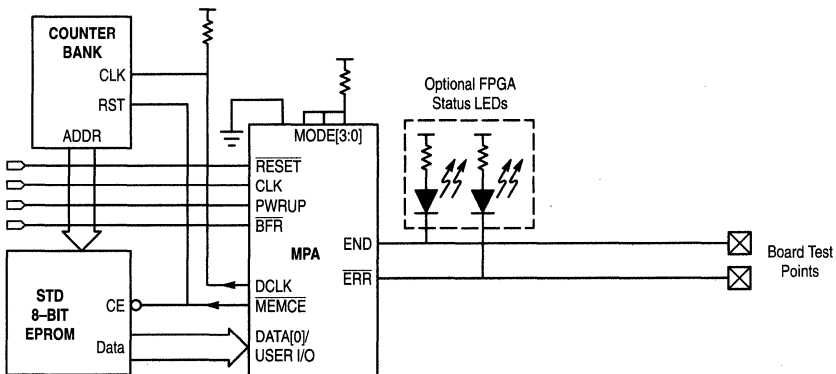


Figure 2–26. BFR Mode 3: 8–Bit Data, External Address, External Clock



MPA1000 Product Description

2

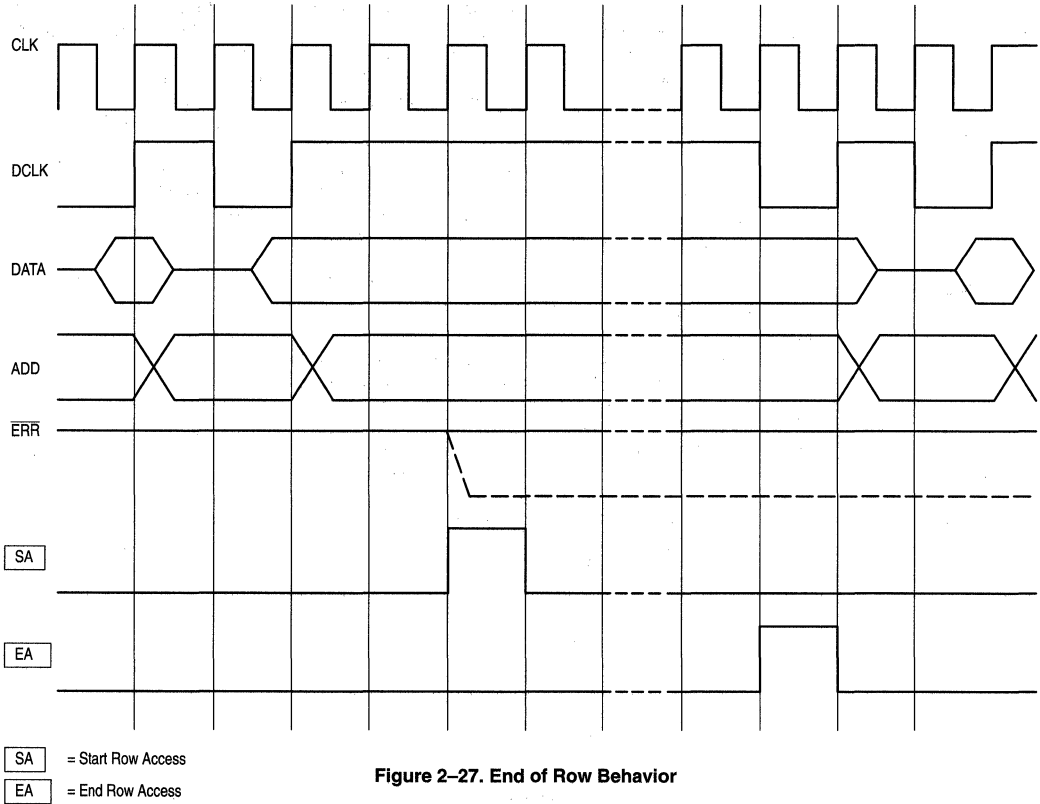


Figure 2-27. End of Row Behavior

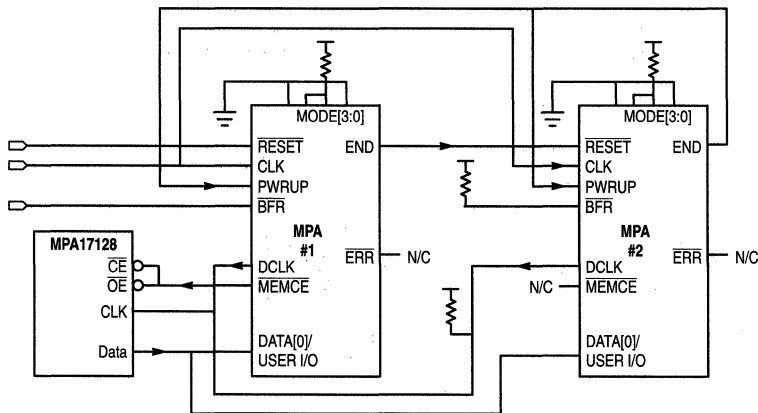


Figure 2-28. Multiple Device Subsystem: BFR2; Serial Data, External Address, External Clock



BFR Multiple Device Subsystems

If multiple devices are used together in BFR mode, the first device loads first and the END signal on each device is connected to the RESET pin of the next device. As an upstream device completes configuration, a configuration sequence is initiated on the next device. This daisy chain extends to the last device. This devices END is connected to the PWRUP pins of all subsystem MPA devices. All devices enter user mode when the last device successfully configures.

Care must be taken to insure proper operation. $\overline{\text{BFR}}$ on all but the first device must be tied high and the subsystems composite DCLK line must be pulled up to eliminate spurious clock signals as one device tristates DCLK and the next device asserts it. Figure 2-29 illustrates the control signal hand off.

When constructing a subsystem in which the first device asserts the 18 bit address (BFR mode 1), this device provides address generation for all devices in the subsystem. The DCLK pin of the first device becomes an input when it successfully configures and its internal counter remains active. Positive edges applied to this pin will increment the first devices internal address counter and present the resulting address on the first devices 18 bit address bus. Subsequent devices in this subsystem should use BFR mode 3 (external address, 8 bit data).

Examples of multiple device boot configurations are shown in Figure 2-28, Figure 2-30 and Figure 2-31. The last device's END signal is fed back into the first device's PWRUP pin. Holding PWRUP low, holds the MEMCE output low.

2

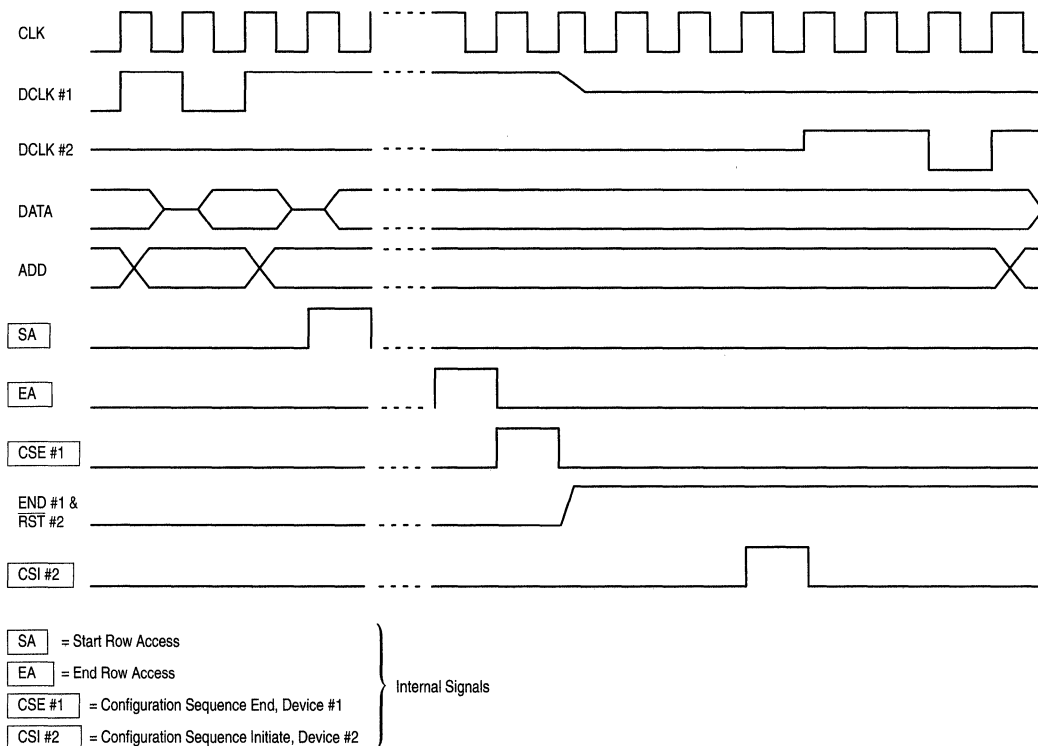


Figure 2-29. BFR Mode Daisy Chain Timing



2

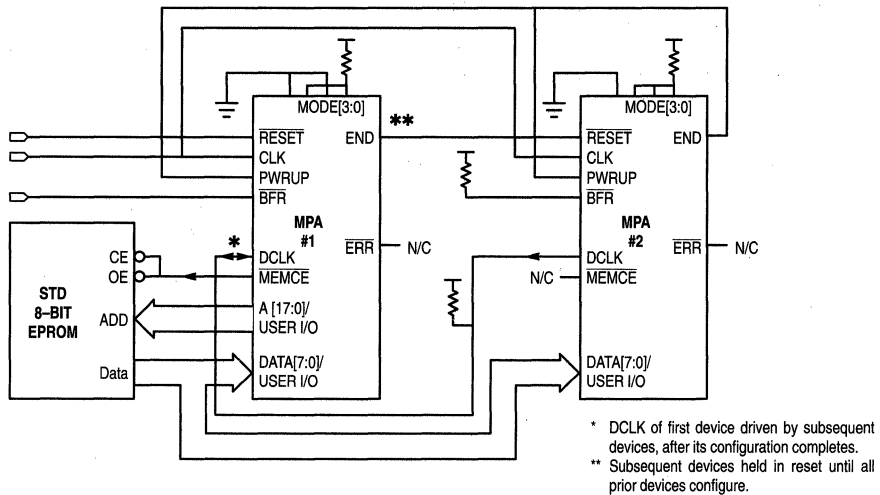


Figure 2-30. Multiple Device Subsystem: BFR1 and BFR3; 8 Bit Data, Internal Address, External Clock

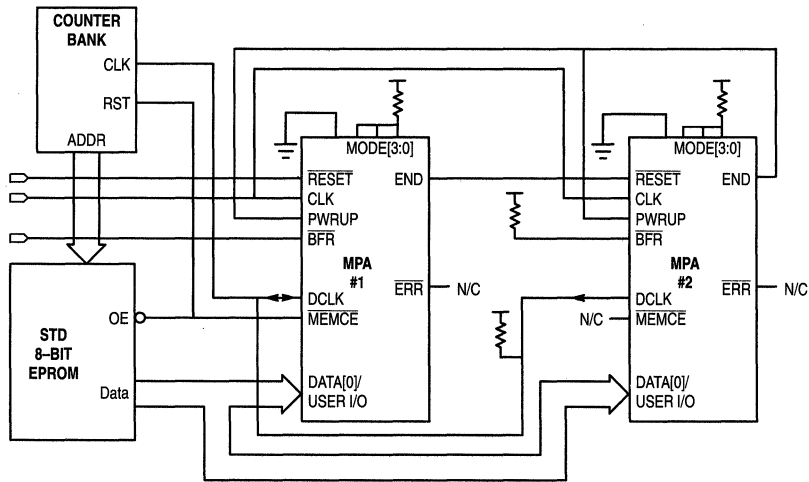


Figure 2-31. Multiple Device Subsystem: BFR3, 2; 8 Bit Data, External Address, External Clock



MICRO Mode

In MICRO Mode the MPA device behaves as an asynchronous microprocessor peripheral. Table 2-7 details MICRO Mode configuration pin function. A chip select (/CS) is derived from the processor address and enables a single MPA device. In a multiple device subsystem, a chip select for each MPA device is required. When a device is selected, the data bus is used to write commands, read status, write configuration data and read configuration data. There are two device configuration registers, the function register (RS=0) and the data/status register (RS=1). Configuration commands are written to the function register. Subsequent behavior is specific to the command issued and is documented in Table 2-8. The data register is either used to read device status, read device configuration data or write device configuration data. RS is normally connected to the least significant address line to map the function register to address A and the data/status register to address A+1.

Configuration data format information can be found in the

“Device Configuration Data Format” section on page 2-33. Configuration data is generated by the MPA design system configuration generator after a layout is complete.

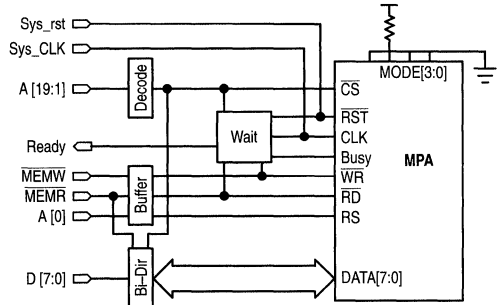


Figure 2-32. MICRO Mode: Single Device With External Clock and Wait State Insertion

2

Table 2-7. MICRO Mode Configuration Control Pins

Pin Name	Micro	I/O		Description
MODE[3:0]	MODE[3:0]	I	Dedicated*	Mode Pins
RESET	RESET	I	Dedicated	General configuration reset
CLK	CLK	I/O	Dedicated	Clock for configuration circuitry — If external clock is selected, pin is an input. If not selected internal configuration clock is used and output through this pin.
F0	CS	I	Dedicated	Chip select for device in MICRO Mode.
F1	RD	I	Dedicated	Micro read signal
F2	WR	I	Dedicated	Micro write signal
F3	RS	I	Dedicated	Register select — Two register locations are active: Function Register (RS = 0) and Data/Status Register (RS = 1).
F4	Busy	O	Dedicated	Busy signal — Active high when device is not ready to accept data, i.e. while device is resetting data in array or a data register to array transfer is taking place.
DATA[7:0]	DATA[7:0]	I/O	Dedicated	Micro data port — for configuration logic.
JTAG [4:0]	J [4:0]	I/O	User/JTAG	JTAG pins — JTAG or User I/O is selected by MODE[3].

* Dedicated — Pins used for configuration. Not available for user I/O.



Table 2–8. MICRO Mode Function Register (RS=0)

DATA					Function
7	6	5	4	[3:0]	
				0000	Normal operation — No function performed.
				0001	Reset Device — Entire device configuration is reset. BUSY is asserted until reset completes.
				0010	Load Configuration — After writing this command, an entire normal format device configuration is presented to the data register in 8 bit segments starting with the configuration header block. At any time during the loading process, a read from the data register will return status register contents. As complete rows including ECB are loaded, BUSY is temporarily asserted while row data is internally transferred from the internal data register to the currently addressed memory row. Once this write operation is complete, BUSY is deasserted and additional data can be written. Each time BUSY is deasserted, the status register should be checked for incorrect ID or row configuration data error(s). Once an error is detected, NO further write accesses to the data register will be accepted until the device is reset or another load configuration command is issued.
				0011	Reset Row — Indicates that the next data written to the data register will be a device row address. After the address is written, the contents of that configuration memory row are reset. BUSY is asserted after the address is written and deasserted when the operation is complete.
				0100	Load Row — The next data written to the data register consists of a row address followed by configuration data for that row including the terminating ECB. After the ECB is written, BUSY will be asserted during internal write and deasserted when the write completes. Reading the data register returns status register contents. The status register should be checked for row configuration data error(s). Once an error has been detected, NO further write accesses to the data register will be accepted until the device is reset or a load configuration command is issued.
				0101	Read Row — The next data written to the data register will be interpreted as a row address. After the row address is written, BUSY is asserted while row data is read into the internal data register. BUSY is deasserted when the transfer is completed. Subsequent successive reads from the data register will return row configuration data. No ECB is returned. The row data read back is in the same order as it is written, rightmost byte first.
				0110	Read Device ID — 4 subsequent reads from the data register return device ID. The most significant ID byte is read first. Refer to "configuration data format" for individual device ID values.
				0111	Bits [3:0] — Reserved pattern.
				1XXX	Bits [3:0] — Reserved pattern.
			1		User Outputs Enabled — Normally user outputs are enabled one or more clocks after user inputs are enabled to insure valid input values have propagated into the device.
		1			User Inputs Enabled — Normally user inputs are enabled after a configuration is successfully loaded into the device.
	1				Internal Oscillator Disabled — Normally always enabled. May be disabled if external clock and Vpp are user supplied. If internal configuration clock is used (Mode[2]=0), oscillator cannot be disabled. Internal oscillator drives a charge pump that generates bootstrap Vpp. If turned off, then back on, allow 100µs restart time.
1					Bootstrap Enabled (Vpp) — Should be enabled after configuration is completed and disabled during configuration. (Disable ties Vpp to VDD.) Only a few package types bond out Vpp to a pin. The Vpp pin can be used to monitor Vpp. The Vpp pin may be driven between VDD and 6.5V externally if Bootstrap is enabled and internal oscillator is disabled. Vpp is applied to the pass gate transistors inside the array, to ensure the lowest possible R _{DN} .

2

Table 2–9. MICRO Mode Data/Status Register (RS=1)

Bit Position								Function
[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]	
R	R	R	R	R			1	Incorrect Device ID.
R	R	R	R	R		1		Row configuration data error. ECB mismatch.
R	R	R	R	R	1			Busy signal asserted. Allows software handshaking if hardware wait states are not to be used.

R = Unspecified, reserved for factory use.



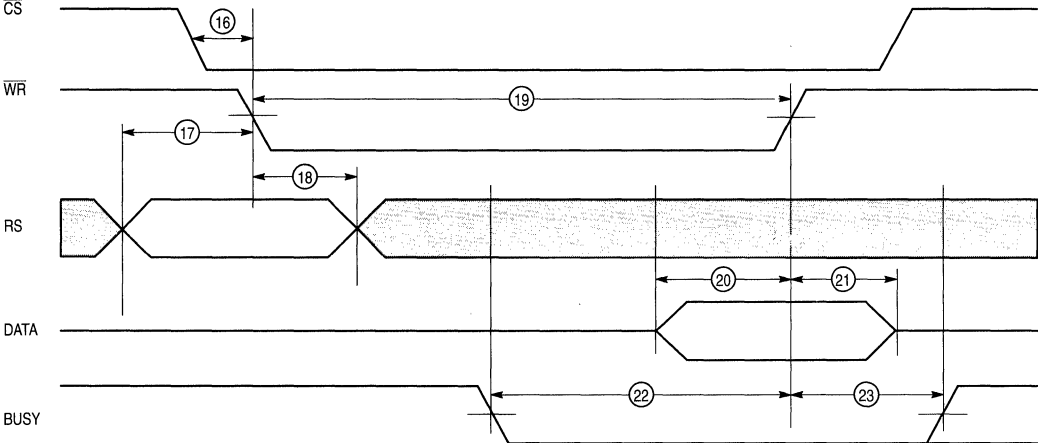


Figure 2-33. MICRO Mode External Timings (Write Cycle)

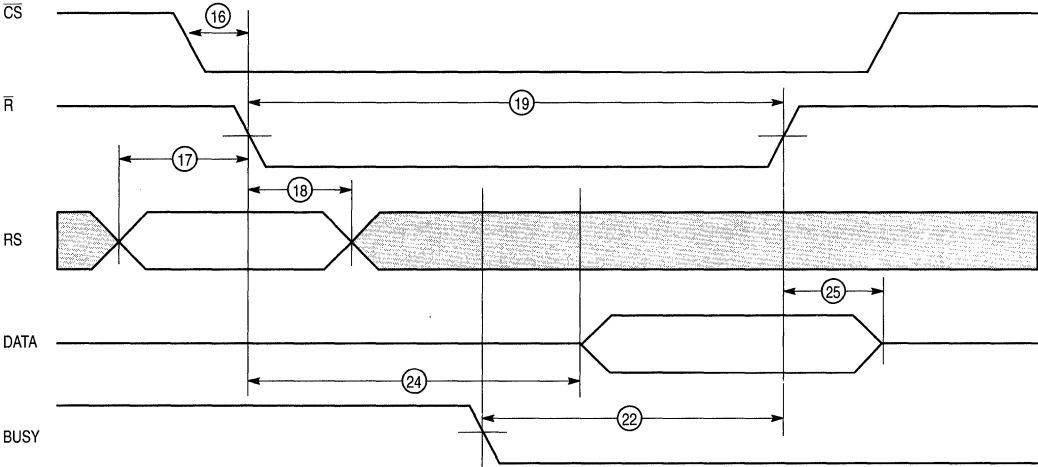


Figure 2-34. MICRO Mode External Timings (Read Cycle)

Number	Characteristic	Min	Max	Unit	Notes
16	CS Setup before Read/Write Falling Edge	10		ns	
17	RS Setup before Read/Write Falling Edge	10		ns	
18	RS Hold after Read/Write Falling Edge	10		ns	
19	Read/Write Pulse Width	50		ns	
20	Data Setup to End of Write	20		ns	
21	Data Hold after Write	10		ns	
22	Busy Inactive before End of Read/Write	50		ns	
23	Busy Active after Write	0	20	ns	
24	Data Access Time	20	40	ns	
25	Data Hold Time after Read	0	10	ns	

The configuration clock is still used to drive the MPA's internal configuration logic in MICRO mode. The length of BUSY is, therefore, a function of the configuration clock.



MICRO Mode Maximum Data Transfer Rate

The maximum MICRO Mode data transfer rate is governed by the R/W timing described in Figure 2-33 and Figure 2-34. The processor must only write data when BUSY is inactive. BUSY is only asserted when data cannot be accepted at the maximum rate. The specific behavior of BUSY for each MICRO Mode function is described in Table 2-8. When the device is powered up, an internal reset sequence is initiated and BUSY is asserted (see "Behavior During Power-On-Reset"). BUSY will be deasserted when the internal reset sequence completes. The processor can monitor BUSY directly or the status register can be read.

2

If processor R/W cycles are faster than the timing shown, external circuitry must be used to insert wait states. Figure 2-32 and Figure 2-39 show an application circuit consisting of one or more MPA devices and an optional wait state insertion block used to lengthen R/W timing based on CS, MEMW, MEMR, BUSY, and RESET using an externally provided clock.

Using MICRO Mode to Read Configuration SRAM

An interesting side benefit of using MICRO mode is the ability to go back to the MPA after the normal boot process completes and read back out the configuration SRAM. While under spec operating conditions, there is no possibility of configuration SRAM corruption. Some applications, however, may have out-of-spec operating conditions, such as extreme noise on power rails or rails subject to power dips. In such applications, system level health monitoring can be augmented using this SRAM read out feature. Normal MPA device operation is not disturbed during configuration SRAM reads.

Multiple Devices in MICRO Mode

If multiple devices are used in MICRO Mode, external

logic is required to individually address each MPA device using CS (chip select) signals. After configuration, the processor must write bootstrap enable, enable inputs and enable outputs commands to each device. A subsystem BUSY signal can be derived by OR-ing the BUSY signals from each individual device. Refer to Figure 2-39.

Internal Clock Specification

The internal ring oscillator is a clock source with possible frequencies ranging from 10MHz to 40Mhz. This variation is expected and does not present a problem for proper charge pump or configuration operation. The internal configuration clock is derived by dividing the oscillator frequency by 8. The internal configuration clock can be used for user mode operation and is presented on the CLK pin when MODE[2] is low.

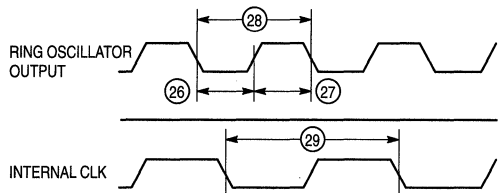


Figure 2-35. Internal Oscillator and Clock Specification

Num	Characteristic	Min	Typ	Max	Unit
26	Ring Oscillator Low	10	25	50	ns
27	Ring Oscillator High	10	25	50	ns
28	Ring Oscillator Period	25	50	100	ns
29	Internal Config Clock Period	200	400	800	ns



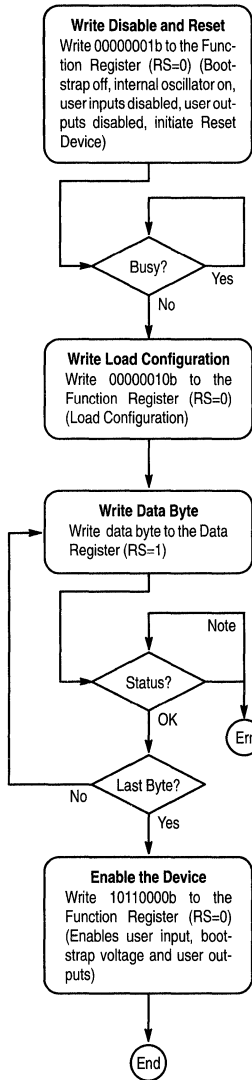


Figure 2-36. MICRO Mode Configuration Load Sequence Example

External Clock Specification

To improve configuration performance an external clock can be connected to the CLK pin when MODE[2] is asserted. The specifications for this clock are given in Figure 2-37.

The maximum external clock frequency is 40MHz. At this frequency, boot time in the BFR modes will decrease by a factor between 8 and 32 times.

Busy? – There are two different ways Busy can be checked. The first is to examine the state of the physical BUSY signal. The second is to read the contents of the Data/Status Register (RS=1)

Note – The designer has a fair number of options with regards to what code to put in this “Status” test block. The code may simply check that neither busy nor any error flags have been set and move on (this is what is shown) or the code may be optimized for higher speed.

To decrease the total load time, you might want to only check for busy at the end of the every row of data. (When counting bytes to keep track of where you are in the load, remember that the first row has an additional 5 data bytes, the first four are JTAG ID and the next is the Data Type tag.)

If for some reason you also shut off the internal oscillator during this boot process, this would be the time to turn it back on (preferably prior to enabling the bootstrap; the oscillator drives the charge pump that provides bootstrap voltage). Start up time for the oscillator is not greater than 100µs.

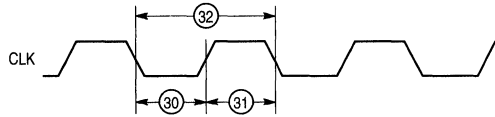


Figure 2-37. External Clock Specification

Num	Characteristic	Min	Max	Unit
30	External Clock Low	10		ns
31	External Clock High	10		ns
32	External Clock Period	25		ns

2

Power On Reset Operation

The MPA1000 devices contain circuitry to insure reliable self configuration when power is applied to the device. An counter clocked by the internal configuration clock and triggered by an analog power on reset circuit delays configuration until the power supply has been given sufficient settling time (Figure 2-40).

The analog power on reset circuit provides a reliable signal (APOR) to indicate that V_{DD} is sufficient to reliably operate device logic. While APOR is low a 17 bit counter is held reset. When APOR is asserted, the counter is enabled and POR occurs when the most significant counter bit reaches 1. Between APOR assertion and POR, the configuration circuitry is continuously resetting configuration memory row by row. When POR is asserted, a final internal reset sequence is performed (Figure 2-40). If an external clock is selected (by asserting MODE[2]) this final internal reset sequence will begin four internal clocks after POR, and will run using the external clock.

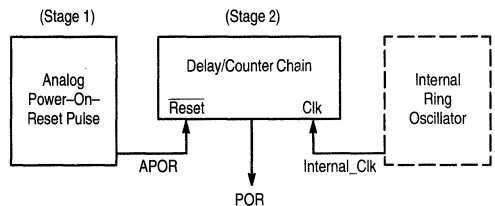


Figure 2-38. 2-Stage Power-On-Reset

External Reset

An external reset sequence can only be initiated by a falling edge on RESET. If an external clock is selected (by asserting MODE[2]), it must be active in order for the reset sequence to complete successfully. Once a reset sequence is initiated it cannot be terminated by a subsequent rising edge of RESET.

If RESET is low when the internal reset sequence completes, configuration will not commence until RESET is deasserted (Figure 2-41). This feature can be used to hold off configuration until other external events occur. This



MPA1000 Product Description

feature is used in conjunction with the multiple device daisy configuration mode, internal reset and internal configuration signals.

2

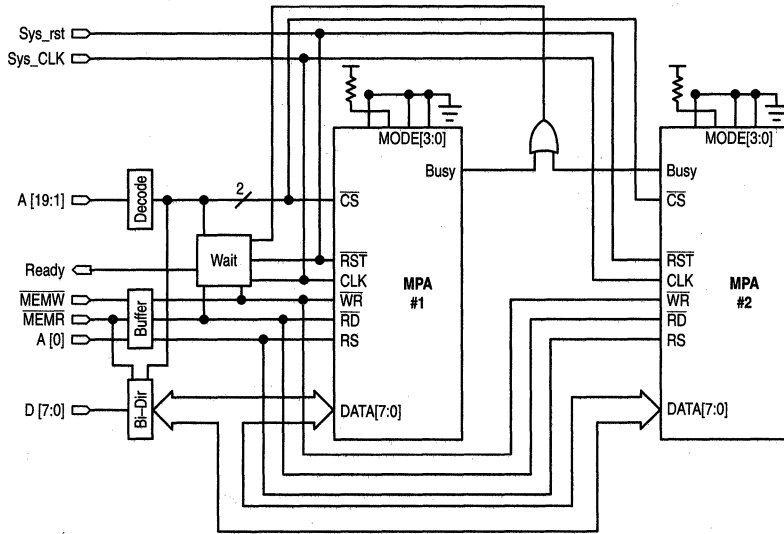


Figure 2-39. MICRO Mode: Multiple Devices With External Clock and Wait State Insertion

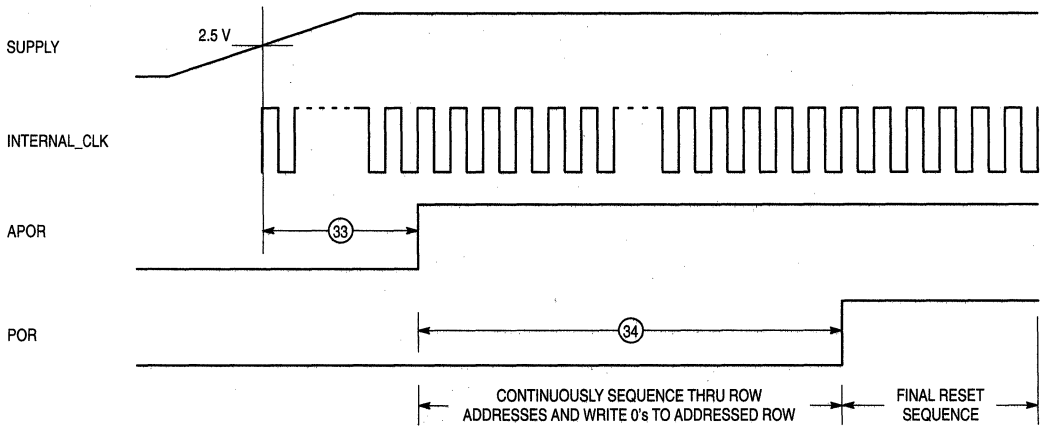
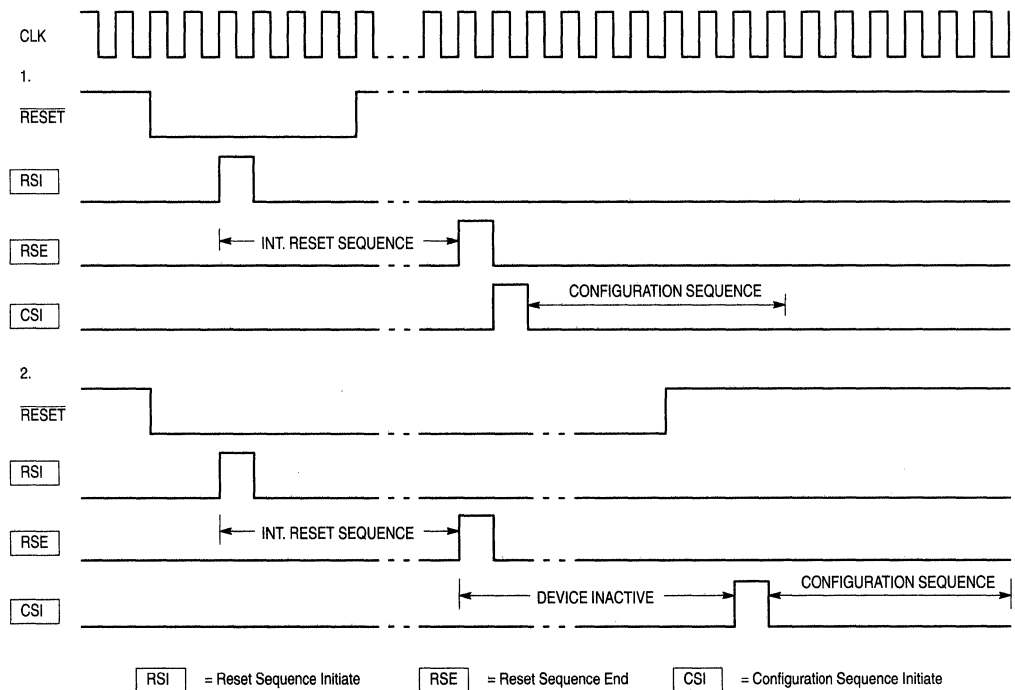


Figure 2-40. Power-On-Reset Circuitry Timing

Number	Characteristic	Min	Max	Unit	Notes
33	APOR	10	1000	μs	
34	POR (Active)	13.2	52.4	ms	





2

Figure 2-41. External Reset Behavior



MPA1000 Product Description

2

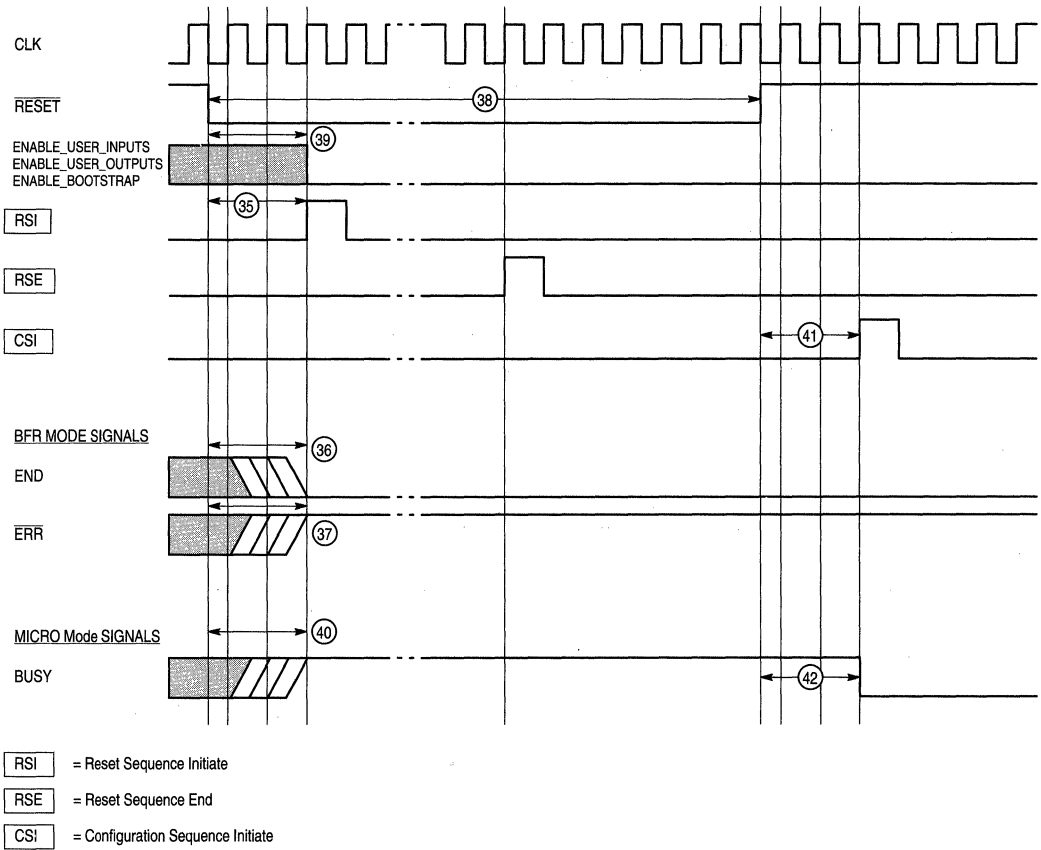


Figure 2-42. External Reset Timing

Number	Characteristic	Min	Max	Unit	Notes
35	RESET Low to Reset Sequence	2	3	CLK	
36	RESET Low to END Low	0	3	CLK	
37	RESET Low to ERR High	0	3	CLK	
38	RESET Pulse Width	50		ns	
39	RESET Low to Internal Disable	0	3	CLK	
40	RESET Low to Busy Active	0	3	CLK	
41	RESET High to CSI Pulse	2		CLK	RESET Released After RSE
42	RESET High to Busy Inactive	2		CLK	RESET Released After RSE

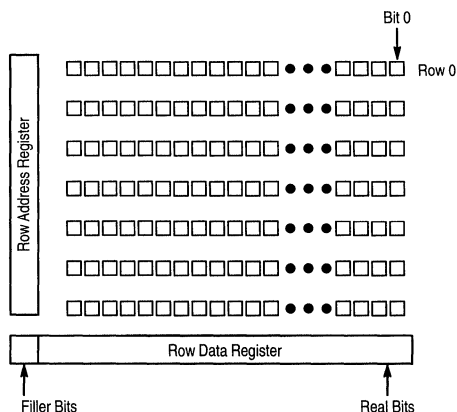
In MICRO Mode, the busy signal remains high while the reset signal is asserted and until the internal reset sequence is completed.



MPA1000 Configuration Data Format

Device Configuration Memory Organization

The MPA1000 devices are programmed by loading configuration data into on chip configuration memory constructed of SRAM cells. This memory is organized differently from standard memory products. The configuration SRAM is distributed throughout the MPA device. Data is read and written to the device 1 row at a time via the internal row data register (RDR). Individual rows are addressed via the row address register (RAR). Each device has a different size RDR and RAR (Figure 2–43). The configuration logic is responsible for the control of these resources.



Device	# Rows	# Real Bits	# Filler	# ECB	Total Row Bits
1016	95	562	6	8	576
1036	139	828	4	8	840
1064	183	1090	6	8	1104
1100	227	1352	0	8	1360

Figure 2–43. Device Memory Organization

A complete configuration image includes the total row bits shown in Figure 2–43, prefaced by the 5 byte header block. The total configuration image size is given in Table 2–10.

Table 2–10. Configuration Image Size

Device	Total Bits	Decimal Bytes	Hex Bytes
1016	54,760	6,845	1ABD
1036	116,800	14,600	3908
1064	202,072	25,259	62AB
1100	308,760	38,595	96C3

Configuration logic writes data to the leftmost (most significant) RDR byte and reads from the rightmost (least significant) RDR byte. Each of these transfers occurs in 8 bit

increments. When serial data is presented, the bits are accumulated into a byte before RDR transfer. Each configuration logic RDR write operation first shifts the RDR eight 8 bits and transfers the new byte into the leftmost RDR byte position. Configuration read operations transfer the rightmost RDR byte to the configuration logic and then shifts RDR contents right 8 bits.

The RAR enables a single configuration memory row. MPA configuration logic writes a row addresses into the RAR. Subsequent read or write operations are performed between the RDR and the RAR selected row in parallel.

Filler bits are used to round the RDR up to the nearest byte boundary. The ECB is not part of the RDR. During configuration a single row data vector is written to the RDR and an ECB is calculated from the data written. The calculated value is compared to the ECB contained in the data vector. If a mismatch is detected, ERR is asserted and the configuration process terminates. The ECB mechanism prevents data write disturbances from causing unpredictable device function.

Device Configuration Data Formats

When whole configurations are loaded into a device, the first 40 bits contain a 32 bit device ID followed by an 8 bit data type field. The device ID is the same as the JTAG device ID described in "JTAG Boundary Scan". If an incorrect ID is presented, ERR is asserted and configuration stops. Device ID comparison prevents incompatible configurations from causing unpredictable device behavior. The data type field identifies subsequent data format. Recognized data types are shown in Table 2–11.

Table 2–11.

[7:3]	[2]	[1]	[0]	Data Type
00000			0	Sequential data (Normal data)
00000			1	Test data – Multiple row access
00000		0		Unencrypted data
00000		1		Encrypted data – Not supported on first product. Reserved for future implementations
00000	0			Uncompressed data
00000	1			Compressed data – Not supported on first product. Reserved for future implementations

Header Block

Device ID [3]	00000000b = Normal 00000001b = Test
Device ID [2]	
Device ID [1]	
Device ID [0]	
Data Type	

2



MPA1000 Product Description

Two data formats are supported; Normal data and test data. Normal data is generated by the MPA Design System and is the only data type users are expected to use. Test data is a special format developed to aid device testing where many very regular configuration patterns must be rapidly loaded during production test. Test mode data only results in a memory savings when many rows of configuration memory contain identical information. Since this is unlikely for real designs, test mode data offers little or no benefit for reducing user configuration memory storage requirements.

Normal data consists of a series of configuration memory row images including filler bits and ECB. Each device has a different number of bytes per row and a different number of rows. A generalized normal data representation is shown in Figure 2-44. Bytes are presented to the device from left to right and from top-most row (row 0) to bottom-most row. The ECB is calculated by summing the row data byte by byte, complementing the carry and using this as the carry into the next addition.

Data 0 (Row 0)	Data 1 (Row 0)	~	~	~	ECB 0
Data 0 (Row 1)	Data 1 (Row 1)	~	~	~	ECB 1
"	"	"	"	"	"
"	"	"	"	"	"
Data 0 (Row x)	Data 1 (Row x)	~	~	~	ECB x
Data 0 (Row y)	Data 1 (Row y)	~	~	~	ECB y

Figure 2-44. Configuration Data Block (Normal Data)

Test data format is similar to normal data except that a row count and address list follows the ECB. The RDR is loaded and the ECB calculated normally. Each address is written to the RAR, a write cycle initiated to transfer the RDR to the addressed configuration memory row, the expected address count is decremented and the next address is loaded until the expected address count reaches zero. The next byte is assumed to be the first byte of a new row data vector. Configuration ends when a row address of 255 is presented. Figure 2-45 shows the generalized test data format.

Data 0	Data 1	~	ECB M	No. Rows	Row A
					Row B
					Row C
					Row D
					"
Data 0	Data 1	~	ECB N	No. Rows	Row E
					Row F
					Row G
					Row 255

(Row 255 = Configuration Terminating Byte)

Figure 2-45. Test Data Configuration



MPA1000 JTAG Boundary Scan

JTAG Boundary Scan Functions

JTAG is a standardized boundary scan methodology used for board level testing to detect faults in package and board connections, as well as internal circuitry. The MPA1000 JTAG boundary scan cell is designed to meet the IEEE std. 1149.1 for testability test of an integrated circuit.

IEEE 1149.1 Architecture

Figure 2-46 shows the general diagram of the IEEE 1149.1 MPA1000 JTAG system. Its design is compatible to Motorola H4C and H4C+ family of arrays. The MPA1000 JTAG design is hard wired.

A more detailed description of the MPA1000 JTAG system can be found in Motorola Application Note AN1618/D *Using JTAG Boundary Scan with the Motorola MPA1000 Family of FPGAs* in Ch 4. on page 4-209.

TAP and I/O Peripheral Signals

The TAP (Test Access Port) consists of five externally accessible signals which are used to control and observe boundary scan data. These five pins; TCK, TMS, TDI, TRSTB, and TDO are multiplexed with normal signal pins. After JTAG testing, these pins can be programmed as normal I/O pins when MODE[3] is deasserted. The test clock pin, TCK, is used to synchronize all JTAG functions. The TCK, TMS and TRSTB control the TAP controller. TDI is the test data input pin and TDO is the test data output pin.

JTAG Control and Test Register

The **TAP Controller** is a synchronous, 16-state machine, which selects the mode of operation for the test circuitry. An example of the operation of the TAP controller is shown in Figure 2-47 where the TAP controller is sequenced through most of its test states.

2

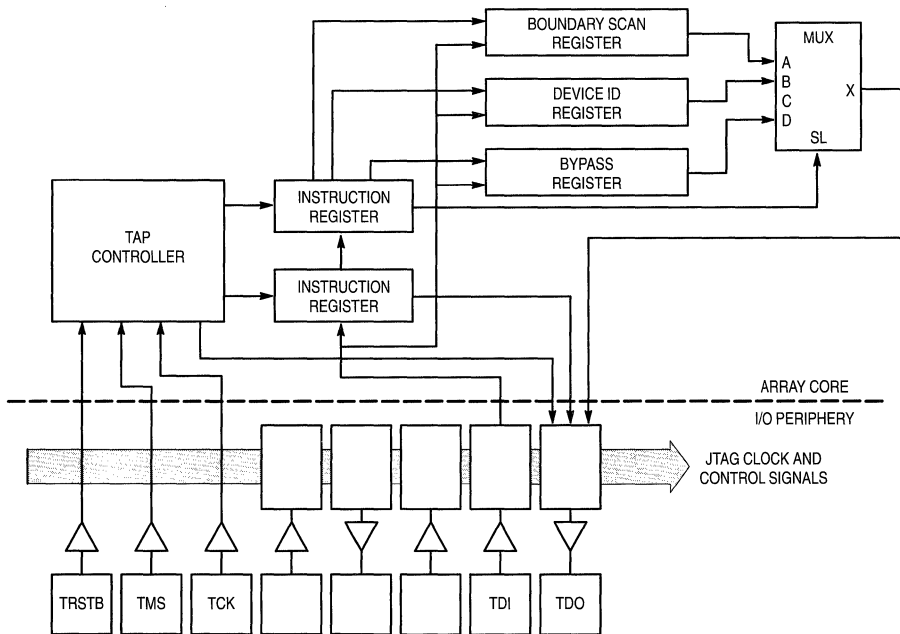


Figure 2-46. JTAG System



2

Ⓓ = "D" STATE OF TAP CONTROLLER
 0 = LOGIC STATE OF TMS
 "0" = OFF/LOW
 "1" = ON/HIGH
 DR = DATA REGISTER
 IR = INSTRUCTION REGISTER

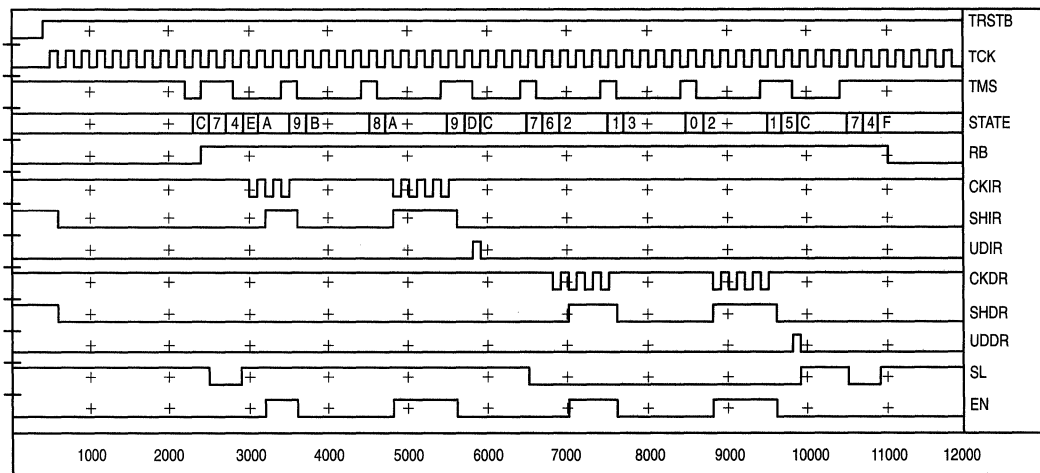
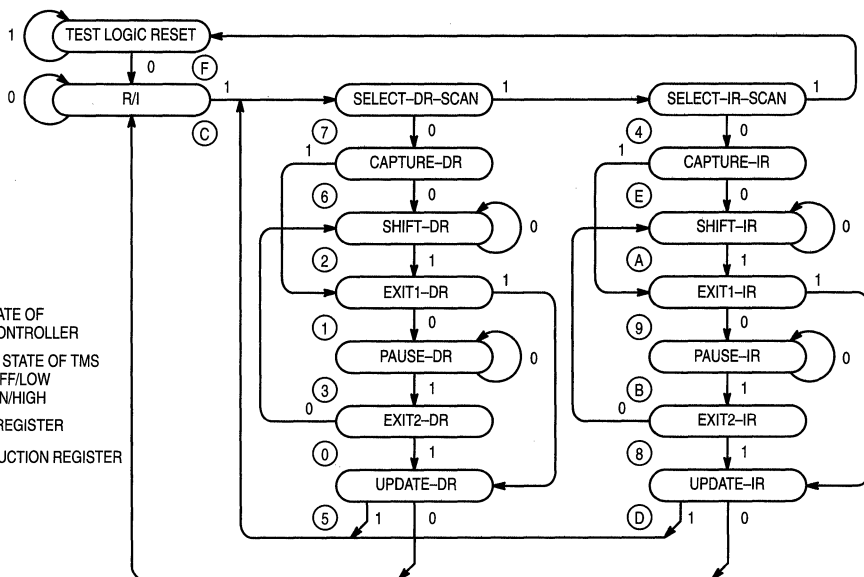


Figure 2-47. TAP Controller and Test Cycle



The **Instruction Register** is a 3-bit shift register, which permits an instruction to be shifted into the design to select the test to be performed. The **Instruction Decode** translates the instruction into separate control signals. Table 2–12 shows the basic public instructions supported by Motorola's FPGA:

Table 2–12. Basic Public Instructions

I ₂	I ₁	I ₀	Public Instruction	Register Selected
0	0	0	EXTEST	Boundary Scan Cell
0	0	1	INTEST	Boundary Scan Cell
0	1	0	SAMPLE	Boundary Scan Cell
1	0	0	IDCODE	Device Register
1	1	1	BYPASS	Bypass Register

- **EXTEST** (external test) is the boundary scan test that checks board interconnections between integrated circuits (I.C.s).
- **INTEST** (internal test) checks the logic internal to I.C.s.
- **SAMPLE** test samples data at the I/O pins of an I.C. during normal operating mode.
- **IDCODE** instruction outputs the identification code of the I.C.
- **BYPASS** instruction redirects the test data from TDI directly to TDO, effectively removing the I.C. from the boundary scan chain.

The **Bypass Register** is a single-bit shift register used to provide a shortest path between TDI and TDO.

Table 2–13. Device Register ID Codes

Bit Number	Code Use
0–11	Motorola Identification
12–21	Array Identification
22–27	Programmable Logic Products Identification
28–31	Version Number

The **Device Identification Register** is a 32-bit register which holds a manufacturer's identity code, part number and version code. The bit assignment for the ID code is given in Table 2–13.

For example, for MPA1036 & MPA1064, the ID codes are listed as follows:

Array	ID code (Binary)
MPA1016	0001 001110 0100001110 000000011101
MPA1036	0001 001110 0100011110 000000011101
MPA1064	0001 001110 0100110100 000000011101
MPA1100	0001 001110 0101000000 000000011101

Array	Hex ID code	Hex ID Code (Bit Order Reversed)
MPA1016	1390E01D	C8 05 07 B8
MPA1036	1391E01D	C8 85 07 B8
MPA1064	1393401D	C8 C5 02 B8
MPA1100	1394001D	C8 25 00 B8

The JTAG ID code can easily be located when viewing a configuration file with a text editor. The ID code is always the first four data bytes. The bit order reversed version of the code shows up in configuration images targeted to serial EPROMs.

The **Boundary Scan Register** is the chain of JTAG boundary scan cells that are linked together to form a shift register around the periphery of the array. The test data enters the boundary scan register through the TDI pin, the rising edge of CKDR when SHDR is asserted, then is shifted around the array through each I/O cell in a counter clockwise direction, and finally exits through the TDO pin. Since each I/O pin is designed as a bidirectional pin, a 2-bit shift register resides in each I/O cell, one for monitor either the input or output, and the other to monitor the enable pin of the 3-state output buffer. For every two clock cycles, the data shifts from one I/O site to the other. The boundary scan cell resides in every I/O site with the exception of TDI, TCK, TMS, TRSTB and TDO pins.

2



MPA1000 Pin Definitions

Table 2-14. MPA1000 Package Pinout Compatibility

Device	FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP	DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP
MPA1016	•	•		
MPA1036	•	•	•	
MPA1064			•	•
MPA1100				•

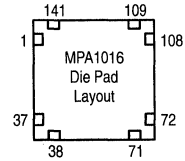
Table 2-15. Pin Definitions

Pin	Definition
5V Int Vdd	Internal array power (VDD)
5V Ext Vdd	Pad driver power for I/Os programmed to 5V
3V Ext Vdd	Pad driver power for I/Os programmed to 3V. If no I/Os are programmed to 3V, tie to 5V Ext Vdd. If 3V I/Os are used, connect to a 3.0V or 3.3V supply. These pins must be \leq 5V Ext Vdd.
Ext Vss	Pad driver VSS
Int Vss	Internal array VSS
I/O	User I/O
I/O Clk	User I/O with optional clock input



MPA1000 Pin Assignments

Pinouts for MPA1016



Edge	Pad	Pad Type	Pin Location	
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP
	1	5V Int Vdd		1
	2	Ext Vss		
	3	3V Ext Vdd	12	2
	4	5V Ext Vdd		
	5	Ext Vss		3
L	6	I/O (A16)	13	4
L	7	I/O (A15)	14	5
L	8	I/O (A14)	15	6
L	9	I/O (A13)	16	7
L	10	I/O (A12)	17	8
	11	5V Ext Vdd		
L	12	I/O (A11)	18	9
L	13	I/O		10
L	14	I/O (A10)	19	11
L	15	I/O		12
L	16	I/O Clk	20	13
	17	Int Vss	21	14
L	18	I/O Clk	22	15
L	19	I/O (A9)	23	16
L	20	I/O (A8)	24	17
L	21	I/O		18
L	22	I/O (A7)	25	19
	23	Ext Vss	26	20
L	24	I/O		21
L	25	I/O (A6)	27	22
L	26	I/O		23
L	27	I/O (A5)	28	24
L	28	I/O		25
	29	F[4]	29	26
	30	Int Vss		27
	31	F[3]	30	28
	32	Ext Vss		
	33	F[2]	31	29
	34	5V Ext Vdd		30
	35	F[0]	32	31
	36	3V Ext Vdd		32

Edge	Pad	Pad Type	Pin Location	
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP
	37	Ext Vss		
	38	Ext Vss		33
	39	5V Ext Vdd		
	40	RESET	33	34
	41	3V Ext Vdd		35
	42	F[1]	34	36
B	43	I/O (A4)	35	37
B	44	I/O (A3)	36	38
B	45	I/O (A2)	37	39
B	46	I/O (A1)	38	40
B	47	I/O (A0)	39	41
	48	5V Ext Vdd	40	42
B	49	I/O		43
B	50	I/O		44
B	51	I/O (D7)	41	45
B	52	I/O		46
B	53	I/O Clk	42	47
	54	5V Int Vdd	43	48
B	55	I/O Clk	44	49
B	56	I/O		50
B	57	I/O		51
B	58	I/O (D6)	45	52
B	59	I/O		53
	60	Ext Vss	46	54
B	61	I/O (D5)	47	55
B	62	I/O (D4)	48	56
B	63	I/O (D3)	49	57
B	64	I/O (D2)	50	58
B	65	I/O (D1)	51	59
	66	MODE[0]	52	60
	67	Ext Vss		61
	68	MODE[1]	53	62
	69	3V Ext Vdd		
	70	5V Ext Vdd		63
	71	Probe Pad	NOT BONDED	
	72	Ext Vss		66

2

Shaded areas indicate Alternate I/O Zones.



Pinouts for MPA1016 (continued)

2

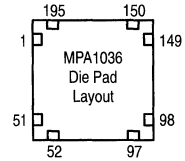
Edge	Pad	Pad Type	Pin Location	
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP
	73	5V Ext Vdd		
	74	MODE[2]	54	67
	75	5V Int Vdd		
	76	MODE[3]	55	68
	77	Vpp		
	78	Clk	56	69
	79	3V Ext Vdd	57	70
	80	Ext Vss		71
R	81	I/O (DCLK)	58	72
R	82	I/O		73
R	83	I/O (D0)	59	74
R	84	I/O		75
R	85	I/O (TDO)	60	76
	86	Ext Vss		77
R	87	I/O (TDI)	61	78
R	88	I/O (TMS)	62	79
R	89	I/O		80
R	90	I/O (TRSTB)	63	81
R	91	I/O Clk	64	82
	92	Int Vss	65	83
R	93	I/O Clk	66	84
R	94	I/O		85
R	95	I/O	67	86
R	96	I/O		87
R	97	I/O	68	88
	98	Ext Vss		89
R	99	I/O (TCK)	69	90
R	100	I/O	70	91
R	101	I/O	71	92
R	102	I/O	72	93
R	103	I/O	73	94
	104	5V Ext Vdd		
	105	3V Ext Vdd		
	106	Ext Vss	74	95
	107	5V Ext Vdd		
	108	5V Int Vdd		96
	109	Int Vss		97
	110	Ext Vss		

Edge	Pad	Pad Type	Pin Location	
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP
	111	5V Ext Vdd	75	98
	112	3V Ext Vdd		99
	113	Ext Vss		
T	114	I/O	76	100
T	115	I/O	77	101
T	116	I/O	78	102
T	117	I/O	79	103
T	118	I/O	80	104
	119	Ext Vss		105
T	120	I/O	81	106
T	121	I/O	82	107
T	122	I/O	83	108
T	123	I/O		109
T	124	I/O Clk	84	110
	125	5V Int Vdd	1	111
T	126	I/O Clk	2	112
T	127	I/O		113
T	128	I/O	3	114
T	129	I/O	4	115
T	130	I/O	5	116
	131	5V Ext Vdd		117
T	132	I/O	6	118
T	133	I/O	7	119
T	134	I/O	8	120
T	135	I/O	9	121
T	136	I/O (A17)	10	122
	137	Ext Vss		123
	138	5V Ext Vdd		124
	139	Ext Vss	11	125
	140	3V Ext Vdd		126
	141	Int Vss		127
		NC		64,65,128

Shaded areas indicate Alternate I/O Zones.



Pinouts for MPA1036



Edge	Pad	Pad Type	Pin Location			
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP	DH Suffix 160-Pin QFP	HI Suffix 181-Pin PGA
	1	5V Int Vdd		1		A2
	2	Ext Vss				VSSE
	3	3V Ext Vdd	12	2	40	A1
	4	5V Ext Vdd				VDDE
	5	Ext Vss		3		VSSE
L	6	I/O (A16)	13	4	39	B2
L	7	I/O			38	C2
L	8	I/O (A15)	14	5	37	D4
L	9	I/O			36	B1
L	10	I/O (A14)	15	6	35	C3
	11	5V Ext Vdd				VDDE
L	12	I/O (A13)	16	7	34	D3
L	13	I/O			33	C1
L	14	I/O (A12)	17	8	32	D2
L	15	I/O			31	D1
L	16	I/O (A11)	18	9	30	E3
	17	Ext Vss			29	VSSE
L	18	I/O		10	28	F3
L	19	I/O (A10)	19	11	27	E1
L	20	I/O			26	E2
L	21	I/O		12	25	F1
L	22	I/O Clk	20	13	24	G3
	23	5V Int Vdd			23	G1
	24	Int Vss	21	14	22	VSSI
L	25	I/O Clk	22	15	21	G2
L	26	I/O			20	F2
L	27	I/O			19	H1
L	28	I/O (A9)	23	16	18	H3
L	29	I/O			17	H2
	30	5V Ext Vdd			16	VDDE
L	31	I/O (A8)	24	17	15	J1
L	32	I/O		18	14	J2
L	33	I/O (A7)	25	19	13	K1
L	34	I/O			12	K2
L	35	I/O			11	L1
	36	Ext Vss	26	20	10	VSSE

Shaded areas indicate Alternate I/O Zones.

Edge	Pad	Pad Type	Pin Location			
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP	DH Suffix 160-Pin QFP	HI Suffix 181-Pin PGA
L	37	I/O		21	9	M1
L	38	I/O (A6)	27	22	8	L2
L	39	I/O		23	7	N1
L	40	I/O (A5)	28	24	6	J3
L	41	I/O		25	5	P1
	42	F[4]	29	26	4	K3
	43	Int Vss		27		VSSI
	44	F[3]	30	28	3	M2
	45	Ext Vss				VSSE
	46	F[2]	31	29	2	L3
	47	5V Ext Vdd		30		VDDE
	48	F[0]	32	31	1	M3
	49	3V Ext Vdd		32		P2
	50	Ext Vss				VSSE
	51	Ext Vss				VSSE
	52	Ext Vss		33		VSSE
	53	5V Ext Vdd				VDDE
	54	RESET	33	34	160	R1
	55	3V Ext Vdd		35		N2
	56	F[1]	34	36	159	R2
B	57	I/O (A4)	35	37	158	N3
B	58	I/O			157	R3
B	59	I/O (A3)	36	38	156	N4
B	60	I/O			155	R4
B	61	I/O (A2)	37	39	154	P3
	62	Ext Vss			153	VSSE
B	63	I/O			152	N5
B	64	I/O (A1)	38	40	151	R5
B	65	I/O			150	P4
B	66	I/O (A0)	39	41	149	R6
B	67	I/O			148	N6
	68	5V Ext Vdd	40	42	147	VDDE
B	69	I/O		43	146	P5
B	70	I/O		44	145	R7
B	71	I/O (D7)	41	45	144	N7
B	72	I/O		46	143	R8

2



Pinouts for MPA1036 (continued)

2

Edge	Pad	Pad Type	Pin Location			
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP	DH Suffix 160-Pin QFP	HI Suffix 181-Pin PGA
B	73	I/O Clk	42	47	142	N8
	74	Int Vss			141	VSSI
	75	5V Int Vdd	43	48	140	P6
B	76	I/O Clk	44	49	139	P8
B	77	I/O		50	138	P7
B	78	I/O		51	137	R9
B	79	I/O (D6)	45	52	136	P9
B	80	I/O		53	135	R10
	81	Ext Vss	46	54	134	VSSE
B	82	I/O (D5)	47	55	133	R11
B	83	I/O			132	N9
B	84	I/O (D4)	48	56	131	R12
B	85	I/O			130	P10
B	86	I/O (D3)	49	57	129	P11
	87	5V Ext Vdd				VDDE
B	88	I/O			128	R13
B	89	I/O (D2)	50	58	127	N10
B	90	I/O			126	R14
B	91	I/O (D1)	51	59	125	N11
B	92	I/O			124	P13
	93	MODE[0]	52	60	123	P12
	94	Ext Vss		61		VSSE
	95	MODE[1]	53	62	122	N12
	96	3V Ext Vdd			121	P14
	97	5V Ext Vdd		63		VDDE
	98	Probe Pad	NOT BONDED			
	99	Ext Vss				VSSE
	100	Ext Vss		66		VSSE
	101	5V Ext Vdd				VDDE
	102	MODE[2]	54	67	120	M12
	103	5V Int Vdd				R15
	104	MODE[3]	55	68	119	N13
	105	Vpp				P15
	106	Clk	56	69	118	L13
	107	3V Ext Vdd	57	70	117	N15
	108	Ext Vss		71		VSSE

Shaded areas indicate Alternate I/O Zones.

Edge	Pad	Pad Type	Pin Location			
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP	DH Suffix 160-Pin QFP	HI Suffix 181-Pin PGA
R	109	I/O (Dclk)	58	72	116	L14
R	110	I/O		73	115	M13
R	111	I/O (D0)	59	74	114	M15
R	112	I/O		75	113	N14
R	113	I/O (Tdo)	60	76	112	K14
	114	Ext Vss		77	111	VSSE
R	115	I/O (Tdi)	61	78	110	L15
R	116	I/O			109	K13
R	117	I/O			108	K15
R	118	I/O (Tms)	62	79	107	M14
R	119	I/O			106	J15
	120	5V Ext Vdd			105	VDDE
R	121	I/O		80	104	H14
R	122	I/O (Trstb)	63	81	103	J13
R	123	I/O			102	H15
R	124	I/O			101	J14
R	125	I/O Clk	64	82	100	G14
	126	Int Vss	65	83	99	VSSI
	127	5V Int Vdd			98	G15
R	128	I/O Clk	66	84	97	H13
R	129	I/O		85	96	F15
R	130	I/O	67	86	95	G13
R	131	I/O		87	94	E15
R	132	I/O	68	88	93	F14
	133	Ext Vss		89	92	VSSE
R	134	I/O (Tck)	69	90	91	F13
R	135	I/O			90	D15
R	136	I/O	70		89	E14
R	137	I/O		91	88	C15
R	138	I/O	71		87	E13
	139	5V Ext Vdd				VDDE
R	140	I/O		92	86	D13
R	141	I/O	72		85	D14
R	142	I/O		93	84	C13
R	143	I/O			83	B15
R	144	I/O	73	94	82	D12



Pinouts for MPA1036 (continued)

E d g e	Pad	Pad Type	Pin Location			
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP	DH Suffix 160-Pin QFP	HI Suffix 181-Pin PGA
	145	Ext Vss				VSSE
	146	3V Ext Vdd				C12
	147	Ext Vss	74	95	81	VSSE
	148	5V Ext Vdd				VDDE
	149	5V Int Vdd		96		C14
	150	Int Vss		97		VSSI
	151	Ext Vss				VSSE
	152	5V Ext Vdd	75	98	80	VDDE
	153	3V Ext Vdd		99		A15
	154	Ext Vss			79	VSSE
T	155	I/O	76	100	78	B14
T	156	I/O			77	C11
T	157	I/O	77	101	76	B13
T	158	I/O			75	B12
T	159	I/O	78	102	74	A14
	160	5V Ext Vdd			73	VDDE
T	161	I/O	79	103	72	A13
T	162	I/O			71	C10
T	163	I/O	80		70	A12
T	164	I/O		104	69	B11
T	165	I/O	81		68	A11
	166	Ext Vss		105	67	VSSE
T	167	I/O	82	106	66	A10
T	168	I/O		107	65	B10
T	169	I/O	83	108	64	A9
T	170	I/O		109	63	C9
T	171	I/O Clk	84	110	62	B8
	172	5V Int Vdd	1	111	61	B9
	173	Int Vss			60	VSSI
T	174	I/O Clk	2	112	59	C8
T	175	I/O		113	58	A8
T	176	I/O	3	114	57	B7
T	177	I/O		115	56	A7
T	178	I/O	4	116	55	C7
	179	5V Ext Vdd		117		VDDE
T	180	I/O	5		54	B6

Shaded areas indicate Alternate I/O Zones.

E d g e	Pad	Pad Type	Pin Location			
			FN Suffix 84-Pin PLCC	DD Suffix 128-Pin QFP	DH Suffix 160-Pin QFP	HI Suffix 181-Pin PGA
T	181	I/O		118	53	A6
T	182	I/O	6		52	C6
T	183	I/O			51	A5
T	184	I/O	7	119	50	B5
	185	Ext Vss			49	VSSE
T	186	I/O	8	120	48	C5
T	187	I/O			47	A4
T	188	I/O	9	121	46	B4
T	189	I/O			45	A3
T	190	I/O (A17)	10	122	44	C4
	191	Ext Vss		123	43	VSSE
	192	5V Ext Vdd		124	42	VDDE
	193	Ext Vss	11	125		VSSE
	194	3V Ext Vdd		126	41	B3
	195	Int Vss		127		VSSI
		NC		64,65, 128		E5

181PGA NOTES:

VSSE Plane: G12, E12, K12, D10, M10, G4, E4, K4, D6, M6

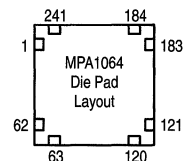
VSSI Plane: E8, L8, H11, M11, H5, D5

VDDE Plane: D8, M8, H12, F12, J12, L12, D9, M9, D11, H4, F4, M4,
J4, L4, D7, M7, M5

2



Pinouts for MPA1064



2

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
	1	5V Int Vdd			VDDI
	2	Ext Vss			VSSE
	3	3V Ext Vdd	40		E4
	4	5V Ext Vdd			VDDE
	5	Ext Vss			VSSE
L	6	I/O (A16)	39	1	C4
L	7	I/O		2	B2
L	8	I/O		3	D4
L	9	I/O	38	4	C2
L	10	I/O		5	C3
	11	Ext Vss		6	VSSE
L	12	I/O (A15)	37	7	D3
L	13	I/O		8	B1
L	14	I/O	36	9	D2
L	15	I/O		10	C1
L	16	I/O (A14)	35	11	G4
	17	5V Ext Vdd		12	VDDE
L	18	I/O (A13)	34	13	E3
L	19	I/O	33	14	D1
L	20	I/O (A12)	32	15	E2
L	21	I/O	31	16	E1
L	22	I/O (A11)	30	17	F3
	23	Ext Vss	29	18	VSSE
L	24	I/O	28	19	G3
L	25	I/O (A10)	27	20	F1
L	26	I/O	26	21	G2
L	27	I/O	25	22	G1
L	28	I/O Clk	24	23	J4
	29	5V Int Vdd	23	24	VDDI
	30	Int Vss	22	25	VSSI
L	31	I/O Clk	21	26	H1
L	32	I/O		27	H3
L	33	I/O	20	28	J2
L	34	I/O		29	H2
L	35	I/O	19	30	K1
	36	Ext Vss		31	VSSE

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
L	37	I/O		32	L1
L	38	I/O (A9)	18	33	J3
L	39	I/O		34	L2
L	40	I/O	17	35	K3
L	41	I/O		36	M1
	42	5V Ext Vdd	16	37	VDDE
L	43	I/O (A8)	15	38	N1
L	44	I/O	14	39	K2
L	45	I/O (A7)	13	40	P1
L	46	I/O	12	41	L3
L	47	I/O	11	42	N2
	48	Ext Vss	10	43	VSSE
L	49	I/O	9	44	R1
L	50	I/O (A6)	8	45	M3
L	51	I/O	7	46	T1
L	52	I/O (A5)	6	47	L4
L	53	I/O	5	48	R2
	54	F[4]	4	49	N3
	55	Int Vss			VSSI
	56	F[3]	3	50	P2
	57	Ext Vss			VSSE
	58	F[2]	2	51	P3
	59	5V Ext Vdd			VDDE
	60	F[0]	1	52	P4
	61	3V Ext Vdd			N4
	62	Ext Vss			VSSE
	63	Ext Vss		53	VSSE
	64	5V Ext Vdd			VDDE
	65	RESET	160	54	R3
	66	3V Ext Vdd		55	P5
	67	F[1]	159	56	T2
B	68	I/O (A4)	158	57	R4
B	69	I/O		58	T3
B	70	I/O	157	59	P6
B	71	I/O		60	U2
B	72	I/O (A3)	156	61	T4

Shaded areas indicate Alternate I/O Zones.



Pinouts for MPA1064 (continued)

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
	73	Ext Vss			VSSE
B	74	I/O		62	R5
B	75	I/O	155	63	U3
B	76	I/O		64	T5
B	77	I/O (A2)	154	65	U4
B	78	I/O		66	P7
	79	Ext Vss	153	67	VSSE
B	80	I/O	152	68	R6
B	81	I/O (A1)	151	69	U5
B	82	I/O	150	70	R7
B	83	I/O (A0)	149	71	U6
B	84	I/O	148	72	P8
	85	5V Ext Vdd	147	73	VDDE
B	86	I/O	146	74	T7
B	87	I/O	145	75	U7
B	88	I/O (D7)	144	76	R8
B	89	I/O	143	77	U8
B	90	I/O Clk	142	78	T8
	91	Int Vss	141	79	VSSI
	92	5V Int Vdd	140	80	VDDI
B	93	I/O Clk	139	81	T9
B	94	I/O	138	82	R9
B	95	I/O	137	83	U10
B	96	I/O (D6)	136	84	R10
B	97	I/O	135	85	T10
	98	Ext Vss	134	86	VSSE
B	99	I/O (D5)	133	87	U11
B	100	I/O	132	88	P10
B	101	I/O (D4)	131	89	T11
B	102	I/O	130	90	R11
B	103	I/O (D3)	129	91	U12
	104	5V Ext Vdd			VDDE
B	105	I/O	128	92	U13
B	106	I/O		93	P11
B	107	I/O (D2)	127	94	U14
B	108	I/O		95	R12

Shaded areas indicate Alternate I/O Zones.

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
B	109	I/O	126	96	T13
	110	Ext Vss		97	VSSE
B	111	I/O		98	U15
B	112	I/O (D1)	125	99	R13
B	113	I/O		100	U16
B	114	I/O	124	101	T14
B	115	I/O		102	T15
	116	MODE[0]	123	103	R14
	117	Ext Vss			VSSE
	118	MODE[1]	122	104	R15
	119	3V Ext Vdd	121		P12
	120	5V Ext Vdd			VDDE
	121	Probe Pad	NOT BONDED		
	122	Ext Vss			VSSE
	123	5V Ext Vdd		105	VDDE
	124	MODE[2]	120	106	T16
	125	5V Int Vdd			P13
	126	MODE[3]	119	107	T17
	127	Vpp			P14
	128	Clk	118	108	P16
	129	3V Ext Vdd	117	109	N14
	130	Ext Vss		110	VSSE
R	131	I/O (DCLK)	116	111	R16
R	132	I/O		112	R17
R	133	I/O	115	113	L14
R	134	I/O		114	N16
R	135	I/O (D0)	114	115	P15
	136	Ext Vss			VSSE
R	137	I/O	113	116	N15
R	138	I/O		117	P17
R	139	I/O		118	M15
R	140	I/O		119	N17
R	141	I/O (TDO)	112	120	L15
	142	Ext Vss	111	121	VSSE
R	143	I/O (TDI)	110	122	L16
R	144	I/O	109	123	M17

2



Pinouts for MPA1064 (continued)

2

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
R	145	I/O	108	124	K15
R	146	I/O (TMS)	107	125	L17
R	147	I/O	106	126	K16
	148	5V Ext Vdd	105	127	VDDE
R	149	I/O	104	128	J15
R	150	I/O (TRSTB)	103	129	K17
R	151	I/O	102	130	J14
R	152	I/O	101	131	J16
R	153	I/O Clk	100	132	H16
	154	Int Vss	99	133	VSSI
	155	5V Int Vdd	98	134	VDDI
R	156	I/O Clk	97	135	H17
R	157	I/O	96	136	H15
R	158	I/O	95	137	G17
R	159	I/O	94	138	G16
R	160	I/O	93	139	F17
	161	Ext Vss	92	140	VSSE
R	162	I/O		141	E17
R	163	I/O		142	G15
R	164	I/O (TCK)	91	143	D17
R	165	I/O	90	144	F15
R	166	I/O	89	145	C17
	167	5V Ext Vdd		146	VDDE
R	168	I/O	88	147	E16
R	169	I/O		148	G14
R	170	I/O	87	149	D16
R	171	I/O	86	150	E15
R	172	I/O	85	151	B17
	173	Ext Vss			VSSE
R	174	I/O	84	152	C16
R	175	I/O		153	D15
R	176	I/O	83	154	B16
R	177	I/O		155	D14
R	178	I/O	82	156	C15
	179	Ext Vss			VSSE
	180	3V Ext Vdd			E14

Shaded areas indicate Alternate I/O Zones.

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
	181	Ext Vss	81		VSSE
	182	5V Ext Vdd			VDDE
	183	5V Int Vdd			VDDI
	184	Int Vss			VSSI
	185	Ext Vss		157	VSSE
	186	5V Ext Vdd	80		VDDE
	187	3V Ext Vdd		158	D13
	188	Ext Vss	79		VSSE
T	189	I/O		159	C14
T	190	I/O	78	160	B15
T	191	I/O		161	D12
T	192	I/O	77	162	A16
T	193	I/O	76	163	C13
	194	Ext Vss		164	VSSE
T	195	I/O		165	B13
T	196	I/O	75	166	B14
T	197	I/O		167	D11
T	198	I/O	74	168	A15
T	199	I/O		169	C12
	200	5V Ext Vdd	73	170	VDDE
T	201	I/O	72	171	C11
T	202	I/O	71	172	A14
T	203	I/O	70	173	B11
T	204	I/O	69	174	A13
T	205	I/O	68	175	C10
	206	Ext Vss	67	176	VSSE
T	207	I/O	66	177	B10
T	208	I/O	65	178	A12
T	209	I/O	64	179	C9
T	210	I/O	63	180	A11
T	211	I/O Clk	62	181	D9
	212	5V Int Vdd	61	182	VDDI
	213	Int Vss	60	183	VSSI
T	214	I/O Clk	59	184	A10
T	215	I/O	58	185	B9
T	216	I/O	57	186	A8



Pinouts for MPA1064 (continued)

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
T	217	I/O	56	187	B8
T	218	I/O	55	188	A7
	219	5V Ext Vdd		189	VDDE
T	220	I/O	54	190	B7
T	221	I/O	53	191	C8
T	222	I/O	52	192	A6
T	223	I/O	51	193	C7
T	224	I/O	50	194	A5
	225	Ext Vss	49	195	VSSE
T	226	I/O		196	A4
T	227	I/O	48	197	D7
T	228	I/O		198	B5
T	229	I/O	47	199	C6
T	230	I/O	46	200	A3
	231	Ext Vss			VSSE
T	232	I/O		201	B4

Shaded areas indicate Alternate I/O Zones.

Edge	Pad	Pad Type	Pin Location		
			DH Suffix 160-Pin QFP	DK Suffix 208-Pin QFP	KE Suffix 224-Pin PGA
T	233	I/O	45	202	D6
T	234	I/O		203	A2
T	235	I/O (A17)	44	204	C5
T	236	I/O		205	B3
	237	Ext Vss	43		VSSE
	238	5V Ext Vdd	42		VDDE
	239	Ext Vss		206	VSSE
	240	3V Ext Vdd	41	207	D5
	241	Int Vss		208	VSSI

224 PGA NOTES:

VSSI Plane: E8, E10, H5, J13, K5, N9

VSSE Plane: A1, A9, A17, E7, E9, E11, F4, F14, H13, J1, J5, J17, K13, M4, M14, N7, N11, P9, U1, U9, U17

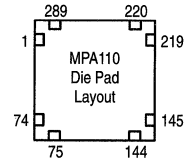
VDDI Plane: D8, D10, K4, K14, N8

VDDE Plane: B6, B12, F2, F16, H4, H14, M2, M16, N10, T6, T12

2



Pinouts for MPA1100



2

Edge	Pad	Pad Type	Pin Location	
			DK Suffix 208-Pin QFP	HV Suffix 299-Pin PGA
	1	5V Int Vdd		VDDI
	2	Ext Vss		VSSE
	3	3V Ext Vdd		F5
	4	5V Ext Vdd		VDDE
	5	Ext Vss		VSSE
L	6	I/O		C2
L	7	I/O		D3
L	8	I/O (A16)	1	B1
L	9	I/O		E3
L	10	I/O	2	C1
	11	Ext Vss		VSSE
L	12	I/O	3	D1
L	13	I/O		F3
L	14	I/O	4	E2
L	15	I/O		G4
L	16	I/O	5	E1
	17	Ext Vss	6	VSSE
L	18	I/O (A15)	7	F2
L	19	I/O	8	H4
L	20	I/O	9	F1
L	21	I/O	10	H3
L	22	I/O (A14)	11	G2
	23	5V Ext Vdd	12	VDDE
L	24	I/O (A13)	13	G1
L	25	I/O	14	J4
L	26	I/O (A12)	15	H2
L	27	I/O	16	J3
L	28	I/O (A11)	17	H1
	29	Ext Vss	18	VSSE
L	30	I/O	19	J1
L	31	I/O (A10)	20	K4
L	32	I/O	21	K2
L	33	I/O	22	K3
L	34	I/O Clk	23	K1
	35	5V Int Vdd	24	VDDI
	36	Int Vss	25	VSSI

Edge	Pad	Pad Type	Pin Location	
			DK Suffix 208-Pin QFP	HV Suffix 299-Pin PGA
L	37	I/O Clk	26	L3
L	38	I/O	27	L1
L	39	I/O	28	L4
L	40	I/O	29	L2
L	41	I/O	30	M2
	42	Ext Vss	31	VSSE
L	43	I/O	32	M3
L	44	I/O (A9)	33	M1
L	45	I/O	34	M4
L	46	I/O	35	N1
L	47	I/O	36	N2
	48	5V Ext Vdd	37	VDDE
L	49	I/O (A8)	38	N3
L	50	I/O	39	P1
L	51	I/O (A7)	40	N4
L	52	I/O	41	P2
L	53	I/O	42	P3
	54	Ext Vss	43	VSSE
L	55	I/O	44	P4
L	56	I/O		R1
L	57	I/O (A6)	45	R3
L	58	I/O		R2
L	59	I/O	46	R4
	60	Ext Vss		VSSE
L	61	I/O		T3
L	62	I/O (A5)	47	T1
L	63	I/O		T4
L	64	I/O	48	T2
L	65	I/O		U3
	66	F[4]	49	U1
	67	Int Vss		VSSI
	68	F[3]	50	V1
	69	Ext Vss		VSSE
	70	F[2]	51	W1
	71	5V Ext Vdd		VDDE
	72	F[0]	52	V2

Shaded areas indicate Alternate I/O Zones.



Pinouts for MPA1100 (continued)

Edge	Pad	Pad Type	Pin Location	
			DK Suffix 208-Pin QFP	HV Suffix 299-Pin PGA
	73	3V Ext Vdd		R5
	74	Ext Vss		VSSE
	75	Ext Vss	53	VSSE
	76	5V Ext Vdd		VDDE
	77	RESET	54	V3
	78	3V Ext Vdd	55	T6
	79	F[1]	56	U5
B	80	I/O		W2
B	81	I/O		V5
B	82	I/O (A4)	57	Y2
B	83	I/O		V6
B	84	I/O	58	Y3
	85	Ext Vss		VSSE
B	86	I/O	59	Y4
B	87	I/O	60	U7
B	88	I/O		W5
B	89	I/O (A3)	61	V7
B	90	I/O		Y5
	91	Ext Vss		VSSE
B	92	I/O	62	Y6
B	93	I/O	63	U8
B	94	I/O	64	W7
B	95	I/O (A2)	65	V8
B	96	I/O	66	Y7
	97	Ext Vss	67	VSSE
B	98	I/O	68	W8
B	99	I/O (A1)	69	U9
B	100	I/O	70	Y8
B	101	I/O (A0)	71	V9
B	102	I/O	72	W9
	103	5V Ext Vdd	73	VDDE
B	104	I/O	74	Y9
B	105	I/O	75	U10
B	106	I/O (D7)	76	W10
B	107	I/O	77	V10
B	108	I/O Clk	78	Y10
	109	Int Vss	79	VSSI
	110	5V Int Vdd	80	VDDI

Shaded areas indicate Alternate I/O Zones.

Edge	Pad	Pad Type	Pin Location	
			DK Suffix 208-Pin QFP	HV Suffix 299-Pin PGA
B	111	I/O Clk	81	V11
B	112	I/O	82	Y11
B	113	I/O	83	U11
B	114	I/O (D6)	84	W11
B	115	I/O	85	W12
	116	Ext Vss	86	VSSE
B	117	I/O (D5)	87	V12
B	118	I/O	88	Y12
B	119	I/O (D4)	89	U12
B	120	I/O	90	Y13
B	121	I/O (D3)	91	W13
	122	5V Ext Vdd		VDDE
B	123	I/O	92	V13
B	124	I/O	93	Y14
B	125	I/O (D2)	94	U13
B	126	I/O	95	W14
B	127	I/O	96	V14
	128	Ext Vss	97	VSSE
B	129	I/O	98	U14
B	130	I/O		Y15
B	131	I/O (D1)	99	V15
B	132	I/O		Y16
B	133	I/O	100	U15
	134	Ext Vss		VSSE
B	135	I/O		V16
B	136	I/O	101	Y17
B	137	I/O		V17
B	138	I/O	102	Y18
B	139	I/O		V18
	140	MODE[0]	103	Y19
	141	Ext Vss		VSSE
	142	MODE[1]	104	W19
	143	3V Ext Vdd		T16
	144	5V Ext Vdd		VDDE
	145	Probe Pad	NOT BONDED	
	146	Ext Vss		VSSE
	147	5V Ext Vdd	105	VDDE
	148	MODE[2]	106	V19

2



Pinouts for MPA1100 (continued)

2

E d g e			Pin Location	
	Pad	Pad Type	DK Suffix 208-Pin QFP	HV Suffix 299-Pin PGA
	149	5V Int Vdd		U17
	150	MODE[3]	107	W20
	151	Vpp		U18
	152	Clk	108	V20
	153	3V Ext Vdd	109	R16
	154	Ext Vss	110	VSSE
R	155	I/O		T17
R	156	I/O		U20
R	157	I/O (DCLK)	111	T18
R	158	I/O		T19
R	159	I/O	112	R17
	160	Ext Vss		VSSE
R	161	I/O	113	R18
R	162	I/O	114	T20
R	163	I/O		R19
R	164	I/O (D0)	115	R20
R	165	I/O		P17
	166	Ext Vss		VSSE
R	167	I/O	116	P18
R	168	I/O	117	P19
R	169	I/O	118	N17
R	170	I/O	119	P20
R	171	I/O (TDO)	120	N18
	172	Ext Vss	121	VSSE
R	173	I/O (TDI)	122	N19
R	174	I/O	123	N20
R	175	I/O	124	M17
R	176	I/O (TMS)	125	M20
R	177	I/O	126	M18
	178	5V Ext Vdd	127	VDDE
R	179	I/O	128	M19
R	180	I/O (TRSTB)	129	L19
R	181	I/O	130	L17
R	182	I/O	131	L20
R	183	I/O Clk	132	L18
	184	Int Vss	133	VSSI
	185	5V Int Vdd	134	VDDI
R	186	I/O Clk	135	K20

E d g e			Pin Location	
	Pad	Pad Type	DK Suffix 208-Pin QFP	HV Suffix 299-Pin PGA
R	187	I/O	136	K18
R	188	I/O	137	K19
R	189	I/O	138	K17
R	190	I/O	139	J20
	191	Ext Vss	140	VSSE
R	192	I/O	141	H20
R	193	I/O	142	J18
R	194	I/O (TCK)	143	H19
R	195	I/O	144	J17
R	196	I/O	145	G20
	197	5V Ext Vdd	146	VDDE
R	198	I/O	147	G19
R	199	I/O	148	H18
R	200	I/O	149	F20
R	201	I/O	150	H17
R	202	I/O	151	F19
	203	Ext Vss		VSSE
R	204	I/O	152	E20
R	205	I/O		G17
R	206	I/O	153	E19
R	207	I/O		F18
R	208	I/O	154	D20
	209	Ext Vss		VSSE
R	210	I/O	155	C20
R	211	I/O		E18
R	212	I/O		B20
R	213	I/O	156	D18
R	214	I/O		C19
	215	Ext Vss		VSSE
	216	3V Ext Vdd		F16
	217	Ext Vss		VSSE
	218	5V Ext Vdd		VDDE
	219	5V Int Vdd		VDDI
	220	Int Vss		VSSI
	221	Ext Vss	157	VSSE
	222	5V Ext Vdd		VDDE
	223	3V Ext Vdd	158	E15
	224	Ext Vss		VSSE

Shaded areas indicate Alternate I/O Zones.



Pinouts for MPA1100 (continued)

Edge	Pad	Pad Type	Pin Location	
			DK Suffix 208–Pin QFP	HV Suffix 299–Pin PGA
T	225	I/O		C18
T	226	I/O	159	B19
T	227	I/O		C17
T	228	I/O	160	A19
T	229	I/O		C16
T	230	Ext Vss		VSSE
T	231	I/O	161	D15
T	232	I/O		A18
T	233	I/O	162	C15
T	234	I/O		A17
T	235	I/O	163	D14
T	236	Ext Vss	164	VSSE
T	237	I/O	165	C14
T	238	I/O	166	A16
T	239	I/O	167	D13
T	240	I/O	168	A15
T	241	I/O	169	C13
T	242	5V Ext Vdd	170	VDDE
T	243	I/O	171	B13
T	244	I/O	172	A14
T	245	I/O	173	D12
T	246	I/O	174	A13
T	247	I/O	175	C12
T	248	Ext Vss	176	VSSE
T	249	I/O	177	B12
T	250	I/O	178	A12
T	251	I/O	179	D11
T	252	I/O	180	A11
T	253	I/O Clk	181	C11
T	254	5V Int Vdd	182	VDDI
T	255	Int Vss	183	VSSI
T	256	I/O Clk	184	A10
T	257	I/O	185	C10
T	258	I/O	186	B10
T	259	I/O	187	D10
T	260	I/O	188	A9
T	261	5V Ext Vdd	189	VDDE

Shaded areas indicate Alternate I/O Zones.

Edge	Pad	Pad Type	Pin Location	
			DK Suffix 208–Pin QFP	HV Suffix 299–Pin PGA
T	262	I/O	190	B9
T	263	I/O	191	C9
T	264	I/O	192	A8
T	265	I/O	193	D9
T	266	I/O	194	B8
T	267	Ext Vss	195	VSSE
T	268	I/O	196	A7
T	269	I/O	197	C8
T	270	I/O	198	B7
T	271	I/O	199	D8
T	272	I/O	200	A6
T	273	5V Ext Vdd		VDDE
T	274	I/O	201	A5
T	275	I/O		C7
T	276	I/O	202	B5
T	277	I/O		C6
T	278	I/O	203	A4
T	279	Ext Vss		VSSE
T	280	I/O		A3
T	281	I/O (A17)	204	C5
T	282	I/O		A2
T	283	I/O		C3
T	284	I/O	205	B2
T	285	Ext Vss		VSSE
T	286	5V Ext Vdd		VDDE
T	287	Ext Vss	206	VSSE
T	288	3V Ext Vdd	207	E6
T	289	Int Vss	208	VSSI

299 PGA NOTES:

VSSE Plane: A1, A20, B3, B6, B14, B16, B18, C4, D2, D5, D7, D16, D19, E4, E8, E13, E17, G3, G18, H5, H16, J2, J19, N5, N16, T5, T8, T13, U2, U6, U16, U19, V4, W3, W16, W18, Y1, Y20

VSSI Plane: E10, E12, J16, K5, L16, M5, T10, T12

VDDE Plane: B4, B11, B15, B17, D4, D6, D17, E5, E7, E14, E16, F4, F17, G5, G16, P5, P16, T7, T14, U4, W4, W6, W15, W17

VDDI Plane: E9, E11, J5, K16, L5, M16, T9, T11

2



MPA1000 Electrical Specifications

Absolute Maximum Ratings*

Symbol	Parameter	Min	Max	Unit
V _{dd} , V _{ddo}	DC Supply Voltage	-0.5	6.5	V
V _{out}	DC Output Voltage	-0.5	V _{DD} + 0.5	V
V _{in}	DC Input Voltage	-0.5	V _{DD} + 0.5	V
I	DC Current Drain per Pin, Any Single Input or Output		50	mA
T _A	Operating Temperature Range (In Free Air)	Commercial Industrial	0 70 85	°C
T _{stg}	Storage Temperature Range	-65	150	°C

* Maximum Ratings are those values beyond which damage to the device may occur. Functional operation should be restricted to the Recommended Operating Conditions. This device contains circuitry to protect the inputs against damage due to high static voltages or electric fields; however, it is advised that normal precautions be taken to avoid application of any voltage higher than maximum rated voltages to this high impedance circuit.

Recommended Operating Conditions

Symbol	Parameter	Min	Max	Unit	
V _{dd}	DC Supply Voltage	Commercial Industrial	4.75 4.50	5.25 5.50	V
V _{ddo}	Output Supply Voltage	5V Output Supply (5V ext) 3V Output Supply, No I/Os Programmed to 3V 3V Output Supply, 1 or more I/Os Programmed to 3V	V _{dd} V _{dd} 3.0	V _{dd} V _{dd} 3.6	V
V _{IH}	High Level Input Voltage	TTL CMOS	2.0 70	V _{dd} 100	V %V _{ddo}
V _{IL}	Low Level Input Voltage	TTL CMOS	0 0	0.8 20	V %V _{ddo}

DC Electrical Characteristics – Inputs

Symbol	Parameter	Min	Max	Unit	
V _{IH}	High Level Input Voltage	TTL CMOS	2.0 70	V _{dd} 100	V %V _{ddo}
V _{IL}	Low Level Input Voltage	TTL CMOS	0 0	0.8 20	V %V _{ddo}
I _{IN}	Input Leakage Current		-10	10	μA
I _{OZ}	3-State Leakage Current		-10	10	μA
I _{PD}	Pad Pull-Down (When Selected) at V _{in} = V _{ddo}		40	110	μA
I _{PU}	Pad Pull-Up (When Selected) at V _{in} = 0V		20	200	μA
C _{IN}	Input Capacitance (Sample Tested)	PGA Package Plastic Packages		15 10	pF

DC Electrical Characteristics – Outputs (4.5 < 5_Ext_V_{dd} < 5.5; -40°C < T_A < +85°C; V_{dd} = 5V; 3.0 < 3V_Ext_V_{dd} < 3.6)

Symbol	Parameter	Min	Max	Unit	
V _{OH}	High Level Output Voltage	DPLD_OPLEVEL=5V; DPLD_OPDRIVE=high; I _{OH} =6mA DPLD_OPLEVEL=5V; DPLD_OPDRIVE=low; I _{OH} =4mA DPLD_OPLEVEL=3V; DPLD_OPDRIVE=high; I _{OH} =6mA DPLD_OPLEVEL=3V; DPLD_OPDRIVE=low; I _{OH} =4mA (Note 6.) DPLD_OPLEVEL=5V; DPLD_OPDRIVE=high; I _{OH} =44mA	2.4 2.4 2.1 2.1 2.2		V
V _{OL}	High Level Output Voltage	DPLD_OPLEVEL=5V; DPLD_OPDRIVE=high; I _{OH} =-6mA DPLD_OPLEVEL=5V; DPLD_OPDRIVE=low; I _{OH} =-4mA DPLD_OPLEVEL=3V; DPLD_OPDRIVE=high; I _{OH} =-6mA DPLD_OPLEVEL=3V; DPLD_OPDRIVE=low; I _{OH} =-4mA (Note 6.) DPLD_OPLEVEL=5V; DPLD_OPDRIVE=high; I _{OH} =-95mA		0.4 0.4 0.4 0.4 1.4	V

6. Not available on all family members. Please ask your sales representative for more information.



MPA1000 Primary Clock Characteristics

Symbol	Parameter	Min	Typ	Max	Unit
T _{ckin}	Primary Clock Pad to Register Delay		5.6		ns
T _{cks}	Primary Clock Skew			1.0	ns
T _{cwh}	Clock High Time		2.0		ns
T _{cwl}	Clock Low Time		2.0		ns

MPA1000 JTAG Clock Characteristics

Symbol	Parameter	Min	Typ	Max	Unit
F _{cjtag}	Shift Clock Frequency			16	MHz

2

AC Characteristics (These are Maximum values based on worst case operating points for an MPA1036.)

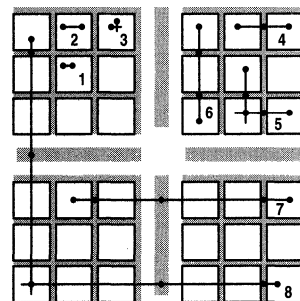
Symbol	Parameter	Speed Grade						Unit
		2I	4I	6I	2	4	6	
T _{cn}	Core Cell AND (Note 7.)	1.1	1.3	1.4	1.0	1.2	1.3	ns
T _{cx}	Core Cell XOR (Note 7.)	1.7	2.0	2.2	1.6	1.8	2.1	ns
T _{ip}	Input Pad Delay	2.7	3.1	3.5	2.6	2.9	3.3	ns
T _{li}	Local Interconnect (Note 8.)	NEG	NEG	NEG	NEG	NEG	NEG	ns
T _{mb}	Medium Bus	0.8	0.9	1.0	0.8	0.9	1.0	ns
T _{mbt}	Medium Bus Turn	2.1	2.4	2.7	2.0	2.2	2.5	ns
T _g	Global Bus (Same Quadrant)	1.4	1.7	1.9	1.4	1.6	1.8	ns
T _{gg}	Global Bus (Adjacent Quadrant)	3.3	3.8	4.3	3.2	3.6	4.0	ns
T _{xt}	X-Bus Turn	1.4	1.6	1.8	1.3	1.5	1.7	ns
T _{pb}	Peripheral Bus (Generic)	6.2	7.2	8.1	5.9	6.7	7.6	ns
T _{iwo}	Internal Wired-OR (Full Device)	8.1	9.5	10.6	7.8	8.8	10.0	ns

7. Includes routing.

8. Value is negligible – value is lumped into core cell delays.

Example Delay Paths (These are Maximum values based on worst case operating points for an MPA1036.)

Path	Parameter	Speed Grade						Unit
		2I	4I	6I	2	4	6	
Path 1	Local	1.1	1.3	1.4	1.0	1.2	1.3	ns
Path 2	Medium	1.9	2.2	2.4	1.8	2.1	2.3	ns
Path 3	Medium Turn	3.1	3.6	4.0	3.0	3.5	3.8	ns
Path 4	M-Port-M	3.0	3.5	3.9	2.9	3.4	3.7	ns
Path 5	M-Xturn-M	6.1	7.1	7.7	5.9	6.6	7.4	ns
Path 6	M-G-M	5.9	6.8	7.6	5.6	6.5	7.2	ns
Path 7	M-G-IQ-G-M	7.6	8.8	9.8	7.2	8.3	9.3	ns
Path 8	M-G-IQ-G-Xturn-G-IQ-G-M	14.4	16.8	18.8	13.8	15.8	17.8	ns



M = Medium; G = Global; IQ = Inter-Quadrant Switch. All paths include 1 cell delay. Path is from cell input to next cell input.



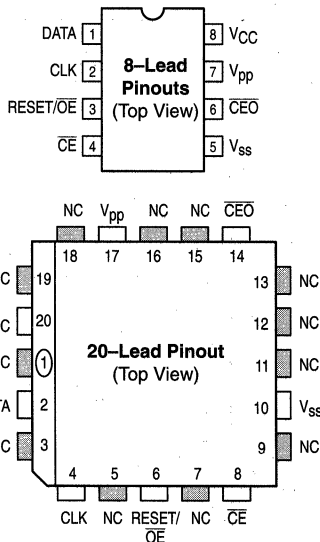
MPA17000 Serial EPROMs

The MPA17128 and MPA1765 are serial OTP EPROMs. They provide a compact, low pin count, non-volatile configuration store for the MPA1000 devices.

MPA17000 devices can be cascaded for increased memory capacity when needed. They are available in the standard 8-pin plastic DIP (P suffix), 8-pin SOIC (D suffix) and 20-pin PLCC (FN suffix) packages.

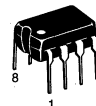
2

- Configuration EPROM for MPA1000 Devices
- Voltage Range — 4.5 to 6.0V
- Maximum Read Current of 10mA
- Standby Current of 10 μ A, Typical
- Industry Standard Synchronous Serial Interface
- Full Static Operation
- 10MHz Maximum Clock Rate at 5.0V
- Programmable Polarity on Hardware Reset
- Programs With Industry Standard Programmers
- Electrostatic Discharge Protection > 2000 Volts
- 8-Pin PDIP and SOIC; 20-Pin PLCC Packages
- Commercial (0 to +70°C) and Industrial (-40 to +85°C)



MPA17128 MPA1765

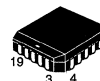
128K, 64K SERIAL EPROM



P SUFFIX
PLASTIC PACKAGE
CASE 626-05



D SUFFIX
PLASTIC SOIC PACKAGE
CASE 751-05



FN SUFFIX
PLCC PACKAGE
CASE 775-02

PIN NAMES

Pins	Function
DATA	Data I/O
CLK	Clock
RESET/OE	Reset Input and Output Enable
CE	Chip Enable Input
VSS	Ground
CEO	Chip Enable Output
Vpp	Programming Voltage Supply
VCC	+4.5 to 6.0V Power Supply
NC	Not Connected



Table 2–16. MAXIMUM RATINGS*

Parameter	Value	Unit
V _{CC} and Input Voltages W.R.T. V _{SS}	-6.0 to V _{DD} + 0.6	V
V _{PP} Voltage W.R.T. V _{SS} During Programming	-0.6 to +14.0	V
Output Voltage W.R.T. V _{SS}	-0.6 to V _{CC} + 0.6	V
Storage Temperature Range	-65 to +150	°C
Ambient Temperature With Power Applied	-65 to +125	°C
Soldering Temperature of Leads (10 Seconds)	+300	°C
ESD Protection on All Leads	≥2	kV

NOTE: Maximum Ratings are those values beyond which damage to the device may occur. Functional operation should be restricted to the Recommended Operating Conditions.

Table 2–17. DC CHARACTERISTICS (V_{CC} = 4.5 to 6.0V; Commercial (C) T_A = 0 to +70°C; Industrial (I) T_A = -40 to +85°C)

Symbol	Characteristic	Min	Max	Unit	Condition
V _{IH}	Input Voltage High DATA, \overline{CE} , \overline{CEO} , Reset	2.0	V _{CC}	V	
V _{IL}	Input Voltage Low DATA, \overline{CE} , \overline{CEO} , Reset	-0.3	0.8	V	
V _{OH}	Output Voltage High DATA, \overline{CE} , \overline{CEO} , Reset	3.86 2.40		V	I _{OH} = -4mA; V _{CC} ≥ 4.5V
V _{OL}	Output Voltage Low DATA, \overline{CE} , \overline{CEO} , Reset		0.32	V	I _{OL} = 4.0mA
I _{LI}	Input Leakage Current	-10	10	μA	V _{IN} = 0.1V to V _{CC}
I _{LO}	Output Leakage Current	-10	10	μA	V _{OUT} = 0.1V to V _{CC}
C _{INT}	Internal Capacitance (All Inputs/Outputs)		10	pF	V _{CC} = 5.0V (Note 1); T _A = 25°C; f _{clk} = 1MHz
I _{CC} Read	Operating Current		10	mA	V _{CC} = 6.0V; CLK = 10MHz
I _{CCS}	Standby Current		500	μA	V _{CC} = 6.0V

1. This parameter is initially characterized and not 100% tested.

Applications Information

DATA

Three-state DATA output for reading and function as the input during programming.

CLOCK

Clock input. Used to increment the internal address and bit counters for reading and programming.

RESET/ \overline{OE}

Reset and Output Enable input. A Low level both the \overline{CE} and RESET/ \overline{OE} inputs enables the data output driver. A High level on RESET/ \overline{OE} resets both the address and bit counters. In the MPA17128, the logic polarity of this input is programmable as either RESET/ \overline{OE} or \overline{OE} /RESET. This document describes the pin as RESET/ \overline{OE} although the opposite polarity is also possible, this option is defined and set at device program time.

\overline{CE}

Chip Enable input. Used for device selection. A Low level on both \overline{CE} and \overline{OE} enables the data output driver. A High level on \overline{CE} disables both the address and bit counters and forces the device into a low power mode.

\overline{CEO}

Chip Enable Out output. This signal is asserted Low on the clock cycle following the last bit read from the memory. It will stay Low as long as \overline{CE} and \overline{OE} are both Low. It will then follow \overline{CE} until \overline{OE} goes High. Thereafter \overline{CEO} will stay High until the entire PROM is read again. This pin also used to sense the status of RESET polarity when program mode is entered.

V_{PP}

Programming Voltage Supply. Used to enter programming mode (+10V) and to program the memory (+13V) Must be connected directly to V_{CC} for normal Read operation. No overshoot above +15.5V permitted.



USING THE MPA17000 WITH MPA1000 DEVICES

Connections between the MPA devices and the Serial EPROMs are:

- The DATA output of the MPA17000 drives D0 (data in).
- The CLK input of the MPA17000 is driven by the data clock DCLK output.
- MPA17000s can be cascaded using the \overline{CEO} output to drive the \overline{CE} input of the next MPA17000.
- For normal Read operations V_{pp} must be connected to V_{CC} .

Do not leave V_{pp} open.

The connections between an MPA device and an MPA17000 device are shown in Figure 2-48. The MPA D[0] line is connected to the MPA17000 CLK. At power-up or upon reconfiguration, the MEMCE signal goes Low, enabling the MPA17000 DATA output. During the configuration process, D[0] reads data from the MPA17000 on every rising DCLK edge. The MEMCE signal goes High at the end of configuration and resets the internal address counters of the MPA17000.

2

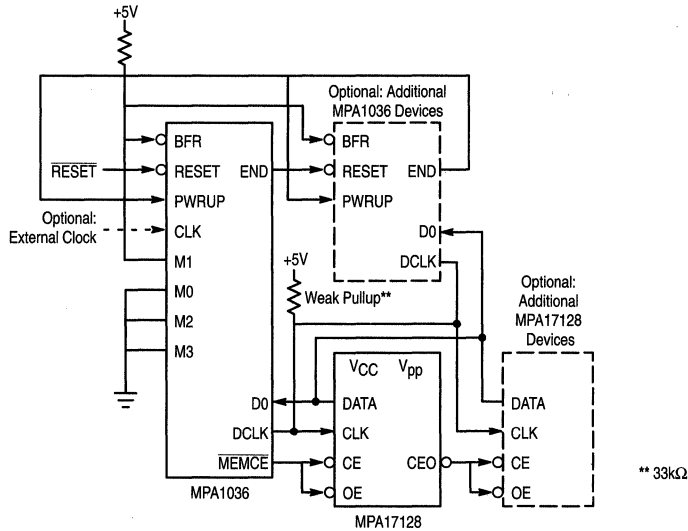


Figure 2-48. MPA1036 Configuration Using MPA17128 Serial EPROM

Cascading Serial Configuration PROMs

Cascading MPA17000s provide additional memory for multiple MPA1000s or for MPA1000s requiring larger configuration memories.

When the last bit from the first MPA17000 is read, the next clock signal to the MPA17000 asserts its \overline{CEO} output Low and disables its DATA line. The second MPA17000 recognizes the Low level on its \overline{CE} input and enables its DATA output. (See Figure 2-48).

Additional logic may be required if cascaded memories are so large that the rippled chip enable is not fast enough to activate successive MPA17000s.

STANDBY MODE

The MPA17128 enters a low power standby mode whenever \overline{CE} is High. In standby mode, the MPA17000 consumes less than 500 μ A of current. The output will remain in a high impedance state regardless of the state of the \overline{OE} input.

PROGRAMMING MODE

Programming mode is entered by holding V_{pp} High for at least two clock edges and is exited by removing power from the device or by a Low on both \overline{CE} and \overline{OE} . Figure 2-51 through Figure 2-56 shows the programming algorithm.

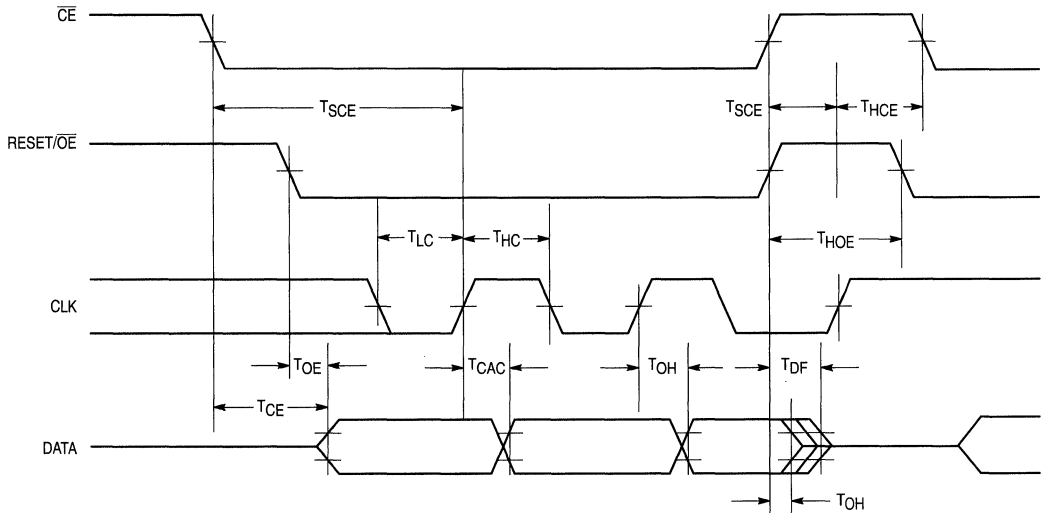
MPA17128 RESET POLARITY

The MPA17128 lets the user choose the reset polarity as either RESET/ \overline{OE} or \overline{OE} /RESET. Any third-party commercial programmer should prompt the user for the desired reset polarity.

The programming of the overflow word should be handled transparently by the PROM programmer; it is mentioned here as supplemental information only.

The polarity is programmed into the first overflow word location, max address+1. 00000000 in these locations makes the reset active Low, FFFFFFFF in these locations makes the reset active High. The default condition is RESET active High.





2

Figure 2-49. AC Characteristics Over Operating Conditions

Table 2-18. AC OPERATING CONDITIONS

Symbol	Parameter	Limit $4.5V \leq V_{CC} \leq 6.0V$		Unit	Condition
		Min	Max		
T_{OE}	\overline{OE} to Data Delay		45	ns	
T_{CE}	\overline{CE} to Data Delay		50	ns	
T_{CAC}	CLK to Data Delay		60	ns	
T_{OH}	Data Hold From \overline{OE} , \overline{CE} or CLK	0		ns	
T_{DF}	\overline{OE} or \overline{CE} to Data Float Delay		50	ns	Note 1
T_{LC}	CLK Low Time	25		ns	Note 2
T_{HC}	CLK High Time	25		ns	Note 2
T_{SCE}	\overline{CE} Setup Time to CLK (To Guarantee Proper Counting)	25		ns	
T_{HCE}	\overline{CE} Hold Time to CLK (To Guarantee Proper Counting)	0		ns	Note 2
T_{HOE}	\overline{OE} High Time (Guarantees Counters are Reset)	20		ns	Note 2
CLK_{max}	Clock Frequency		10	MHz	

1. Float delays are measured with minimum tester AC load and maximum DC load.

2. Guarantee by design, not tested.



2

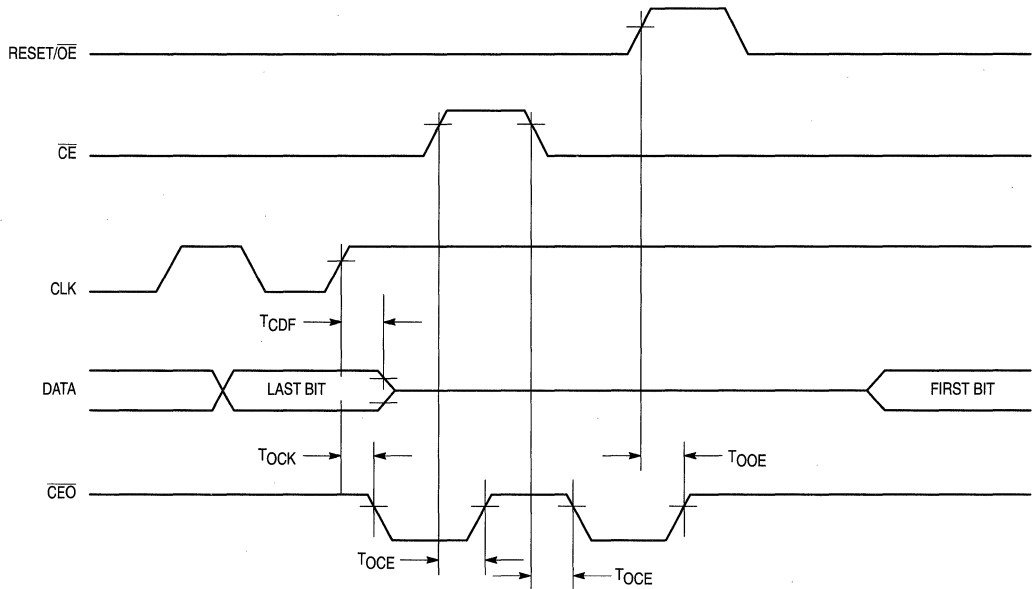


Figure 2-50.

Table 2-19.

Symbol	Parameter	Limit 4.5V ≤ V _{CC} ≤ 6.0V		Unit	Condition
		Min	Max		
T _{CDF}	CLK to Data Float Delay		50	ns	
T _{OCK}	CLK to CEO Delay		40	ns	
T _{OCE}	CE to CEO Delay		40	ns	
T _{OOE}	RESET/OE to CEO Delay		40	ns	



Table 2–20. PIN ASSIGNMENTS IN THE PROGRAMMING MODE

Pin Name	DIP	PLCC	I/O	Function
DATA	1	2	I/O	The rising edge of the clock shifts a data word in or out of the PROM one bit at a time.
CLK	2	4	I	Clock input. Used to increment the internal address/word counter for reading and programming operation.
RESET/ \overline{OE}	3	6	I	The rising edge of CLK shifts a data word into the PROM when \overline{CE} and \overline{OE} are High; it shifts a data word out of the PROM when \overline{CE} is Low and \overline{OE} is High. The address/word counter is incremented on the rising edge of CLK while \overline{CE} is held High and \overline{OE} is held Low. Note: Any modified polarity of the RESET/ \overline{OE} pin is ignored in the programming mode.
\overline{CE}	4	8	I	The rising edge of CLK shifts a data word into the PROM when \overline{CE} and \overline{OE} are High; it shifts a data word out of the PROM when \overline{CE} is Low and \overline{OE} is High. The address/word counter is incremented on the rising edge of CLK while \overline{CE} is held High and \overline{OE} is held Low.
GND	5	10	—	Ground pin.
\overline{CEO}	6	14	O	The polarity of the RESET/ \overline{OE} pin can be read by sensing the \overline{CEO} pin. Note: The polarity of the RESET/ \overline{OE} pin is ignored while in the programming mode. In final verification, this pin must be monitored to go Low one clock cycle after the last data bit has been read.
V _{PP}	7	17	—	Programming Voltage Supply. Programming mode is entered by holding \overline{CE} and \overline{OE} High and V _{PP} at V _{PP1} for two rising clock edges and then lowering V _{PP} to V _{PP2} for one more rising clock edge. A word is programmed by strobing the device with V _{PP} for the duration T _{PGM} V _{PP} must be tied to V _{CC} for normal operation.
V _{CC}	8	20	—	+5 V power supply input.

2

Table 2–21. DC PROGRAMMING SPECIFICATIONS

Symbol	Parameter	Limit		Unit	Condition
		Min	Max		
V _{CCP}	Supply Voltage During Programming	5.0	6.0	V	
V _{IL}	Input Voltage Low	0	0.5	V	
V _{IH}	Input Voltage High	2.4	V _{CC}	V	
V _{OL}	Output Voltage Low		0.4	V	
V _{OH}	Output Voltage High	3.7		V	
V _{PP1}	Programming Voltage	12.5	13.5	V	Note 1
V _{PP2}	Programming Mode Access Voltage	V _{CCP}	V _{CCP} + 1	V	
I _{PPP}	Supply Current in Programming Mode		100	mA	
I _L	Input or Output Leakage Current	–10	10	μA	
V _{CCL}	First Pass Supply Voltage Low for Final Verification	2.8	3.0	V	
V _{CCH}	Second Pass Supply Voltage High for Final Verification	6.0	6.2	V	

1. No overshoot is permitted on this signal. V_{PP} must not be allowed to exceed V_{PP1} max.



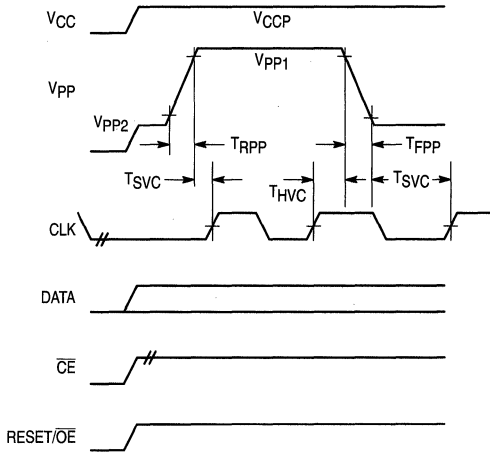


Figure 2-51. Enter Programming Mode

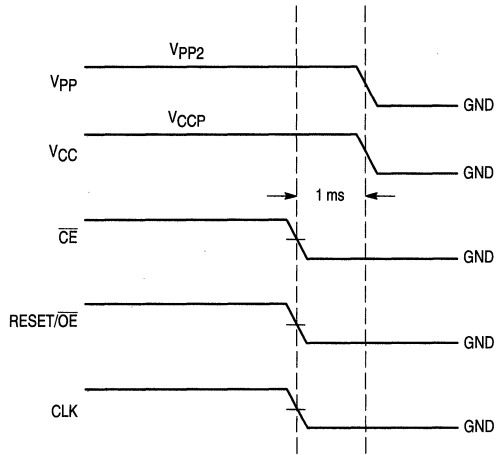


Figure 2-52. Exit Programming Mode

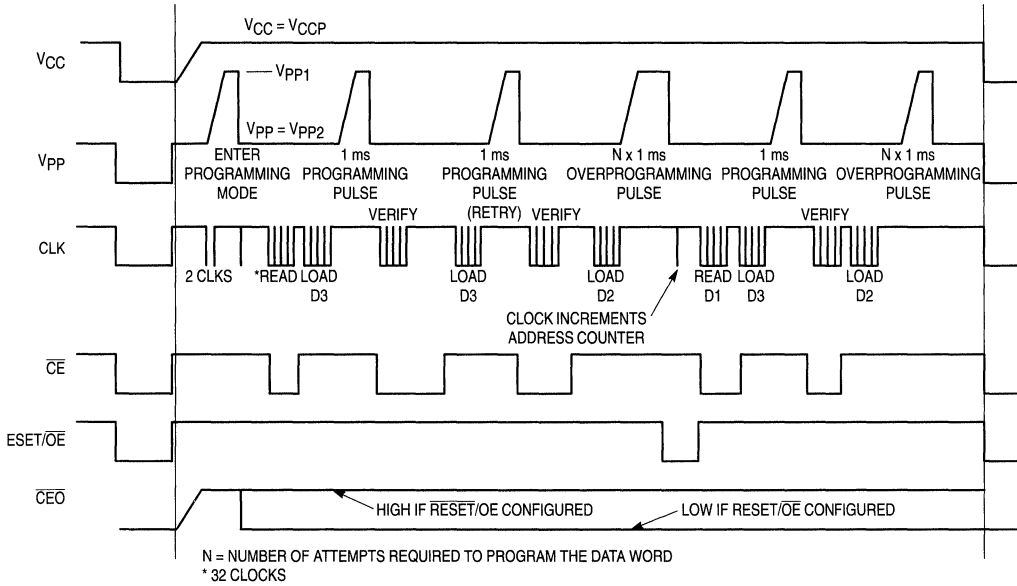
2

Table 2-22. AC PROGRAMMING SPECIFICATIONS

Symbol	Parameter	Limit		Unit	Condition
		Min	Max		
T_{RPP}	Rise Time of V_{pp} (10 to 90%)	50		ns	
T_{FPP}	Fall Time of V_{pp} (90 to 10%)	50		ns	
T_{PGM}	V_{pp} Programming Pulse Width	0.95	1.05	ms	
T_{SVC}	V_{pp} Setup to CLK for Entering Programming Mode	100		ns	
T_{HVC}	V_{pp} Hold from CLK for Entering Programming Mode	300		ns	
T_{SDP}	Data Setup to CLK for Programming	50		ns	
T_{HDP}	Data Hold from CLK for Programming	0		ns	
T_{SCC}	\overline{CE} Setup to CLK for Programming/Verifying	100		ns	Note 1
T_{HCC}	\overline{CE} Hold from CLK for Programming/Verifying	200		ns	
T_{SCV}	\overline{CE} Setup to V_{pp} for Programming	100		ns	
T_{HCV}	\overline{CE} Hold from V_{pp} for Programming	50		ns	
T_{SIC}	\overline{OE} Setup to CLK for Incrementing Address Counter	100		ns	
T_{HIC}	\overline{OE} Hold from CLK for Incrementing Address Counter	0		ns	
T_{CAC}	CLK to Data Valid		400	ns	
T_{OH}	Data Hold from CLK	0		ns	
T_{CE}	\overline{CE} Low to Data Valid		250	ns	

1. While in programming mode, \overline{CE} should only be changed while CLK is High and has been High for 200ns.





2

Figure 2-53. Programming Cycle Overview

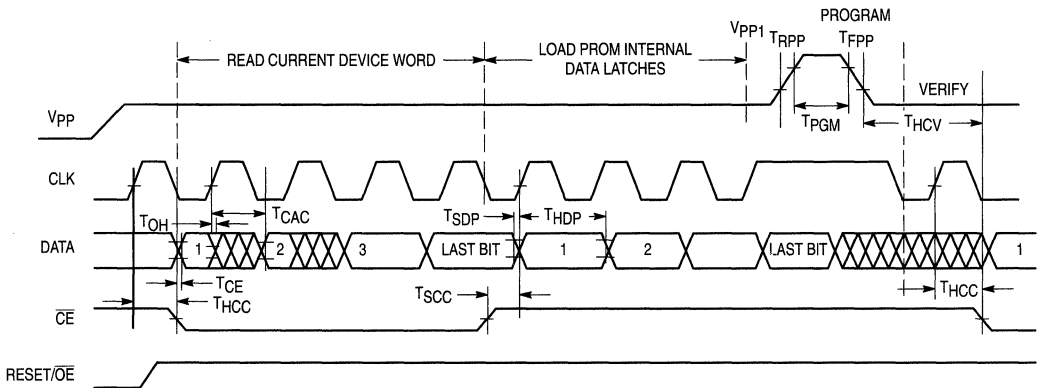


Figure 2-54. Details of Read/Program/Verify Cycle



2

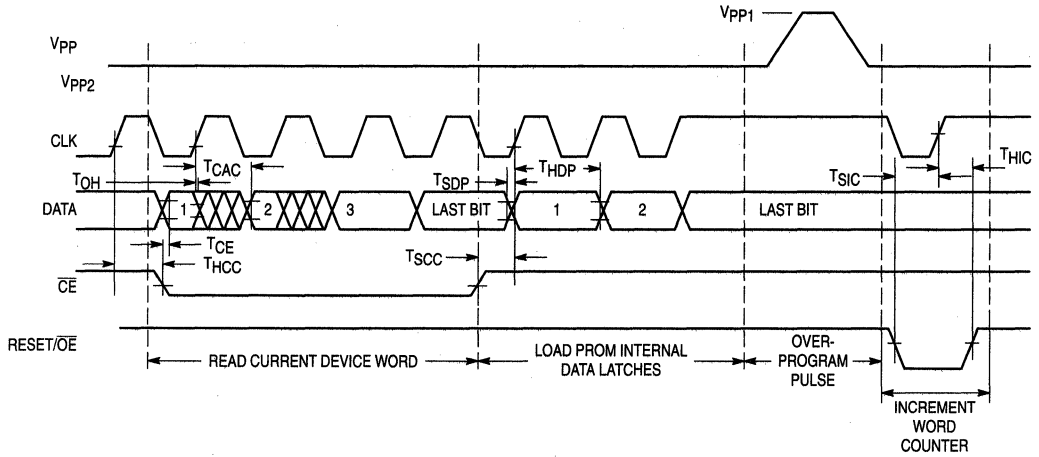


Figure 2-55. Overprogramming Detail



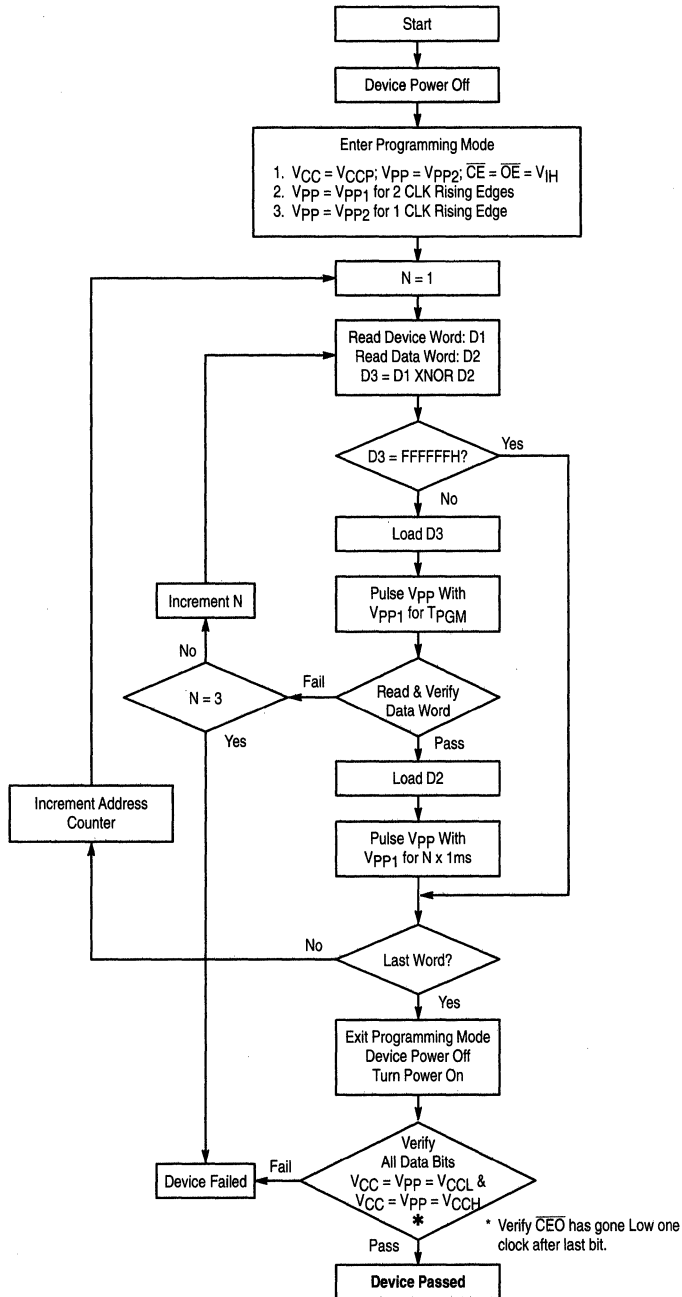


Figure 2-56. MPA17128 Programming Spec



Advance Information

MPA17000 Serial EEPROM

The MPA17C256 is an easy to use and cost effective serial configuration memory ideally suited for use with today's popular SRAM based FPGAs. The MPA17C256 is available in 8-pin PDIP and 20-pin SOIC and PLCC packages, adhering to industry standard pinouts. The device interfaces downstream FPGA(s) with a very simple enable, clock and data interface. The MPA17C256 is reprogrammable with no need for a higher programming "super voltage"; it may even be reprogrammed on board. The MPA17C256 also has user programmable RESET/OE polarity.

2

- EE Programmable 262,144 x 1 bit Serial Memories Designed to Store Configuration Programs for FPGAs
- Simple Interface to SRAM FPGAs
- Cascadable to Support Additional Configurations or Future Higher Density FPGAs
- Low Power CMOS EEPROM Process
- Programmable Reset Polarity
- Available in Space Efficient 8-Pin PDIP, 20-Pin SOIC and 20-Pin PLCC Packages
- In-System Programmable via 2-Wire Bus

Controlling the MPA17C256 Serial EEPROM

Most connections between the FPGA device and the Serial EEPROM are simple and self-explanatory:

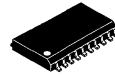
- The DATA output of the MPA17C256 drives DIN of the FPGA devices
- The master FPGA DCLK output drives the CLK input of the MPA17C256
- The CEO output of the first MPA17C256 drives the CE input of the next MPA17C256 in a cascade chain of EEPROMs.
- $\overline{\text{SER_EN}}$ must be connected to VCC
- $\overline{\text{CE}}$ enables the chip and is required to enable the DATA output pin
- RESET/OE is chip reset and is part of the DATA output enable structure

The simplest connection to a Motorola Programmable array (MPA) is shown below. In this configurations, the $\overline{\text{MEMCE}}$ output of the MPA enables the MPA17C256 and takes it out of reset. Shortly after, the first bit of configuration data will appear on DATA. Subsequent rising edges of DCLK bring out the next bits. For more complex connections, including cascading of EEPROMs and cascading of MPA devices, please consult the MPA databook (DL201/D).

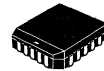
MPA17C256



P SUFFIX
8-LEAD PLASTIC PACKAGE
CASE 626-05



DW SUFFIX
20-LEAD PLASTIC SOIC WIDE PACKAGE
CASE 751D-04



FN SUFFIX
20-LEAD PLCC PACKAGE
CASE 775-02

PIN NAMES

Pins	Function
DATA	Data I/O
CLK	Clock
RESET/OE	Reset Input and Output Enable
CE	Chip Enable Input
VSS	Ground
CEO	Chip Enable Output
SER_EN	Programming Enable
VCC	+4.5 to 6.0V Power Supply
NC	Not Connected

This document contains information on a new product. Specifications and information herein are subject to change without notice.



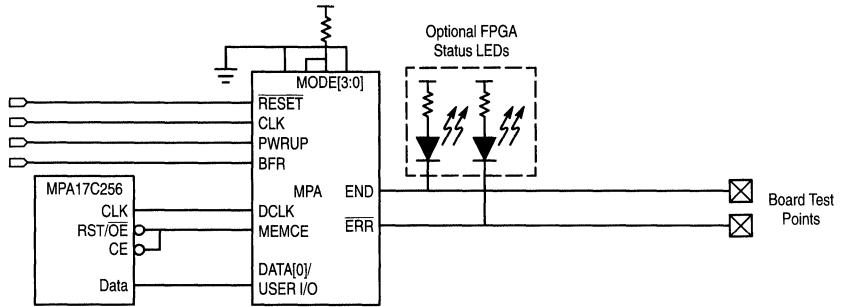


Figure 2-57. BFR Mode 2: 1-Bit (Serial) Data, External Address, External Clock

2

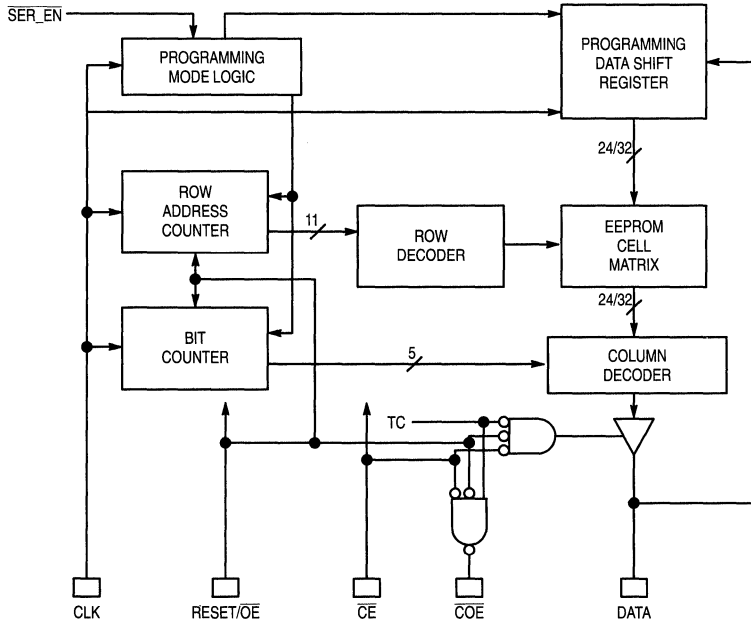


Figure 2-58. Block Diagram



2

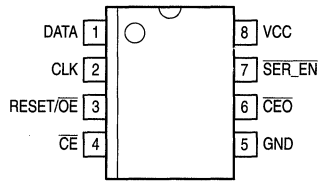


Figure 2-59. 8-Lead DIP Pinout

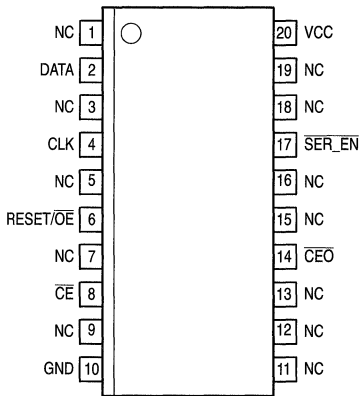


Figure 2-60. 20-Lead SOIC Pinout

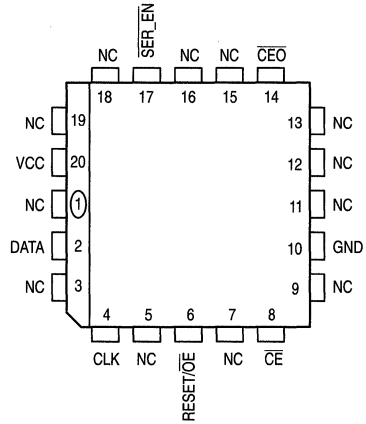


Figure 2-61. 20-Lead PLCC Pinout



Table 2-23. PIN DESCRIPTIONS

PLCC/ SOIC	DIP	Name	I/O	Description
2	1	DATA	I/O	Three-State DATA output for reading. Input/Output pin for programming
4	2	CLK	I	Clock Input. Rising edge used to increment the internal address and bit counter for reading and programming
6	3	RESET/OE	I	RESET/Output Enable input (when SER_EN is High). A Low level on both the CE and RESET/OE inputs enables the data output driver. A High level on RESET/OE resets both the address and bit counters. The logic polarity of this input is programmable as either RESET/OE or RESET/OE. This document describes the pin as RESET/OE.
		RESET Polarity Selected	I	RESET Polarity Select Input. During programming, when CE is High, this input is used to determine the polarity of the pin when SER_EN is High.
		WP	I	Write Protect (WP) input. When WP is Low, the entire memory can be written. When WP is enabled (High), the lowest 1/4 of the memory cannot be written; i.e., 64K in MPA17C256. Note that when WP is High, the chip will still acknowledge the receipt of data, but it will not write it into memory.
8	4	CE	I	Chip Enable input. Used for device selection only when SER_EN is High. A Low level on both CE and OE enables the data output driver. A High level on CE disables both the address and bit counters and forces the device into a low power mode. Note this pin will not enable/disable the device in 2-wire Serial mode (i.e., when SER_EN is Low). During programming, and when CE is Low, the main array is read and written. When CE is High, the main array is deselected and a Serial WRITE operation will change the polarity of the RESET pin.
10	5	GND	—	Ground Pin
14	6	CEO	O	Chip Enable Out output. This signal is asserted Low on the clock cycle following the last bit read from the memory. It will stay Low as long as CE and OE are both low. It will then follow CE until OE goes High. Thereafter, CEO will stay High until the entire PROM is read again and senses the status of RESET polarity.
		A2	I	Device selection input, A2. Used to enable (select) the device during programming. When SER_EN is Low, this pin MUST be at either a logic level '1' or '0' (i.e., not 3-state) and the A2 contents of the Device Address must match the condition of the pin for the device to be selected.
17	7	SER_EN	I	Serial enable is normally high during FPGA loading operations. Bringing SER_EN Low, enables the two wire serial interface mode for programming.
20	8	VCC	—	+5V Power Supply input

2

Table 2-24. MAXIMUM RATINGS*

Symbol	Parameter	Value	Unit
V _{CC}	DC Supply Voltage (Referenced to GND)	-0.5 to +7.0	V
	Voltage Applied to Output in High Output State	-0.1 to V _{CC} +0.5V	V
T _A	Operating Temperature Range (In Free-Air)	-55 to +125	°C
T _{STG}	Storage Temperature Range	-65 to +150	°C
T _{SOL}	Maximum Soldering Temperature (10s @ 1/16in)	260	°C
ESD	RZAP = 1.5K, CZAP = 100pF	2000	V

* Maximum Ratings are those values beyond which damage to the device may occur. Functional operation should be restricted to the Recommended Operating Conditions.

Table 2-25. RECOMMENDED OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Unit
V _{CC}	Supply Voltage Relative to Ground	Commercial (-0 to +70°C) Industrial (-40 to +85°C)	4.75 5.25	V
		4.50	5.50	



Table 2-26. DC CHARACTERISTICS OVER OPERATING CONDITIONS

Symbol	Parameter	Min	Max	Unit
V _{IH}	High Level Input Voltage	2.0	V _{CC}	V
V _{IL}	Low Level Input Voltage	0	0.8	V
V _{OH}	High Level Output Voltage	Commercial Industrial		V
V _{OL}	Low Level Output Voltage	Commercial Industrial	0.32 0.37	V
I _{CCA}	Supply Current, Active Mode		10	mA
I _{IL}	Input/Output Leakage Current (V _{in} = V _{CC} or GND)	-10	10	μA
I _{CCS}	Supply Current, Standby Mode	Commercial Industrial	1 2	mA

2

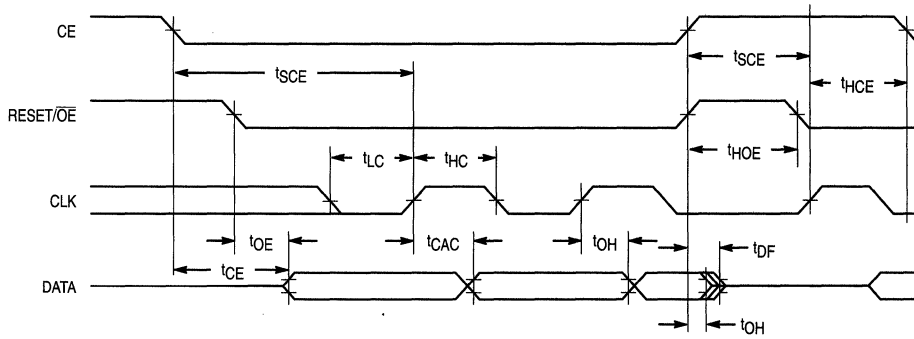


Figure 2-62. AC Characteristics Over Operating Conditions

Table 2-27. AC CHARACTERISTICS OVER OPERATING CONDITIONS (Note 1.)

Symbol	Parameter	Commercial		Industrial		Unit
		Min	Max	Min	Max	
t _{OE}	\overline{OE} to Data Delay		110		150	ns
t _{CE}	\overline{CE} to Data Delay		50		50	ns
t _{CAC}	CLK to Data Delay		50		55	ns
t _{OH}	Data Hold from \overline{CE} , \overline{OE} , or CLK	0		0		ns
t _{DF}	\overline{CE} or \overline{OE} to Data Float Delay (Note 2.)		50		50	ns
t _{LC}	CLK Low Time	30		35		ns
t _{HC}	CLK High Time	30		35		ns
t _{SCE}	\overline{CE} Setup Time to CLK (To Guarantee Proper Counting)	45		50		ns
t _{HCE}	\overline{CE} Hold Time to CLK (To Guarantee Proper Counting)	0		5		ns
t _{HOE}	\overline{OE} High Time (Guarantees Counter Is Reset)	50		60		ns
f _{max}	Maximum Input Clock Frequency		10		10	MHz

1. AC test load = 50pF.

2. Float delays are measured with 5pF AC loads. Transition is measured ±500mV from steady state active levels.



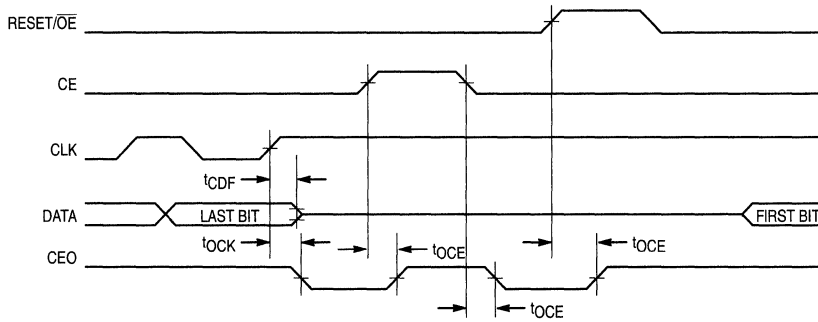


Figure 2-63. AC Characteristics Over Operating Conditions When Cascading

2

Table 2-28. AC CHARACTERISTICS OVER OPERATING CONDITIONS WHEN CASCADING

Symbol	Parameter	Commercial		Industrial		Unit
		Min	Max	Min	Max	
t_{CDF}	CLK to Data Float Delay		50		50	ns
t_{OCK}	CLK to \overline{CEO} Delay		65		75	ns
t_{OCE}	CE to \overline{CEO} Delay		55		60	ns
t_{OOE}	RESET/ \overline{OE} to \overline{CEO} Delay		55		55	ns

Table 2-29. DC CHARACTERISTICS ($V_{CC} = 5.0V \pm 5\%$)

Symbol	Parameter	Min	Typ	Max	Unit	Condition
V_{CC}	Supply Voltage	4.75	5.00	5.25	V	
I_{CC}	Supply Current		2.0	5.0	mA	$V_{CC} = 5V$
I_{LL}	Input Leakage Current		0.10	3.00	μA	$V_{in} = V_{CC}$ or V_{SS}
I_{LO}	Output Leakage Current		0.05	3.00	μA	$V_{out} = V_{CC}$ or V_{SS}
V_{IH}	High Level Input Voltage	$V_{CC} \times 0.7$		$V_{CC} + 0.5$	V	
V_{IL}	Low Level Input Voltage	-0.5		0.4	V	
V_{OL}	Output Low Level Voltage			0.4	V	$I_{OL} = 3mA$

Table 2-30. DC CHARACTERISTICS ($V_{CC} = 3.3V \pm 10\%$)

Symbol	Parameter	Min	Typ	Max	Unit	Condition
V_{CC}	Supply Voltage	3.0	3.3	3.6	V	
I_{CC}	Supply Current		2.0	3.0	mA	$V_{CC} = 3.6V$
I_{LL}	Input Leakage Current		0.10	3.00	μA	$V_{in} = V_{CC}$ or V_{SS}
I_{LO}	Output Leakage Current		0.05	3.00	μA	$V_{out} = V_{CC}$ or V_{SS}
V_{IH}	High Level Input Voltage	$V_{CC} \times 0.7$		$V_{CC} + 0.5$	V	
V_{IL}	Low Level Input Voltage	-0.5		0.2	V	
V_{OL}	Output Low Level Voltage			0.4	V	$I_{OL} = 2.1mA$



2

Table 2-31. AC CHARACTERISTICS ($V_{CC} = 5.0V \pm 5\%$)

Symbol	Parameter	Min	Max	Unit
f_{Clock}	Clock Frequency, Clock		400	kHz
t_{Low}	Clock Pulse Width Low	1.2		μs
t_{High}	Clock Pulse Width High	0.8		μs
t_{AA}	Clock Low to Data Out Valid	0.1	0.9	μs
t_{Buf}	Time the Bus Must Be Free Before a New Transmission Can Start	1.2		μs
t_{HST}	Start Hold Time	0.6		μs
t_{SST}	Start Setup Time	0.6		μs
t_{HDA}	Data In Hold Time	0		μs
t_{SDA}	Data In Setup Time	100		ns
t_r	Input Rise Time		0.3	μs
t_f	Input Fall Time		300	ns
t_{SSSTP}	Stop Setup Time	0.6		μs
t_{DH}	Data Out Hold Time	50		ns
t_{WR}	Write Cycle Time		10	ms

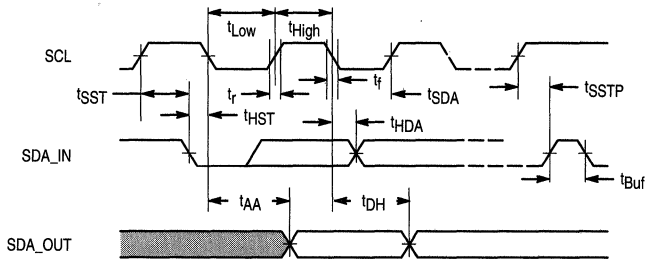


Figure 2-64. Serial Data Timing Diagram

Table 2-32. AC CHARACTERISTICS ($V_{CC} = 3.3V \pm 10\%$)

Symbol	Parameter	Min	Max	Unit
f_{Clock}	Clock Frequency, Clock		100	kHz
t_{Low}	Clock Pulse Width Low	4.0		μs
t_{High}	Clock Pulse Width High	4.0		μs
t_{AA}	Clock Low to Data Out Valid	0.1	1.0	μs
t_{Buf}	Time the Bus Must Be Free Before a New Transmission Can Start	4.5		μs
t_{HST}	Start Hold Time	2.0		μs
t_{SST}	Start Setup Time	2.0		μs
t_{HDA}	Data In Hold Time	0		μs
t_{SDA}	Data In Setup Time	200		ns
t_r	Input Rise Time		0.3	μs
t_f	Input Fall Time		300	ns
t_{SSSTP}	Stop Setup Time	2.0		μs



Table 2–32. AC CHARACTERISTICS ($V_{CC} = 3.3V \pm 10\%$)

Symbol	Parameter	Min	Max	Unit
t_{DH}	Data Out Hold Time	100		ns
t_{WR}	Write Cycle Time		20	ms

Cascading Serial Configuration EEPROMs

For multiple FPGAs configured as a daisy-chain, or for future FPGAs requiring larger configuration memories, cascading MPA17C256s provides additional memory.

After the last bit from the first MPA17C256 is read, the next clock signal asserts its \overline{CEO} output LOW and disables its DATA line. The second MPA17C256 recognizes the LOW level on its \overline{CE} input and enables its DATA output.

Standby Mode

The MPA17C256 enters a low power standby mode whenever \overline{CE} is asserted HIGH. In this mode, it consumes

less than 1.0mA of current. The output remains in a high impedance state regardless of the state of the \overline{OE} input.

MPA17C256 Reset Polarity

The MPA17C256 lets the user choose the reset polarity as either $\overline{RESET}/\overline{OE}$ or \overline{RESET}/OE .

Programming Mode

The programming mode is entered by bringing $\overline{SER_EN}$ LOW. In this mode, the chip can be programmed by a 2-wire interface. The programming is done at V_{CC} supply only. Programming (high) voltages are generated inside the chip. For additional programming information, see the Programmer's Guide section on page 2–71.

2

Programmer's Guide

Serial Bus Overview

The serial bus is a two wire bus; one wire (CLOCK) functions as a clock and is provided by the programmer, the second wire (DATA) is a bi-directional signal and is used to provide data and control information.

Information is transmitted on the serial bus in messages. Each MESSAGE is preceded by a START BIT and is ended with a STOP BIT. The message consists of an integer number of bytes, each byte consists of 8 bits of data and is followed by a ninth ACKNOWLEDGE BIT. This ACKNOWLEDGE BIT is provided by the recipient of the data, This is possible because devices only drive DATA low, the system (in the programming case the Programmer) provides a small pull-up current (1k Ohm equivalent) for the Data Pin.

The MESSAGE FORMAT consists of the bytes shown in the Message Bytes table below. The MESSAGE FORMAT is preceded by a start bit and ended by a stop bit.

The programmer provides all the bytes except for the data bytes when the device is being read. Note that each byte is individually acknowledged. This acknowledgment is provided by the MPA17C256 in all cases except for the data bytes in the read mode, in which case the acknowledge is provided by the programmer.

Bit Format

Data on the DATA pin may change only during CLOCK low times.

Figure 2–65. Message Bytes

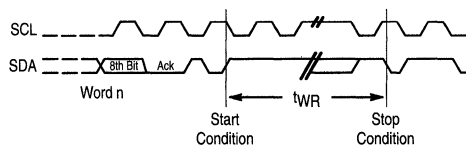
DEVICE ADDRESS	1ST ADDRESS WORD	2ND ADDRESS WORD	DATA BYTE(S)
----------------	------------------	------------------	--------------

Figure 2–66. Message Format

START BIT	DEVICE ADDRESS	1ST ADDRESS WORD	2ND ADDRESS WORD	DATA BYTE(S)	STOP BIT
-----------	----------------	------------------	------------------	--------------	----------

Start and Stop Bits

The START BIT is indicated by a high-to-low transition of DATA when CLOCK is high. Similarly, the STOP BIT is generated by a low-to-high transition of DATA when CLOCK is high, as shown below.

**Figure 2–67. Start and Stop Bits****Acknowledge Bit**

The ACKNOWLEDGE BIT is shown in the above figure. Note that the ACKNOWLEDGE BIT is provided by the device receiving the byte. The receiving device can accept the byte, by asserting a low value, on DATA or it can refuse the byte by asserting (not driving the signal) a 1 on DATA. All bytes must be terminated by either the ACKNOWLEDGE BIT or a STOP BIT.



Bit Ordering Protocol

The most significant bit is the first bit of a byte transmitted on DATA for the DEVICE ADDRESS BYTE and the EEPROM ADDRESS BYTES. It is followed by the lesser significant bits until the eighth bit, the least significant bit is transmitted. This is followed by the acknowledge bit. However, for DATA BYTES (both writing and reading) the first bit transmitted is the least significant bit. This protocol is shown in the tables below.

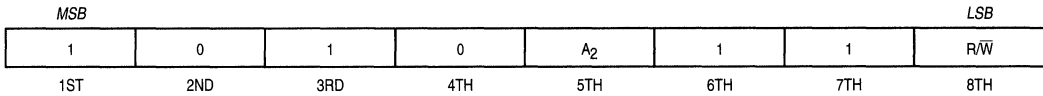
Device Address Byte

The contents of the Device Address Byte are shown below, along with the order in which the bits are clocked into the device. The A2 bit is provided to allow 2 devices to share a common bus; when programming a single device, the A2 bit must be forced to a logic '0' or '1' level. It is recommended that this pin be connected to 0V using a 4.7K ohm resistor pulldown

2



Figure 2-68. Device Address Byte



Where: $R/\bar{W} = 1$ Read
 $\quad\quad\quad = 0$ Write
 A₂ = 1 if \overline{CEO} pin is at VCC
 $\quad = 0$ if \overline{CEO} pin is at GROUND

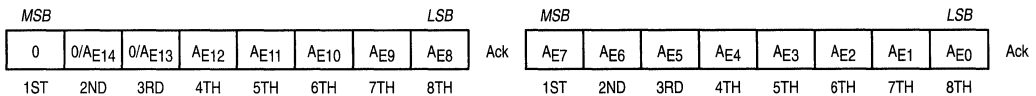
EEPROM Address

The EEPROM address consists of two bytes, each of which is followed by an acknowledge bit. These two bytes define a

15-bit address AE14 – AE0. The order in which each byte is clocked into the device is also indicated. AE14 is MSB for 17C256.

2

Figure 2-69.



Data Byte

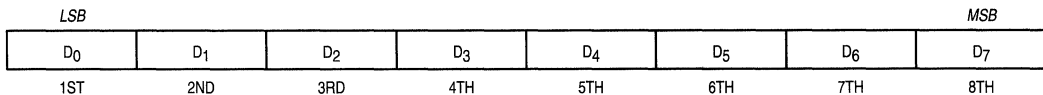
The organization of the Data Byte is shown below. Note that in this case, the data byte is clocked into the device LSB first and MSB last.

zero. Writing can start at any address within a page and the number of bytes written must be 64. The first byte is written at the transmitted address. The address is incremented in the device following the receipt of each data word received. Only the lower six bits of the address are incremented and if the address is incremented after the 64th byte in the page is sent, then the next byte to be written is the first byte of the page.

Writing

All writing takes place in pages. A page is 64-bytes long and the page boundaries are addresses where A5–A0 are all

Figure 2-70.



A write action consists of

- a Start Bit
- a Device Address with $R/\bar{W} = 0$
 - An Acknowledge Bit From the device
 - First Word of the Address
 - An Acknowledge Bit From the device
 - Second Word of the Address
 - An Acknowledge Bit From the device
 - One or more data bytes (sent to the device)
 - Each followed by an Acknowledge Bit From the device
- a Stop Bit

WRITE POLLING: On receipt of the stop bit, the device enters an internally timed write cycle. While the device is busy with this write cycle it will not acknowledge any transfers. Thus the programmer can start the next page write by sending the Start Bit followed by the Device Address. If this is not acknowledged, then the programmer should abandon the transfer without asserting a stop bit. The programmer can then repeat this until an acknowledge is received. When this is received the write action can proceed, i.e., the next byte to be sent is the device address.

Reading

Read operations are initiated the same way as write operations with the exception that the R/\bar{W} bit in the device address is set to one. There are three read operations: current address read, random read and sequential read.

CURRENT ADDRESS READ: The internal data word address counter maintains the last address accessed during the last read or write operation, incremented by one. This address stays valid between operations as long as the chip



power is maintained and the device remains in 2-wire access mode. If the last operation was a read at address n , then the current address would be $n + 1$. If the final operation was a write at address n , then the current address would again be $n + 1$ with one exception. If address n was the 64th byte address in the page, the incremented address $n + 1$ would "roll over" to the first byte address on the next page.

Once the device address with the R/\bar{W} select bit set is clocked in and acknowledged by the device the current address word is serially clocked out. The programmer does not acknowledge the read but does generate a following stop condition.

2

A current address read action consists of

- a Start Bit
- a Device Address with $R/\bar{W} = 1$
 - An Acknowledge Bit From the device
- a data byte from the device
- a Stop Bit from the programmer

RANDOM READ: A random read requires a "dummy" byte write sequence to load in the data word address. Once the device address word and data word address are clocked in and acknowledged by the device, the programmer must generate another start condition. The programmer now initiates a current address read by sending a device address with the R/\bar{W} bit high. The device acknowledges the device address and serially clocks out the data word. The programmer does not acknowledge the read but does generate a following stop condition.

A random address read action consists of

- a Start Bit
- a Device Address with $R/\bar{W} = 0$
 - An Acknowledge Bit From the device
- First Word of the Address
 - An Acknowledge Bit From the device
- Second Word of the Address
 - An Acknowledge Bit From the device
- a Start Bit
- a Device Address with $R/\bar{W} = 1$
 - An Acknowledge Bit From the device
- a data byte from the device
- a stop bit from the programmer

SEQUENTIAL READ: Sequential reads are initiated by either a current address read or a random address read. After the programmer receives a data word, it responds with an acknowledge. As long as the device receives an acknowledge, it will continue to increment the data word address and serially clock out sequential data words. When the memory address limit is reached, the data word address will "roll over". The sequential read operation is terminated when the programmer does not respond with an acknowledge but generates a stop condition.

Programming Pins

Eight pins are used to program the devices. These eight pins, and their mapping to the package pins are shown in the following table:

Table 2-33. Programming Pins

Pin	8-Pin Device	20-Pin Device
DATA	1	2
CLOCK	2	4
RESET/OE	3	6
CE	4	8
GROUND	5	10
A2 (CEO)	6	14
SER_EN	7	17
VCC	8	20

Programmer Functions

The programmer needs to perform the following functions:

1. Check the Manufacturers Code and the Device Code (Not necessary for In-System Programming)
2. Program the device
3. Verify the device
4. Set the Reset Polarity option

In the order given above. They are performed in the following manner.

Reading Manufacturers and Device Code

These two bytes are read from addresses 0 and 1, respectively, by performing a "read" as specified in this spec, with the following DC voltages set:

RESET/ \bar{CE}	=	0V
\bar{CE}	=	11.5 \pm 0.5V
A2 (CEO)	=	(Same as applied to A2 Pin, usually 0V)
SER_EN	=	0V

The correct codes are (Note 1.)

Manufacturers Code	-	Byte 0	1E
Device Code	-	Byte 1	FF 17C128 7F 17C65 77 17C256

1. The Manufacturer's Code and Device Code are read using the same byte ordering specified in the beginning of this document: i.e., LSB first, MSB last.

Programming the Device

All the bytes in the device's 64-byte page must be written. The order is not important but it is suggested that the device be written sequentially from Byte 0. Writing is accomplished by using the DATA and CLOCK pins and setting the other programming pins as follows:



$\overline{\text{RESET/CE}}$ = 0V
 $\overline{\text{CE}}$ = 0V
 $\text{A2} (\overline{\text{CEO}})$ = (Same as applied to A2 Pin,
usually 0V)
 $\overline{\text{SER_EN}}$ = 0V

Verifying the Device

All bytes in the device must be read and compared to their intended values. Reading is done using the CLOCK and DATA pins with the other programming pins set to the same value as in programming:

$\overline{\text{RESET/CE}}$ = 0V
 $\overline{\text{CE}}$ = 0V
 $\text{A2} (\overline{\text{CEO}})$ = (Same as applied to A2 Pin,
usually 0V)
 $\overline{\text{SER_EN}}$ = 0V

MPA17C256 Setting the Polarity Option

Setting the Polarity Option Active High

Write a byte of data set to FE to address 3FFF, using the previously defined 2-wire write algorithm, with the other programming pins set to the following:

$\overline{\text{RESET/CE}}$ = $V_{CC} \pm 0.25V$
 $\overline{\text{CE}}$ = $V_{CC} \pm 0.25V$
 $\text{A2} (\overline{\text{CEO}})$ = (Same as applied to A2 Pin,
usually 0V)
 $\overline{\text{SER_EN}}$ = 0V

This will change $\overline{\text{RESET/OE}}$ pin functionality to $\overline{\text{RESET/OE}}$, i.e., active high OE and active low RESET.

Setting the Polarity Option Active Low:

Write a byte of data set to FE to address 3FFF, using the previously defined 2-wire write algorithm, with the other programming pins set to the following:

$\overline{\text{RESET/CE}}$ = 0V
 $\overline{\text{CE}}$ = $V_{CC} \pm 0.25V$
 $\text{A2} (\overline{\text{CEO}})$ = (Same as applied to A2 Pin,
usually 0V)
 $\overline{\text{SER_EN}}$ = 0V

This will change $\overline{\text{RESET/OE}}$ functionality to $\overline{\text{RESET/OE}}$ i.e., active low OE and active high RESET (the default condition).

After RESET polarity has been modified the MPA17C256 device must be powered down before the modified RESET polarity takes effect.

Verifying the RESET/OE Polarity

If a programmed (master) device is to be used as the source for the data to be programmed into some new devices, then the programmer can read the data from the master. The polarity of the $\overline{\text{RESET/OE}}$ must be known before this can be done successfully for the MPA17C256. Depending on the capabilities of the programming device, one of the following algorithms can be used to read the programmed polarity of the $\overline{\text{RESET/OE}}$ pin.

1. If the programmer is able to sense a tri-state condition:

Switch the power on with

$\overline{\text{RESET/CE}}$ = 0V
 $\overline{\text{CE}}$ = 0V
 $\text{A2} (\overline{\text{CEO}})$ = Input to programmer (High Z)
 $\overline{\text{SER_EN}}$ = $V_{CC} \pm 0.25V$
 CLOCK = 0
 INPUT = Input to programmer

In this condition, if the SDA pin is 3-stated then the $\overline{\text{RESET/OE}}$ fuse is active high: if the SDA pin reads a "0" or a "1", then the $\overline{\text{RESET/OE}}$ fuse is active low.

2. If the programmer is NOT able to sense a 3-state condition:

Switch the power on with

$\overline{\text{RESET/CE}}$ = $V_{CC} \pm 0.25V$
 $\overline{\text{CE}}$ = 0V
 $\text{A2} (\overline{\text{CEO}})$ = Input to programmer (High Z)
 $\overline{\text{SER_EN}}$ = $V_{CC} \pm 0.25V$
 CLOCK = 0
 INPUT = Input to programmer

Hold this configuration for t_{WR} time after V_{CC} reaches nominal level. Then, set $\overline{\text{RESET/OE}}$ to low and pulse the clock 262,144 times for the MPA17C256, reading the data provided at each clock pulse. After the last clock has been issued $\overline{\text{CEO}}$ should drop from high to low. If it does so then the polarity is $\overline{\text{RESET/OE}}$ (active low). If $\overline{\text{CEO}}$ remains high, then the polarity is $\overline{\text{RESET/OE}}$ (active high). In this latter case, none of the data read is reliable and it should be discarded. The procedure should be redone with $\overline{\text{RESET/OE}} = 0V$ on power up and switched to $V_{CC} \pm 0.25V$ before starting the clock. The data read is now good data.

2



MPA1000 Design System Product Description

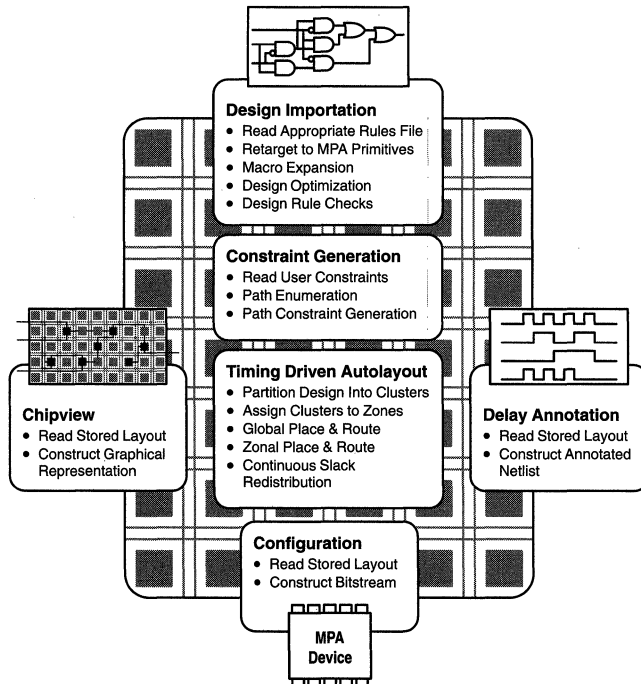
Overview

The Motorola Programmable Array (MPA) design system is a bridge between a design capture environment and Motorola field programmable arrays. The MPA design system automatically transforms designs into device configurations which, when loaded into an MPA device, realize a design. A design is automatically analyzed, optimized, transformed into MPA cells, partitioned, placed and routed based on timing constraints for every path in the design. MPA design tools understand and optimally utilize the MPA device architecture; this eliminates the need to learn a new set of rules and makes these tools ideally suited for use with logic synthesis. Full incremental design support reduces design implementation time and powerful library retargeting capabilities allow you to reuse designs which may have been implemented on less capable devices. The MPA design system operates on existing hardware platforms and supports design capture and simulation tools from more than 10 vendors. All these features plus on-line, hypermedia, help make the MPA design system a powerful yet extremely easy to use design implementation engine.

2

Features

- Push Button Implementation
- Optimal Use of MPA Device Resources
- Optimal Results with Gate Level Design Input
- Library of Common MSI Functions
- Design Flow Manager
- Design Retargeter
- Timing Driven with Integrated Static Timing Analysis
- Layout Delay extraction for post layout simulation
- Layout viewer
- Incremental design support
- On-line, hypermedia, documentation
- Supports all popular design capture and simulation tools
- Lowest cost FPGA development systems.
- Instant access; Downloading via the internet (WWW, ftp).



Push Button Design Implementation

The MPA design system minimizes training investment and automatically generates design implementations which meet timing constraints.

The gate level logic and abundant hierarchical routing resources of the MPA device present a rich implementation media for design implementation. MPA design tools understand and optimally utilize the MPA device resources so there are no elaborate rules to learn or design modifications required to begin design capture. Staying focused on end product design rather than implementation tools or device architecture gets the design done faster and, unlike other programmable solutions, without programmable logic device specificity to impede future design migration efforts. The combination of automatic tools and gate level architecture is ideal for traditional schematic driven or high level language based design capture methods. In fact, logic synthesis tools were originally designed for and produce the most efficient results for targeting gate level devices.

A design is analyzed, optimized, transformed into MPA cells, partitioned, placed and routed based on timing constraints for all paths in the design – automatically. A netlist from one of the popular design capture systems or an existing LPM netlist is imported into the MPA design system. The logic is mapped to a series of MPA cells and the entire resulting netlist is optimized and checked. Based on a simple clock specification, the MPA design system generates timing constraints for all paths in the design. During automatic partitioning, placement and routing path slack time is constantly redistributed insuring only the resources required to meet timing requirements are consumed. Because MPA tools implement the design according to constraints, tool induced design iterations are virtually eliminated. Completed layouts can be transformed into device configurations, as well as annotated simulation netlists. A layout browser is also available.

The MPA design system also includes complete on–line, hypermedia, help covers the device, the design system and the integration kits. Integration kits for Viewlogic, Exemplar, Synopsys, VeriBest, Verilog–SDF, VITAL–SDF, VHDL (1076), Verilog (OVI) and OrCAD are included (contact your vendor for additional kits). All these features add up to a powerful yet extremely easy to use design implementation engine for the MPA product family.

Design Importation

Designs can be captured using schematics, a high level language, or a combination of these entry methods using commercially available design capture and logic synthesis software and the appropriate interface kit. Alternatively, existing designs can be retargeted from other programmable logic devices to the MPA device using commercial logic synthesis tools or the powerful retargeting capabilities provided with MPA design system.

Design importation begins with a netlist and an optional clock specification file. The clock specification file provides a mechanism for the user or design capture tools to document system level timing requirements. In addition, a rich set of attributes can be attached to specific components or nets

within the design to specify timing and design pinout constraints.

A retargeting rules file is read and the input netlist is transformed into a series of MPA cells and associated interconnections. Rules files provide a mechanism to perform attribute mapping, cell mapping and macro expansion. By creating custom rule files, the user can extend the importation process from arbitrary sources. The MPA design system comes with rules for its native library/EDIF. The resulting netlist is optimized to clip unused logic and remove redundant logic. For example: each MPA cell has programmable input inversion capability. All inverters or non–inverting buffers can be removed from the netlist and replaced with signal sense information attached to each input.

A series of design rule checks are performed to insure design integrity before the layout process begins.

Constraint Generation

Timing constraints, the optimized MPA netlist and static timing analysis is used to generate path slack constraints for all paths in the design. Each unique signal pathway between a register output and a register input throughout the design are enumerated. The total logic and estimated or real wire delays along the path are summed. The time between the active upstream register clock edge and the next active downstream clock edge minus the downstream register setup time is subtracted from the total path delay. This difference is called path slack. If any path in the design has a negative slack value, the implementation will not function at the required clock rate(s).

Path constraints are utilized throughout the layout process to insure that a design implementation which meets timing constraints is automatically generated. If no clock or timing specifications are provided, the MPA design system uses the fastest possible clock based on very small net delay estimates to generate the path constraints. This usually results in the best possible implementation, but may take longer than the time required to generate a satisfactory rather than best possible result.

Contrast this to other programmable logic design tools which only provide manual net constraint annotation or net criticality assignment. In these cases significant effort is necessary to generate constraints and many costly iterations are required to tune these constraints for a given design. If any changes are made to the design, another costly round of iterations is required.

Autolayout

The autolayout process makes use of the hierarchical organization of the MPA device to minimize run time and deliver implementations that meet timing requirements. Designs which have diverse timing requirements are ideally implemented because path slack estimates are refined throughout the autolayout process insuring only the resources required to meet timing requirements are consumed.

The process begins by flattening the design and partitioning it into small component groups of approximately

2



MPA1000 Design System Product Description

the same size called clusters. A cluster boundary delay estimation is applied to pull the most tightly constrained paths into a minimum number of clusters. The clusters are then assigned to zones taking into account zonal boundary delay cost and relative zone placement delay costs. Other costs like total number of port connections per zone and are also considered. As assignment proceeds, cluster and zone boundary delay costs are added to each path and slack is recomputed.

Next global placement and routing is done. Global routes begin and end on either I/O cells or port cells. Intrazone placement and routing is deferred to a later phase. During global routing all the port cell and I/O cell locations are fixed and the connections between them established. High fanout nets are constructed in a highly regular manner to insure efficient resource utilization. As in partitioning, slack estimates are refined throughout global routing.

Finally the intrazonal placement and routing is done. Cells assigned to a particular zone are placed and routed to other zone cells or zone port cells. Port cells and core cells are constructed to allow port swapping. Core cells can be routed through if necessary. Allowing core cells to act as routing cells allows dynamic adjustment of routing resources within the zone. Dynamic resource adjustment is a powerful design specific adaptation mechanism.

This process produces a layout from which device configurations, delay back annotations, and chipviews can be generated.

Incremental Design Support

When specification changes necessitate design iterations, simply push the button again. Constraints are automatically recalculated and autolayout only reworks those portions of the design which have changed. Full incremental design support means simple design changes to facilitate design verification can be made quickly and easily.

Delay Back Annotation

Designs can be verified through numerous methods. One particularly useful method is the annotation of device and implementation specific delays back into the original simulation environment to improve system or device level simulation accuracy. A MPA device layout can be transformed into an appropriately formatted delay annotation

file or annotated netlist quickly and easily. The annotated delay information represents the worst case delays for a given device speed grade.

Chipview

While the MPA design system provides a rich set of reports describing the implementation of a design, a graphical view of the implementation can be indispensable for reviewing overall layout quality. Chipview provides a graphical view of a completed layout. Chipview can be useful during initial design iterations to visually verify I/O pin placements before commencing PCB layout, for example.

Configuration

A layout can be transformed into a device configuration which, when loaded into the appropriate MPA device, produces a physical design realization. Many formatting options are available. The MPA download pod can be used to emulate a serial PROM. Using the pod, device configuration files can be downloaded to a device directly from the PC or workstation development environment.

Integration Kits

The MPA design system can be used with a large number of commercial electronic design automation software. The Vendor Software List on page 2-79 shows the currently supported vendors and tools. For each supported vendor, an integration kit is provided which facilitates MPA design within that vendors' environment. Many of these kits are available from Motorola and included at no charge on the MPA design system CD-ROM. Other kits can be acquired directly from the vendor. Refer to the MPA Design System Product List for more information.

Low Cost, Easy Access

MPA Design systems are easy to use, competitively priced and widely available. Copies of MPA design system software supporting 1016 and 1036 can be downloaded from the World Wide Web (WWW) @ <http://sps.motorola.com/fpga>. Complete kits including download pod, evaluation board, MPA device, CD-ROM and documentation can be ordered from your local authorized Motorola distributor or Motorola sales representative (see Motorola Distributor and Worldwide Sales Office listings in Ch 5. on page 5-3).

*Fast, Efficient Design Implementation With Minimal Investment.
That's MPA!*



Vendor Software List

Vendor	Package, Revision	Synthesis	Schematic	Simulation	Timing Analysis
Viewlogic	Workview Office, 7.31 (7.4 Aug 97)	Q2-97	Yes	Yes	Q3-97
Viewlogic	Workview Office, 7.12	Q2-97	Yes	Yes	No
Synopsys	Design Compiler, 3.1	Yes	Yes (Generated)	Yes	No
Exemplar	Galileo, 3.2.5 (4.1 Aug 97)	Yes	Yes (Generated)	Yes	No
Exemplar	Leonardo 4.0.3 (4.1 Aug 97)	Yes	Yes (Generated)	Yes	No
Model Tech	3Q97	No	No	Yes	Yes
Data I/O	Synario, 3.0	Yes	Yes	Yes	No
Cadence	FPGA Designer, TBD	No	No	Yes	Yes
OrCAD	Capture, 7.0	No	Yes	Yes	No
OrCAD	Express, TBD	No	Yes	Yes	No
Protel	Advanced Schematics, 3.23	No	Yes	No	No
VeriBest	VeriBest, VB97	Yes	Yes	Yes	No
Mentor Graphics	Design Architect, A3	Q2-97	Q2-97	Q2-97	Q2-97

2

Design System Product List

Part Number	Description	Platform		MPADS CD-ROM	Supports 1016/1036	Supports All MPAs	Eval Board	POD	Maintenance	S.R.P.
		PC	WS							
MPA1E/P	Entry Level Kit	X		X	X		X	X		\$295
MPA1E/W	Entry Level Kit		X	X	X		X	X		\$595
MPA1S/P	Standard Level Kit	X		X		X	X	X	1 Year	\$2,295
MPA1S/W	Standard Level Kit		X	X		X	X	X	1 Year	\$3,995
MPA1CD/P	Design Software CD	X		X	X					\$12.95
MPA1CD/W	Design Software CD		X	X	X					\$295
MPA1/POD	Download Pod	X	X			X		X		\$195
MPA1/BRD	Evaluation Board	X	X		X (FN)		X			\$145
MPA1M12P	Maintenance	X							1 Year	\$595
MPA1M12W	Maintenance		X						1 Year	\$795

2

Motorola Programmable Design System Descriptions

MPA1CD – Motorola Design System Software. Single CD-ROM containing MPADS software for both PC and workstation including MPA databook, application notes and all supported EDA vendor integration kits. MPA1CD/D is available from the Motorola Literature Distribution Center – Call 1-800-441-2447 or 303-675-2140. MPA1CD and software downloaded from our web site do not include maintenance.

MPA1E/P – PC Entry Kit. MPADS software with full support for MPA1016 and MPA1036 devices, download pod (MPA1/POD) and an evaluation board (MPA1/BRD), with an 84-pin MPA device, and a CD-ROM, at an attractive price.

MPA1S/P – PC Standard Kit. MPADS software with support for all MPA devices, maintenance for 1 year, download pod (MPA1/POD) and evaluation board (MPA1/BRD).

MPA1E/W – Workstation Entry Kit. MPA Entry kit for workstations – Sun OS V 4.1 (Now), HPUX V 8 (2Q/97).

MPA1S/W – Workstation Standard Kit. MPA standard kit for workstations.

MPA1/POD – Serial port download cable for both workstation and PC. Downloads configuration directly to a MPA device.

MPA1/BRD – Evaluation board including EPROM socket and an MPA1000 device in the 84-pin PLCC package.

MPA1M12P, MPA1M12W – 1 year maintenance (Standard Level Kit includes 1 year of maintenance).



MPA1000 Design System Product Description

Motorola Integrated Design System Packages for Windows '95/NT			
Kit Type	Part Number	Description	Suggested Retail Price
Basic	MPA1WV/BSC	<ul style="list-style-type: none"> • Schematic Entry & Simulation • MPA Design System Software (MPADS) for MPA1016, MPA1036 • Note: Maintenance Is Not Included 	\$749
Basic Plus	MPA1WV/BSCPL	<ul style="list-style-type: none"> • Schematic Entry, Simulation & VHDL Editor/Compiler • MPA Design System Software (MPADS) for MPA1016, MPA1036 • Note: Maintenance Is Not Included 	\$1,049
Standard	MPA1WV/STD	<ul style="list-style-type: none"> • Schematic Entry, Simulation & VHDL Editor/Compiler • MPA Design System Software (MPADS) for All MPA1000 Devices • One Year of Maintenance on All Software • Download Pod and Evaluation Board 	\$3,149
Deluxe	MPA1WV/DLX	<ul style="list-style-type: none"> • Schematic Entry, VHDL Editor/Compiler, Speedwave VHDL Simulator, Mixed Mode VHDL/Gate Level Simulator • MPA Design System Software (MPADS) for All MPA1000 Devices • One Year of Maintenance on All Software • Download Pod and Evaluation Board 	\$5,149

2

For additional information on Workview Office, visit the Viewlogic Web page at: <http://www.viewlogic.com/products>

The Motorola Integrated Design System incorporates Viewlogic's Workview Office Tool Suite with Motorola's Programmable Array Design System (MPADS) providing an integrated, easy to use, complete design environment for MPA1000 FPGAs. Support for other popular design capture and simulation tools, as well as stand-alone MPADS kits and accessories, are also available.

The Motorola Integrated Design System includes:

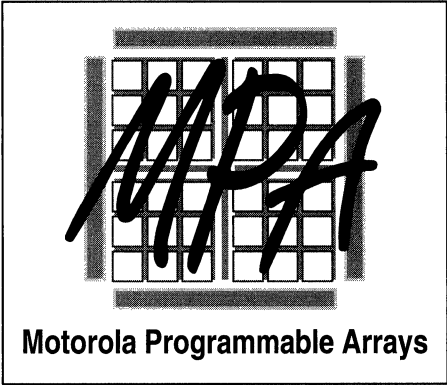
- Hierarchical Schematic Entry
- Gate Level Simulation
- Simulation Waveform Viewing
- VHDL and Mixed Mode Simulation
- VHDL Entry and VHDL Compilation
- Schematic Generation
- EDIF Netlist Writer
- Design Optimization
- Automatic, Timing Driven, Layout
- Layout Viewing
- Configuration Generation
- Download Hardware and Demo Board

A 30-day evaluation copy of the Integrated Design System with MPA1016 and MPA1036 support is available. Consult factory.



2

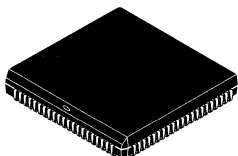




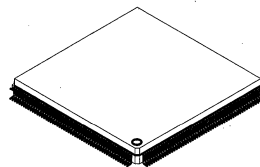
Packaging, Quality & Reliability

3

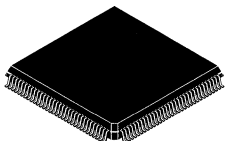
MPA1000 Family Packaging & Case Information



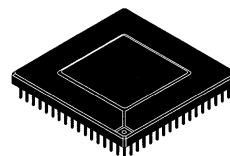
84-LEAD PLASTIC PLCC
CASE 780A-01
ISSUE A
FN SUFFIX



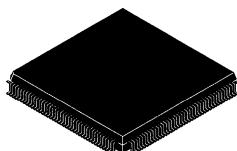
208-LEAD PLASTIC QFP
CASE 872A-01
ISSUE O
DK SUFFIX



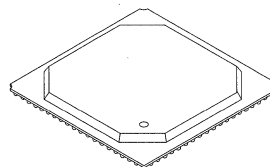
128-LEAD PLASTIC QFP
CASE 862A-01
ISSUE B
DD SUFFIX



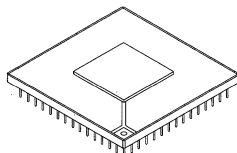
224-LEAD CERAMIC PGA
CASE 860F-01
ISSUE O
KE SUFFIX



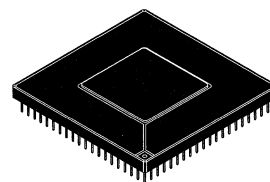
160-LEAD PLASTIC QFP
CASE 864A-03
ISSUE C
DH SUFFIX



256-LEAD PLASTIC BGA
CASE 1208A-01
ISSUE O
BG SUFFIX



181-LEAD CERAMIC PGA
CASE 768N-01
ISSUE O
HI SUFFIX

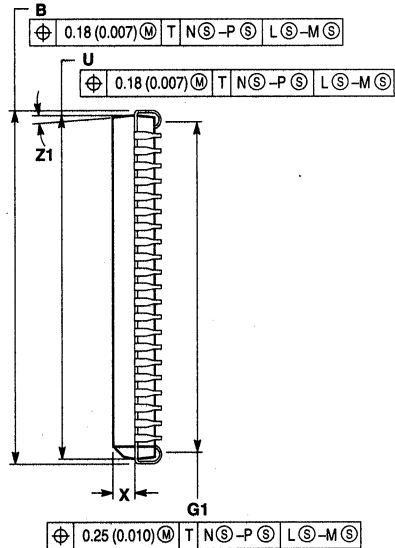
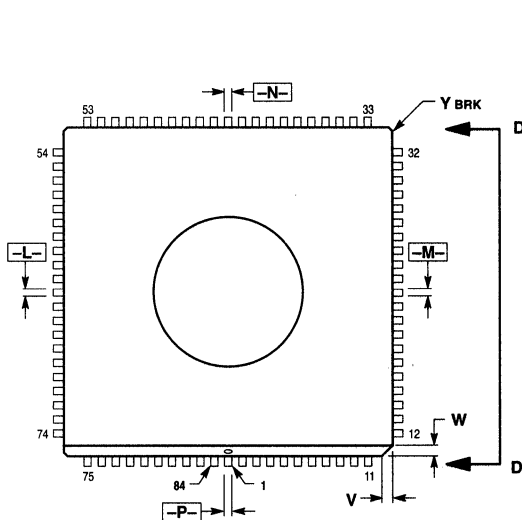


299-LEAD CERAMIC PGA
CASE 861B-01
ISSUE O
HV SUFFIX

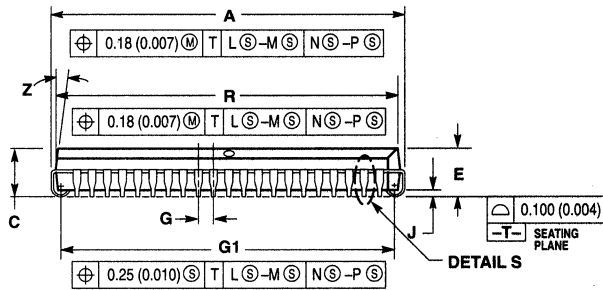
PICTORIALS NOT TO SCALE



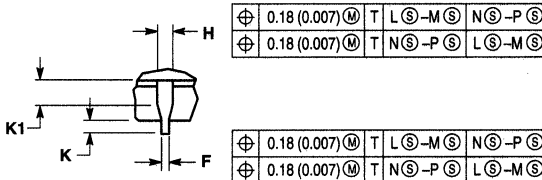
FN SUFFIX
84-LEAD PLASTIC PLCC PACKAGE
CASE 780A-01
ISSUE A



DETAIL D-D



DETAIL S



DETAIL S

- NOTES:**
1. DATUMS -L-, -M-, -N-, AND -P- DETERMINED WHERE TOP OF LEAD SHOULDER EXITS PACKAGE BODY AT GLASS PARTING LINE.
 2. DIMENSION G1, TRUE POSITION TO BE MEASURED AT DATUM -T-, SEATING PLANE.
 3. DIMENSIONS R AND U DO NOT INCLUDE GLASS PROTRUSION. ALLOWABLE GLASS PROTRUSION IS 0.25 (0.010) PER SIDE.
 4. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
 5. CONTROLLING DIMENSION: INCH.

DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	1.185	1.195	30.10	30.35
B	1.185	1.195	30.10	30.35
C	0.165	0.180	4.20	4.57
E	0.090	0.110	2.29	2.79
F	0.013	0.021	0.33	0.53
G	0.050 BSC		1.27 BSC	
H	0.026	0.032	0.66	0.81
J	0.020	—	0.51	—
K	0.025	—	0.64	—
R	1.150	1.156	29.21	29.36
U	1.150	1.156	29.21	29.36
V	0.042	0.048	1.07	1.21
W	0.042	0.048	1.07	1.21
X	0.042	0.056	1.07	1.42
Y	—	0.020	—	0.50
Z	2°	10°	2°	10°
G1	1.110	1.130	28.20	28.70
K1	0.040	—	1.02	—
Z1	2°	10°	2°	10°

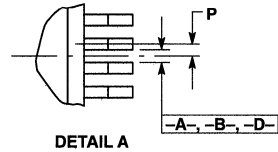
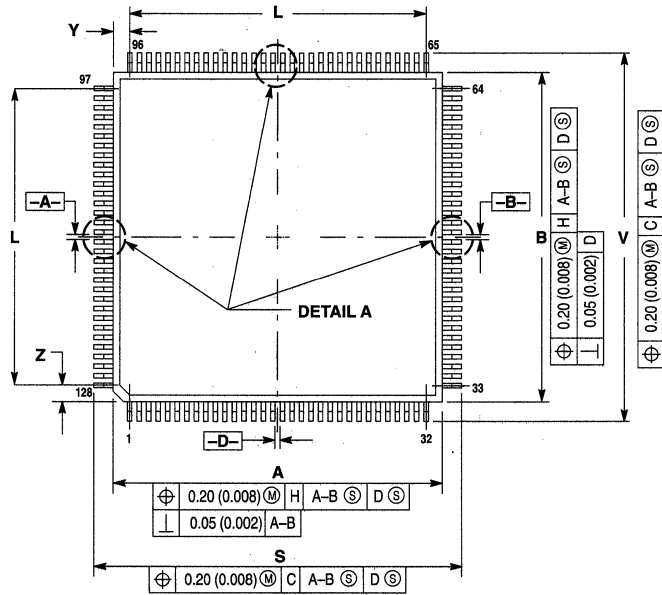
3



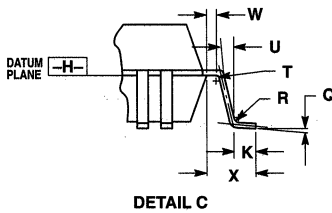
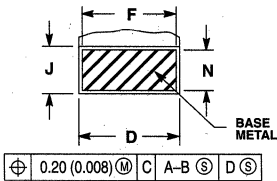
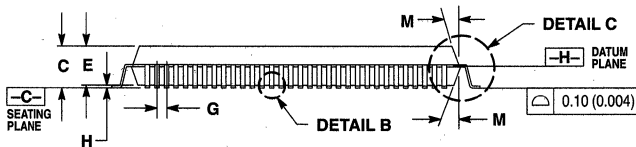
Packaging & Case Information

DD SUFFIX 128-LEAD PLASTIC QFP PACKAGE CASE 862A-02 ISSUE B

3



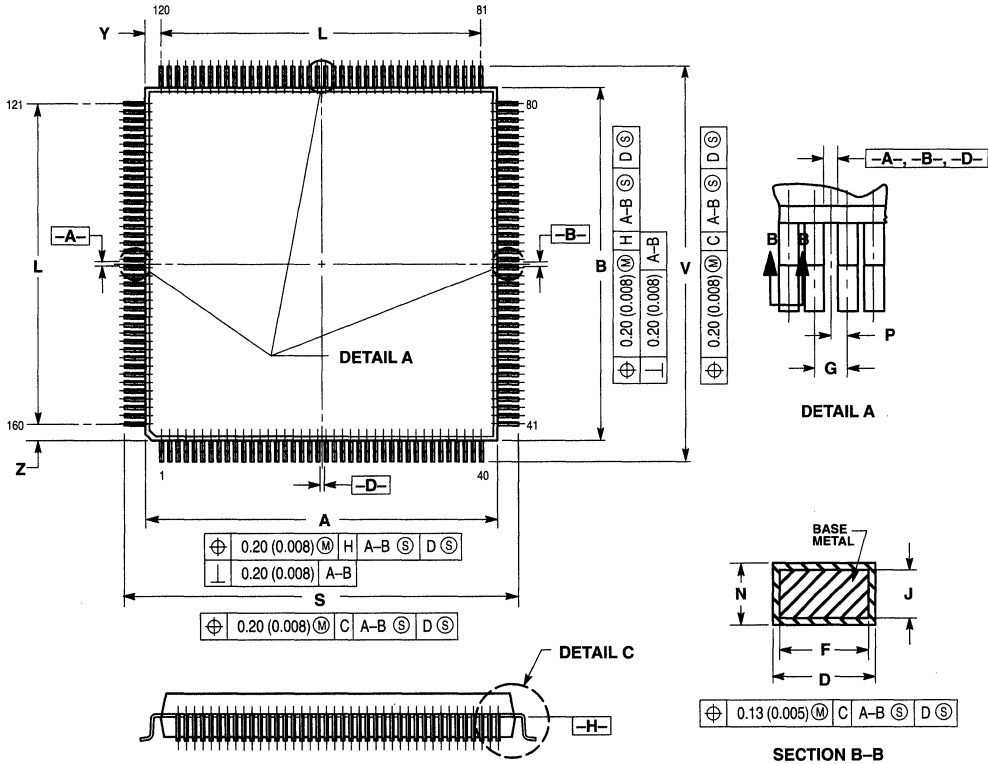
- NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
 2. CONTROLLING DIMENSION: MILLIMETER
 3. DATUM PLANE -H- IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
 4. DATUMS -A-, -B- AND -D- TO BE DETERMINED AT DATUM PLANE -H-.
 5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE -C-.
 6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 (0.010) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE -H-.
 7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OF THE FOOT.



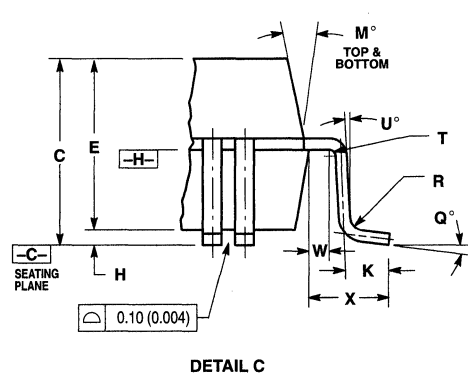
DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	27.90	28.10	1.098	1.106
B	27.90	28.10	1.098	1.106
C	—	4.07	—	0.160
D	0.30	0.45	0.012	0.018
E	3.17	3.67	0.125	0.144
F	0.30	0.40	0.012	0.016
G	0.80	BSC	0.032	BSC
H	0.25	0.35	0.010	0.014
J	0.13	0.23	0.005	0.009
K	0.65	0.95	0.026	0.037
L	24.80	REF	0.976	REF
M	5°	18°	5°	16°
N	0.13	0.17	0.005	0.007
P	0.40	BSC	0.016	BSC
Q	0°	7°	0°	7°
R	0.13	0.30	0.005	0.012
S	30.95	31.45	1.219	1.238
T	0.13	—	0.005	—
U	0°	—	0°	—
V	30.95	31.45	1.219	1.238
W	0.40	—	0.016	—
X	1.60	REF	0.063	REF
Y	1.60	REF	0.063	REF
Z	1.60	REF	0.063	REF



DH SUFFIX
160-LEAD PLASTIC QFP PACKAGE
CASE 864A-03
ISSUE C



3



- NOTES:**
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
 2. CONTROLLING DIMENSION: MILLIMETER.
 3. DATUM PLANE -H- IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
 4. DATUMS -A-, -B- AND -D- TO BE DETERMINED AT DATUM PLANE -H-.
 5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE -C-.
 6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 (0.010) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE -H-.
 7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION. DAMBAR CANNOT BE LOCATED ON THE LOWER RADIUS OR THE FOOT.

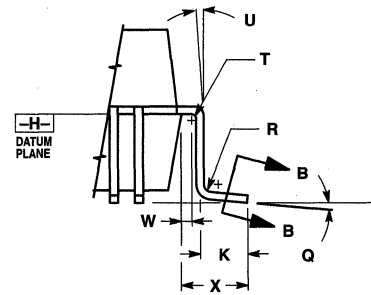
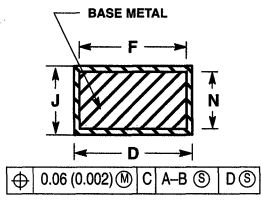
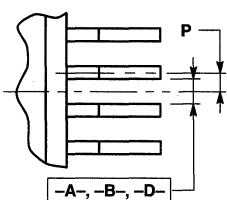
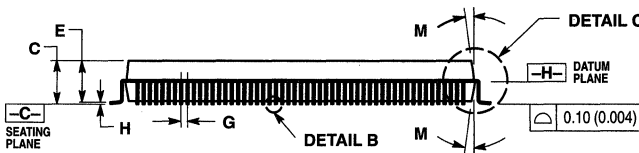
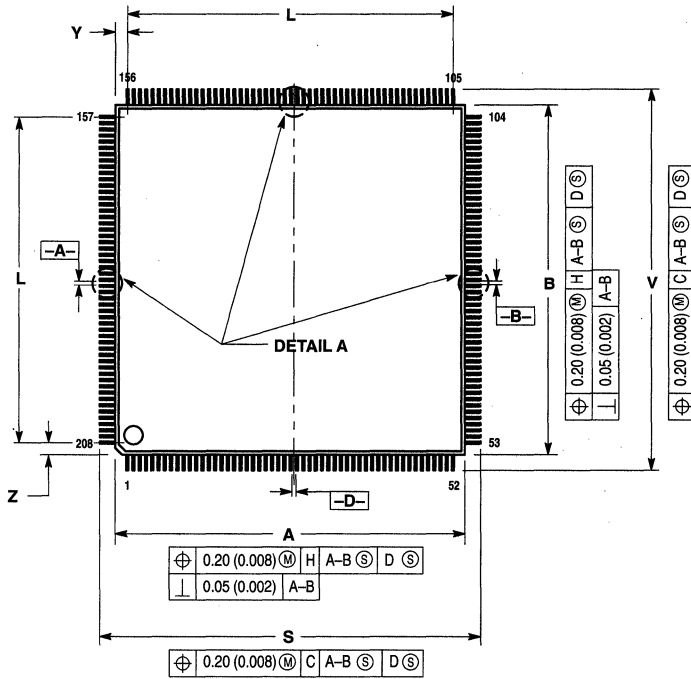
SECTION B-B

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	27.90	28.10	1.098	1.106
B	27.90	28.10	1.098	1.106
C	3.35	3.65	0.132	0.152
D	0.22	0.38	0.009	0.015
E	3.20	3.50	0.126	0.138
F	0.22	0.33	0.009	0.013
G	0.65 BSC		0.026 REF	
H	0.25	0.35	0.010	0.014
J	0.11	0.23	0.004	0.009
K	0.70	0.90	0.028	0.035
L	25.35 REF		0.998 REF	
M	5°	16°	5°	16°
N	0.11	0.19	0.004	0.007
P	0.325 BSC		0.013 BSC	
Q	0°	7°	0°	7°
R	0.13	0.30	0.005	0.012
S	31.00	31.40	1.220	1.236
T	0.13		0.005	
U	0°		0°	
V	31.00	31.40	1.220	1.236
W	0.40		0.016	
X	1.60 REF		0.063 REF	
Y	1.33 REF		0.052 REF	
Z	1.33 REF		0.052 REF	



DK SUFFIX
208-LEAD PLASTIC QFP PACKAGE
CASE 872A-01
ISSUE O

3



NOTES:

1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DATUM PLANE -H- IS LOCATED AT BOTTOM OF LEAD AND IS COINCIDENT WITH THE LEAD WHERE THE LEAD EXITS THE PLASTIC BODY AT THE BOTTOM OF THE PARTING LINE.
4. DATUMS -A-, -B- AND -D- TO BE DETERMINED AT DATUM PLANE -H-.
5. DIMENSIONS S AND V TO BE DETERMINED AT SEATING PLANE -C-.
6. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION. ALLOWABLE PROTRUSION IS 0.25 (0.010) PER SIDE. DIMENSIONS A AND B DO INCLUDE MOLD MISMATCH AND ARE DETERMINED AT DATUM PLANE -H-.
7. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. DAMBAR PROTRUSION SHALL NOT CAUSE THE D DIMENSION TO EXCEED 0.38 (0.015).

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	27.90	28.10	1.098	1.106
B	27.90	28.10	1.098	1.106
C	3.45	4.10	0.136	0.161
D	0.14	0.30	0.005	0.012
E	3.20	3.60	1.126	0.142
F	0.14	0.26	0.005	0.010
G	0.50 BSC		0.020 BSC	
H	0.25	0.35	0.010	0.014
J	0.09	0.20	0.003	0.008
K	0.70	0.80	0.027	0.036
L	25.50 REF		1.004 REF	
M	5°	9°	5°	9°
N	0.09	0.18	0.003	0.007
P	0.25 BSC		0.010 BSC	
Q	0°	7°	0°	7°
R	0.13	0.30	0.005	0.012
S	31.00	31.40	1.220	1.236
T	0.13		0.005	
U	0°		0°	
V	31.00	31.40	1.220	1.236
W	0.40		0.016	
X	1.60 REF		0.063 REF	
Y	1.25 REF		0.049 REF	
Z	1.25 REF		0.049 REF	

DETAIL A

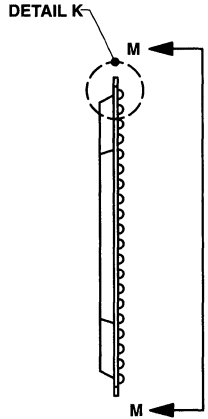
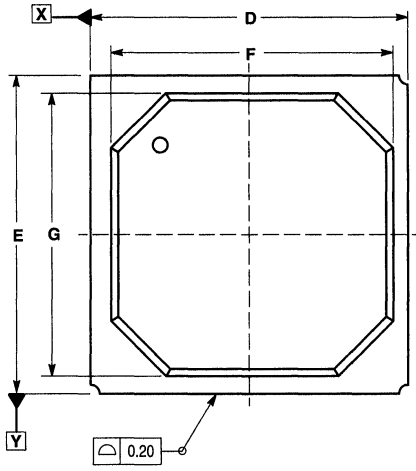
DETAIL B
SECTION B-B
ROTATED 7° CCW

DETAIL C



Packaging & Case Information

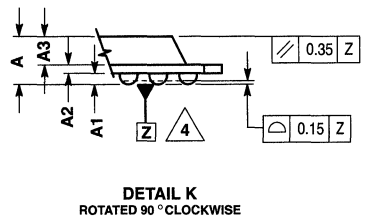
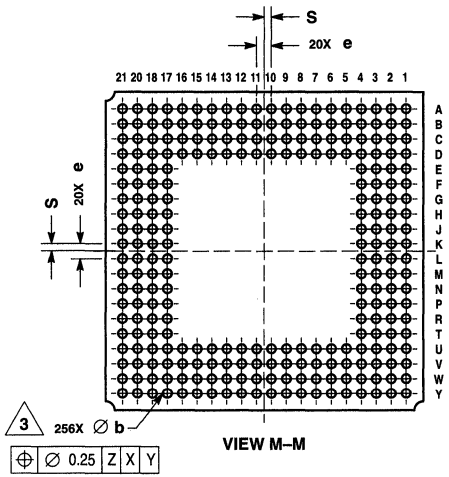
BG SUFFIX
256-LEAD PLASTIC BGA PACKAGE
CASE 1208A-01
ISSUE O



- NOTES:
1. DIMENSIONS ARE IN MILLIMETERS.
 2. INTERPRET DIMENSIONS AND TOLERANCES PER ASME Y14.5M, 1994.
 3. DIMENSION b IS MEASURED AT THE MAXIMUM SOLDER BALL DIAMETER, PARALLEL TO DATUM PLANE Z.
 4. DATUM Z (SEATING PLANE) IS DEFINED BY THE SPHERICAL CROWNS OF THE SOLDER BALLS.

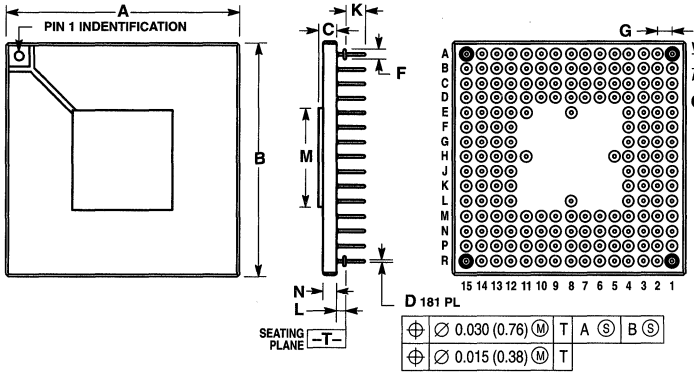
MILLIMETERS		
DIM	MIN	MAX
A	1.92	2.32
A1	0.50	0.70
A2	0.38 REF	
A3	1.12	1.22
b	0.80	0.90
D	27.00 BSC	
E	27.00 BSC	
F	24.00	24.70
G	24.00	24.70
e	1.27 BSC	
S	0.635 BSC	

3



Packaging & Case Information

HI SUFFIX 181-LEAD CERAMIC PGA PACKAGE CASE 768N-01 ISSUE O



- NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.

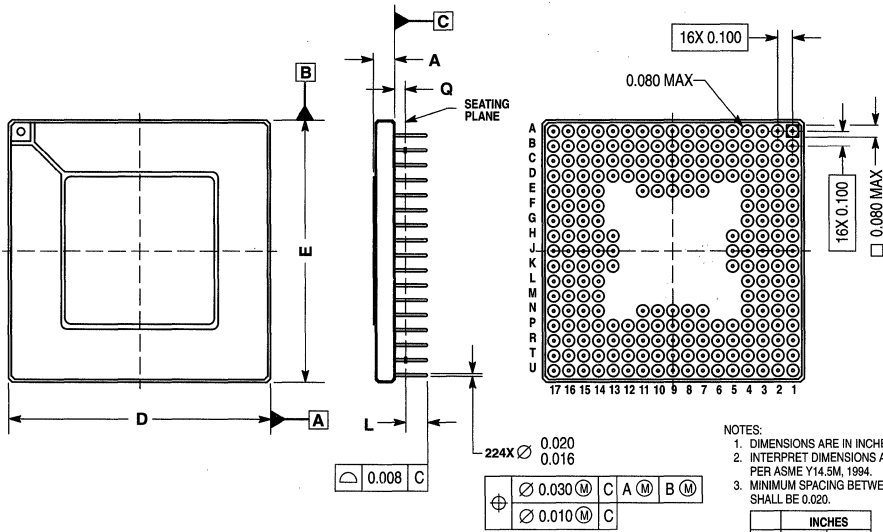
DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	1.555	1.595	39.50	40.51
B	1.555	1.595	39.50	40.51
C	0.102	0.124	2.59	3.15
D	0.016	0.020	0.41	0.51
F	0.040	0.060	1.02	1.52
G	0.100 BSC		2.54 BSC	
K	0.110	0.150	2.79	3.81
L	0.043	0.057	1.09	1.45
M	0.655	0.675	16.64	17.15
N	0.090	0.110	2.29	2.79

3

D 181 PL

⊕	∅ 0.030 (0.76)	M	T	A	Ⓢ	B	Ⓢ
⊕	∅ 0.015 (0.38)	M	T				

KE SUFFIX 224-LEAD CERAMIC PGA PACKAGE CASE 860F-01 ISSUE O



- NOTES:
1. DIMENSIONS ARE IN INCHES.
2. INTERPRET DIMENSIONS AND TOLERANCES PER ASME Y14.5M, 1994.
3. MINIMUM SPACING BETWEEN CONDUCTORS SHALL BE 0.020.

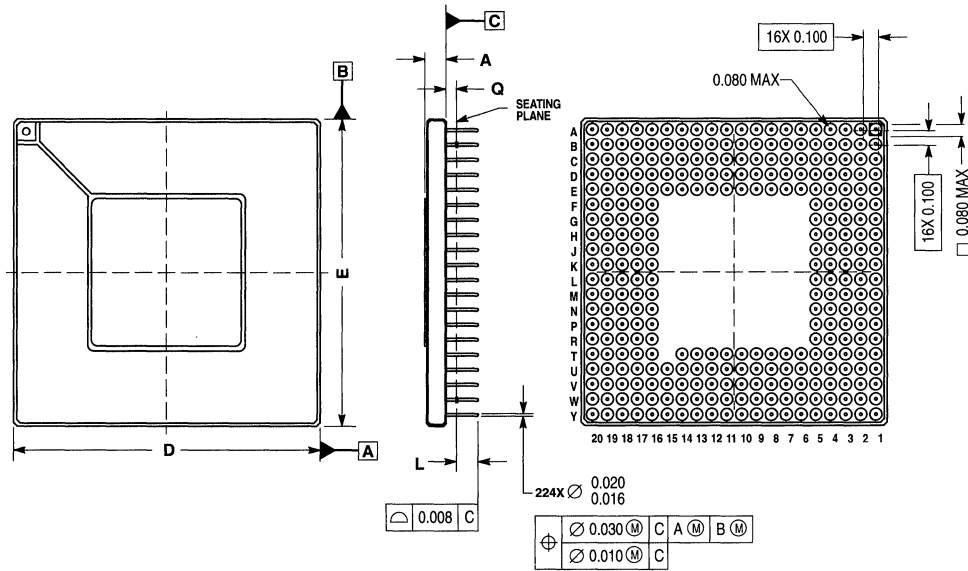
DIM	INCHES	
	MIN	MAX
A	0.070	0.145
D	1.740	1.780
E	1.740	1.780
L	0.100	0.200
Q	0.045	0.075

224X ∅ 0.020
0.016

⊕	∅ 0.030	M	C	A	M	B	M
	∅ 0.010	M	C				



HV SUFFIX
299-LEAD CERAMIC PGA PACKAGE
CASE 861B-01
ISSUE O



3

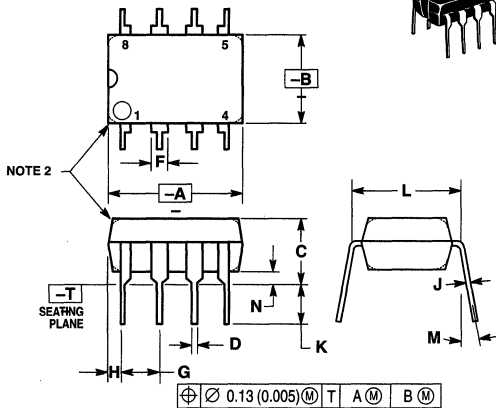
- NOTES:
 1. DIMENSIONS ARE IN INCHES.
 2. INTERPRET DIMENSIONS AND TOLERANCES PER ASME Y14.5M, 1994.
 3. MINIMUM SPACING BETWEEN CONDUCTORS SHALL BE 0.020.

DIM	INCHES	
	MIN	MAX
A	0.070	0.145
D	2.040	2.080
E	2.040	2.080
L	0.100	0.200
Q	0.045	0.075
S	0.050 BSC	



MPA17000 EPROM/EEPROM Packaging & Case Information

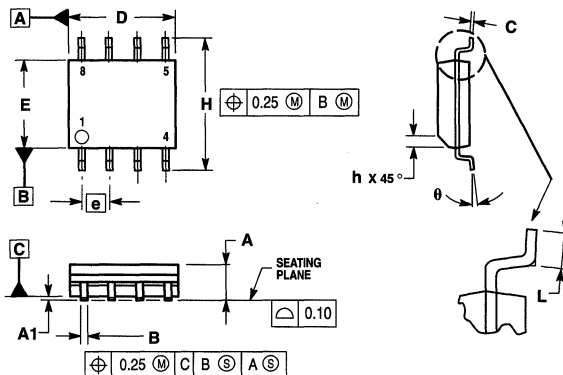
P SUFFIX
8-LEAD PLASTIC DIP PACKAGE
CASE 626-05
ISSUE K



- NOTES:
1. DIMENSION L TO CENTER OF LEAD WHEN FORMED PARALLEL.
 2. PACKAGE CONTOUR OPTIONAL (ROUND OR SQUARE CORNERS).
 3. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	9.40	10.16	0.370	0.400
B	6.10	6.60	0.240	0.260
C	3.94	4.45	0.155	0.175
D	0.38	0.51	0.015	0.020
F	1.02	1.78	0.040	0.070
G	2.54 BSC		0.100 BSC	
H	0.76	1.27	0.030	0.050
J	0.20	0.30	0.008	0.012
K	2.92	3.43	0.115	0.135
L	7.62 BSC		0.300 BSC	
M	—	10°	—	10°
N	0.76	1.01	0.030	0.040

D SUFFIX
8-LEAD PLASTIC SOIC PACKAGE
CASE 751-05
ISSUE S



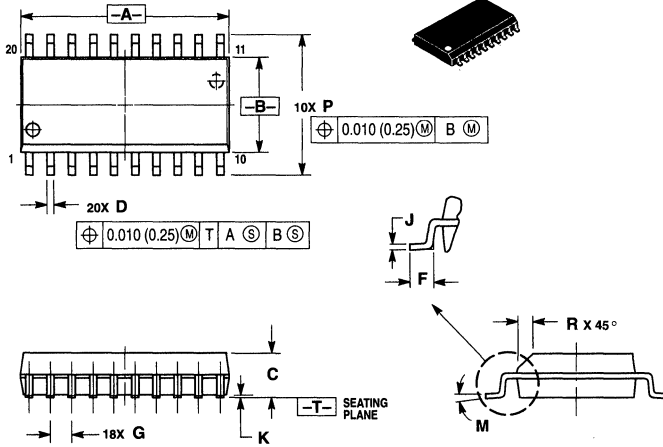
- NOTES:
1. DIMENSIONING AND TOLERANCING PER ASME Y14.5M, 1994.
 2. DIMENSIONS ARE IN MILLIMETERS.
 3. DIMENSION D AND E DO NOT INCLUDE MOLD PROTRUSION.
 4. MAXIMUM MOLD PROTRUSION 0.15 PER SIDE.
 5. DIMENSION B DOES NOT INCLUDE MOLD PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.127 TOTAL IN EXCESS OF THE B DIMENSION AT MAXIMUM MATERIAL CONDITION.

DIM	MILLIMETERS	
	MIN	MAX
A	1.35	1.75
A1	0.10	0.25
B	0.35	0.49
C	0.18	0.25
D	4.80	5.00
E	3.80	4.00
e	1.27 BSC	
H	5.80	6.20
h	0.25	0.50
L	0.40	1.25
θ	0°	7°



Packaging & Case Information

DW SUFFIX 20-LEAD PLASTIC SOIC WIDE PACKAGE CASE 751D-04 ISSUE E



NOTES:

1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.150 (0.006) PER SIDE.
5. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.13 (0.005) TOTAL IN EXCESS OF D DIMENSION AT MAXIMUM MATERIAL CONDITION.

DIM	MILLIMETERS		INCHES	
	MIN	MAX	MIN	MAX
A	12.65	12.95	0.499	0.510
B	7.40	7.60	0.292	0.299
C	2.35	2.65	0.093	0.104
D	0.35	0.49	0.014	0.019
F	0.50	0.90	0.020	0.035
G	1.27 BSC		0.050 BSC	
J	0.25	0.32	0.010	0.012
K	0.10	0.25	0.004	0.009
M	0°	7°	0°	7°
P	10.05	10.55	0.395	0.415
R	0.25	0.75	0.010	0.029

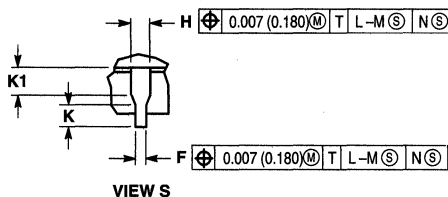
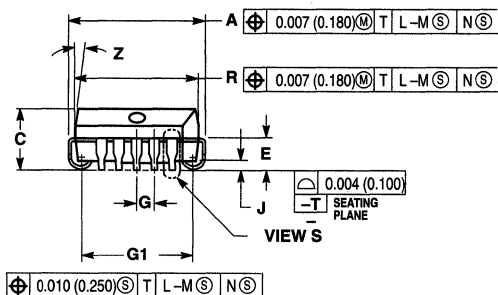
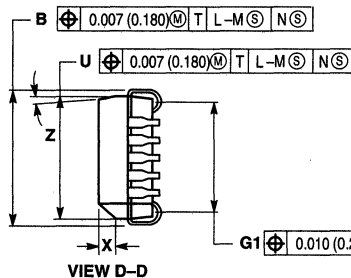
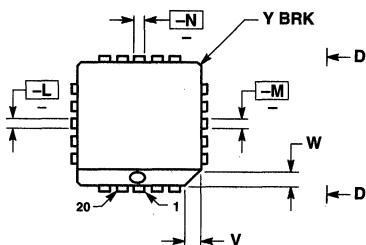
3



FN SUFFIX
20-LEAD PLASTIC PLCC PACKAGE
CASE 775-02
ISSUE C



3



NOTES:

- DATUMS -L-, -M-, AND -N- DETERMINED WHERE TOP OF LEAD SHOULDER EXITS PLASTIC BODY AT MOLD PARTING LINE.
- DIM G1, TRUE POSITION TO BE MEASURED AT DATUM -T-, SEATING PLANE.
- DIM R AND U DO NOT INCLUDE MOLD FLASH. ALLOWABLE MOLD FLASH IS 0.010 (0.250) PER SIDE.
- DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
- CONTROLLING DIMENSION: INCH.
- THE PACKAGE TOP MAY BE SMALLER THAN THE PACKAGE BOTTOM BY UP TO 0.012 (0.300). DIMENSIONS R AND U ARE DETERMINED AT THE OUTERMOST EXTREMES OF THE PLASTIC BODY EXCLUSIVE OF MOLD FLASH, THE BAR BURRS, GATE BURRS AND INTERLEAD FLASH, BUT INCLUDING ANY MISMATCH BETWEEN THE TOP AND BOTTOM OF THE PLASTIC BODY.
- DIMENSION H DOES NOT INCLUDE DAMBAR PROTRUSION OR INTRUSION. THE DAMBAR PROTRUSION(S) SHALL NOT CAUSE THE H DIMENSION TO BE GREATER THAN 0.037 (0.940). THE DAMBAR INTRUSION(S) SHALL NOT CAUSE THE H DIMENSION TO BE SMALLER THAN 0.025 (0.635).

DIM	INCHES		MILLIMETERS	
	MIN	MAX	MIN	MAX
A	0.385	0.395	9.78	10.03
B	0.385	0.395	9.78	10.03
C	0.165	0.180	4.20	4.57
E	0.090	0.110	2.29	2.79
F	0.013	0.019	0.33	0.48
G	0.050 BSC		-1.27 BSC	
H	0.026	0.032	0.66	0.81
J	0.020	—	0.51	—
K	0.025	—	0.64	—
R	0.350	0.356	8.89	9.04
U	0.350	0.356	8.89	9.04
V	0.042	0.048	1.07	1.21
W	0.042	0.048	1.07	1.21
X	0.042	0.056	1.07	1.42
Y	—	0.020	—	0.50
Z	—	2°	—	10°
G1	0.310	0.330	7.88	8.38
K1	0.040	—	1.02	—



Quality and Reliability

Quality

The Motorola culture is a culture of quality. Throughout all phases of product development, from defining and designing to shipping the product, Motorola strives for total customer satisfaction through "Six Sigma" and "On Time Delivery" programs.

Designing Products

Extensive work was done on the 75% UDR CMOS process to ensure a solid platform for quality products. Process reliability studies were performed to uncover any weaknesses in the initial process so that enhancements could be made to strengthen it before it was released to production. In addition, comprehensive characterization and correlation work was completed on the process to ensure the utmost in modeling parameter accuracy.

The design of the products strictly adhered to the design rules set forth by the process designers. Conservative, manufacturable layout rules were followed to minimize the performance variability due to a marginally manufacturable product.

Manufacturing Process

Through SPC and continual engineering work, the manufacture of the CMOS process is both monitored and enhanced on a continuous basis. Statistical data is gathered at both probe and final test through the device data collection to monitor the distribution of a parameter to its specification limits. In addition, final quality assurance gates are set up to guarantee the quality of outgoing product.

Product Characterization

Products are both DC and AC characterized for all data book environmental conditions prior to the release of the product to production. The distributions of the parameters are compared to their specification limits to ensure that Motorola "manufactures" quality products as opposed to "testing" quality products through distribution truncation. In addition, ongoing AC characterization is performed to enhance the distributions of the AC parameters of the device. In doing so, as the distributions warrant, further enhancements to the AC specifications can be achieved.

Reliability

To ensure the long term reliability of MPA products, extensive accelerated life testing is performed prior to production release. This qualification work is performed by Logic Reliability Engineering, an organization specifically dedicated to monitoring and guaranteeing the quality and reliability of logic products. The accelerated life test consists of the following:

Operating Life Test: 145°C, 5.5V Man Supply
 Temperature Cycle: -65°C to 150°C
 Pressure, Temperature, Humidity (Hermeticity)

A minimum of 35 lots, 100 die per lot taken from seven different wafers in the lot constitute a qualification sample. Various intermediate readouts are taken to monitor the performance more closely. In addition, the devices are tested beyond the specification limits to determine where and how they will fail.

Another responsibility of the reliability group is that of failure analysis. This failure analysis service is supported for both internal purposes and for servicing the needs of our customers. Analysis entails everything from simple package examination to internal microprobing to SEM analysis of IC structures. The results of the analysis are returned to the customer and if the analysis suggests a potential problem with the device the information is also passed to the internal product groups.

RAP: Reliability Audit Program

The Reliability Audit Program (RAP) devised in March 1977 is the Motorola internal reliability audit which is designed to assess outgoing product performance under accelerated stress conditions. Logic Reliability Engineering has overall responsibility for RAP, including updating its requirements, interpreting its results, administration at offshore locations and monthly reporting of results. These reports are available at all sales offices. Also available is the "Reliability and Quality Handbook" which contains data for all Motorola semiconductors (BR518/D).

Rap is a system of environmental and electrical tests performed periodically on randomly selected samples of standard products. Each sample receives the tests specified in Figure 3-1. Frequency of testing is specified per internal document 12MRM15301A.

3



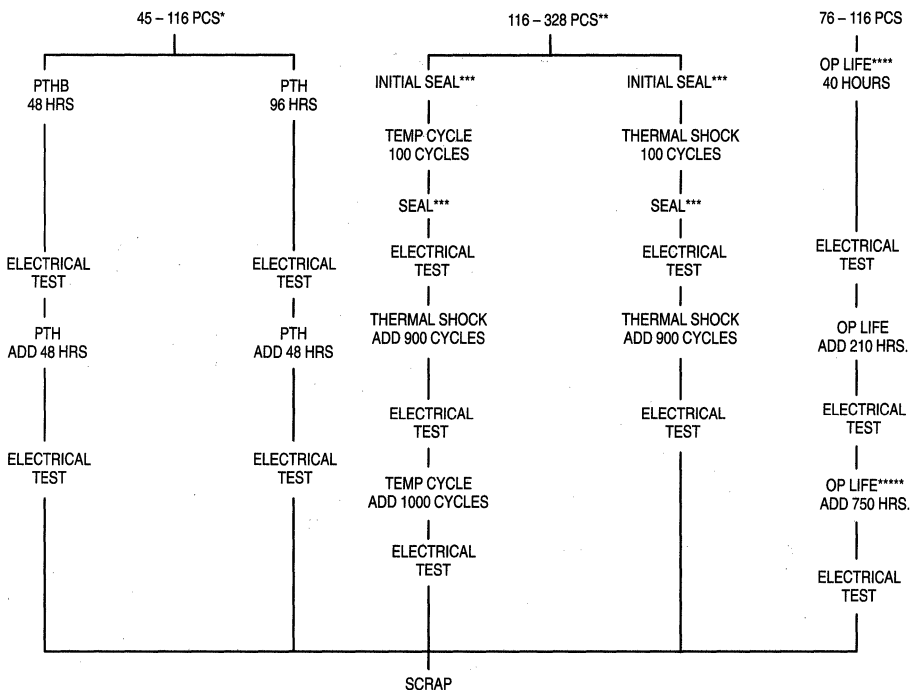


Figure 3-1. Reliability Audit Program Test Flow

- * PTH will be run as a substitute if PTHB sockets are not available. Only required on plastics packages.
- ** Thermal Shock will be run if Temp Cycle is not available.
- *** Seal (fine and gross) only required on hermetic packages.
- **** All units for Op Life to be AC/DC tested before and after being stressed. All units failing AC after stress will be analyzed.
- ***** One sample per month

PTHB

15psig/121°C/100% RH at rated V_{CC} or V_{EE} - to be performed on plastic encapsulated devices only.

Temp Cycle

Mil. Std. 883, Method 1010, Condition C, - 65°C to 150°C

Op Life

Mil. Std. 883, Method 1005, Condition C (Power plus Reverse Bias), T_A = 145°C.

Notes:

1. All standard 25°C DC and functional parameters will be measured Go/NoGo at each readout.
2. Any indicated failure is first verified and then submitted to the Product Analysis Lab for detailed analysis.
3. Sampling to include all package types routinely.
4. Device types sampled will be by generic type and will include all assembly locations.
5. 16 hrs. PTHB is equivalent to ~ 800 hrs. of 85°C/85% RH THB for V_{CC} ≤ 15V.
6. Only moisture related failures (like corrosion) are criteria for failure on PTHB test.
7. Special device specifications (48A's) for digital products will reference 12MRM15301A as a source of generic data for any customer requiring monthly audit reports.



Latch up

Latch up will not be a problem for most designs, but the designer should be aware of it, what causes it, and how to prevent it.

Figure 3–3 shows the cross-section of a typical CMOS inverter and Figure 3–2 shows the parasitic bipolar devices. The circuit formed by the parasitic transistors and resistors is the basic configuration of a silicon controlled rectifier, or SCR. In the latch up condition, transistors Q1 and Q2 are turned ON, each providing the base current necessary for the other to remain in saturation, thereby latching the devices into the ON state. Unlike a conventional SCR, where the device is turned ON by applying a voltage to the base of the NPN transistor, the parasitic SCR is turned ON by applying a voltage to the emitter of either transistor. The two emitters that trigger the SCR are connected to the same point, the CMOS output. Therefore, to latch up the CMOS device, the output voltage must be greater than $V_{DD} + 0.5V$ or less than $V_{SS} - 0.5V$ and have sufficient current to trigger the SCR. The latch-up mechanism is similar for the inputs.

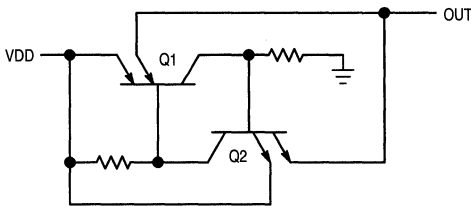


Figure 3–2. Latch-Up Circuit Schematic

To reduce the current for triggering the SCR, guard rings are formed and act as dummy collectors to collect charges

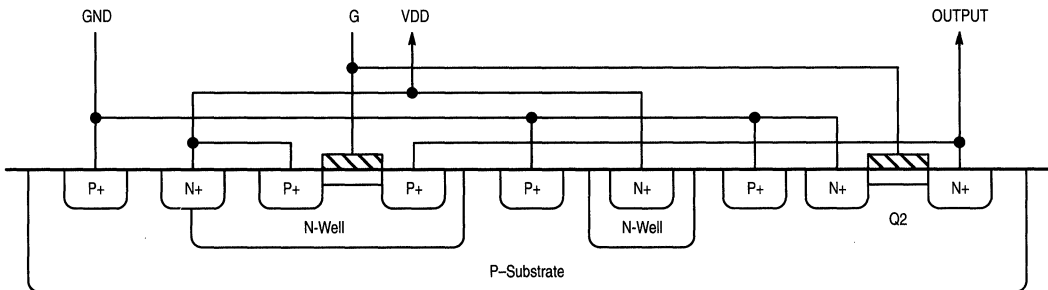


Figure 3–3. CMOS Wafer Cross-Section

directly through V_{CC} and ground, rather than through active circuitry, thereby shunting the parasitic transistors. The guard rings are connected to V_{CC} and ground near the input and output diodes to short out the parasitic SCR. Guard ring diffusion also creates additional parasitic transistors and reduces effective substrate resistance which makes the SCR harder to turn on.

Once a CMOS device is latched up, if the supply current is not limited, the device will be destroyed. Ways to prevent such occurrences are listed below:

3. Insure that inputs and outputs are limited to the maximum rated values, as follows: $-0.5V \leq V_{in}$ or $V_{out} \leq V_{DD} + 0.5V$ (referenced to V_{SS}) I_{in} or $I_{out} \leq 10mA$ (unless otherwise indicated on the data sheet).
4. If voltage transients with sufficient energy to latch up the device is expected on the inputs or outputs, external protection diodes can be used to clamp the voltage. Another method of protection is to use a series resistor to limit the expected worst case current to the maximum rating of 10mA.
5. Sequence power supplies so that the inputs or outputs of CMOS devices are not active before the supply pins are powered up (e.g., recessed edge connectors and/or series resistors may be used in plug-in board applications).
6. Voltage regulating or filtering should be used in board design and layout to insure that power supply lines are free of excessive noise.
7. Limit the available power supply current to the devices that are subject to latch-up conditions. This can be accomplished with a power supply filtering network or a current limiting regulator.

3



Quality and Reliability

Electrostatic Discharge

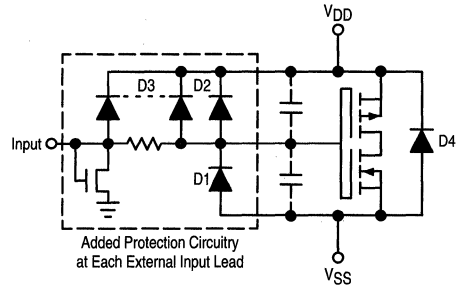
The gate electrode of a CMOS circuit is completely isolated from the substrate by a silicon dioxide layer, which forms the dielectric of the gate-to-substrate capacitor. The thickness of the oxide insulator between the gate and the substrate of a MOS device is about 1000Å and has a typical breakdown voltage in the range of 100 to 120V.

If a voltage higher than the breakdown voltage is applied to the gate, the silicon dioxide beneath the gate will rupture. This can result in permanent damage of the device, causing a short between gate metal and either the substrate or a P or N region. Because of the extremely high resistance of the gate oxide, even a very low energy source (i.e., stray electrostatic charges) is capable of developing this breakdown voltage. The possibility that a CMOS device will be destroyed by static overvoltage exists only during handling and testing. Once the device is mounted in a circuit, normal circuit impedances and voltages make this danger virtually impossible. In order to avoid destruction of CMOS devices by static overvoltage, various input protection circuits were developed.

The protection system used for ESD is the Double Diode Plus Resistor Protection Circuit as shown in Figure 3-4.

It consists of a series isolation resistor R_S , whose average

value is 1.5kohm, and diodes D1 and D2 for clamping excess input voltages to the power supply pins, VDD or VSS. Diode D3 is a distributed parasitic structure resulting from the diffusion fabrication of R_S .

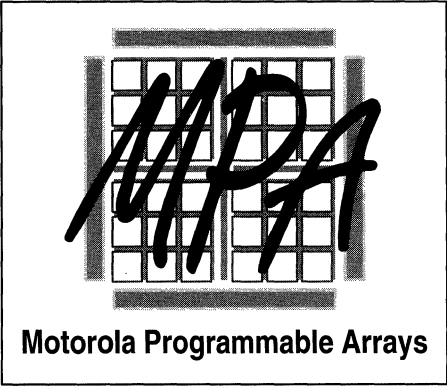


NOTES: $R_S = 1.5k\Omega$ Nominal
Avalanche Voltages
BVD₁ = 30V BVD₂ = 30V
BVD₃ = 80V BVD₄ = 90V

Figure 3-4. Double Diode Plus Resistor Protection Circuit

3





AN1561
Application Note

68030 DRAM Controller Design Using Verilog HDL

Prepared by
Phil Rauba
Field Applications Engineer

4



68030 DRAM Controller Design Using Verilog HDL

Purpose

This article is intended to give a hardware engineer insight into the design methodology of using the Verilog Hardware Descriptive Language (HDL), targeting Motorola's field Programmable Array (MPA) and H4C gate array families. The advantage of using an HDL, such as Verilog, is the ability to retarget the design to other device technologies, by only resynthesizing the design description. A 68030 Dynamic Ram Controller design was used to demonstrate the portability of the Verilog language, and included all of the circuits necessary to interface DRAM to a 68030 microprocessor including: memory decoding, STERM generation, refresh request generation, CAS before RAS refresh, burst address sequencing, DRAM address multiplexing, and bus error time-out.

Design Methodology

The DRAM Logic was designed with a synchronous state machine design technique and described using the Verilog Hardware Descriptive Language, with the intent of providing a portable and easily maintainable design. The design tools used for this project are listed in Appendix A. The steps included in the design process include the system block diagram definition, state diagram generation, Verilog HDL logic definition, Verilog logic simulation, Verilog logic synthesis, place and route, and Verilog post simulation.

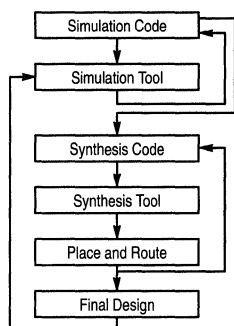


Figure 1. Verilog Design Method

The Verilog Design Methodology, Figure 1, illustrates the design flow beginning with the generation of the Verilog RTL source code. The DRAM design used a hierarchical module development methodology, which partitioned the design into eight submodules and instantiated each of the submodules into the design through a top module description designated as module glue68k. A stimulus module was also created that provided the test bench for verification of simulation code. The stimulus module included a 25MHz clock generator module, a behavioral 68030 bus controller module, and the instantiation of the top glue68k module, designated with the instance name of u1. As each Verilog submodule was written, the code was verified logically with the use of the stimulus module and the waveform capabilities of the Verilog simulator.

Once the design was logically verified, the original Verilog source code, that was used for simulation, was also used for synthesis. Each module of the hierarchical design was synthesized separately so that if a module needed to change, only that module would have to be resynthesized, saving considerable time by not having to resynthesize the entire design.

Since different tools were used for the logic synthesis, when targeting MPA and gate array, different methods were used during the synthesis process. For MPA development, the Exemplar tool was used for synthesizing each of the submodules individually. When synthesizing the top glue68k module, two passes were used with the Exemplar tools, one to generate submodule connectivity and the other to read and reformat the Verilog netlist. Empty submodules were instantiated in the design during the first pass of the Exemplar logic synthesizer with a Verilog netlist being generated. The Verilog glue68k netlist was then edited to add the links to the submodules by using the include command, referencing each of the submodule's file pathname. A final netlist was output from the second pass of the synthesizer, which read the eight Verilog netlists from the links in the top module, and reformatted the file to an EDIF netlist.

For gate array development, Synopsys was used for synthesis of the design. Each of the eight submodules and the top module, glue68k were read into Synopsys and synthesized all at one time without having the need to use the include command.

The EDIF netlist is used by the MPA and gate array place and route tools to generate the final design files. The MPA design procedure was to create a project, select the target device (MPA1036 181 pin PGA), input the EDIF netlist, place and route the design, and back-annotate into a structural Verilog netlist.

After placing and routing the DRAM MPA design, the structural Verilog netlist was used for post simulation. The structural Verilog netlist generated by the place and route back-annotation tool, contains precise MPA1036 gate and path delays to accurately predict the timing behavior of the final placed and routed design. Post simulation is useful for verifying and altering the design, if needed, before a printed circuit board is required. For post simulation, a modified version of the presimulation stimulus file was written to reflect the net name changes that were incurred by use of the design tools, but included the same clock and 68030 bus controller test suites as before. Final simulation of the DRAM design required the structural Verilog netlist module, the stimulus module, and the Verilog MPA1000 series gate primitive library, that was supplied with the MPA design system.

System Description

Bursting is a feature in the newer generation of CISC/RISC microprocessors that is comprised of a memory access of four long words of 32 bits each. The DRAM burst cycle is initiated by first generating a RAS cycle access and a CAS cycle access for the first long word, and then fetching the next three long words by generating only CAS cycles thereafter. The intention of the burst cycle is to divide the RAS cycle

4



generation overhead of the first access amongst all four longword fetches; thereby, providing an overall access performance improvement as compared to single RAS generation for each longword.

The system block diagram is shown in Figure 2 and includes a 68030 microprocessor, a 16MByte dram array using 4Mx4 DRAMs, data bus drivers, and the MPA (or gate array). The MPA provides all of the DRAM interface circuitry needed to support 68030 bursting.

MPA Functional Description

The block diagram functional description of the MPA is shown in Figure 3 and shows all of the modules within the design, including the refresh timing generator, the refresh request state machine, the address decoder, the RAS/CAS state machine, the RAS/CAS decoder logic, the burst control state machine, the burst address generator, the DRAM address multiplexer, and the bus error time-out state machine.

MPA Timing Synchronization

As indicated in the MPA functional block diagram Figure 2, the main clock for all the state machines is the inverted 25MHz

clock to the 68030. Since all of the output timing out of the 68030 is referenced to the falling edge of the processor's 25MHz clock, the clock is inverted and is used for clocking the MPA's internal registers. A delay line (not shown) will be needed for moving the assertion point of the address strobe signal with respect to the internal clock skew within the target device to prevent flip-flop metastable conditions. The delay line value will be dependent on the actual clock skew within the MPA or gate array.

Refresh Timing Generator

The refresh timing generator provides a 97.656KHz refresh request square wave with a period of 10.24usec for the refresh request state machine. The generator is comprised of a eight bit free running up counter with a 25MHz clock source.

Refresh Request State Machine

The refresh state machine receives the 97.656KHz refresh square wave and generates a ref_rq signal to the RAS/CAS state machine. The refresh state machine requests a refresh cycle only once when ref[7] is asserted high and inhibits the request after the RAS/CAS state machine has initiated a CAS before RAS refresh cycle.

4

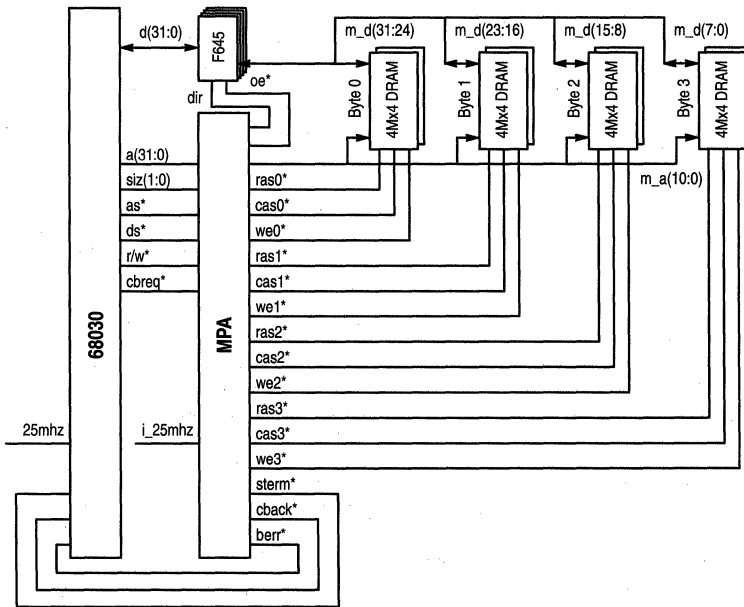


Figure 2. MPU-DRAM Controller Interfacing



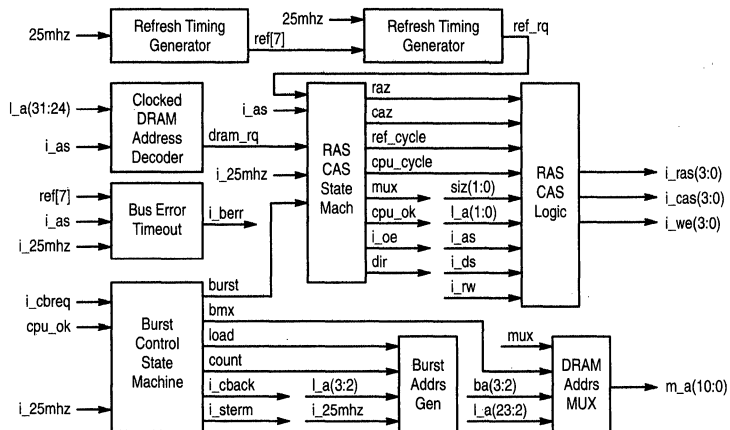


Figure 3. MPA DRAM Controller Detail

Figure 7 shows the refresh cycle that was initiated by the rising edge of ref[7] and by the assertion of ref_rq to the RAS/CAS Controller. The RAS/CAS Controller generates the timing for a CAS before RAS refresh cycle in synchronization with the refresh request state machine sequencing through the request operation. At the end of the refresh cycle the refresh request state machine is in the REFEND state waiting for negation of ref[7].

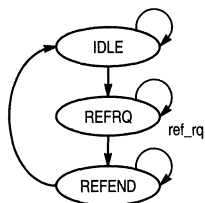


Figure 4. Refresh Request State Diagram

Synchronized DRAM Address Decoder

The DRAM Address decoder is used to decode the 68030 address 01xxxxxxh with qualification of address strobe to generate a access request to the RAS/CAS state machine. The dram_rq timing is shown in Figure 5.

RAS/CAS State Machine

The RAS/CAS State machine arbitrates between refresh requests and 68030 access requests via signals, ref_rq and dram_rq respectively. The arbitration between the two requests occurs in the IDLE state, where refresh has the highest priority. If a refresh request is pending the state machine will take the refresh branch and generate a CAS before RAS refresh timing sequence.

If a DRAM request is pending from the 68030 and a refresh request is not present, the state machine will take the 68030

access branch and generate the RAS, MUX, and CAS timing for a random access into the DRAM array. The assertion of RAS latches the row address into the DRAMs, the MUX signal will present the column address to the DRAM array, and the CAS signal will then latch the column address into the DRAMs. The signal cpu_ok will indicate to the burst controller to start wait state generation. If a burst request sequence is requested by the burst controller via the signal burst, the RAS/CAS state machine will sequence through the burst control states. For a full four long word burst, the burst address generator will provide the column addresses for each of the long word accesses. The RAS/CAS controller state machine will exit bursting upon the negation of address strobe, and has the capability of exiting a burst upon a premature ending of a full four long word burst.

RAS/CAS Logic

The RAS/CAS logic is comprised of combinational logic that encodes the CAS signals for selecting which byte lanes of the DRAM array that are going to be accessed during a cycle. For a CPU write access, the logic supports the misalignment capabilities of the 68030, providing CAS signals only to the bytes of the DRAM array that will be accessed for the write operation. For a CPU read cycle all of the CAS signals will be asserted. During refresh cycles all of the CAS and RAS lines will be asserted.

Burst Control State Machine

The burst control state machine provides all of the bursting control for a 68030 DRAM access and is synchronized to the RAS/CAS controller. Upon the receipt of a dram_rq, the RAS/CAS controller will generate RAS and CAS timing to the DRAM array and will assert the signal cpu_ok, indicating to the burst controller state machine to start a burst cycle. The burst controller will leave an idle state and assert the i_cback signal indicating a synchronous burst access. The burst controller will then insert wait states during the burst operation and be responsible for asserting i_stern indicating the availability of

4



DRAM data. The burst controller will also generate the counting and load controls for the burst address generator and provide the burst multiplexing control to the DRAM address multiplexer.

Burst Address Generator

The burst address generator provides the two least significant bits of the DRAM address during a 68030 burst cycle. The burst controller will initiate the loading of the first burst address into the burst address generator and then control the incrementing of the addresses for the next three long word accesses of the burst cycle.

The burst address generator sequences through the long word addresses, which are generated from the ba(3:2) signals. Entry into the counter state machine can occur at any state and will be defined as the starting 68030 starting address plus one. During the first long word access of the burst, the first address will be supplied by the 68030; the next three long word addresses will be supplied by the burst address generator. After the first 68030 address, the address generator will enter the state of the next address from the load signal from the burst controller. The burst address generator will then be incremented two more times for the next two long word addresses.

4

DRAM Address MUX

The DRAM address MUX provides the row and column addresses on a eleven bit multiplexed address bus to the DRAM array. The 68030 provides the row address to the DRAM array, and the column address of the first long word access of a burst cycle. The two least significant bits of the column address will be supplied by the burst address generator for the next three long word accesses in the burst

cycle. Gating of the addresses onto the DRAM multiplexed address bus is controlled by both the burst controller and the RAS/CAS controller. The DRAM address MUX generates a 68030 burst access to long word address locations 01xxxx0h to 01xxxxCh, when the starting 68030 address is 01xxxx0h.

Bus Error Time-out

A bus error watch dog time-out function is provided by the DRAM controller to keep the 68030 from locking up due to accesses into unused memory. The bus error time-out controller monitors the assertion of address strobe and will generate a bus error to the 68030 if it has kept address strobe asserted from 40.96 usec to 46.08 usec. This time-out may vary depending on where the 68030 started a memory access in relationship to ref[7] of the refresh timing generator. The bus error time-out controller monitors ref[7] for its state transitions, while watching the assertion of address strobe. If address strobe is negated before reaching the state NOACK of the bus error time-out state machine, a bus error will not be generated.

System Timing

The system timing is shown in Figure 5 and gives the overall operation of the modules within the DRAM design for a DRAM array access. The diagram shows the logical implementation of the design with zero propagation delay and is meant to give a relationship of the signal handshaking between the submodules of the design. The system timing diagram shows a four long word burst read access to the DRAM array and is comprised of a 14-7-7-7 burst for a total of 35 cycles. The design has not been optimized for speed at this time, with the intent of generating a reasonable amount of logic for timing verification targeting lower cost designs using slower DRAMs.

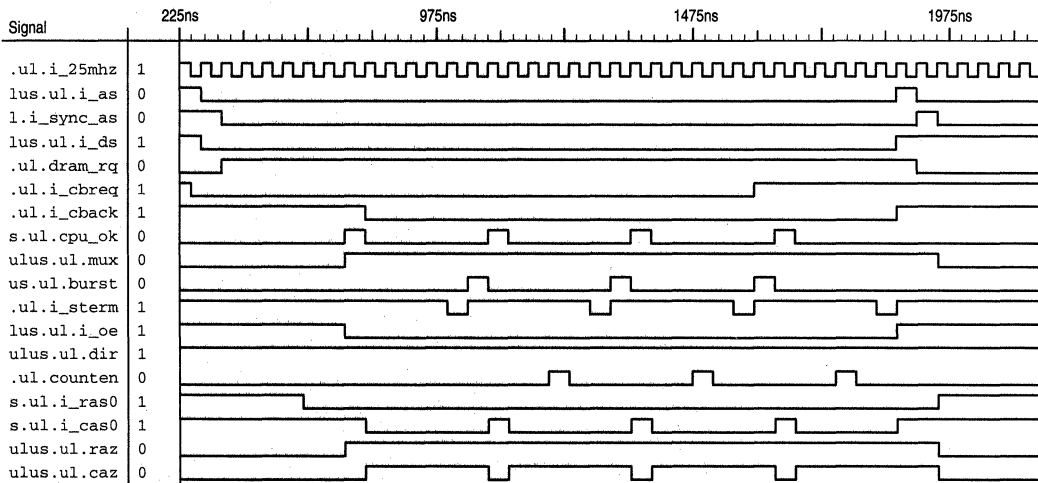


Figure 5. DRAM Burst Timing



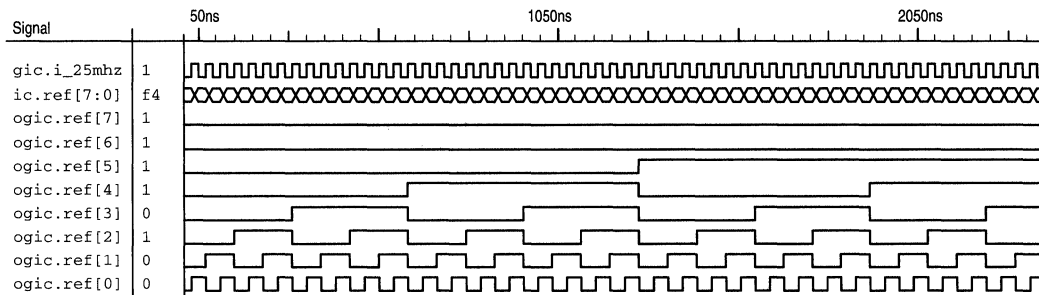


Figure 6. Refresh Counter Timing



Figure 7. Refresh Timing

Figure 5, Figure 6 and Figure 7 timing diagrams are logical simulations of the design copied from the Frontline waveform timing analyzer and do not include final place and route timings.

Verilog Coding Example

The following Verilog synthesis code describes the refresh module:

```
module refresh (i_25mhz, ref_cycle,
               timerclk, ref_rq);
```

```
input i_25mhz;
output timerclk;
wire [7:0] temp;
reg [7:0] ref;
```

```
assign timerclk=ref[7];
```

```
assign temp[0] = ~ref[0];
assign temp[1] = ref[1] ^ ref[0];
assign temp[2] = ref[2] ^ (&ref[1:0]);
assign temp[3] = ref[3] ^ (&ref[2:0]);
assign temp[4] = ref[4] ^ (&ref[3:0]);
assign temp[5] = ref[5] ^ (&ref[4:0]);
assign temp[6] = ref[6] ^ (&ref[5:0]);
assign temp[7] = ref[7] ^ (&ref[6:0]);
```

```
always @ (posedge i_25mhz)
begin
ref[0] = temp[0];
```

```
ref[1] = temp[1];
ref[2] = temp[2];
ref[3] = temp[3];
ref[4] = temp[4];
ref[5] = temp[5];
ref[6] = temp[6];
ref[7] = temp[7];
end
```

```
// Refresh request state machine
```

```
input ref_cycle;
reg [1:0] ref_states;
output ref_rq;
reg ref_rq;
```

```
parameter IDLE = 'b00; // idle state
parameter REFREQ = 'b01; // assert ref rqst
parameter REFEND = 'b10; // wait for end
```

```
always @ (posedge i_25mhz)
begin
```

```
case (ref_states)
IDLE:begin
if (ref[7])
begin
ref_states = REFREQ;
ref_rq = 1;
end
end
if (~ref[7])
```

4



```

begin
  ref_states = IDLE;
  ref_rq = 0;
end
end
REFRQ:begin
  if (ref_cycle)
    begin
      ref_states = REFEND;
      ref_rq = 0;
    end
    if (~ref_cycle)
      begin
        ref_states = REFRQ;
        ref_rq = 1;
      end
    end
  end
REFEND:begin
  if (~ref[7])
    begin
      ref_states = IDLE;
      ref_rq = 0;
    end
    if (ref[7])
      begin
        ref_states = REFEND;
        ref_rq = 0;
      end
    end
  end
endcase
end
endmodule

```

4

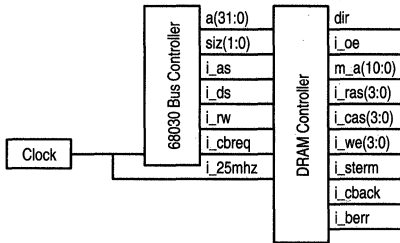


Figure 8. Simulation Model

MPA – Verilog Simulation

Figure 8 shows a block diagram of the Verilog stimulus module used for logic simulation and includes the 25MHz clock module for generating a system clock, the 68030 bus controller module for generating the timing for a 68030 burst cycle, and the glue68k DRAM Controller module. Figure 5 to Figure 7 show the results of the simulation, which was run from 0ns to 10,000ns. The stimulus module included the vectors for generating the 68030 bus signal timing for a burst read cycle. Once the simulation code was verified logically by the simulator and waveform analyzer, the code was determined to be free from syntax errors and matched the

expected timing for the design. Note that at this point, the design has not been verified with the gate and path delays generated from the final place and route tool.

MPA – Exemplar Synthesis

Prior to synthesis, the MPA1000 libraries, that are supplied with the MPA design system place and route software must be properly installed into the directory pathname C:\exemplar\lib and include the filenames p_mpa20.syn and p_mpa23.syn. Exemplar will reference these libraries for gate type selection when targeting a MPA1000 series part.

Although Exemplar has a graphical user interface, this designer preferred to use DOS synthesis commands included in (.bat) batch files. As the design was synthesized, the file manager was used to navigate through the design's subdirectories and to synthesize by double clicking on the batch file contained in a submodule's directory.

The refresh submodule example is synthesized with the DOS command "fpga refresh.v refresh.vg -target=p_mpa -save -macro". The save option stores all the optimization passes of the logic synthesizer allowing the user to select the best pass based on timing and cell count size. The macro option is used for inhibiting the assignment of I/O pads to the inputs and outputs of the submodules, reserving I/O pad assignments to the top module. Each submodule of the design is synthesized separately using the same command, but with different filenames that are identical to the submodule name. The top module glue68k is synthesized with "fpga glue68k.v glue68k.vg -target=p_mpa -pass=2" and uses a control file within its directory called glue68k.ctr, which includes the command:

```

BUFFER_SIG IPCLK i_25mhz

```

to assign the signal i_25mhz to a clock tree within the MPA1036. The glue68k.vg Verilog netlist is edited manually, adding include commands to the end of the file to read in all the Verilog netlists into the top hierarchical module during the Exemplar reformat pass. The include command:

```

#include "c:\fpga\refresh.vg"

```

reads in the refresh submodule into the glue68k.vg module when the DOS command "fpga glue68k.vg glue68k.edif -source=p_mpa -target=p_mpa -effort=reformat", is executed in the fpga directory with the glue68k.ctr control file removed. The final design resulted in 440 gates including 44 DFs, 40 inputs, and 25 outputs, when targeted to a Motorola Programmable Array.

The types of gates that were synthesized by the Exemplar tool for the DRAM Controller design included:

AN2	INV	ONE
BUFF	IPBUF	IPCLK
OPBUF	DF	ND2
OR2	DFR	NR2
YN2	XR2	

The synthesis times for the modules varied with the complexity of the logic, but were relatively fast. For instance the dram module, which is a fairly complex state machine design, took seconds to synthesize as shown:



Pass	Cells	Delay (ns)	Min:Sec
1	153	28.8	00:13
2	93	8.4	00:10
3	155	28.8	00:18
4	88	18.0	00:09
5	153	27.6	00:14
6	88	18.0	00:08
7	155	28.8	00:15
8	88	18.0	00:08
9	187	26.4	00:21
10	98	9.6	00:35
11	91	16.8	00:33

The passes of the Exemplar synthesizer are related to eleven different types of optimization algorithms. The best pass for the dram module, based upon timing, is pass 2, with an estimated gate delay of 8.4ns. In general all of the modules synthesized in this design, had pass two consistently generate the least amount of level delays. One may save some CPU time by specifying that a particular pass be executed by the Exemplar tool.

MPA Design System Place and Route

The design was processed by five steps using the MPA design system graphical user interface: Set Tool Options, Import, Autolayout, Generate Configuration, and Generate Back-Annotation. The input to the place and route tool is an EDIF netlist and requires the Exemplar EDIF netlist file glue68k.edi to be renamed to glue68k.edn to be imported.

MPA design system options that are required to be selected prior to place and route include: part number, package type, and mode. For this design the part was a MPA1036HI, which is a Motorola 181 pin, 8000 MPA equivalent gates, 3600 cell array. The mode determines how the device will be configured upon power up and reserves programming pins on the device to prevent the place and route tools from assigning them to user I/O. The mode selected for the design was "Boot From ROM", where the MPA loads its program from a serial ROM. The Autolayout place and route option was set to use default settings and provided adequate delay timings at 25MHz. Optional parameter settings for high utilization and for minimum delay, which are intended for compact and high speed designs respectively.

The Autolayout tools at default settings, generated a design that used 376 cells, utilizing 20.9% of the device, with an estimated maximum frequency of 30.7MHz. With the minimum delay option selected, the design routed with an estimated maximum frequency of 39.4MHz. The user may experiment with different option settings to generate faster designs, but for this application the 30.5MHz output was adequate and was used for the final design.

Pin assignments for the design can be viewed in the pin report file glue68k.prp with a small section of the report shown here:

I/O Pin report file
Definition: glue68k
Layout: glue68k

Format: Port Name, Net Name, Device Coord, Internal Pad No, Package Pin Name

```
Port   i_25mhz, net   i_25mhz @ ( 36, 0)
IO pad 15 pin n8
Port   i_sterm, net   i_sterm @ ( 20, 0)
IO pad 8 pin p4
Port   siz1, net      siz1 @ ( 78, 40)
IO pad 166 pin h13
```

The Back-Annotation tool is used to generate a structural Verilog netlist for post simulation and assigns the file extension of .vba, which was renamed to .v for input into the simulator. The designer can verify the final design with post simulation or by viewing the timing report generated by the Autolayout tool.

H4C Gate Array

The Verilog netlist files for the design were transferred on a DOS disk to a Sun workstation. Synopsys was used to read in the top hierarchical glue68k module and each of the submodules of the design. The design was synthesized and targeted to the H4C gate array family without any errors. The combinational area of the design was 446 and the noncombinational area was 344 (43 flip-flops using 8 gates per flip-flops) with a total used area of 790.

The types of gates generated by Synopsys include:

AND2	INV2	NOR8H
AND2H	INVB	OA211H
AND3	MUX2A	OA21H
AO22H	MUX2I	OA22H
AOI22H	MUX2IH	OAI211H
DFFP	NAN2	OAI22H
DFFRP	NAN3	ONDAI22H
EXNORA	NAN4	OR2
EXORA	NOR2B	OR3
INV	NOR2H	OR4

One observation of the synthesized design was that the Synopsys synthesizer added buffers to heavily loaded signals to minimize the wire delays and edge rates in the design. Another observation indicated that the gates that were generated for the MPA and H4C gate array were quite similar because of the fine grained nature of the MPA.

Final Simulation

The final placed and routed MPA design was post simulated with the back-annotated structural Verilog netlist using the Verilog simulator. Prior to post simulation, the MPA design system Verilog library file, located in the path C:\dpld\verilog\library.v was edited to enable Verilog XL compliance with the command 'define XL_comp. Another modification of the library needed to eliminate errors encountered during post simulation was that the library description of the module ONE was changed from:

4



```
'ifdef XLcomp
  pullup (strong1,strong0) (PU);
'endif
to:
'ifdef XLcomp
  pullup (strong1) (PU);
'endif
```

The structural Verilog netlist file glue68k.v was also edited to declare the netlist as type back-annotated by enabling the line: 'define source_back_annotation

The post stimulation file stimulus.v is similar to the simulation file, but was edited to change input and output names from lower case to upper case, caused by the renaming of signals from the MPA design system back-annotation tool. The following is a subset of Verilog simulation module stimulus.v, that was used for the test bench and includes a 25MHz free running clock:

```
/
'timescale 1ns / 1ps
module stimulus ;
  wire I_25MHZ;
  ...
  wire DIR;
  clk25  clockGen (I_25MHZ);
  GLUE68K u1(I_25MHZ, I_AS, D_I_AS,
    I_DS, I_RW, I_CBREQ, SI_Z1,
    SI_Z0, L_A31, L_A30, L_A29,
    L_A28, L_A27, L_A26, L_A25,
    L_A24, L_A23, L_A22, L_A21,
    L_A20, L_A19, L_A18, L_A17,
    L_A16, L_A15, L_A14, L_A13,
    L_A12, L_A11, L_A10, L_A9,
    L_A8, L_A7, L_A6, L_A5,
    L_A4, L_A3, L_A2, L_A1, L_A0,
    I_CBACK, I_STERM, BCYCLE,
    M_A10, M_A9, M_A8, M_A7,
    M_A6, M_A5, M_A4, M_A3,
    M_A2, M_A1, M_A0, I_CAS0,
    I_CAS1, I_CAS2, I_CAS3, I_RAS0,
    I_RAS1, I_RAS2, I_RAS3,
    I_BERR, I_OE, DIR);
  initial
  begin
    u1.B_A3 = 0;
    u1.B_A2 = 0;
    ...
```

```
u1.CONTROL_VL8 = 0;
end

// simulate a 68030 DRAM burst cycle
initial
begin
#5  u1.L_A31 = 0; u1.L_A30 = 0;
    u1.L_A29 = 0; u1.L_A28 = 0;
  ...
end
endmodule
module clk25 (clock);
output clock;
reg clock;
initial
  #5 clock = 0;
always
  #20 clock = ~clock;
endmodule
```

Frontline's graphical user interface was invoked and a project called glue68k.dgn was created. Setup of the simulator included setting directory pathnames to the locations of the Verilog source files stimulus.v and glue68k.v and of the MPA design system Verilog library file library.v. The simulator was setup for maximum delay type and to use the +heirinstport command line option to allow the use of hierarchical pathnames used in the glue68k hierarchical design. The simulation was run and the timing of the design was verified with the waveform analyzer as illustrated in Figure 9 and Figure 10. Note that timing waveforms show accurate gate and path delays within the MPA1036.

Appendix A – Design Tools

The design tools were selected for a 486 PC Platform and included Frontline Design Automation, Inc's PureSpeed Verilog Simulator, Exemplar's CORE-TD-DOS PC Topdown Verilog Synthesizer, and MPA design system. The PC was upgraded to 24MBytes of DRAM memory, of which 16MBytes were the minimum required to run the Exemplar software. A CD-ROM drive was used for loading the MPA design system software. Waveforms included in this application note were captured from Frontline's waveform analysis tool for both logical and post simulation figures.

For targeting H4C gate arrays, the design development tool kit was Motorola's Open Architecture CAD System (OACS). Synopsys was used for Verilog logic synthesis on a Sun platform in one of Motorola's ASIC design centers.



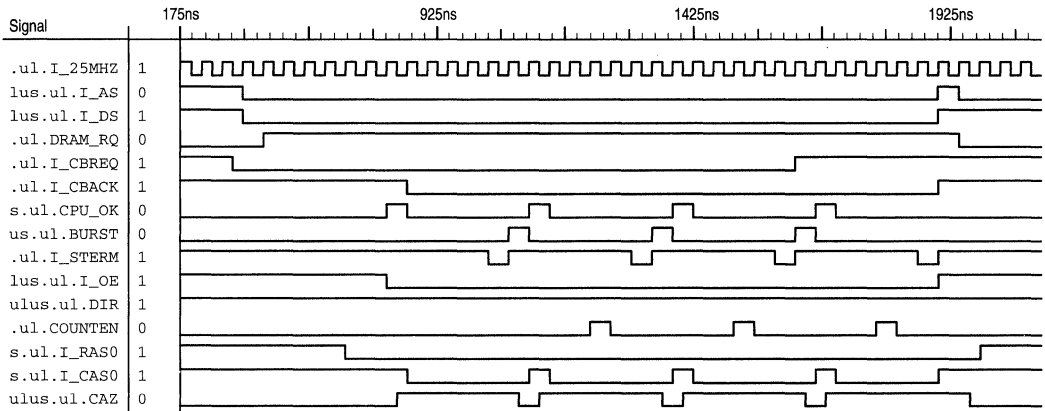


Figure 9. DRAM Burst Timing Final Simulation

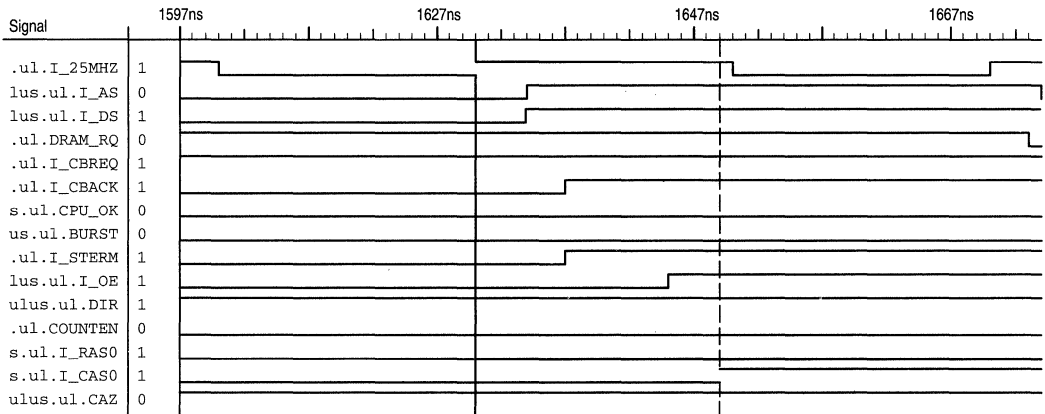


Figure 10. DRAM Burst Timing Final Simulation – Delay Example

4



Programming Multiple MPA1000 Devices Using Serial Peripheral Interface (SPI)

Prepared by
Ajay Matani
Field Applications Engineer

4



Programming Multiple MPA1000 Devices Using Serial Peripheral Interface (SPI)

Introduction

Serial Peripheral Interface (SPI) is an efficient on-board Serial Data Transfer mechanism supported by most of Motorola Microcontrollers. MPA1000 series arrays offer various modes of loading "ConfigWARE" (Configuration data that defines MPA logic functionality and interconnect) data into the device. This application note details a microcontroller MPA configuration control interface using an SPI port.

Why Use SPI for "ConfigWARE" Download?

In-system programmability is not a new concept, as most SRAM based MPA's provide a mechanism for the Microprocessor to configure functionality. For embedded systems, Hardware and Firmware constitute a typical system. Sophisticated embedded systems like Laser Printers provide support for downloading "SoftWARE" (as Fonts, Printer Emulation etc.) for example; while FLASH EEPROMS allow for "FirmWARE" upgrade as is a case in many new PC-motherboards that have BIOS in FLASH. As shown in Figure 1, flexibility offered by these different layers decreases as we approach the HardWARE layer, which is quite fixed.

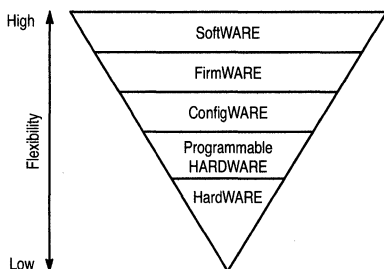


Figure 1. Programming Flexibility

"ConfigWARE" provides the flexibility to use the same "HardWARE" to carry out different functionality. The time and resources required to download "ConfigWARE" into the MPAs becomes critical as device size and number of devices in a system increase. It is also beneficial to store "ConfigWARE" along with "FirmWARE" in non-volatile memory like FLASH, Floppy, HDD or download over a Network connection.

Many of Motorola's highly integrated MCU devices have on-chip resources (such as RAM, PROM, serial ports etc.) that enable independent boot-up and loading of the "ConfigWARE". Considering that most of them also have SPI support, it is worthwhile to examine efficient use of SPI for downloading "ConfigWARE".

MPA1000 Configuration Methods

Four basic methods are available to download "ConfigWARE" into the MPA devices; one Micro Mode with

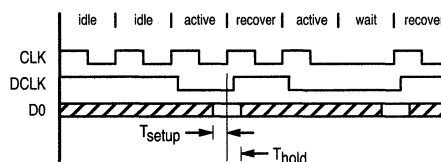
typical peripheral bus interface and three BFR (Boot From ROM) modes. The Micro Mode, BFR Mode(1) and BFR Mode(3) support byte wide data transfer hence BFR Mode(2) which supports serial data transfer is the only one we consider for SPI interface. In fact, MPA configuration logic supports 8-bits at a time and thus accumulates the serial stream into a byte before loading it in the internal RAM array. This arrangement matches well with the SPI support of byte data transfer at a time on the serial protocol.

BFR Mode(2) operation is a very simple serial transfer mechanism that uses 3 signals, CLK(clock in), DCLK(clock out) and D0(data). This mode is intended to load from external serial PROM devices like MPA17128 (page 1-6). The signal relationships are as follows:

CLK (up to 20Mhz) is the master clock used by configuration logic. This could also be generated from the MPA internal Ring Oscillator.

DCLK is output from the MPA and can run as fast as 1/2 the CLK frequency.

A simple way of looking at DCLK is to consider it as a Data Strobe and clock for an Address Counter, where DCLK low to high transition is the critical edge for both operations (Figure 2). As each transition of DCLK is generated by a rising edge of the CLK signal, manipulating CLK allows controlling DCLK operation. D0 is data presented to the MPA.



NOTE: During "idle" phase of CLK, internal Reset or Configuration Sequence logic is being exercised. At the end of "active" phase of CLK, new configuration data is read in. The "wait" phase of CLK is created by stretching the CLK low phase of the active cycle. After every "active" or "wait" condition, a "recover" cycle of CLK is needed.

Figure 2. ConfigWARE Download Timing

By extending DCLK in its low state, wait states can be inserted in the access of serial data on D0. This can be easily achieved by keeping CLK in low state whenever needed. As CLK is used by the configuration logic also, the suggested clock stretching should be applied only during data access cycles denoted by DCLK low state. Since the configuration logic is a static design, there is no minimum operational frequency requirement allowing large number of wait states if needed.

The window for which D0 should be stable with valid data is defined by Figure 2-33, Figure 2-34 and the accompanying table on page 2-27. This relatively narrow window requirement is easily achievable.

4



Serial Peripheral Interface

SPI operation as well, is quite straightforward. The SPI on a MCU can be configured as either a Master or a Slave. The serial transfer operation is carried out on four lines, SCK (Serial Clock), MOSI (Master Out Slave In), MISO (Master In Slave Out) and SS/(Slave Select) supporting synchronous bi-directional serial transfer of byte size data.

The primary difference between the Master and Slave Mode is the source of SCK. Though transfers are synchronous, SPI circuit is required to be a static design which allows the SCK to have no maximum phase or period time requirement. In Master Mode, the MCU based SPI circuit sends a burst of 8-bits, synchronized to a prescaled internal CPU clock.

The programmer's model of SPI consists of SPDR (8-bit SPI data register), SPCR (SPI control register) and SPSR (SPI status register). Refer to MC68HC11RM/AD for detailed discussion of SPI operation.

Once configured as a Master, writing to SPDR starts a transfer of the data byte uninhibited, on the MOSI signal. The data presented on MISO by the slave is de-serialized and made available in the SPDR at the end of byte transfer. As writing to the SPDR also starts a transfer sequence, there is no facility to carry out hand shake with the slave apriori to the transfer.

When SPI on a MCU device is configured as a slave, the SCK supplied by the external master controls the flow of transfer. MOSI now becomes input and MISO becomes output. The SS/ signal plays an important role in this mode, acting as a gate to the SCK. This facilitates selective transfer to multiple slave using common SCK signal.

DESIGN APPROACH

This application note is based on a design implemented in a working system with multiple MPA1036 devices daisy chained. The requirement of this system is to provide a flexible and efficient "ConfigWARE" download capability. The design uses MC68HC11K4 MCU with external FLASH EEPROM to store the "ConfigWARE".

Various possibilities were considered to establish the lower level handshake with MPA configuration logic. BFR Mode(2) was chosen as it sacrifices only one general purpose i/o signal for configuration, namely DCLK.

The easiest, glueless and trivial method of interfacing an MCU to an MPA device in BFR[2] is a single, 8-bit I/O port and 100% software controlled transfer. MCU software overhead results in a long MPA subsystem start-up time if this method is used. As the size of the MPA subsystem increases, this problem is compounded. Using the MCU SPI hardware in Slave Mode and a minimal amount of external logic, relatively fast configuration times can be achieved using a serial data stream.

FUNCTIONAL DESCRIPTION

We have established the basic serial transfers in terms of MPA BFR Mode(2) and SPI in our discussion up to this point, but there are a few more functionalities that need to be examined at system level. Consider that there are n MPA devices daisy chained as shown in Figure 3. The MCU (MC68HC11 in this discussion) and PAL device constitute the controller.

The controller requires only three outputs (CLK, D0 and RESET1/) to carry out the task. The POWRUPn signal may be monitored to confirm the end of download sequence, though any error condition may be detected alternatively by making sure that exact number of bytes are downloaded.

4



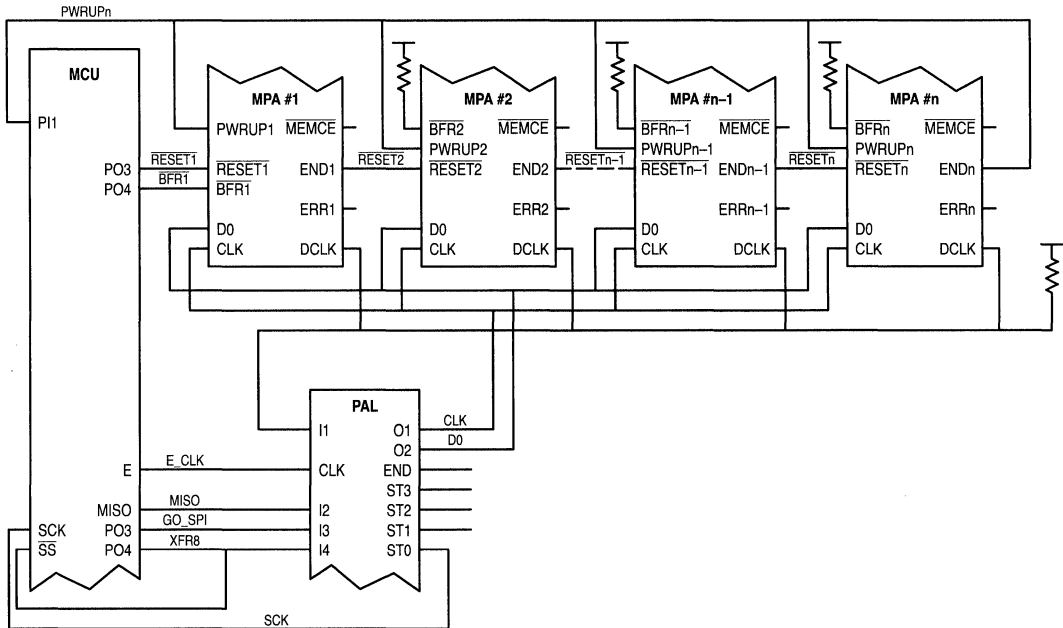


Figure 3. Multiple Daisy Chained MPA Devices

4

Let us look at all the signals in detail.

Signals outside of the controller :

CLK Derived from PAL External clock input. Used during reset and configuration sequence. Clock stretching is used in this application during the Configuration sequence to establish proper handshake between the MPA and program controlled sequence on the MCU.

RESETr Output from MCU Reset to the First MPA in the daisy chain. The falling edge initiates reset sequence. At the end of reset sequence, if the signal is still asserted, configuration sequence is delayed till the rising edge of signal is recognized. Otherwise, the configuration sequence immediately follows the reset sequence. CLK must be active during these sequences.

RESETr Connect ENDn-1 to RESETr as configuration logic keeps END low till the present device completes the configuration sequence and lets the next MPA device start its own reset and configuration sequence.

BFR1 Output from MCU. Active low signal initiates Reset and Configuration sequences. Acts same as RESETr except that the Configuration sequence immediately follows the reset sequence

regardless of the state of BFR1 signal. (The use of this signal is optional).

BFRn These are pulled up.

PWRUPn Last MPA END connects to all PWRUP signals. When this signal on the MPA is low, all the user I/O are disabled (in tri-state). A desirable condition until all the devices are configured properly.

MEMCE Not used.

ERRn These output signal gets asserted during configuration sequence if Device ID mismatch or Checksum error is detected. They can be left open as such error condition stops configuration sequence and can be detected alternatively.

DCLK Input to PAL. Is the wired-or signal (requires external pull-up) from all the MPA devices that gets pulled low by the currently configuring device during data transfer.

D0 Output from PAL. This is the data input to all the MPA devices.

Signals inside the controller (between MCU and PAL) :

E_CLK Output from MCU. System clock.

MISO Output from MCU. SPI data out of MCU in Slave Mode.



- GO_SPI** Output from MCU. Sequence Master Control output, resets and enables configuration sequence logic under program control.
- XFR8** Output from MCU. Byte Transfer Control output, initiates data transfer sequence logic under program control. Also acts as Slave Select for SPI logic with connection external to the MCU.
- SCK** Output from PAL. SPI clock input from MCU. Controls bit data transfer out of SPI shift register.

CONTROL LOGIC IMPLEMENTATION

The MC68HC11K4 and a PAL22V10 device is used for the control logic.

The base clock for the circuit is the "E" clock from MCU that is also used by the SPI logic internal to the MCU. Typical frequencies for 8-bit MCUs for the system clock "E" are 4 Mhz and as high as 25 Mhz for some 32-bit MCUs.

The circuit uses four inputs, three outputs and four state bits on the PAL. One of the outputs, SCK is considered as a state variable too. Refer to Appendix A, where the PAL design in CUPL (need to check trade mark) source language is described. Appendix B lists the Boolean equations in AND-OR form for each of output and state variable as generated by CUPL assembler.

Two general purpose Output Port bits (GO_SPI and XFR8) from the MCU are required besides the SPI signals. Considering that data is flowing from MCU to the MPAs, only data out signal (MISO in slave mode) of the SPI logic is used along with the SCK and SS signals.

Appendix C lists the assembly source code for MC68HC11K4 MCU for the subroutines needed to carry out the ConfigWARE download.

CONTROL FLOW

For ease of understanding, let us follow the firmware in Appendix C to track the operation of control sequence.

The calling function to "LD_FPGA" is assumed to have set the general purpose port bits to the correct direction and have made a call to "RST_FPGA" which forces RESET1/. The first thing LD_FPGA does is to call "EN_FPGA" which makes the control logic ready by negating XFR8, correctly set up the SPI on MCU, assert GO_SPI and negate RESET1/.

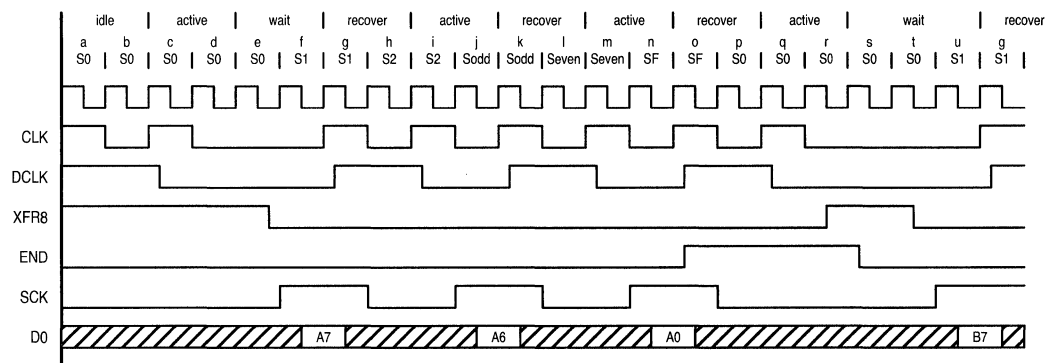
This forces the sequencer in the PAL to STATE S0 and stay there till sequence begins when XFR8 is asserted (active low). The CLK output of PAL keeps on toggling while in STATE S0, defining the "idle" phase of CLK (Fig. 4). Negated XFR8 also negates END state variable.

Next, LD_FPGA makes sure that the correct ConfigWARE for the first device is made available (starting at address \$4000) before it calls the "IM2FPGA" (Image to FPGA) function. IM2FPGA calls "BY2FPGA" for the exact number of times to download the complete image for a single FPGA device. For FPGA1036 for example, the number is 14600 (\$3908) bytes as explained in Appendix D.

BY2FPGA is the lowest level routine that directly controls the transfer of byte to the current FPGA device. The data byte is written to the SPI data register and XFR8 is made active to begin the transfer. While the sequencer in PAL and MPA synchronize and carry out the transfer, the program waits for SPI transfer to complete within a certain period of time. If the code times out, it returns with error condition set. As only the completion of SPI transfer is waited on, there is no need for any external signals to indicate error conditions.

4





NOTE:

a ---> b : S0>S0, as DCLK is high	k ---> l : Sodd>Seven, as DCLK is high
b ---> c : S0>S0, as DCLK is high	l ---> m : Sodd>Seven, as XFR8 is high
c ---> d : S0>S0, as XFR8 is high	m ---> n : Seven>SF, as DCLK is low
d ---> e : S0>S0, as XFR8 is high	n ---> o : SF>SF, as DCLK is low
e ---> f : S0>S1, as DCLK and XFR8 low	o ---> p : SF>S0, as DCLK is high (END set)
f ---> g : S1>S1, as DCLK is low	p ---> q : S0>S0, as DCLK is high
g ---> h : S1>S2, as DCLK is high	q ---> r : S0>S0, as DCLK is low but END is high
h ---> i : S2>S2, as DCLK is high	r ---> s : S0>S0, as DCLK is low but END is high
i ---> j : S2>Sodd, as DCLK is low	s ---> t : S0>S0, as DCLK is low, END is low but XFR8 is high
j ---> k : Sodd>sodd, as DCLK is low	t ---> u : S0>S1, as DCLK is low, END is low and XFR8 is low
	u ---> g : catch the sequence at g again

4

Figure 4. System Training

For the normal transfer, the sequencer in the PAL waits in STATE S0 when MPA device is not ready for transfer as indicated by DCLK high. When FPGA is ready to receive data, it asserts DCLK low. State m/c responds by stopping the transitions on CLK signal, forcing wait condition till MCU holds XFR8 signal high. When MCU program sequence catches up and asserts XFR8 low, STATE S1 is entered.

All odd numbered states are similar and correspond to the time when DCLK is high. Except for STATE S0, all even number states correspond to the time when DCLK is low. For a byte transfer, XFR8 going low should run the sequencer from STATE S0 to STATE SF in a sequence, and back to STATE S0. When the state m/c is in STATE SF, END state variable gets set on the next clock, indicating completion of a transfer. Until XFR8 is negated by BY2MPA when it detects completion of SPI transfer, the END condition forces the state m/c to stay in STATE S0, even when the MPA is ready to receive the next

bit, indicated by DCLK low. This mechanism ensures that one and only one byte is transferred on every MCU controlled cycle of XFR8.

The new transfer will not start till program sequence makes a new call to BY2MPA, which in turn will begin with XFR8 active. Timeout or SPI error condition, if any, makes BY2FPGA return a non zero value indicating error. The address value of the image byte of the erroneous transfer is saved in SPI_ERR variable. IM2FPGA passes the error back to LD_FPGA.

If there is no error, LD_FPGA repeats the above process for additional devices, making sure that the correct ConfigWARE for that device is addressed.

If an error occurs, the sample code jumps to "LD_FERR". The handling of error is left to the calling routine and user interface.



APPENDIX A.

```

/*****
/* MPA1000 series FPGA configuration logic with SPI */
/* HC11 companion PAL */
/*****
Device      p22v101cc;
/** Pin Assignments */
/* Clock and Inputs */
PIN 2      = MCEK;      /* Input - Register Clock */
PIN 3      = MISO;     /* Input - Serial data, HC11 */
PIN 4      = XFR8;     /* Input - Transfer control, HC11 */
PIN 11     = GO_SPI;   /* Input - Master Control, HC11 */
PIN 6      = DCLK;     /* Input - CLK feedback from FPGA */

/* Outputs and State variables */
PIN 20     = CLK;      /* Output - clock to FPGA */
PIN 21     = D0;       /* Output - Data out to FPGA */
PIN 23     = ST3;      /* State - var 3 */
PIN 24     = ST2;      /* State - var 2 */
PIN 25     = ST1;      /* State - var 1 */
PIN 26     = ST0;      /* Output - SPI clk to HC11 */
                /* State - var 0 (Dual function) */
PIN 27     = END; /* State - End condition */

/** Declarations and Intermediate Variable Definitions */
field count = [ST3..0]; /* declare counter bit field */
#define S0 'b'0000 /* define counter states */
#define S1 'b'0001
#define S2 'b'0010
#define S3 'b'0011
#define S4 'b'0100
#define S5 'b'0101
#define S6 'b'0110
#define S7 'b'0111
#define S8 'b'1000
#define S9 'b'1001
#define SA 'b'1010
#define SB 'b'1011
#define SC 'b'1100
#define SD 'b'1101
#define SE 'b'1110
#define SF 'b'1111

/** Logic Equations */

CLK.d      =      !CLK & DCLK
                /* High if idle or restore state */
                #      !CLK & !DCLK & !(ST3 & ST2 & ST1 & ST0);
                /* High if data ready in active state */

CLK.sp     =      'b'0;
CLK.ar     =      'b'0;

END.d      =      ST3 & ST2 & ST1 & ST0
                #      END & !XFR8;
                /* Count has expired but no new XFR8 */

END.ar     =      'b'0;
END.sp     =      'b'0;

D0         =      !GO_SPI & MISO /* active low GO_SPI */
                #      GO_SPI & XFR8; /* Slave Select signal */

```

4



```

ST0.ar      =      'b'0;
ST1.ar      =      'b'0;
ST2.ar      =      'b'0;
ST3.ar      =      'b'0;

ST0.sp      =      'b'0;
ST1.sp      =      'b'0;
ST2.sp      =      'b'0;
ST3.sp      =      'b'0;

ST0.oe      =      !GO_SPI;      /* Master control      */

sequence count {      /* free running counter      */

present S0  if      !DCLK & !END & !XFR8
             next S1;
             if      DCLK
                 # !DCLK & END
                 # !DCLK & XFR8
             next S0;

present S1  if      !XFR8 & DCLK
             next S2;
             if      !XFR8 & !DCLK
             next S1;
             if      XFR8
             next S0;

present S2  if      !XFR8 & !DCLK
             next S3;
             if      !XFR8 & DCLK
             next S2;
             if      XFR8
             next S0;

present S3  if      !XFR8 & DCLK
             next S4;
             if      !XFR8 & !DCLK
             next S3;
             if      XFR8
             next S0;

present S4  if      !XFR8 & !DCLK
             next S5;
             if      !XFR8 & DCLK
             next S4;
             if      XFR8
             next S0;

present S5  if      !XFR8 & DCLK
             next S6;
             if      !XFR8 & !DCLK
             next S5;
             if      XFR8
             next S0;

present S6  if      !XFR8 & !DCLK
             next S7;
             if      !XFR8 & DCLK
             next S6;
             if      XFR8
             next S0;

```

4



AN1562

```

present S7  if  !XFR8 & DCLK
             next S8;
             if  !XFR8 & !DCLK
             next S7;
             if  XFR8
             next S0;

present S8  if  !XFR8 & !DCLK
             next S9;
             if  !XFR8 & DCLK
             next S8;
             if  XFR8
             next S0;

present S9  if  !XFR8 & DCLK
             next SA;
             if  !XFR8 & !DCLK
             next S9;
             if  XFR8
             next S0;

present SA  if  !XFR8 & !DCLK
             next SB;
             if  !XFR8 & DCLK
             next SA;
             if  XFR8
             next S0;

present SB  if  !XFR8 & DCLK
             next SC;
             if  !XFR8 & !DCLK
             next SB;
             if  XFR8
             next S0;

present SC  if  !XFR8 & !DCLK
             next SD;
             if  !XFR8 & DCLK
             next SC;
             if  XFR8
             next S0;

present SD  if  !XFR8 & DCLK
             next SE;
             if  !XFR8 & !DCLK
             next SD;
             if  XFR8
             next S0;

present SE  if  !XFR8 & !DCLK
             next SF;
             if  !XFR8 & DCLK
             next SE;
             if  XFR8
             next S0;

present SF  if  !XFR8 & DCLK
             next S0;
             if  !XFR8 & !DCLK
             next SF;
             if  XFR8
             next S0;
}

```

4



APPENDIX B.

```

/*****
/* LOGIC reduced to AND-OR equations (LISTING) */
/*****

CLK.d =>
    !CLK & DCLK
    # !CLK & !DCLK & ST3
    # !CLK & !DCLK & ST2
    # !CLK & !DCLK & ST0
    # !CLK & !DCLK & ST1
D0 =>
    !GO_SPI & MISO
    # GO_SPI & XFR8
END.d =>
    ST0 & ST1 & ST2 & ST3
    # END & !XFR8
ST0.d =>
    !DCLK & !END & !ST0 & !ST1 & !ST2 & !ST3 & !XFR8
    # !DCLK & !ST0 & ST1 & ST2 & ST3 & !XFR8
    # !DCLK & ST0 & !ST2 & !ST3 & !XFR8
    # !DCLK & !ST0 & ST1 & !ST3 & !XFR8
    # !DCLK & !ST1 & ST2 & !XFR8
    # !DCLK & ST0 & ST1 & ST2 & !XFR8
    # !DCLK & !ST2 & ST3 & !XFR8
ST1.d =>
    !DCLK & ST0 & ST1 & !XFR8
    # DCLK & ST0 & !ST1 & !XFR8
    # !ST0 & ST1 & !XFR8
ST2.d =>
    !DCLK & ST0 & ST1 & ST2 & !XFR8
    # DCLK & ST0 & ST1 & !ST2 & !XFR8
    # !ST0 & ST1 & ST2 & !XFR8
    # !ST1 & ST2 & !XFR8
ST3.d =>
    !DCLK & ST0 & ST1 & ST2 & ST3 & !XFR8
    # DCLK & ST0 & ST1 & ST2 & !ST3 & !XFR8
    # ST0 & ST1 & !ST2 & ST3 & !XFR8
    # !ST1 & ST3 & !XFR8
    # !ST0 & ST1 & ST3 & !XFR8

```

4

APPENDIX C.

```

; Code Excerpts for HC11 using SPI to configure MPA1000 series FPGA
; *****
; *   Define   *
; *****
DDRD: equ  REGBS+$09    ; port D Data Direction reg
SPCR: equ  REGBS+$28    ; spi control reg
SPSR: equ  REGBS+$29    ; spi status reg
SPDR: equ  REGBS+$2A    ; spi data reg
PORTG: equ  REGBS+$7E   ; port G data reg
PORTH: equ  REGBS+$7C   ; port H data reg

CFG1: equ  $11          ; Page 1    image for memory mapper
CFG2: equ  $22          ; Page 2    image for memory mapper
; more or less depending on number of FPGA devices in chain
CFGn: equ  $ff          ; Page n    image for memory mapper

```



```

; *****
; *      RAM      *
; *****

        org      $200          ; FLEXVAL ram area
SPI_ER: ds.b  2                ; Address of failed transfer if any
;SEC_IMG: ds.b  1              ; Sector image number for current 16K byte block
;                               ; in external FLASH EEPROM paged at address $4000
;                               ; Consider writing to this address for proper page
;                               ; selection of the ConfigWARE image.

=====
; *****
; Enable FPGA Transfers via SPI
; Entry      :      none
; Exit      :      SPI is enabled as SLAVE
;           :      GO_SPI is active
; *****
EN_FPGA:
        bset    PORTH,#$00      ; make sure XFR8 is inactive (high)
        bclr   PORTH,#$04      ; assert GO_SPI
        ldaa   #$44             ; SPE = 1 and CPHA = 1, rest 0
        staa   SPCR             ; to SPI control register
        bset   DDRD,#$04        ; MISO is made output PD[2]
        bset   PORTG,#$40       ; negate RESET1* at bit-6
        rts

; *****
; Disable FPGA Transfers via SPI
; Entry      :      none
; Exit      :      SPI is disabled
;           :      GO_SPI is inactive
; *****
DI_FPGA:
        bset    PORTH,#$00      ; make sure XFR8 is inactive (high)
        bset   PORTH,#$04       ; negate GO_SPI (high)
        bclr   DDRD,#$04        ; MISO is made input again PD[2]
        ldaa   #$04             ; SPE = 0 and CPHA = 1, rest 0
        staa   SPCR             ; to SPI control register
        rts

; *****
; Transfer byte to FPGA via SPI (ignore SPI errors if any)
; Entry      :      x = pointer to the byte
; Exit      :      a = 0 if o.k.
;           :      a != 0 if time out
;           :      x = x + 1 if o.k. else x is unchanged
; *****
BY2FPGA:
        ldaa   0,x
        staa   SPDR             ; write to SPI data register
        bclr   PORTH,#$01       ; XFR8 = low (active)
        ldaa   #$ff             ; time out counter
BY2_F1:
        tst    SPSR             ; check if spif is set
        bmi   BY2_F2             ; jump if transfer complete
        deca
        bne   BY2_F1             ; not complete, decrement time out
        bne   BY2_F1             ; check for time out
BY2_F9:
        stx    SPI_ERR          ; address at which error occurred
        ldaa   #$ff             ; indicate error condition
BY2_F8:
        bset   PORTH,#$01       ; XFR8 = high (inactive)
        rts

```

4



```

BY2_F2:
    tst    SPDR        ; dummy read to clear SPIF bit in SPSR
    ldaa  SPSR
    bne   BY2_F9      ; some error ?
    inx
    bra   BY2_F8      ; normal exit

; *****
; Transfer image to single FPGA (MPA1036 case)
; Entry   :      SEC_IMG has the current sector
; Exit   :      passes end condition of BY2FPGA to calling routine
; Notes  :      MPA1036 has 116800 bits
;          :      14600 bytes as follows
;          :          5 bytes of header
;          :      14595 bytes of data as follows
;          :          139 rows containing
;          :          105 bytes/row
;          :      address of byte after end of image
;          :      $3908 (when starting address is $0 )
;          :      $7908 (in this code, as ConfigWARE image is at $4000 )
; *****
IM2FPGA:
    ldx   #$4000      ; start of image in FLASH
IM2_F1:
    jsr   BY2FPGA
    tsta
    beq   IM2_F2      ; jump if no error
    rts              ; pass error up
IM2_F2:
    cpx   #$7908      ; end of image
    bne   IM2_F1      ; keep looping
    rts              ; return with a = 0

; *****
; Reset  FPGAs
; Entry   :      none
; Exit   :      FPGAs are forced into reset state
; *****
RST_FPGA:
    bset  PORTH,$$00   ; make sure XFR8 is inactive (high)
    bset  PORTH,$$04   ; negate GO_SPI
    bclr  PORTG,$$40   ; assert RESET1* at bit-6
    rts

; *****
; Program FPGAs
; Entry   :      none
; Exit   :      FPGAs are loaded and active unless error
;          :      In error case, a jump to LD_FERR routine (not shown here) is
made.
; *****
LD_FPGA:
    jsr   EN_FPGA

LD_F1:
    ldaa  CFG1
    staa  SEC_IMG
    jsr   IM2FPGA      ; do transfer
    tsta
    bne   LD_FERR      ; jump if error

LD_F2:

```

4



AN1562

```

ldaa CFG2
staa SEC_IMG
jsr IM2FPGA
tsta
bne LD_FERR ; jump if error

; more or less segments depending on number of FPGA devices in chain

LD_Fn:
ldaa CFGn
staa SEC_IMG
jsr IM2FPGA
tsta
bne LD_FERR ; jump if error

LD_END:
jsr DI_FPGA
rts

```

APPENDIX D.

ConfigWARE RAM Array Sizes for MPA1000 Device Family

4

	MPA1016	MPA1036	MPA1064	MPA1100
HEADER (bytes)	5	5	5	5
ROWS	95	139	183	227
Bytes/Row	72	105	138	170
Total Bytes	6845	14600	25259	38595
Total Bytes (Hex)	1ABD	3908	62AB	96C3
RAM Array Size (Relative to MPA1016)	1.00	2.13	3.69	5.64
Number of Cells (Relative to MPA1016)	1.00	2.25	4.00	6.25



Effective Synthesis Techniques for MPA1000 Devices

Prepared by
Thomas G. Felske
Wanhao Li
Motorola Programmable Logic Products

4



Effective Synthesis Techniques for MPA1000 Devices

Introduction

Logic synthesis has become an increasingly important issue in the FPGA design area due to the rapid growth of the FPGA design complexity. Many FPGA vendors offer synthesis design flows for designers who prefer synthesis over schematic design entry. This paper presents the critical architectural components of the MPA device and HDL techniques to achieve the best design performance from the MPA10000 synthesis design flow. The synthesis design flow includes Synopsys Design Compiler (Version 3.3b), Exemplar Galileo (Version 3.1), and Mentor Autologic which all map to the Motorola MPA1000 FPGA technology. Although this application note focuses on the Synopsys tool, most of the techniques are tool-independent and shall apply to the other synthesis tools.

Direct Mapping For Design Compilers

One of the biggest advantage of using the MPA1000 synthesis flow is that the MPA architecture requires a single mapping process for synthesis. The MPA1000 architecture is fine-grained and each basic cell can be configured directly to logic gates such as AND, OR, XOR, and Multiplexer. Since the MPA technology mapping is very ASIC-like, the synthesis tools can map the design with regular ASIC logic optimization algorithms. Consequently, this provides an convenient FPGA synthesis environment for designers who have ASIC experience or want to retarget FPGA designs to an ASIC at a later time. This is important to Motorola to be able to use the same tool for both ASIC and FPGA design. It not only saves designers from buying another tool but also prevents them from having to learn another synthesis tool.

To re-target Synopsys' Design Compiler from the FPGA technology to ASIC, Design Compiler's ".synopsys_dc.setup" file is modified to link to the ASIC target library. Upon startup, Design Compiler loads the library setup file to link to the desired synthesis library. The command lines in the setup file that link a target library are:

```
technology = ASIC or FPGA technology name
link_library = {technology + ".db" .....}
target_library = {technology + ".db"}
search_path = {Regular Synopsys path ASIC or FPGA
library path}
```

Logic Optimization and Technology Mapping

MPA1000 Architecture Resources

When partitioning functional blocks at the system level, the design capacity of the MPA device must be known to calculate how many FPGAs may be required for the system design in order to distribute the logic amongst the FPGAs. The calculations are bound by the physical limitations of the MPA device. The following sections define the physical limitations of the MPA architecture for various design resources. The limitations are related to the routing resources available to connect to the clock and reset pins for IOBs, wire-or and wire-and bus limits, clock resources, and set/reset resources for registers.

Although most of the logic optimization is done at the synthesis level, the MPA1000 backend system further optimizes logic by stripping back logic of unused output pins or signals, and optimizing inverters by cancelling out or pushing the inversion into the driven macro's input signal (bubble pushing).

Clock Signal Resources

Clock resources can use up to eight dedicated clock pad sites to connect to the dedicated clock tree resource.

The core array contains routing paths to go from primary clock resources to regular routing resources and vice versa. Therefore, the clock pins of the registers located inside core array can be routed from either primary clock tree or secondary clock tree which are implemented by regular routing resources. However, the clock pins of the I/O registers can only be driven by primary clock resources. If a clock signal is generated inside the core array, it can only drive registers inside the core array unless the signal is connected to the primary clock tree.

Regular routing signals inside the core array can be routed to primary clock resources and back, some high-fanout regular signals can be implemented with the primary clock tree and improve both routing congestion and routing delays. However, the designer must be aware of the limitation of the total number of primary clock trees.

Due to potential routing resource limitation, the total number of clock and reset signals (primary and secondary) should not exceed 15. If the total number exceed 15, place and route is very likely to fail.

Only five different clock/reset signals are allowed in each cell zone (10x10 cells) since each cell column is connected by the same clock signal. Among the five signals, only two can come from primary clock tree. In the I/O area, five I/O macros are aligned as an I/O segment and connected to a cell zone. Each I/O segment only allows 2 clock signals which need to come from primary clock trees. In general, place and route tools handle all the registers and clock signal placement to make sure no violations occur. However, if the designer changes or influences the cell or I/O placement manually by a control file, the clock signal limitation has to be followed. For instance, if all the I/O pins are fixed by a control "<design>.pat" file, it will cause partition failure if three of the I/O pins inside a I/O segment are driving three different clock signals.

Three-State/WOR Resource

The MPA1000 devices offer real three-state signals only in the I/O areas. Inside the core array, WOR structures are supported instead of a three-state structure. The I/O macros can utilize an WOR structure that connects to the P-bus resource. The internal WOR bus requires the designer to utilize the WPUP pull up macro with each bus and the PWPUP pull up macro for each P-bus WOR structure.

IOB Resource

When a designer assigns "port_is_pad" attributes and "insert_pads" to the top-level I/O "ports" that do not contain instantiated I/O macros, each I/O port will be inferred as an I/O macro. The default I/O inference for MPA1000 devices are:

4



```

clock input port   IPCLK
regular input port IPBUF
regular output port OPBUF
three-state port  OPTBUF

```

Each input macro (IPBUF) can be configured as TTL or CMOS levels. Each output macro (OPBUF) can be configured into 3V/5V, high drive or low drive, slow slew or fast slew rate. Those parameters can be inserted into the system through several different ways. The most convenient way is to use Design Compiler's "set attribute" command as follows:

```

set attribute {I/O instance name} attribute_name
attribute_value -type string

```

The I/O properties can also be inserted in a separate ASCII control (<design>.pat) file which will input those properties during the backend "import" stage. Refer to the on-line documentation for details of the "<design>.pat" control file.

There are two I/O structures that cannot be inferred by an behavior description; the open-drain structure and the registers inside I/O macros. Designers who need to use these two structures need to instantiate the desired macros. The following is a VHDL example for an open-drain macro:

```

U01: OPBUF port map (O=>internal_signal,
EXTOUT=>output_port);

```

A list of the special I/O macros with registers are in the on-line manual for more details on the available macros.

Peripheral Bus Resources (to route to IOB clock/reset pins)

The MPA1000 architecture contains an eight bit bus that runs the lengths of the chip next to the IOBs. This is the peripheral bus or P-bus. This resource is used for an interface resource for the IOB pins and global routing. This resource is not automatically used by the routing tool and it is up to the designer to instantiate the special P-bus macros for access to and from the P-bus. The special buffers are the APBUF, PABUF, PWPUP, APWBUFF, and APWINV. The prefix AP refers to an Array to P-bus connection and visa versa for PA. The W refers to the wire-or capability.

Set/Reset Resources

The internal core and the IOB registers have asynchronous set and reset capability that are mutually exclusive. The register contains a single pin that can be programmed for a set or reset function. The dedicated low skew reset tree is physically the same as the clock tree until the signal enters a zone. There the port cell(s) direct the signal to either a clock pin or reset/set pin.

MPA1000 Design Resources

The MPA1000 synthesis library contains special macros that aid in the efficiency of the FPGA design. Efficiency shall be described as guiding critical routes of the design to special or dedicated routing resources, utilizing special function macros optimized for the MPA architecture, and utilizing MPA functional resources that can be only be structurally instantiated from the MPA synthesis library. The following sections describe the special types of macros that can be found in the MPA synthesis libraries. These macros are for

specific design requirements. Please peruse the synthesis libraries for more macro specific details.

Clock Tree macros

The clock tree macro resource consists of the IPCLK clock pad macro and the ACLK internal clock buffer. The "ACLK" macro must be instantiated by the designer to connect internally generated clock signals to the primary clock tree. An example of a structural description applying an internal clock buffer signal to the primary clock tree.

```

(VHDL) Buffer_instance_name: ACLK port map
(A=>clk_input, Q=>clk_output);

```

```

(VERILOG) Buffer_instance_name ACLK
(.A(clk_input), .Q(clk_output));

```

The "clk_output" signal will be driving register clock pins from the primary clock tree.

There are only eight primary clock signals in a MPA1000 device. The total number of IPCLK and ACLK macros must not exceed eight. If it does, partitioning failure will occur.

Three-state/WOR macros

To use WOR structures in the MPA1000 devices, There are several macro resources. They are WBUF, WINV, WND2, WOR2 for an internal WOR bus and there are several IO macros that have the WOR capability that utilize the P-bus resource. An example would be the IPWINV or APWBUFF macros. For each WOR bus created, a pull up macro must be attached. The pull up resources are WPUP for the internal WOR bus, and PWPUP for the P-bus pull up. When creating an internal or P-bus WOR structure, both the WOR buffer and the pull up macros must be instantiated. Currently, there is not a method to inference an WOR structure when using Synopsys. The following is an VHDL example of an internal WOR instantiation :

```

U01: WBUF port map (A=>sig1, W=>worbus);
U02: WBUF port map (A=>sig2, W=>worbus);
U03: WBUF port map (A=>sig3, W=>worbus);
U04: WPUP port map (W=>worbus);

```

The "WPUP" macro is necessary to pull up the wor bus signal. The on-line documentation has more information on how many "WPUP" macros should be placed on the wired-or bus.

Three state functionality can be inferred in an HDL when it will be used in the I/O area of the FPGA. The "m" signal in this example needs to be an external I/O signal which is assigned "out port" " port is pad" attributes . A three-state inference:

```

if (sli = '1') then
m = e; else m = 'z';

```

IO macros

The MPA input output block (IOB) is a complex logic block with data registers on the input and output data signals. The registers cannot be inferred with RTL code, but can be accessed through a structural description. In general when the synthesis tool places pads onto the design's peripheral signals, the pad macros are simple pad buffers e.g. IPBUF and OPBUF. To utilize the registers in the IOB, an structural description must be used. There are more that seventy

4



different IOB macros. The on-line help describes the many different types.

Clock Signals/Clock Tree Generation

Without careful attention, clock tree implementation can cause problems for both synchronous and asynchronous designs. There are several clock tree related issues, the HDL designer should know :

- Design Compiler does have the capability to infer clock pads for those assigned clock signals. Internal clock buffers require a structural description. However, Design Compiler cannot insert internal clock buffers automatically. For instance, a gated clock signal which needs to drive the clock pins in I/O flip-flops will require an internal clock buffer to get onto the primary clock tree. This is an MPA1000 architecture limitation since the I/O register clock pins can only be accessed from the dedicated clock tree. By structurally inserting an internal clock buffer, the gated clock signal will connect to the dedicated primary clock tree through the clock buffer. Since the number of primary clocks (buffers and pads) of MPA1000 is eight, it is important to limit the total number of inferred clock pads/buffers and inserted clocks to eight.
- If a signal is assigned as an "input port" and as "clock", Synopsys will map the signal into a "IPCLK". Even if the designer doesn't assign "clock" for an "input port" signal, if the input signal is driving a clock input of a register directly, it will also be assigned as an "IPCLK" macro automatically. All the IPCLK pads will connect to the primary clock tree by default. The designer must make sure that "don't touch" was assigned to all those primary clock trees. If Synopsys "optimize" or "balance" the clock tree, it might generate many secondary clock trees which ends up causing clock tree routing problems.
- If an internally generated clock signals need to drive both primary clock tree and a regular output pad, a separate output macro "OPBUF" needs to be added to bring the signal to an output pad. The following is a VHDL example:

```
clkbuf1: ack port map (A=>internal_clock,
Q=>primary_clock);
```

```
outbuf1: opbuf port map (O=>internal_clock,
EXTOUT=>output_pad);
```

General HDL Techniques

The description of an design in an HDL is very important. The synthesis tool interprets the description and then maps the design to the target library. Performance of the design depends on the interpretation of the HDL description and the style of HDL coding. These influence the logic that will be mapped to the design.

Synopsys does not optimize logic based on XOR logic reduction (Mueller-Reed). However, since 50% of the MPA1000 cells can be implemented as an "XOR", the designers can build macros based on XOR logic. The outcome of the XOR based logic can potentially be faster and use fewer number of cells.

- Merge registers into the I/O macros. Registers that are directly connected to input macros or output macros with out feedback can be pulled into an I/O location by instantiating

MPA I/O macros such as "OPDFR" and "OPDLR" in an HDL design. If specialized I/O macros are not used, the register resource will come from inside the core array area.

- For multiple clocks within a design, it is recommended to implement a clock enable signal for the various registers and have the registers all clocked with the top level clock.
- For state machines, an important issue when compiling a state machine is that the one-hot state assignment approach is very viable for a fine-grained architecture such as the MPA1000. In general, Design Compiler supports five types of state assignment: manual, auto, one-hot, binary, and gray. The one-hot state assignment assigns one unique flip-flop per state. In coarse-grained architectures such as Xilinx 3000/4000, there are large combinational circuits attached before the flip-flops in each block (CLB). One-hot state assignment will cause significant penalties of areas for 3000/4000 architectures. On the other hand, the MPA1000 architecture has no combinational circuits attached to each flip-flop within a cell. The area penalty is, therefore, minimal. Since our both designs are utilizing less than 30% of the cells, In general, it is recommended to use the one-hot state assignment if timing is more critical than area. The "one-hot" mode in most cases achieves faster timing due to its much simplified combinational logic.

Synchronous Design Style

The synchronous design style is the preferred method of design recommended by synthesis tool and FPGA vendors. Few designers use the asynchronous design style since it can result in a little bit faster performance in an ASIC or custom design environment. In an FPGA environment, this assumption can be false. In general, asynchronous clock signals are not driven from primary clock resources and are therefore implemented in an "secondary clock tree" in MPA1000. The secondary clock trees use the regular routing resources to route the signal and have significantly longer clock delays and clock skews. The extra clock signals also occupy routing resources such as ports and global busses that result in increased routing congestion. All those combined, the asynchronous design typically will not get the faster-speed advantage the designer might expect.

The only asynchronous design styles suggested by Synopsys are designs with gated-clock and designs with asynchronous reset. Designers who use gated-clocks have to be aware of all the asynchronous clock tree problems that exist.

The asynchronous reset, however, is a popular design style and generally will not cause serious design problem since it is usually implemented as a global signal and tends to be mapped into primary clock/reset tree.

Area Estimation and Area Optimization

The designer should do some rough area estimations before trying to fit the design into the target MPA device. The rule of thumb is :

- The total number of cells (components) created by synthesis should not exceed 40% of the total raw cells of each device since some cells will be used as routing resources. The routability is usually design-specific. However, if there are more high-fanout nets, the routability usually decreases.

4



There is still some chance to route designs with more than 40% cell utilization. However, the performance usually is not as good and the probability of successful routing is much lower.

- The total number of flip-flops should not exceed 80% of raw flip-flops.
- The total number of clock/reset signals should not exceed 15.
- In general, device area is not a concern for FPGA unless it cannot be fitted into a device. There are some techniques to improve device areas:
- Instantiate special I/O macros to utilize registers inside I/O macros. There are two registers available inside each I/O macro. It will reduce the device area, reduce the required routing resource, and help in synchronizing on/off-chip timing.
- Use wired-or structure can reduce the number of gates. However, special attention should be given to timing and routability issues.
- Using the primary clock tree to implement high-fanout net will not only improve the wiring delays but also save a lot of routing resources. However, an active primary clock tree will consume more power than an idle clock tree. If power consumption is an important issue, designers should do some calculations as to the trade offs of using the clock tree.
- Use Synopsys techniques of resource sharing and area optimization options.

Techniques for Timing Optimization

In general, most of the timing improvement will come from the design and HDL code changes. By knowing the behavior of HDL compilers will generally lead to big improvements in device timing, the following describes some of the common timing optimization techniques used in HDL code structure and design synthesis:

- Avoid long "if_elsif" or "case" statements. Each elsif or "when" statement usually will add at least one logic level to the circuit. A better technique would be to see if the statements can be broken down and inserted into different concurrent processes or blocks. The same thing applies to the other statements inside each process. Since they are all implemented sequentially, try to see if they can be implemented as concurrent statements outside of the process.
- Be aware of the default inferred latches and registers. Make sure that the if/elsif statements always close with an "else" statement if no latch element is expected.
- High-fanout nets usually cause very long wiring delays. Try to reduce the number of fanouts if possible.
- Use clock file to differentiate slow clock and fast clock. It will help timing-driven layout to optimize and report the right critical paths.
- Use Synopsys "timing optimization" option.

Timing Analysis and Delay Estimation

Since FPGA place & route can take a long time for larger designs, it is important to do timing analysis and delay estimation during the synthesis stage to avoid many design

iterations. In general, accurate FPGA delay estimation is very difficult to achieve due to large metal wire delays and large switching element delays. However, by using component delays and wiring delays derived from statistical analysis, some obvious problems can be identified and avoided in the early design stages.

Critical path analysis is the most popular way to estimate device frequency. The post-layout critical-path delays are reported in the "timing file" (design.tim). The path delays in the timing file include both the component and wiring delays. Each level of logic in the critical path needs at least 1.2ns (0.7ns cell delay and 0.5ns direct interconnect delay). In general, for delay paths longer than 5 logic levels, the average delay for each level of logic is between 2.5ns to 3.5ns.

To estimate the critical path delay, the designer should count the levels of logic of the critical path by using the "highlight_path" command to highlight the critical path. Assuming the number of logic levels for the critical path is N, the theoretical lower bound of the critical path delay (LD) can be calculated with the following equation:

$$LD = 1.2 * N + \text{setup} + \text{clk_to_q}$$

For the MPA1000 family, the setup time is about 1ns while the clk_to_q is about 1.5ns.

In a real design, the average delay for each level of logic is around 3ns. Therefore, the expected delay (ED) for the critical path should be:

$$ED = 3 * N + \text{setup} + \text{clk_to_q}$$

General timing analysis guidelines for MPA1000 synthesis designers are:

- If timing budget is smaller than or only slightly more than LD, there is no chance to reach the timing goal. The designer should try to modify the design by modifying the HDL code structure, or modifying the timing budget to solve the problem.
- If the timing budget is close to ED, the place&route tool has a good chance to reach the timing goal. The designer should utilize the timing optimization techniques in the Synopsys environment and use timing driven layout options for the place and route software. For a multiple clock system, the slower clock usually will not create a critical path. However, the software will not be able to differentiate the less critical clock signal unless the designer creates a timing group for each clock in the "clock file". Please refer to the online documentation for detailed information regarding the "clock file" format.
- If the timing budget is far exceeding ED, the designer will have a large slack time for the critical path. Even though timing-driven layout is still suggested for place&route, the designer may have some flexibility in using a less aggressive timing goal to reduce run time and focus more on the area optimization when synthesizing the design.

Behavioral vs. Structure Synthesis

In an traditional synthesis flow, structure design is generally recommended at the top-level for constructing logic hierarchies. Behavioral design, however, is usually much easier and maintainable for designers. For example, in an FPGA design, critical portions of the design utilized the structural description method to capture specific logic

4



modules. One area that used an structural description was the I/O logic module. Although some FPGA technology mappers have the capability to do special mapping processes for the I/O, such as mapping flip-flops into an I/O, it is generally much more controllable to construct an I/O logic module with an structural description. It is also easier to insert attributes to the macros which are described structurally. The current MPA1000 synthesis library includes a large variety of I/O macros which have many combinations of input/output, clock, flip-flop/latch, and delay elements. By instantiating I/O macros directly from the MPA1000 library, the designer can avoid some potential problems caused by the synthesis logic

inference process.

An interesting note when using an structural description is that Design Compiler is still capable of optimizing the structural description even it is supposed to be optimized by the designer already. It can be concluded that the software is usually capable of performing a much more thorough logic optimization search compared to human beings under a well-defined environment. In one example, an 20% improvement was achieved by running optimization on hand crafted structural modules.

4

Interfacing to the PowerPC™ with a Motorola Programmable Array

Prepared by
Rich Rejmaniak
Field Applications Engineer

4

PowerPC is a trademark of International Business Machines Corporation.



Interfacing to the PowerPC™ with a Motorola Programmable Array

Introduction

With the higher speeds and wider busses for modern RISC processors, designing bus peripherals can be a daunting task. The need to decode an ever expanding address range in an ever shrinking period of time can overwhelm many logic designers. In addition, the new generations of processors have far more complex bus protocols aimed at increasing overall bus throughput. The pressures on a particular design to meet performance specifications within economic boundaries are quickly outpacing standard design solutions. In the past, PALs or other small programmable logic devices were used to perform this function. Current processor bus complexity is now outdistancing the pinouts of these devices, let alone their logic functionality. The solution to these problems in high volume applications is through the use of ASICs. However, ASICs are not an economical solution for the small and medium production runs (<5000 units) which characterize the majority of processor designs.

The solution to this problem is the FPGA. It sits between the PAL and the ASIC in both cost and complexity at this production scale. This article demonstrates the use of a Motorola Programmable Array as a bus peripheral to the PowerPC family of RISC microprocessors. The design contained herein is a bus cycle analyzer capable of triggering on a bus event and capturing a snapshot of 5 bus cycles. This device operates in two modes: One is a voyeuristic mode where the MPA is silently monitoring and recording bus cycles, waiting for the trigger event. The other is as an addressable bus device to allow the host processor to program the trigger event and retrieve the captured bus cycles.

To accomplish this, the design must handle all aspects of the PowerPC bus protocol. The design is exhaustive in this sense, ignoring only two aspects of the bus: It does not monitor bus arbitration, nor does it support enveloped writes. It does not monitor bus arbitration because it never assumes address bus mastership. Enveloped write support was not implemented for complexity reasons, thus limiting this device in designs that have multiple caching bus masters.

The PowerPC Bus

To fully understand the design of a peripheral to a particular bus, it is first necessary to understand the bus itself. The PowerPC family of processors has a bus that is quite different from that of typical CISC processors. Most CISC processors have a single processor bus composed of address and data lines under the auspices of a single set of control signals. The PowerPC actually has two busses. This should not be confused with a Harvard Architecture, which has two data pathways. The PowerPC follows the standard single pathway design for instructions and data. What the PowerPC does that is different than older processors is to provide for independent mastership of the address and data buses. Each of these buses has its own control lines and can operate with some degree of independence. The processors that conform to the bus model being studied are the MPC601, MPC603 (In 64 bit data bus mode), and the MPC602. The MPC620, MPC602,

MPC5xx, and the MPC8xx processors have differing bus structures that are not applicable to this design.

All control and timing signals on the PowerPC bus are active low and are designated in this text with a preceding exclamation point. i.e. Transfer Start is designated as !TS. The only exception to this is obviously the processor system clock, SYSCLK, which is symmetrical in time.

All transaction control signals on the PowerPC bus are synchronous to the SYSCLK signal. All asserted signals are driven by the rising edge of the clock and their timing specifications are defined by this edge. Just as well, all sampled input signals are defined with their setup and hold times relative to the rising edge of SYSCLK. While the PowerPC bus allows most synchronous bus signals to occur at any number of clock edges in order to extend the length of a memory cycle, all signals must be stable in their active or inactive state during the rising edge of SYSCLK.

The reader should be cautioned that this application note doesn't contain the full specification for the PowerPC bus. Described herein are only the bus signals needed to interface to this bus analyzer circuit. For the reader to properly design a PowerPC based processor system, the detailed electrical and architectural specifications must be referenced. These documents are available through the local Motorola Semiconductor sales offices.

Bus Arbitration

The arbitration of the PowerPC bus is done without the knowledge of the MPA devices used in this design. The only time a bus device needs to participate in the arbitration cycle, is when it intends to assume mastership of the address bus to initiate host (i.e. processor) to slave (i.e. memory) transfers. This device never initiates such transfers but simply monitors and responds to these transfers initiated by other bus masters.

Address Bus Tenure

There are two major classes of address bus cycles. Data bus preambles imply that a data bus cycle will follow the address bus cycle. Address only transactions are for cache synchronizing purposes. Enveloped writes are considered to be data bus preambles by the processor. This design treats them as address only transactions, limiting the ability to completely monitor them.

An address bus tenure starts with the assertion of Transfer Start (ITS) and Address Bus Busy (!ABB) by the current bus master. These signals are asserted synchronously with the processor SYSCLK signal. All address and transfer control lines (TT[0:4], TBST, TSI[0:2], TC[0:2], and WT) become valid within ten nanoseconds (typical: plus or minus) of the control lines.

The ITS signal is only asserted for one clock cycle, at the start of the address bus tenure. The !ABB signal remains asserted for the duration of the address tenure, signaling other masters in the arbitration phase that they are denied the bus, as well as for use by the memory subsystem to determine the stability of the address and transfer control signals.

The final step in a successful address cycle is the assertion of Address Acknowledge (!AACK) by the address bus slave.

4



This slave is usually the memory or I/O subsystem. When the bus master samples !AACK asserted on the rising edge of SYSCLK, it immediately negates !ABB and places all address and transfer control signals into the high impedance state. If the bus slave doesn't assert Address Retry (!ARTRY) on the next clock cycle, the address tenure is considered to be retired successfully. This bus analyzer circuit does not monitor the !ARTRY signal. If a bus slave asserts !ARTRY, the address tenure will be re-run, and will be captured again.

Data Bus Preambles

Data bus preambles are the start of a 'typical' processor bus cycle. It is the address phase of a data transfer action between a bus master and a bus slave. If the processor is initiating a single beat transfer, there will be a separate address bus cycle for each data bus cycle. In practice, most bus transactions on the PowerPC (indeed on most RISC processors) are burst transactions. Burst transactions involve the instigation of a number of data bus cycles (four on the PowerPC) from a single address bus cycle.

Another type of data bus preamble is the enveloped write. The PowerPC uses this transaction to maintain horizontal cache synchronization among caching bus masters.

Address Only Transactions

Address only transactions are used by the PowerPC to maintain horizontal cache synchronization among multiple caching bus masters. An example of such a transaction is when a particular bus master updates an internal cache line that is flagged as write back and is in the 'shared' state. In this instance, the processor that is altering its' internal cache line must broadcast this action to any other processor that has the same data in its' cache. Such a notification is done with an address only transaction. This address only transaction can occur while the data bus is busy completing the data phase of a data bus preamble address cycle. This overlap can only occur when the system architecture supports a pipelined bus structure.

Both data bus preambles and address only transactions differ only in the state of the Transfer Type (TT[0:4]) signals asserted during the cycles. The timing remains the same for both types of cycles.

Data Bus Tenure

The data bus is capable of operating under different bus mastership from the address bus. While systems that optimize cost over performance can avoid this feature, high performance systems will generally support this via SDRAMs. The processors supported by this design allow only one level of pipelining; that is, only one address bus cycle can be outstanding (waiting for corresponding data bus cycle) at a time.

The data bus tenure timing is very similar to the address bus tenure timing with a few exceptions. The data bus has no Transfer Start signal as does the address bus. Instead, there is an infer transfer start during the first cycle of Data Bus Busy (!DBB). !DBB is preceded by a Data Bus Grant (!DBG) to the bus master from the arbitration logic. This device is not connected to the !DBG signal. Data Bus Busy is asserted by the bus master for one or more integral SYSCLK cycles until

Transfer Acknowledge (!TA) is asserted by the bus slave. During a store operation (a.k.a. a bus write cycle), the bus master will drive the data lines for the duration of the assertion of the !DBB signal. During a load operation (a.k.a. a bus read cycle), the bus master assumes that the bus slave will have the data lines driven and stable for the rising edge of SYSCLK coincident with the assertion of !TA. The data bus also has a Data Retry (!DRTRY) signal that operates in the same manner as the !ARTRY signal on the address bus.

Single Beat Transactions

A single beat transaction occurs when there is only one data bus cycle for a particular address bus cycle. This is caused by the processor accessing a non-cacheable memory address. Single beat transactions can be from one to eight bytes in width. The width of the access is determined by the processor. The memory is assumed to be 64 bits wide and is expected to support byte lane write selection. Depending on the bus timing imposed by the system design engineer, the address may be present on the address bus for none, some, or all of the data bus cycle.

The duration of a single beat transfer data bus tenure can be as short as one SYSCLK cycle. In this case, the bus master asserts !DBB (and drives the data bus on a write cycle) following the rising edge of SYSCLK that sampled the !DBG signal. The bus slave responds within the constraints of one SYSCLK cycle and has either latched or driven the data bus and drives !TA by the next rising clock edge.

Burst Transactions

A burst transaction is the result of a cache line push or fill. This type of access comprises the vast majority of bus cycles on a high performance design. The external memory system does not have the option of declining a burst transaction. The only way to protect a memory device from a burst notification from the processor is to designate the area as non-cacheable.

A burst transaction can be pipelined or non-pipelined. Non-pipelined bursts provide an address bus cycle for each of the four data bus cycles. A pipelined burst will only post the address of the initial data bus cycle. The memory device is expected to latch and auto-increment the address provided by the processor. The address is modulo four (on a double word boundary) and can specify any double word in the cache line as the first transaction.

In high performance systems, the pipelined burst transaction will be the dominant type of bus cycle. The timing of a burst cycle is that of four back-to-back single beat transactions. In the case of a burst, !DBB is asserted unbroken for the entire duration of the bus tenure. The bus master will drive or latch the data bus for the next data cycle as determined by the modulation of the !TA signal from the bus slave. The bus slave can cause the bus master to extend any of the four data transfer sub-cycles an arbitrary number of SYSCLK cycles. The Bus master will hold the data bus in the current transfer sub-cycle until it receives a !TA signal from the bus slave. At that time the master will proceed to the next sub-cycle or terminate bus ownership after the fourth transfer.

Endianess

It must be remembered that the PowerPC is a big endian machine, both in byte ordering and in bit ordering. On all

4



busses and in all registers' bit [0] is the most significant bit. This design follows this convention with one noticeable exception: the numbering of the register selection lines in the address decode logic. The address lines follow the big endian bit format, however, the write and read enable lines use the little endian (LSB = 0) format. This was done to maintain the convention as marked on the multiplexers and demultiplexers in the component library. When data is routed to and from the internal registers, big endian alignment is maintained.

The user must wire the device to the processor bus using big endian bit ordering. While, in theory, reversing the bit ordering would carry through the entire pattern matching scheme, it would convolute the address decoding and the mapping of the internal setup and history registers.

The Design

This design demonstrates solutions to a number of PowerPC processor design criteria using the Motorola programmable array. This particular implementation is targeted to an MPA1064. An MPA1064 contains 6400 logic cells, providing 1600 internal flip-flops in addition to the flip-flops in each of the I/O cells. The MPA1064 contains 160 I/O cells, each containing one flip-flop for input and one flip-flop for output. This design fits into the MPA1064 placed in a 160 pin QFP, which only bonds out 120 of the 160 I/O pads. The remaining 80 flip-flops in the unbonded 40 I/O pads are available for internal logic realization. This size MPA is necessary because of two aspects of the processor bus: signal count and bus speed.

Bus Speed

The bus speed supported on this design is 50MHz. When the MPA is acting as a bus slave, it responds with wait states to slow the bus down. This is because the only time that the device is addressed is for setup and result retrieval. Both of which are not time critical operations.

When the MPA is monitoring the bus in voyeuristic mode, it must keep up with the bus at full speed. At 50 MHz, there can be at most three levels of logic between clock edges to maintain a reasonable design margin. This results in a heavily pipelined design.

Pipelining

Pipelining impacts this design in two ways: There is an element of latency in the reaction of this circuit to the triggering events. There is also a large consumption of register logic in the target device.

As the monitored bus data flows through this device, it is processed in stages of a nine level pipeline. The first four stages process and synchronize the incoming data, compensating for bursting, split bus tenures, and address only transfers. The last five stages contain the triggering event, the two bus cycles prior to the trigger, and the two bus cycles after.

Latency

The latency inherent in this device means that it is incapable of halting the processor on a trigger event as is common with a breakpoint feature of a development system. When a trigger event occurs, the processor can be interrupted to service the event, however, several bus cycles will have

transpired between the trigger and the interrupt. It is for this reason that the circuit maintains a record of the two bus cycles preceding and following the trigger event.

Register Consumption

The captured data stream is 109 bits wide, multiplied by nine pipeline stages results in 981 register elements for pipelining alone. This does not include the various elements used for setup registers (218 bits) and state control. The MPA architecture is naturally rich in registers, making it extremely useful for pipelined designs.

Bus Interface

There are two basic types of signals on the PowerPC bus that connect to this design. The first group of signals are captured by the design when it is in voyeuristic mode. The second group are signals that interact with the device to provide timing and control. When the device is being addressed as a bus peripheral, some the signals that are part of the capture group become interactive with the device.

Signals Captured

The captured signals are the address bits, data bits, transfer type, transfer size, transfer control, write, and burst. Any of the captured signals can be used to construct the trigger event. All captured bits are present in the trace memory regardless as to their role in the trigger event.

All of the signals mentioned in the preceding paragraph, with the exception of the data bits, are captured on the same clock edge. The data bits corresponding to the other signals may be captured at some indeterminate time after the address cycle occurs. This design will synchronize the signals, aligning the data bus information with the proper address and control information. It should be noted, however, that if address only transactions are permitted, they will be synchronized with an arbitrary data bus capture result. In many cases, this captured data bus information will be data from a burst transaction. If the trigger event is set to detect an address only transaction, the data portion of the trigger should be masked off.

Another caveat of allowing address only transactions, is that the address may be captured during a burst, thereby preventing the compare logic from seeing that particular burst address. In these cases the burst address will be internally generated, but will be obliterated by the captured address bus information. If the obliterated address bus cycle matches the trigger event, it will not be detected.

As a result, this circuit should be programmed to trigger on address only transactions and ignore the data bus, or it should be programmed to disallow the capture of address only transactions.

Signals Interacted

Interacting signals are signals that provide timing and control to the design to allow it to capture the needed signals. The signals in this group are: Transfer start, Transfer acknowledge, Data bus busy, Address acknowledge, and Bus clock.

These signals, with the exception of Bus clock, are generated by the bus master or the bus slave. Some of these signals are generated by this design when it is responding to a load or store from the processor.

4



Transfer start (!TS) marks the beginning of an address tenure and is used by the analyzer to capture the address and control signals. After the address signals are captured, they start to pass through the monitoring pipeline while being simultaneously applied to the address decode logic. If a bus slave responds to the address tenure with an Address acknowledge (!AACK) signal prior to complete decoding, this design assumes that it is not the target of the address and stops the address decoding process. If a successful decode occurs, the analyzer will generate an !AACK signal and standby to respond to the corresponding data tenure. The analyzer never drives Transfer start.

Data bus busy (!DBB) is generated by the data bus master to indicate the start and duration of a data bus tenure. It is used by this analyzer to communicate with the processor when the MPA is addressed as a bus peripheral. Transfer acknowledge (!TA) is used by the analyzer to latch the data from a bus transaction being monitored and is driven by the analyzer when it is acting as a bus slave.

Address bus busy (!ABB), Bus request (!BR), Bus grant (!BG), Data retry (!DRTRY), and Address retry (!ARTRY) are not connected to this analyzer.

The Implementation

The elements used to construct the final circuit are brought together in the MPA. Each of the aforementioned criteria are met using the basic gates elements of this device. Again, the high register density and fine grained flexibility of the MPA device made it an ideal choice for this design.

The design was done using ViewLogic schematic capture. It is a hierarchical design with each section of the circuit represented as components in the next higher level of the schematic.

Compound Components

The components constructed from the basic gate elements are the same type of functions found in most logic libraries. The main difference is that they are pipelined and therefore synchronous.

This analyzer can be modified to operate at a higher bus speed, using more register elements, or at a lower speed, thus saving register elements by editing the pipeline depth of the lower level components. This will not change the overall structure of the design.

Data Pipes

A couple of new elements are synchronous pipeline chains. These have been constructed in both 45 bit and 64 bit widths. The pipeline chains are basically synchronous FIFO stages. Each will clock in data when an enable is present and provide the data and the enable bit at the output at the completion of the unclocking operation. While the data will remain at the output until additional data is clocked in, the enable bit will last only one clock cycle at the output of this logic function.

These pipelines are used to carry the data to be captured and compared while control signals are processed through their respective pipelines. They are used again to store the captured data during the comparison process.

Pipelined Comparator

The 109 bit width of the trigger event meant that the comparator had to be pipelined. The first stage masks off the bits that are not part of the trigger and does a 109 by 109 bit exclusive OR. The remaining stages reduce the 109 by 109 bit result by successive ORing down to a single bit.

Each stage of the comparator uses three levels of logic between registers. The first stage that compares the 109 bit by 109 bit values implements an XOR and an OR, considering the XOR to introduce two levels of logic delay.

Monitoring Sections

The most complex sections of this design are those that deal with the monitoring of the bus activity for the trigger event. The most pressing issue is that of synchronizing the split pipelined address and data transactions. The monitoring circuit must be able to retain and optionally increment the address that corresponds to the current data transaction while it latches and starts processing the current address bus cycles for later data bus activity. In addition, there may be an address only transaction on the address bus while current data bus activity is relying on a previous address bus cycle.

Transfer Type Decoding

This circuit decodes the transfer type to determine if the current address cycle is a single beat, burst, or address only transaction. The complexity of the decoding has this circuit heavily pipelined. In most designs, address only transactions can be decoded with only one or two bits, but since this is a potential debug circuit, complete decoding is implemented.

Burst Transaction Address Generation

Burst transactions are always aligned and transfer 64 bits per data cycle. After the address is latched, decoded, and synchronized with the data bus cycle, it is incremented for future data bus cycles. The address counter uses modulo counting to maintain compatibility with the critical word first nature of cache line fills.

Address Only Transaction Override

Address only transactions produce a unique situation for the trigger event. In a pipelined bus system, a processor can issue and retire an address only transaction while the data bus is active using the contents of the previous address bus cycle. In fact, address only transactions can occur in situations where the data bus is active for all of the time surrounding the address only cycle. When the data bus is active, there is a corresponding address bus cycle inside of the MPA that is either captured off of the address bus, or is generated by the burst address generation logic. If the MPA device is generating the burst address internally while there is an external address only transaction, the MPA device is faced with deciding which address to use in its' comparison logic. For this reason, there is a bit in the control register that resolves this issue. When this bit is set, the processor will process the address only transaction. If this transaction happens to align itself with a data bus cycle, the address only transaction will be paired with the randomly occurring data cycle when presented to the trigger event comparator. In this case, when triggering on an address only transaction, the contents of the data bus should be masked out of the trigger event. When the address only

4



transaction override bit is clear, address only transactions will be blocked from reaching the event trigger comparator stages.

Bus Peripheral Sections

In order for the processor to set up the trigger event and to retrieve the results of a trigger, the MPA device must behave as a slave memory device on the processor bus. During this time the setup and history registers are memory mapped into the processors address space. This design provides for two methods of placing these registers into the appropriate address locations. In the chip select mode, a control line decodes the processor address externally to the MPA and provides an active low chip select to indicate that the device is being addressed by the processor. In the internal decode mode, the MPA decodes the address bus itself to determine if it's being addressed. In this case the chip select line is used to select one of two address locations for the MPA. In both modes, individual register addresses are decoded internally.

Processor Store Cycles

A store cycle from the processor to this design must be a 32 bit word aligned on a 64 bit boundary. All internal setup registers are 32 bits wide, except for the control register.

4

Processor Load Cycles

A load cycle from the processor to this design must be a 32 bit word aligned on a 64 bit boundary. All internal history registers are 32 bits wide, however, not all bits read will contain useful information.

Schematics

The schematics are drawn in a hierarchical topography. Figure 7 shows the structure of the schematics as they are assembled in the design. Pages that are shaded appear more than once in the design, due to the fact that a hierarchical design allows the use of a schematic page as design component.

ANALYZER

Figure 8 is the top page of the bus analyzer design. It provides the high level interconnect between the main operational sections of the design.

I/O_PINS

Figure 9 contains all of the external interface pins for the analyzer. They are grouped into three sections: Data bus interface, Address and transfer control, and timing and operational control.

The top left corner of the page shows the data bus interface. The upper bits [0:31] are input only and the lower bits [32:63] are input and output pins. (Remember: the PowerPC uses big endian bit ordering with bit[0] as the MSB.) The reason for the difference in input and output types is that the device captures the full 64 bit data bus in voyeuristic mode, but responds as a 32 bit peripheral in bus slave mode.

The upper center and right areas of the schematic page are the timing and operational pins. These include the signals that regulate the bus activity as well as the MODE, SELECT, !START, !STOP, and !RESET operational pins.

The bottom left quarter of the page contains the interface to the address lines and the transfer type signal lines. These lines control the non-timing aspects of the transaction, such as data bus direction (WT), transaction size, and transaction type.

The bottom of the page, just to the left of the title block contains the designation for the processor bus clock input pin.

INVBI PAD

Figure 10 is a bi-directional inverting I/O pin. This component was created by modifying the noninverting bi-directional pad for use with active low control signals.

INVINPAD

Figure 11 is an inverting input pin. This component was created by modifying the noninverting input pad for use with active low control signals.

PIN32BI

Figure 12 shows the creation of a 32 bit wide array of I/O pads used to interface to the low 32 bits of the data bus. Each pad is drawn independently, as opposed to an array of components, to allow individual pin assignments if it became necessary.

PIN32IN

Shown in Figure 13, a 32 bit input only array of pins. Again, called out separately to allow individual pin assignment. They are used to monitor the address and upper data bus bits.

IN_PROC

The schematic in Figure 14 is the input processor of the bus monitoring section. The function of this section is to process and synchronize the address and data bus tenures. The resulting output is a unified 109 bit wide snapshot of an aggregate bus cycle.

The incoming address cycle is stored in three 45 bit wide synchronous pipeline stages while the transfer type is decoded. Selected bits of the final pipeline stage can be taken from the burst counter to provide auto incrementing addresses for burst cycles.

The signals captured from the data bus are stored in a three deep pipeline as well, this one being 64 bits wide. The data from the data bus isn't processed during this time, but is pipelined to provide synchronization with the corresponding address signals.

PIPE45

The 45 bit wide pipeline stage shown in Figure 15 is comprised of a 45 bit wide data register and a single flag bit. The flag bit is present for one clock cycle following the latching of data into the register.

REG45

Figure 16 is a 45 bit wide register. It is composed of 8 bit wide standard library components as well as a 32 bit wide composite component. Logic trimming in the place and route software removes the unused logic from the second 8 bit wide register from the final circuit.

This composite component is used as the data register in the 45 bit pipeline as well as a section of the 109 bit wide pipe.



REG32

The 32 bit wide register in Figure 24 is used wherever a register of this width is needed in the design

PIPE64

Figure 17 is a 64 bit version of Pipe45. It is used to provide a delay to the data bus signals for synchronization to the address bus signals.

REG64

Figure 18 is a 64 bit register. It is used to construct the 64 bit wide pipe component and it is used in conjunction with the 45 bit register to create the 109 bit registers.

TTDEC1

Figure 19 is the first stage of the transfer type decoding circuitry. The balance of the decoding is performed discretely at one level higher in the hierarchy, on the IN_PROC schematic.

The only two cases that are decoded from the transfer type are burst cycles and address only cycles. The user is free to program the pattern detection circuitry to trap on other types of transfers. This detection circuitry is independent of the trap pattern and is used to control burst address generation and blocking or passing of address only signals.

BURSTCTR

Figure 20 is the burst counter. It operates in parallel with the third address pipeline stage. Its purpose is to latch the address bus signals, just as the third pipeline stage does. However, it retains this information and increments the address bits that are implied by the burst cycle. It is a modulo counter, as required by the PowerPC specification to allow critical word first loading of cache lines.

CNTCTRL

Figure 21 is the state machine to control the sequencing of the burst counter. It is a three bit unary counter whose output is true if any of the count bits are nonzero.

SELECTOR

The selector circuit shown in Figure 22 is a 45 bit by two way gate. Its purpose is to determine if data from the third stage of the address processing pipeline or data from the burst counter should enter the pattern matching section of the design.

REG109

The 109 bit register in Figure 23 is used throughout the design to carry the full trigger pattern through its processing stages.

TRC_CTRL

Figure 25 is the state machine that controls the operation of the pattern matching section. It accepts input signals to start, stop, reset, and to detect a trigger match. It outputs an all clear signal and a run signal.

The reset signal is from an external pin to the MPA. It forces a clear signal and places the pattern matching section in stop mode.

The start signal generates a pulse on the clear signal and asserts the run signal. The clear pulse is one clock cycle in length and is separated from the run signal by three clock cycles. A start can be generated by an external pin or a bit in the control register. The start signal is edge sensitive.

The stop signal halts the pattern matching logic and suppresses any applied start signal. The stop signal must be removed before the start signal will be accepted. Like the start signal, the stop signal can be applied from the control register or an external pin. The stop signal is edge sensitive.

A match signal halts the pattern matching section, but doesn't affect the reception of a start signal as does the stop signal. The match signal is generated by the trigger circuitry.

The clear signal is distributed to all registers in the processing pipeline. It doesn't affect the contents of the setup registers.

Run is an active low signal that allows the pattern matching section to clock in unified data from the input processing section.

REG_FILE

The schematic shown in Figure 26 is the setup register file. It consists of the two 109 bit registers for the match pattern and the pattern mask. It also contains the three bit setup register which contains the start, stop, and address only control bits. All of these registers are write only.

SETUPREG

Figure 27 is a 109 bit register used for storing the trigger pattern and the trigger mask.

REG13

The 13 bit register in Figure 28 is used to store bits 96 through 108 in the 109 bit setup registers.

REG_SEL

Figure 29 is the schematic of the register selection logic. This allow the contents of five 109 bit history registers to drive the data bus in 32 bit blocks.

Selection bits [0, 1] break the 109 bit registers into four 32 bit sections, the last section containing 13 bits of valid data. Selection bits [2, 4] are then used to determine which register is chosen. These bits can select up to eight registers, therefore not all values are meaningful.

Register selection bits are generated by the address decoding logic.

109SEL32

Shown in Figure 30 is the logic that delivers the chosen 32 bit subset of the 109 bit history register. When the most significant bits are selected, the unused bits are driven to zero.

32SEL8

Figure 31 is used by the logic in Figure 30 to break down the history registers.

MUX_BY32

Figure 32 is used to select one 32 bit history register slice from the array of five presented to it.

MUX_BY8

The logic in Figure 33 is used by Figure 32 to break down the history registers.

4



MUX5TO1

The five input muxes in Figure 34 are derived from the library of parameterized modules entry for an eight to one multiplexer.

BUS_SLV

Figure 35 is the control logic used to place the setup and history register into the processor's address space.

The latches at the bottom of the page, the shift register, and associated logic comprise a state machine that is continually monitoring the processor buses. If no other bus slave responds to an access before the address decode circuitry in Figure 36 detects a hit, then either the write control in Figure 38 or the read control in Figure 39 will transfer the data to or from the proper registers and generate the proper control signals to complete the bus cycle.

This process can take up to 14 clock cycles to complete. It was not considered critical to have this device respond quickly to bus accesses.

ASYNDEC

This schematic in Figure 36 is used to determine if the MPA is being addressed by the processor, and which internal register is being selected.

Address decoding can operate in one of two modes, internal or external. Each mode has a different function assigned to the select pin.

Internal decoding (mode = 1) uses the logic on this page to decipher the address generated by the processor. The MPA will reside at a base address of FFFF 0000 or FF00 0000 as determined by the select signal. The MPA reserves 256 addresses off of the base address for its registers. Thus the actual address that cause an access are FFFF 00XX or FF00 00XX. Note that there are not 256 registers in this device, so not all of these addresses are legal.

If external addressing is used (mode = 0), then the select pin is used to indicate that the MPA has been addressed. In both cases, the read addresses of the history registers and the write address of the setup registers maintain the same offset from the resulting MPA base address.

WHICHREG

Figure 37 is the address decoding section that generates the individual register selection lines.

There are nine lines to select the setup registers. Four for each of the mask and pattern registers, and one for the control register.

There is a selection signal for reading each of the history registers. The particular 32 bit word from each register is determined in the schematic hierarchy starting at Figure 29.

WRITEREG

Figure 38 is the control for the writing of the setup registers. After a launch signal is received from the master decode logic, the system waits for a data bus busy to indicate that data is present on the data bus. The logic then waits for six clocks to

latch the data and generate the transfer acknowledge. It then generates an idle signal to reset the main state machine.

READREG

Figure 39 is the read register control. Upon receiving a launch signal, it drives the data bus and then drives the data bus busy. It holds these signals until the transfer acknowledge is received.

MATCH

Figure 40 is the pattern matching logic. It is heavily pipelined to compensate for the width and speed of the processor bus. This circuit actually maintains two pipelines. The first is the pattern matching pipe, and the second is a parallel path for the captured data to be recalled later.

The first pipeline accepts the unified 109 bit wide bus data. This data is AND'ed with the contents of the mask register. The result of this AND is then XOR'ed with the contents of the pattern register. The result is a 109 bit wide comparison of the data bus and the trigger pattern for the selected bits. It should be noted that the contents of the reference pattern is not AND'ed with the mask register. It is up to the user to make sure that there are no bit positions with a zero in the mask register and a corresponding one in the pattern reference register. Such a combination would preclude a trigger event from occurring.

The resulting 109 bit wide result is successively ORED down to a single bit, which is the unary addition of all of the bits in the result of the XOR. This takes three clocks to work its way through the pipeline. The final result is an active high match signal.

The second pipeline is actually the history buffer register file. It stores the result of the match at T_ZERO[0:108] with two samples before and after the trigger. When a match occurs, the logic in Figure 25 suppresses the enable signal, freezing the contents of the pipelines.

REG_2

Figure 41 is a two bit wide register used in the pattern comparison pipeline.

REG_14

Figure 42 is a fourteen bit wide register used in the pattern comparison pipeline.

MASK

Figure 43 is a 109 bit wide array of AND gates. It is used as a single component in Figure 40 to allow the contents of the mask register to suppress bits captured from the processor bus.

XOR

Figure 44 is a 109 bit wide array of XOR gates. It is used as a single component in Figure 40 to compare the contents of the captured data with the contents of the pattern reference register.

OR109_14

Figure 45 OR's 109 bits down to 14 bits. It is the second of three stages in the pattern matching pipeline shown in Figure 40.

4



OR14_2

Figure 46 OR's 14 bits down to 2 bits. It is the third and final stage in the pattern matching pipeline in Figure 40.

Usage

Control of the bus analyzer consists of programming the mask and compare registers for the trigger event, followed by the control register to set the environment for the trigger. The monitoring of bus activity can be started or stopped either by programming the control register or by activating external control lines to the MPA.

This analyzer will attempt to monitor and trigger on accesses to itself if it is in capture mode while it is being accessed. This could result in unstable driving of the data bus during a load operation or a false trigger during a store operation. The control register can safely be written to while the device is capturing in order to stop operation. The analyzer will attempt to capture this bus cycle, however, it will shut down before the aforementioned cycle reaches the history buffers.

Figure 5 shows the addresses of the internal setup and history registers. The setup registers share the same addresses as the history registers with a read or a write used to differentiate accesses.

Compare Registers

The total trigger compare width is 109 bits. The compare value is stored in four 32 bit registers. The first three registers use all 32 bits, the fourth uses 13 bits. The remainder of the bits in the fourth register are ignored. It is possible to program a trigger pattern that cannot occur. An example of such a pattern would be a burst access from an odd address. It is the responsibility of the user to avoid programming to trigger on such events.

The compare register resides at locations 0x10, 0x14, 0x18, and 0x1C off of the base address of the MPA in the processors memory map.

Mask Registers

The mask registers allow the user to exclude specific bits of the captured bus signals from the comparison process. Each bit in the compare registers has a corresponding bit in the mask registers. Any bit masked off will not be considered in the comparison, but will still be captured and recorded. If all bits in the mask registers are set to zero, all bits in the compare register will be ignored and a match will occur as soon as the capture process is enabled.

The mask register resides at locations 0x0, 0x4, 0x8, and 0xC off of the base address of the MPA in the processors memory map.

Control Register

The control register consists of three bits. Bit Zero starts the bus monitoring process, and is equivalent to applying the !START signal to the MPA. Bit One stops the bus monitoring process, and is equivalent to applying the !STOP signal to the MPA. Bit Two is the address only override selection bit. When this bit is set to a one, address only cycles are processed by the bus analyzer. When it is set to a zero, address only cycles are ignored.

The control register resides at location 0x20 off of the base address of the MPA in the processors memory map.

Input Control Signals

There are three input control signals: Start, Stop, and Reset. Start and stop are falling edge triggered inputs. The start signal causes the MPA to clear out its history buffers and start capturing and comparing the processor bus cycles. The stop signal has the effect of a trigger event becoming true. All capturing stops and the contents of the history buffers are frozen. Reset, which is active low level controlled, causes a stop command to be executed as well as clearing all history buffers and setup registers. It must remain low for one full bus clock cycle.

Output Result Signals

Two output signals indicate the state of the capture control logic: Run and match. Run indicates that the circuit is currently capturing the processor bus. It is active low, a high indicates that the circuit is stopped. The second signal is the match signal. This signal is also active low and indicates that the contents of the trigger history register matches the trigger word. When a trigger occurs, the run signal becomes high (inactive) and the match signal goes low (active). The match signal will remain low until a reset or a start command is executed.

4

Trigger History Buffer

There is a five entry trigger history buffer. Each entry is 109 bits wide. The entries are named Tminus2 (Tm2), Tminus1 (Tm1), TZERO (TZ), Tplus1 (Tp1), and Tplus2 (Tp2). Entry TZERO contains the trigger word. Tminus1 contains the bus cycle immediately preceding the trigger event and Tminus2 contains the cycle preceding Tminus1. Tplus1 is the cycle following the trigger event, and Tplus2 follows Tplus1. Thus, the history buffer contains the bus cycle that caused the trigger event and the two cycles preceding and the two cycles following the event.

Programming bit fields

Table 1 through Table 5 shows meanings of the various transfer control bit fields. Figure 6 provides the locations of the bits as they are assigned in the trigger word. These bit positions hold true for both setup registers and the history buffers.

Summary

This application note attacks a number of PowerPC design issues. The most significant being the speed and bus width criteria.

This design shows that there is very little opportunity to implement a great deal of external logic on a modern RISC processor bus. While at the same time, a wide bus can require a great deal of decoding.

It is hoped that it demonstrates that, while the design of a RISC based system is not trivial, state of the art support devices such as Motorola's Programmable Array can resolve all design issues.



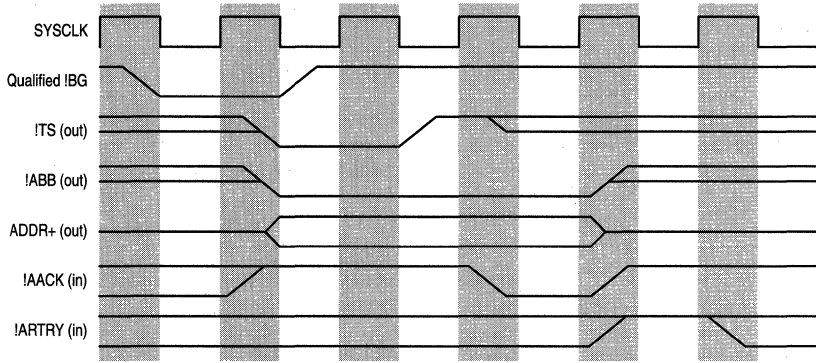


Figure 1. Address Bus Tenure

4

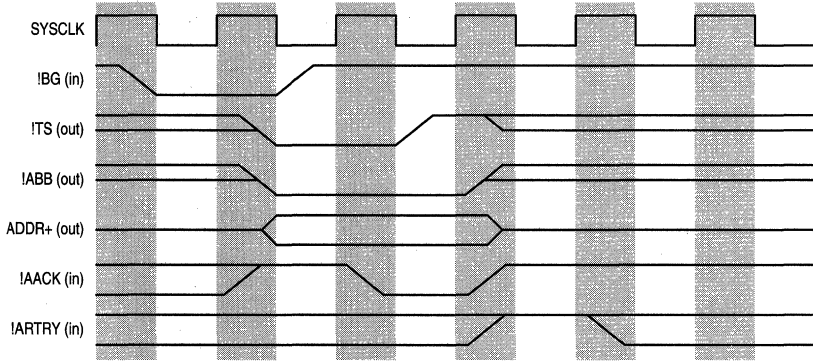


Figure 2. Address-Only Bus Transaction

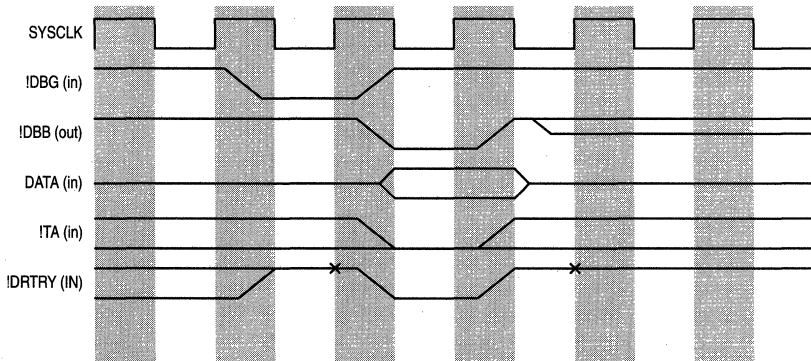


Figure 3. Data Bus Read Cycle



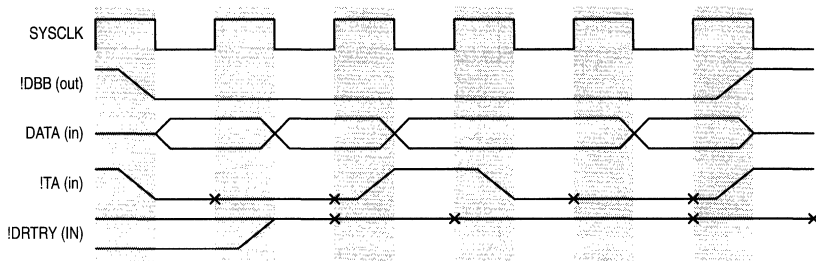


Figure 4. Data Bus Burst Cycle

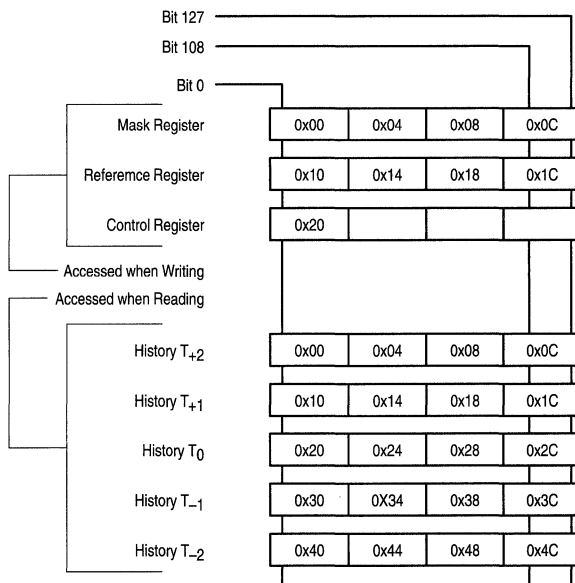


Figure 5. Internal Register Addressing

4



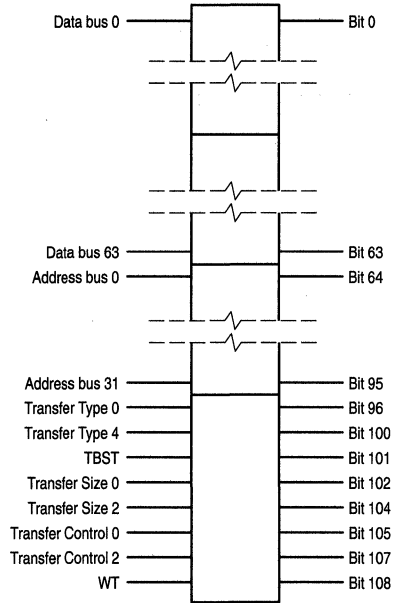


Figure 6. Bit Positions in Trigger Word

Table 1. Transfer Type Encoding

TBST	TSIZ0	TSIZ1	TSIZ2	Transfer Size
Asserted	0	0	0	Not Defined
Asserted	0	0	1	Not Defined
Asserted	0	1	0	Eight Word Burst
Asserted	0	1	1	Not Defined
Asserted	1	0	0	Not Defined
Asserted	1	0	1	Not Defined
Asserted	1	1	0	Not Defined
Asserted	1	1	1	Not Defined
Asserted	0	0	0	Not Defined
Asserted	0	0	1	Eight Bytes
Asserted	0	1	0	One Byte
Asserted	0	1	1	Two Bytes
Asserted	1	0	0	Three Bytes
Asserted	1	0	1	Four Bytes
Asserted	1	1	0	Five Bytes
Asserted	1	1	1	Six Bytes

4



Table 2. Transfer Type Encoding

TT[0:4]	Operation	Bus Transaction	Run Transaction	Snoop Hit on Transaction
00000	Clean Block	Address only	601, 604	
00100	Flush Block	Address only	601, 604	
01000	sync	Address only	601, 604	
01100	Kill Block	Address only	601, 603, 604	601, 603, 604
10000	eieio	Address only	604	
10100	ecowx	SBW	601, 603, 604	
11000	TLB Invalidate	Address only	601, 604	
11100	eciwx	SBR	601, 603, 604	
00001	larx rsrv set	Address only	604	
00101	(reserved)			
01001	tlbsync	Address only	604	
01101	icbi	Address only	604	
1xx01	(res for customer)			
00010	WR w/ FLUSH	SBW or Burst	601, 603, 604	601, 603, 604
00110	WR w/ Kill	Burst	601, 603, 604	601, 603, 604
01010	Read	SBR or Burst	601, 603, 604	601, 603, 604
01110	RWITM	Burst	601, 603, 604	601, 603, 604
100010	WR w/ Flush Atomic	SBW	601, 603, 604	601, 603, 604
10110	(reserved)			
11010	Read Atomic	SBR or Burst	601, 603, 604	601, 603, 604
11110	RWITM Atomic	Burst	601, 603, 604	601, 603, 604
00x11	(reserved)			
01011	RWITMC	SBR or Burst		603, 604
01111	(reserved)			
1xx11	(res for customer)			

SBW = Single Beat Write; SBR = Single Beat Read; RWITM = Read With Intent to Modify; RWITM = Read With Intent to Modify Cache

Table 3. Transfer Codes for MPC601

TCx	Function
TC0	On a read: 1 = Instruction fetch, 0 = Data operation
	On a write: 1 = Response to a snoop hit to modified data, 0 = Not a snoop push
TC1	An operation to reload the other sector is queued

Table 4. Transfer Codes for MPC603

TC[0:1]	Read	Write
00	Data Transfer	Not Snoop Copyback
01	Touch Load	Reserved
10	Instruction Fetch	Snoop Copyback
11	Reserved	Reserved

4

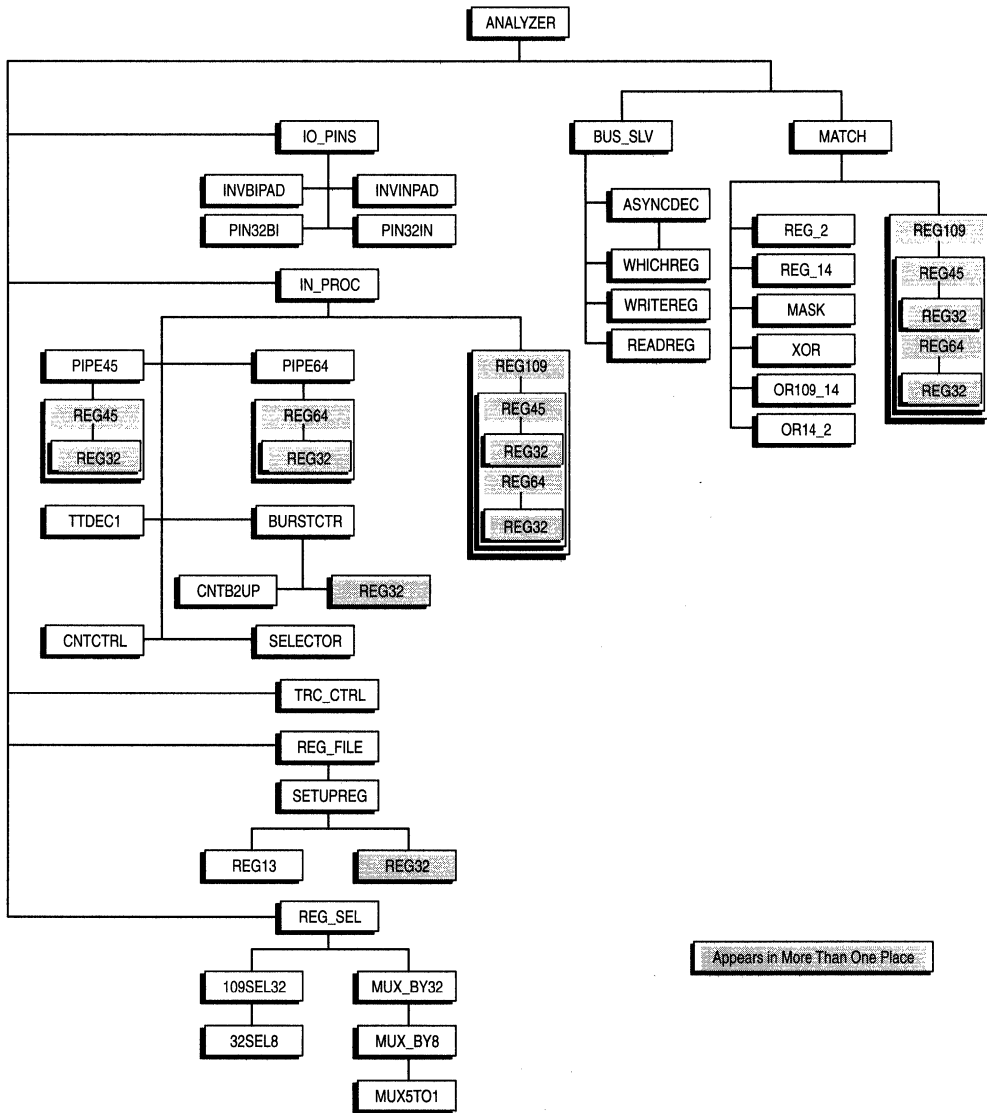


Table 5. Transfer Codes for MPC604

TC[0:2]	WT	TT[0:4]	Function
100	1	00110 (WRITE w/ Kill)	Cache Copyback
100	0	00110 (WRITE w/ Kill)	Block Invalidate (DCBF)
000	0	00110 (WRITE w/ Kill)	Block Clean (DCBST)
010	1	00110 (WRITE w/ Kill)	Snoop Push (READ)
100	1	00110 (WRITE w/ Kill)	Snoop Push (RWITM)
000	1	00110 (WRITE w/ Kill)	Snoop Push (CLEAN)
100	1	00110 (WRITE w/ Kill)	Snoop Push (FLUSH)
100	Don't Care	01100 (Kill Block)	Kill Block Deallocate (DCBI)
000	1	01100 (Kill Block)	Kill Block and Allocate no Castout Required (DCBZ)
001	1	01100 (Kill Block)	Kill Block and Allocate Castout Required (DCBZ)
000	1	01100 (Kill Block)	Kill Block (Allocate) Write to SH Block
0x0	Write through Bit Value	x1x10 (READ)	Data Read no Castout Required
0x1	Write through Bit Value	x1x10 (READ)	Data Read Castout Required
1x0	Write through Bit Value	x1x10 (READ)	Instruction Read
100	Don't Care	01101 (icbi)	Kill Block Deallocate (icbi)

4





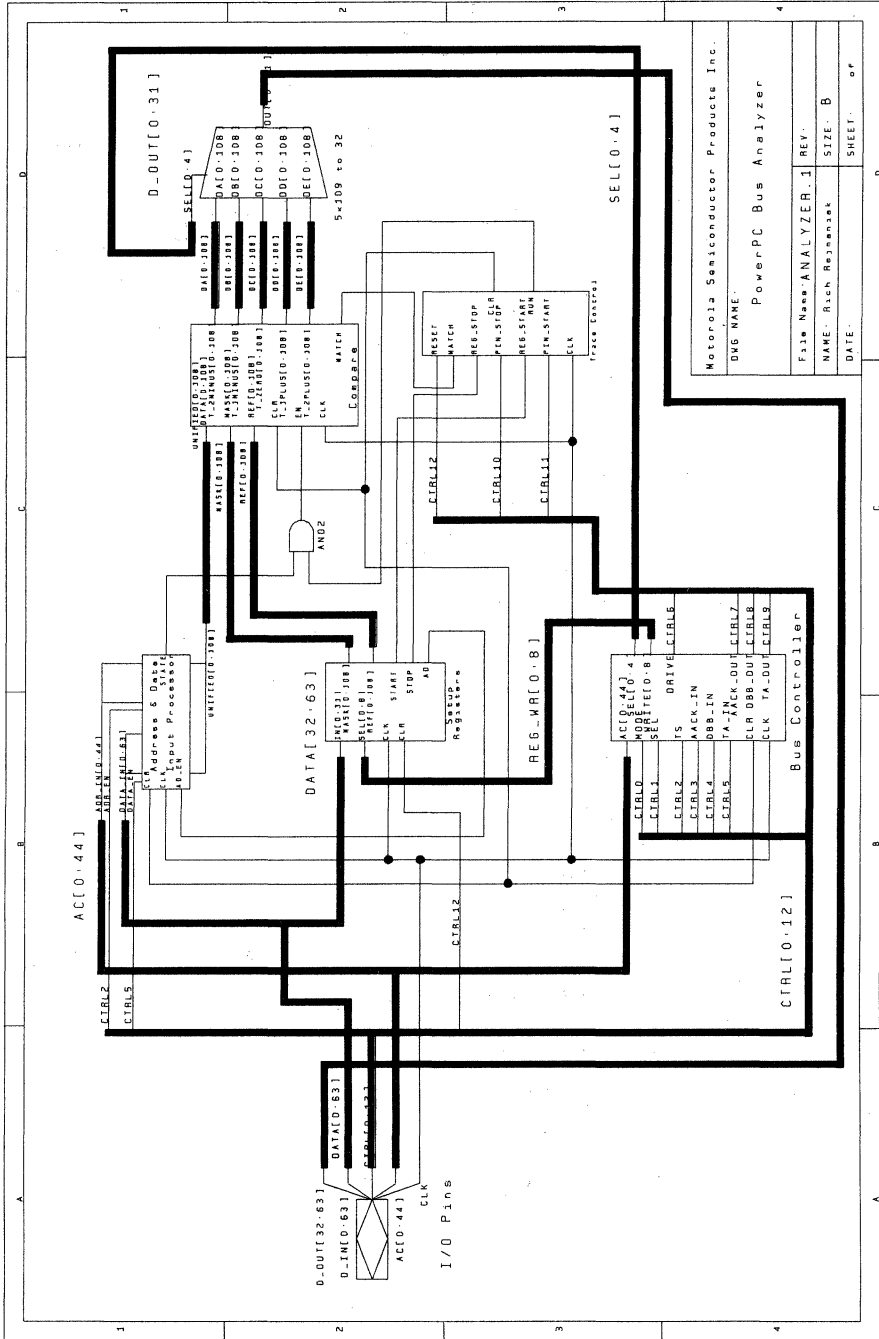
4

Appears in More Than One Place

Figure 7. Schematic Hierarchy



4

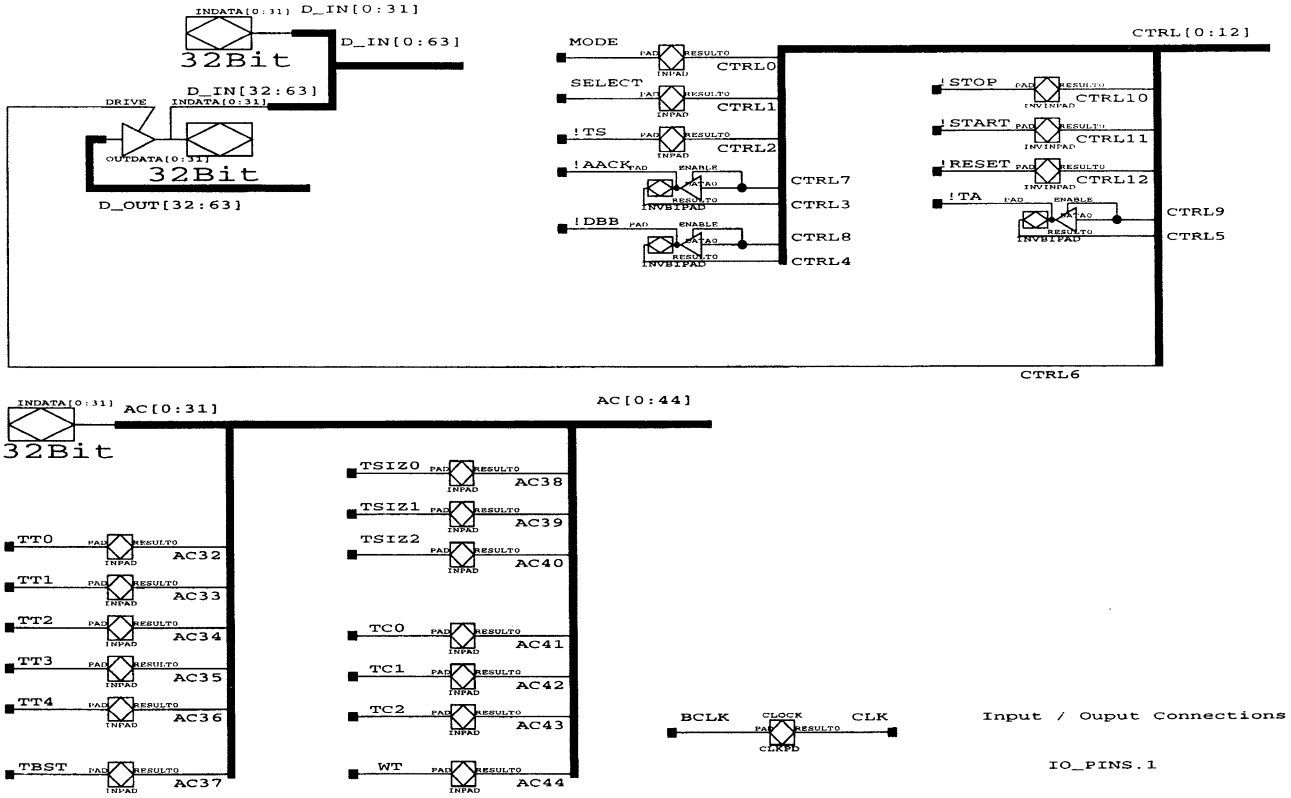


Motorola Semiconductor Products Inc.	
DWG NAME	PowerPC Bus Analyzer
File Name	ANALYZER.1 REV.
NAME - Rack	Rejehansk
DATE	SIZE: B
	SHEET: of

Figure 8. Analyzer



Figure 9. IO_PINS



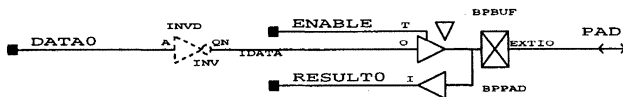


Figure 10. INVBI PAD

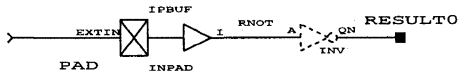
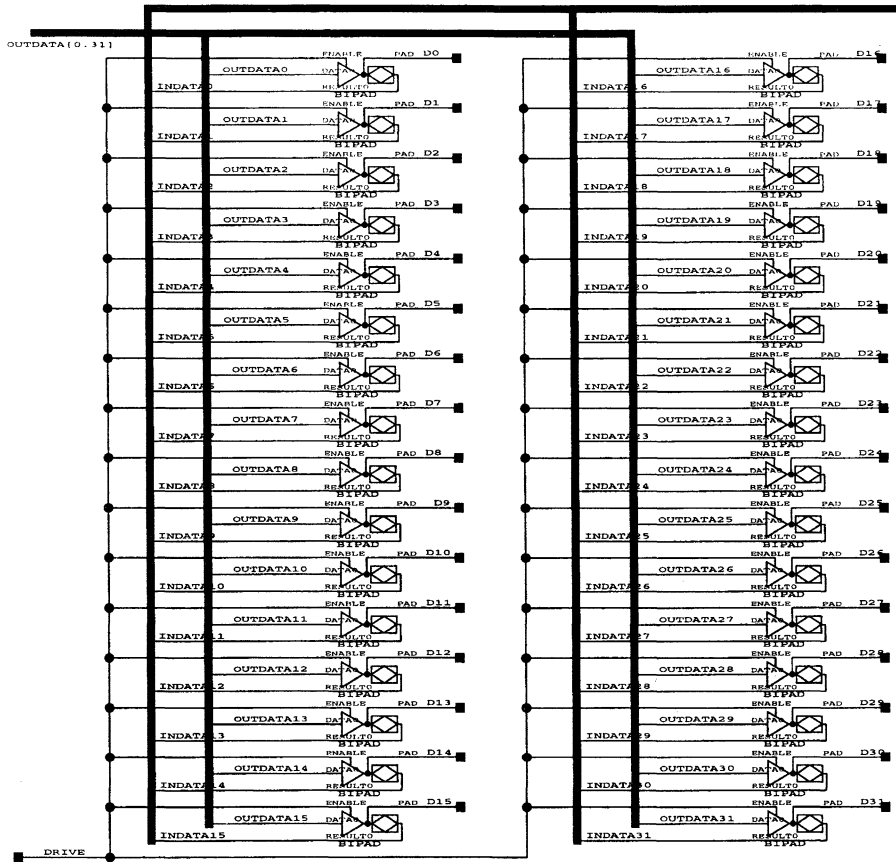


Figure 11. INVIN PAD

4





32 Bit Bidirectional Pir

PINS32BI.1

Figure 12. PINS32BI

MOTOROLA MPA DATA — DL201 REV 2
4-49



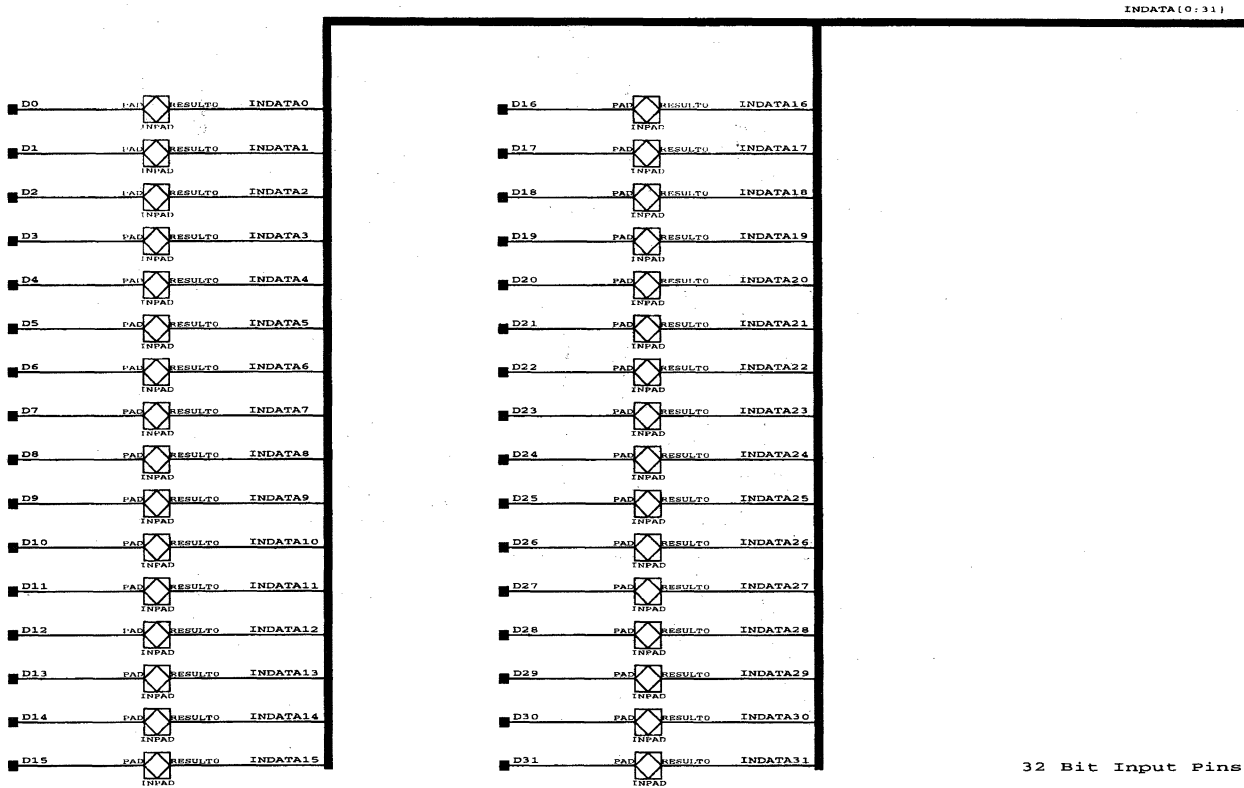
MPA



MOTOROLA MPA DATA — DL201 REV 2

4-50

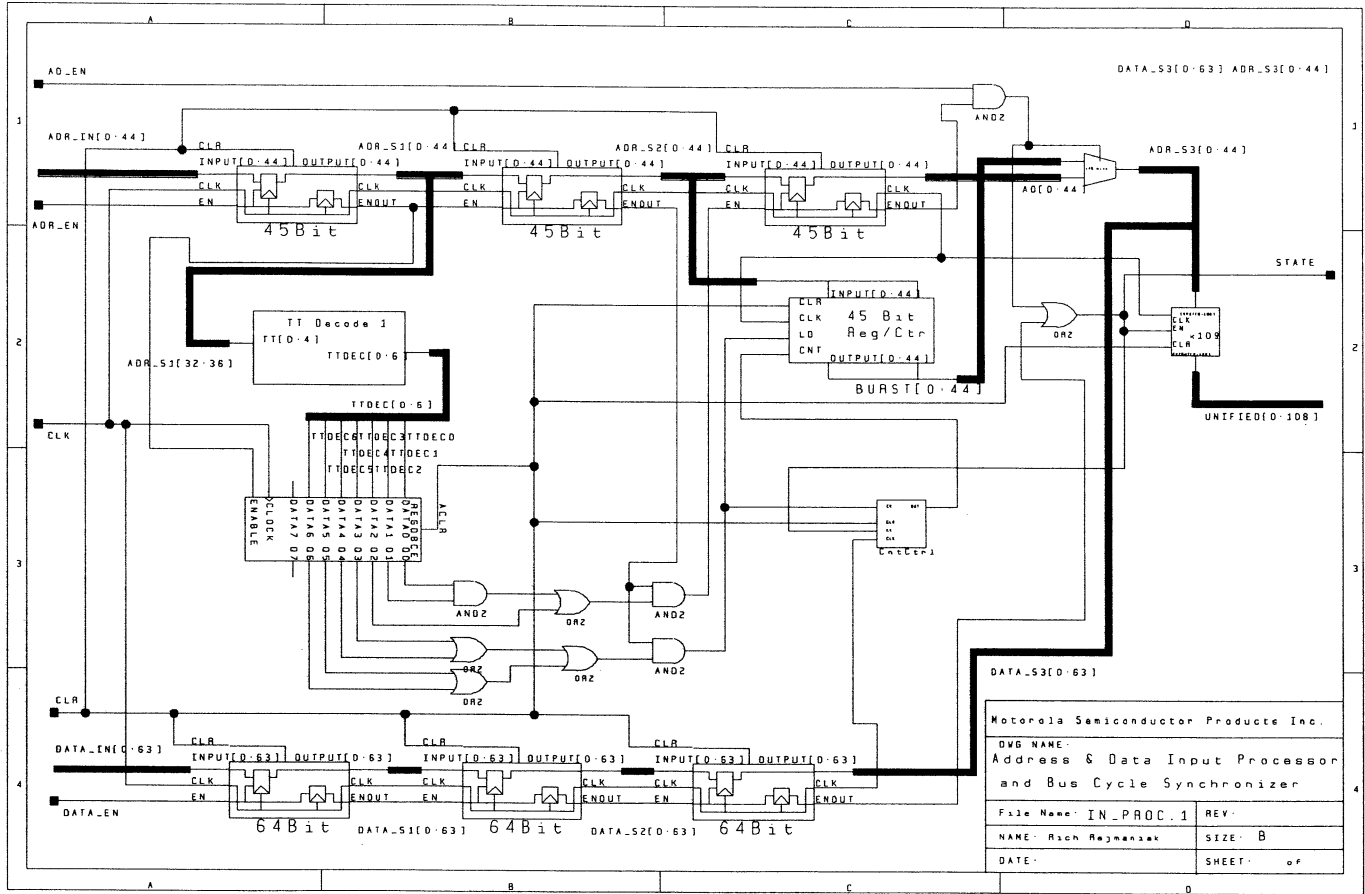
Figure 13. PIN32IN



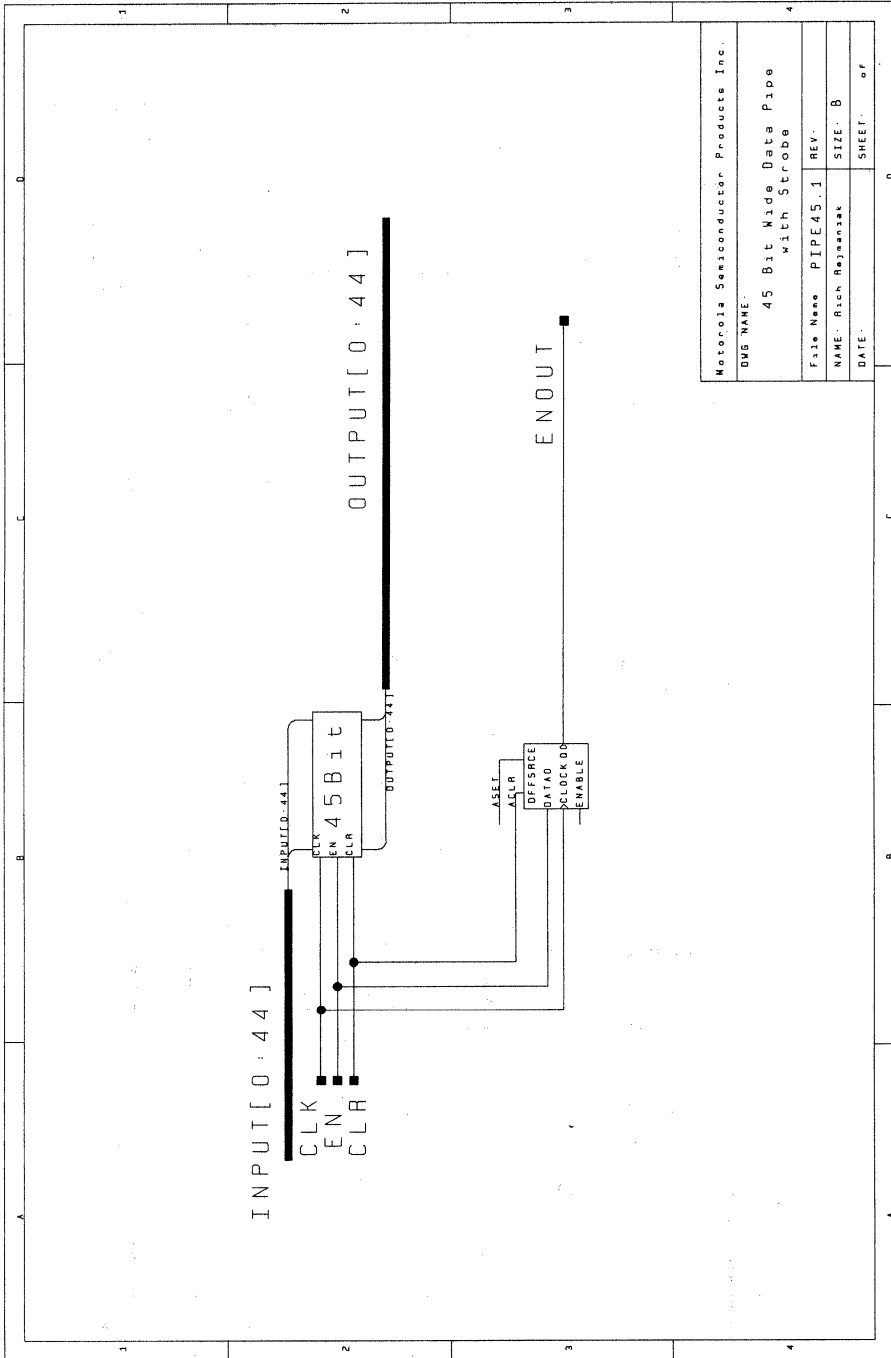
32 Bit Input Pins

PIN32IN.1

Figure 14. IN_PROC



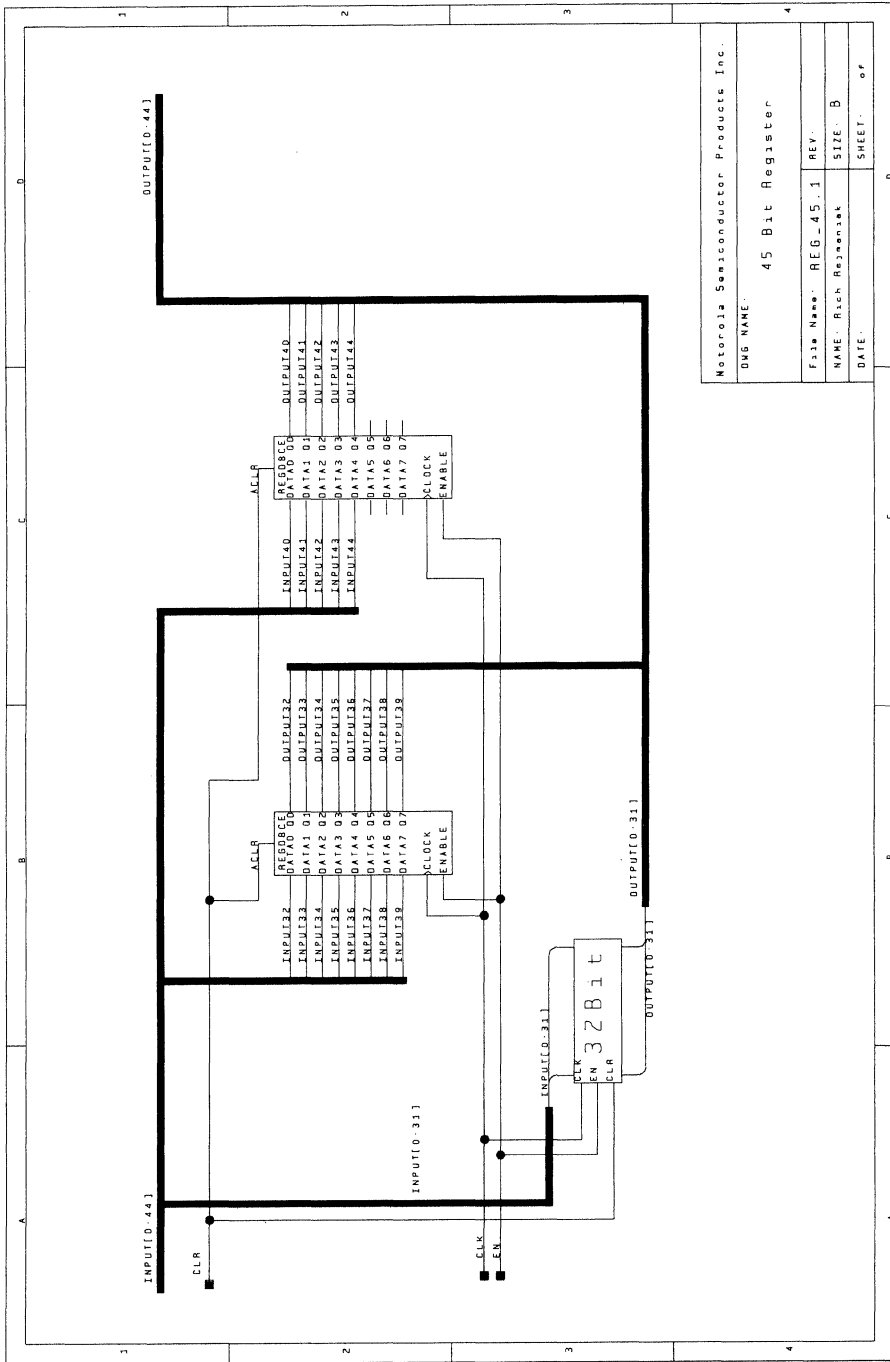
4



Motorola Semiconductor Products Inc.			
DWG NAME:		45 Bit Wide Data Pipe	
File Name:		PIPE45.1	
NAME: Rich Rajanank		REV:	SIZE: B
DATE:		SHEET:	of

Figure 15. PIPE45





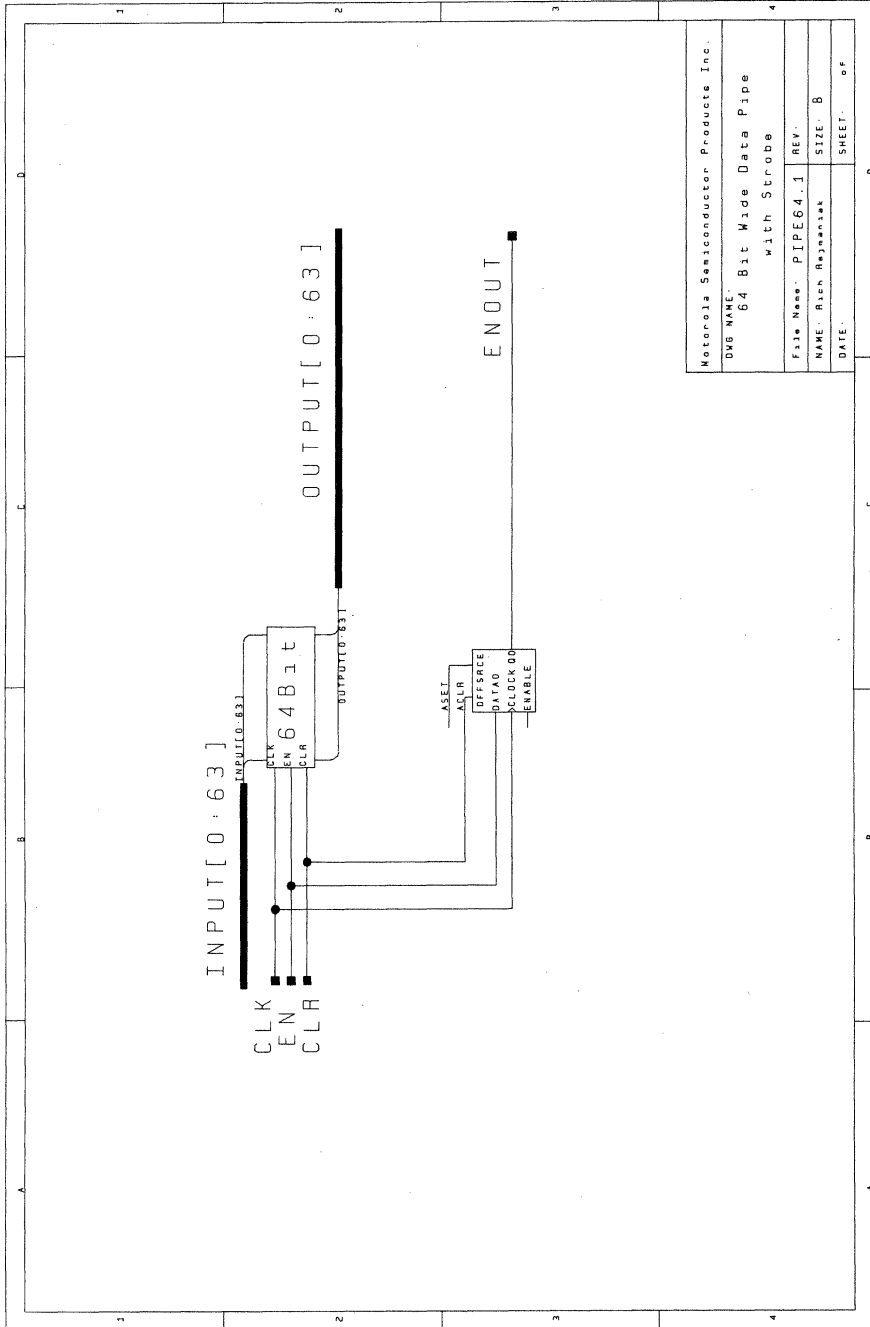
Motorola Semiconductor Products Inc.	
DWG NAME:	45 Bit Register
File Name:	REG_45.1 REV.
NAME:	Rechnungsk
DATE:	SHEET
	of

4

Figure 16. REG_45



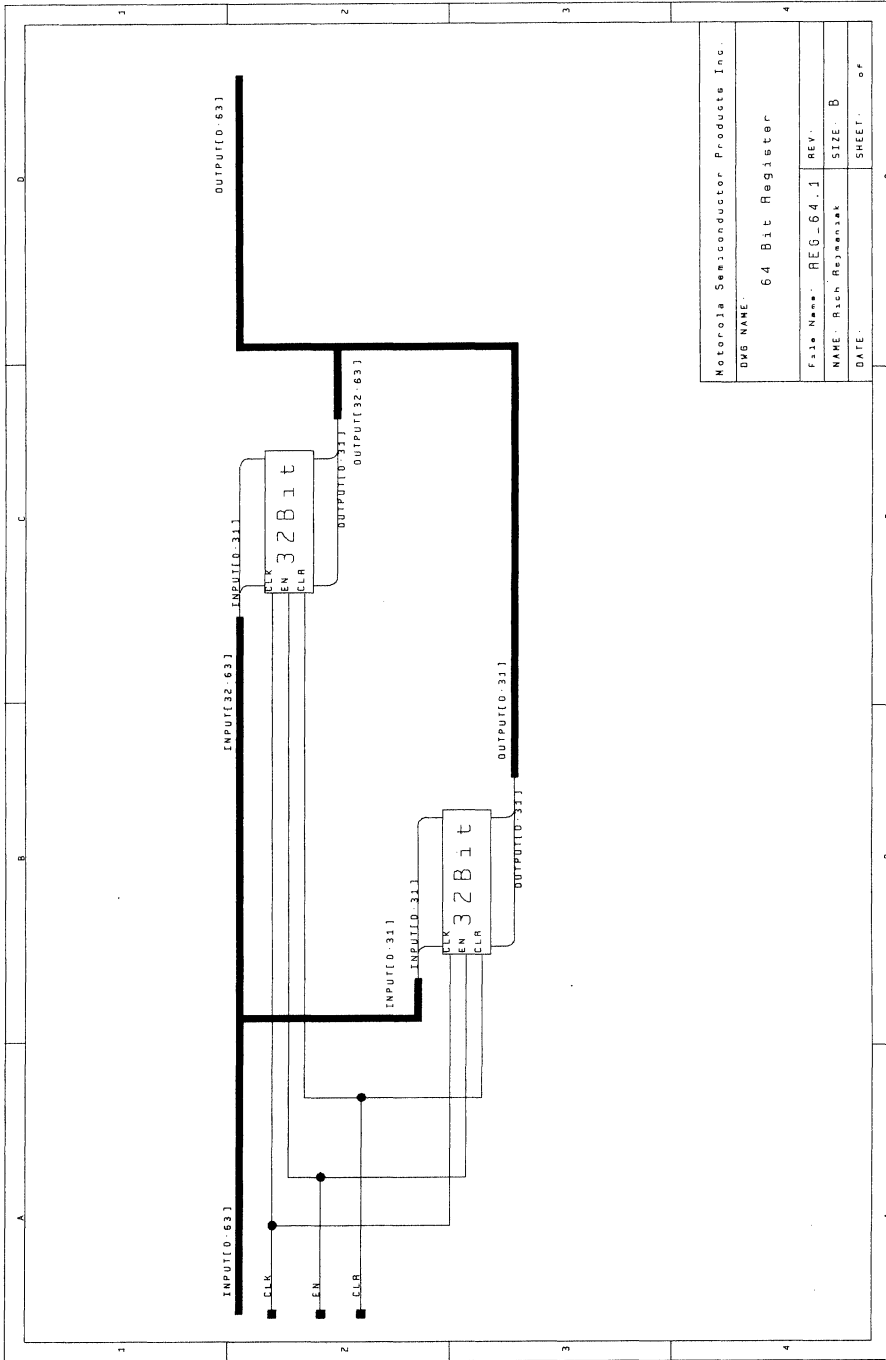
4



Motorola Semiconductor Products Inc.	
DS# NAME:	64 Bit Wide Data Pipe
File Name:	PIPE64.1
REV:	with Strobb
NAME - Rich Resnikus	SIZE - B
DATE:	SHEET - of

Figure 17. PIPE64





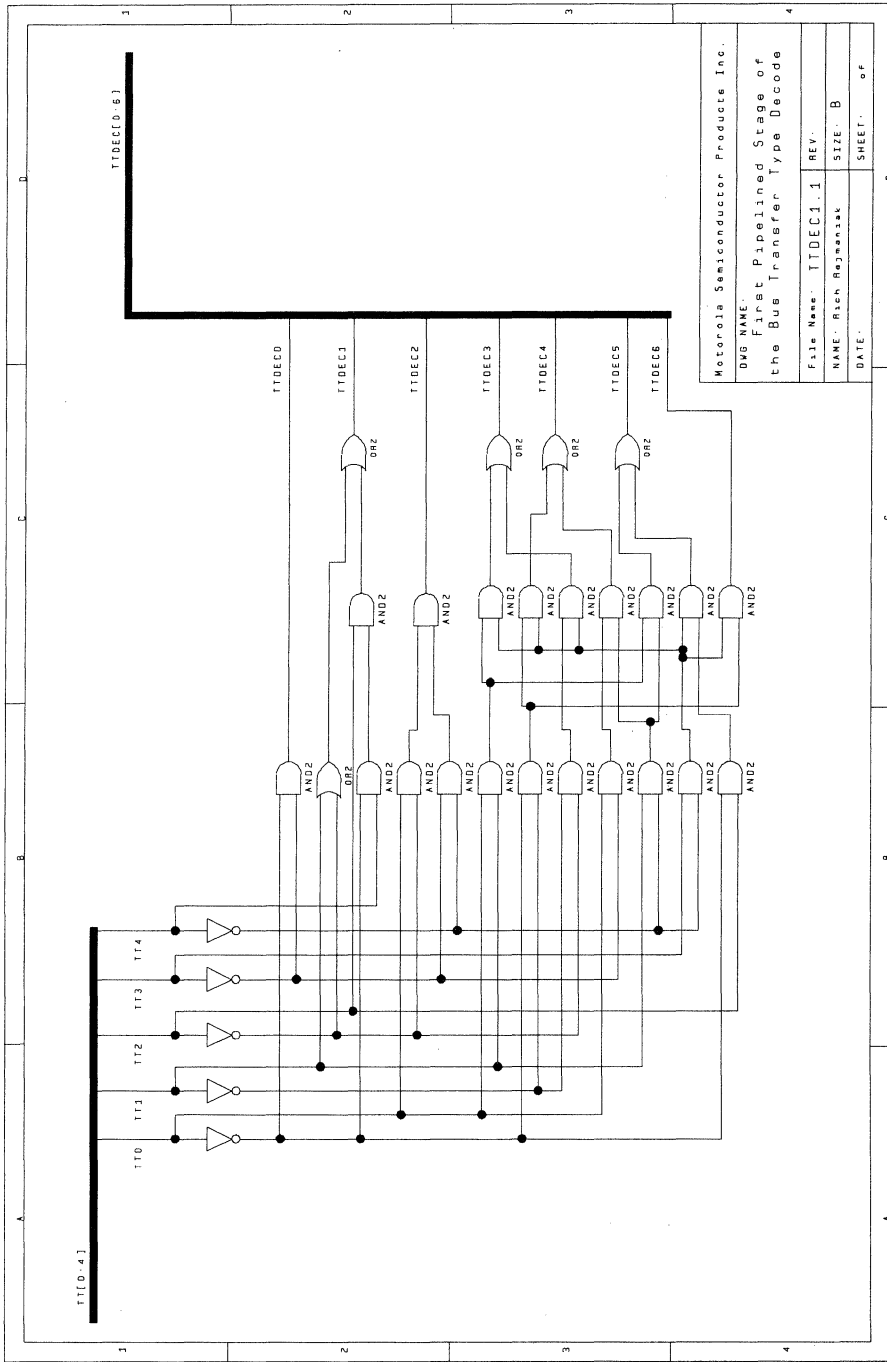
Motorola Semiconductor Products Inc.			
DSB NAME: 64 Bit Register			
File Name:	REG_64.1	REV:	
NAME:	Reg Register	SIZE:	B
DATE:		SHEET:	of

4

Figure 18. REG_64



4

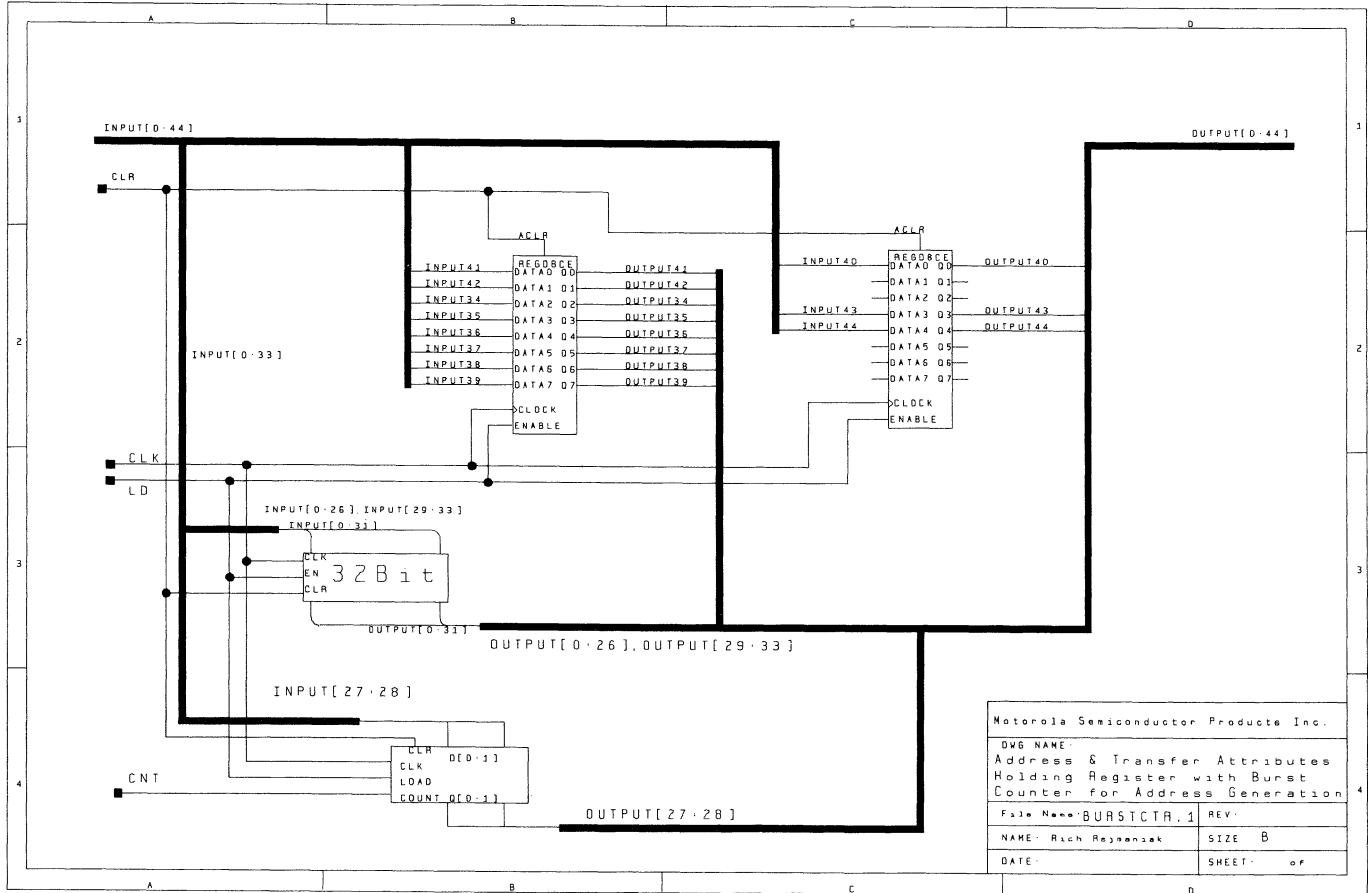


Motorola Semiconductor Products Inc.	
Dwg Name: TTDEC1	
First Pipelined Stage of the Bus Transfer Type Decode	
File Name: TTDEC1.1	REV:
NAME: Rich Rajanank	SIZE: B
DATE:	SHEET: of

Figure 19. TTDEC1

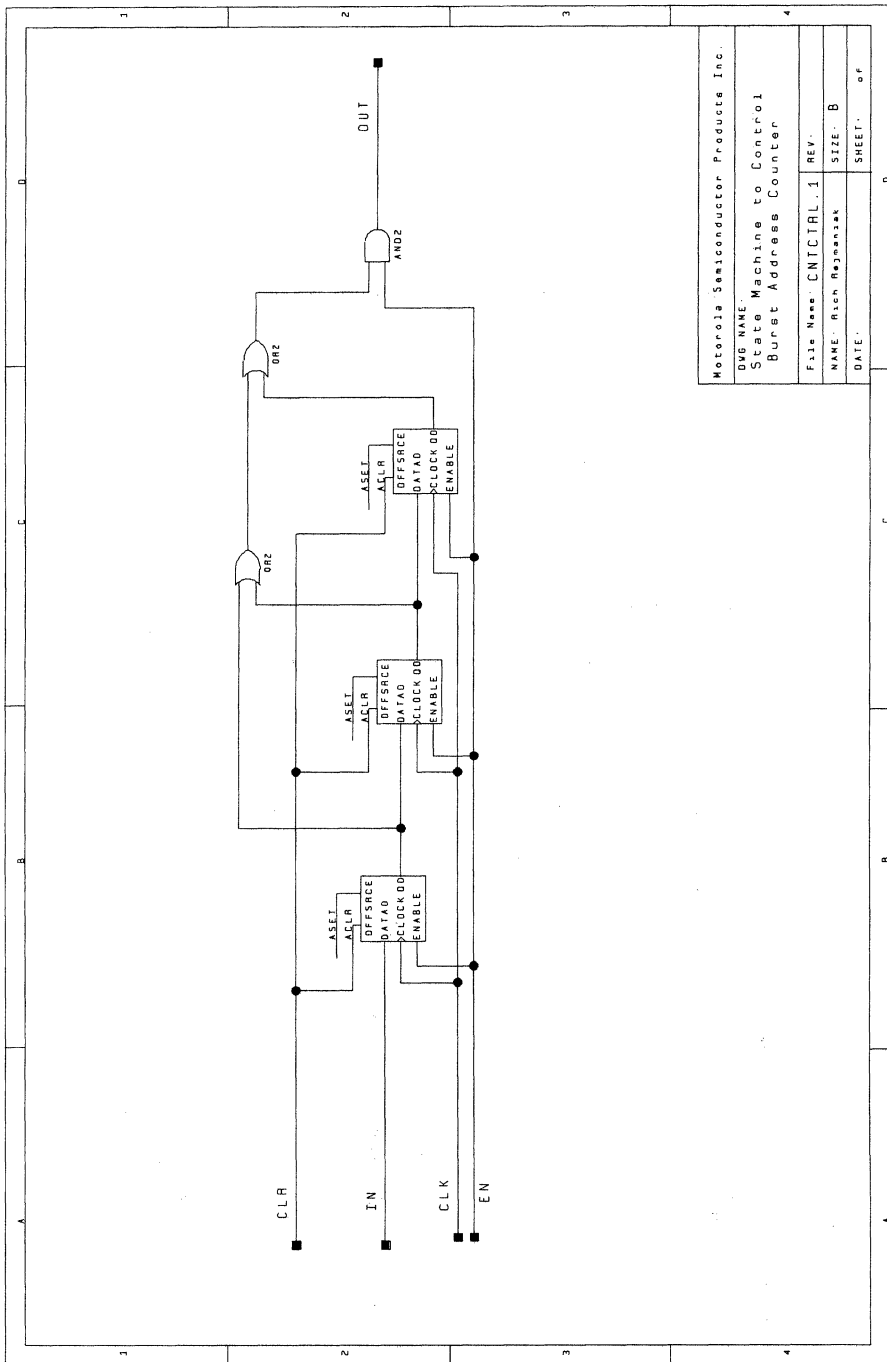


Figure 20. BURSTCTR



Motorola Semiconductor Products Inc.	
DWG NAME: Address & Transfer Attributes Holding Register with Burst Counter for Address Generation	
File Name: BURSTCTR.1	REV:
NAME: Rich Rejnensak	SIZE: B
DATE:	SHEET: of

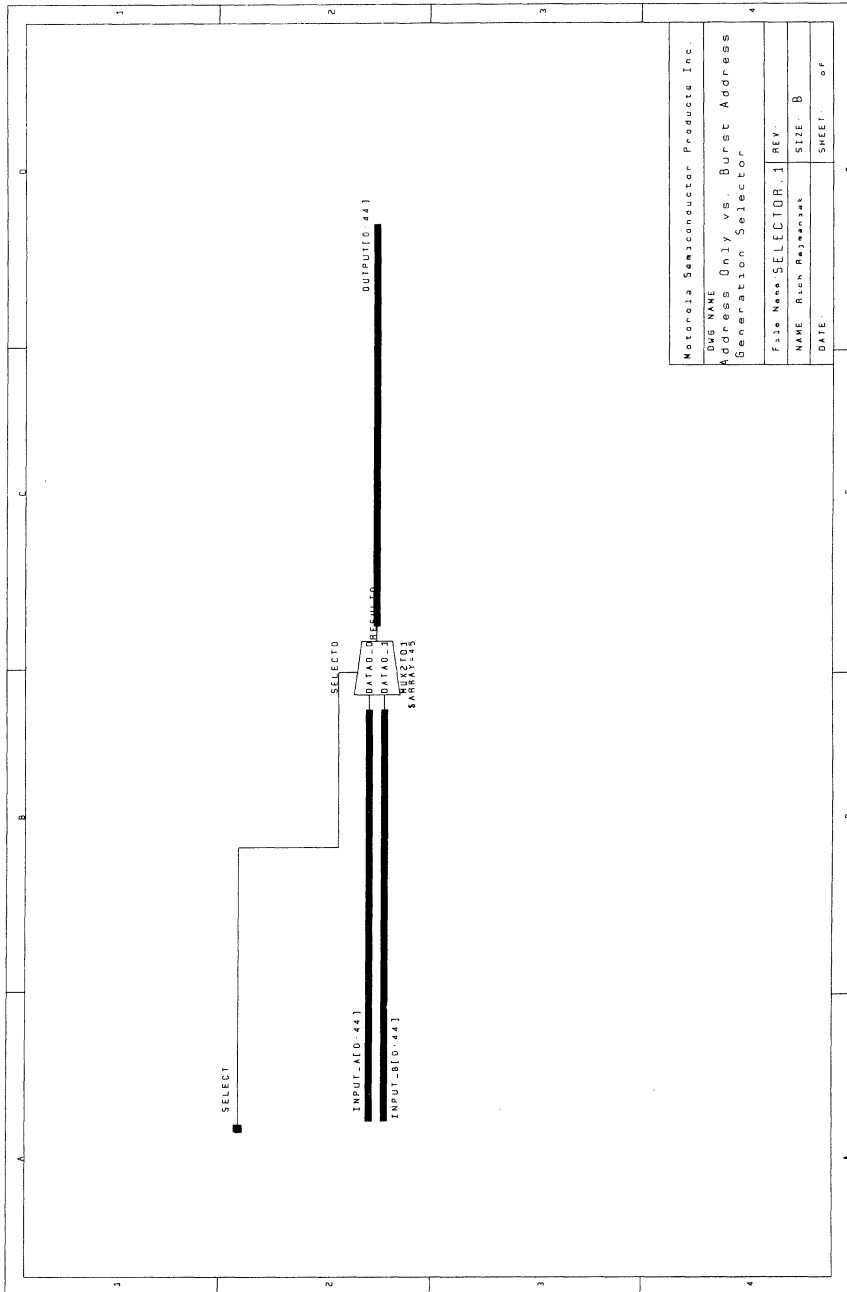
4



Motorola Semiconductor Products Inc.	
DWG NAME: State Machine to Control Burst Address Counter	
File Name: CNTCTRL.1 REV:	SIZE: B
NAME: Rich Rajananiak	DATE:
SHEET: of	

Figure 21. CNTCTRL





Motorola Semiconductor Products Inc.	
DSG NAME	Address Only vs. Burst Address Generation Selector
FILE NAME	SELECTOR.1
REV.	1
NAME	Rich Rajanath
DATE	
SIZE	B
SHEET	of

4

Figure 22. SELECTOR



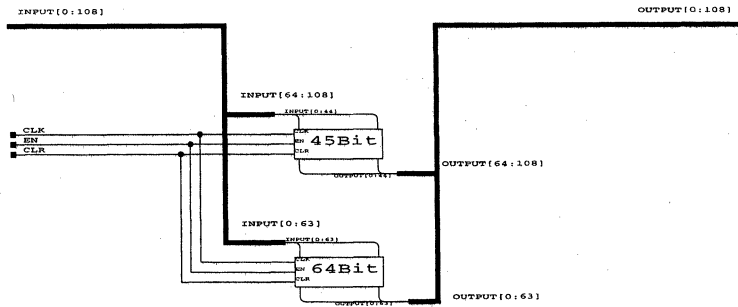


Figure 23. REG109

4

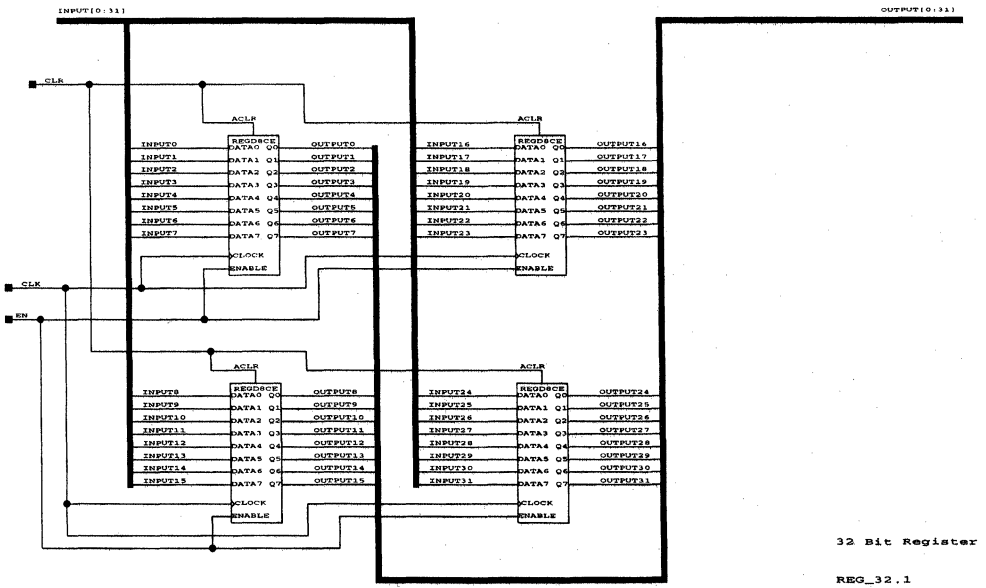
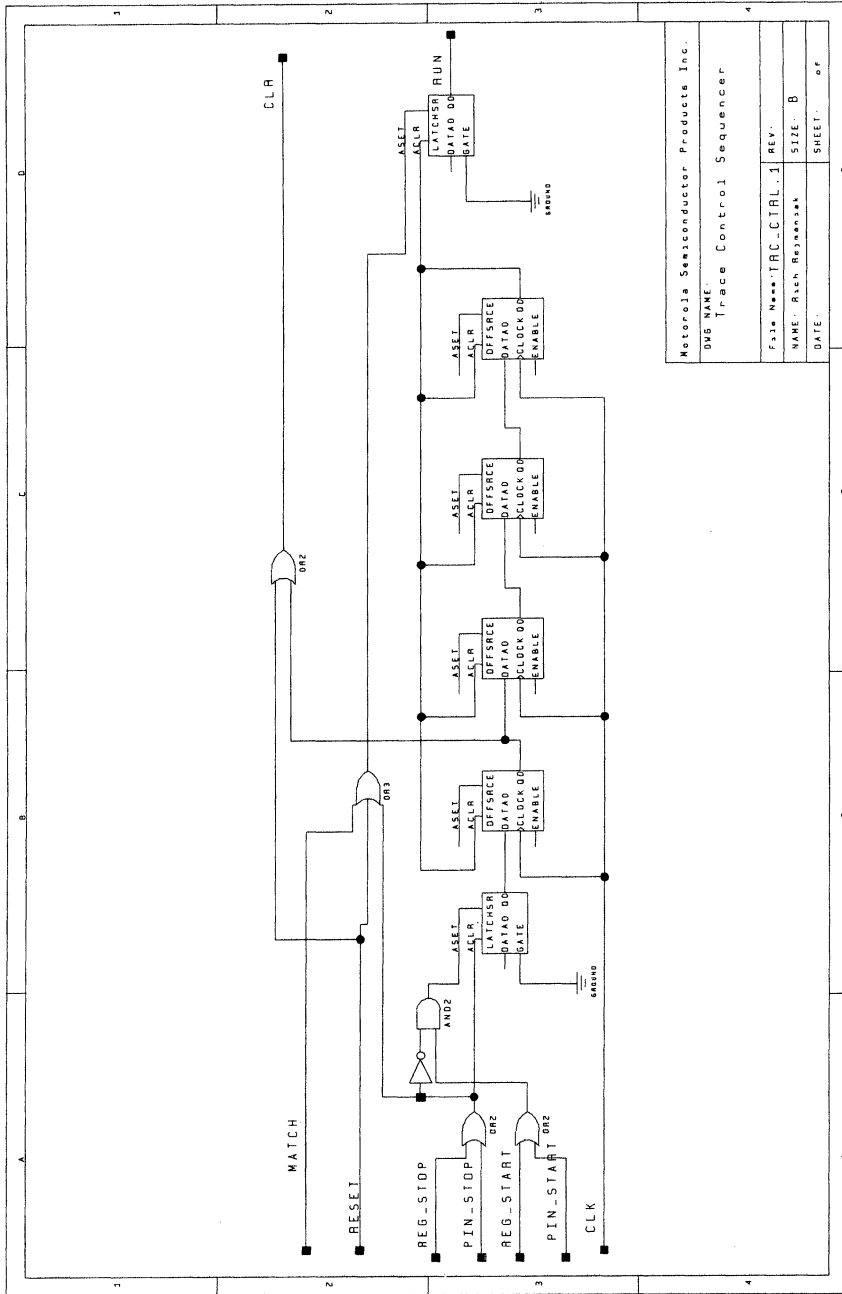


Figure 24. REG32





Motorola Semiconductor Products Inc.	
DWG NAME:	Trace Control Sequencer
FILE NAME:	TRC_CTRL.1
NAME:	Rich Johnson
DATE:	
REV:	
SIZE:	B
SHEET:	of

4

Figure 25. TRC_CTRL



4

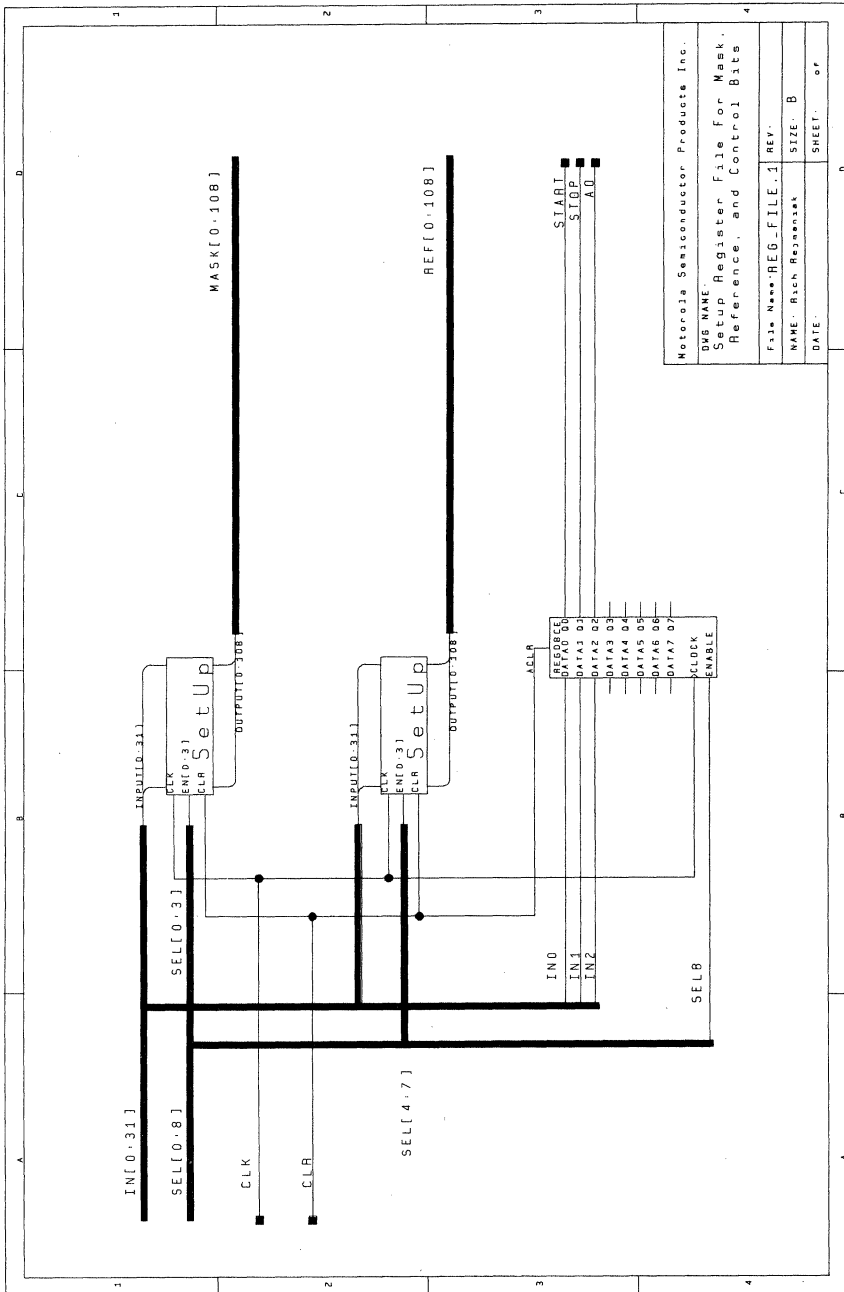
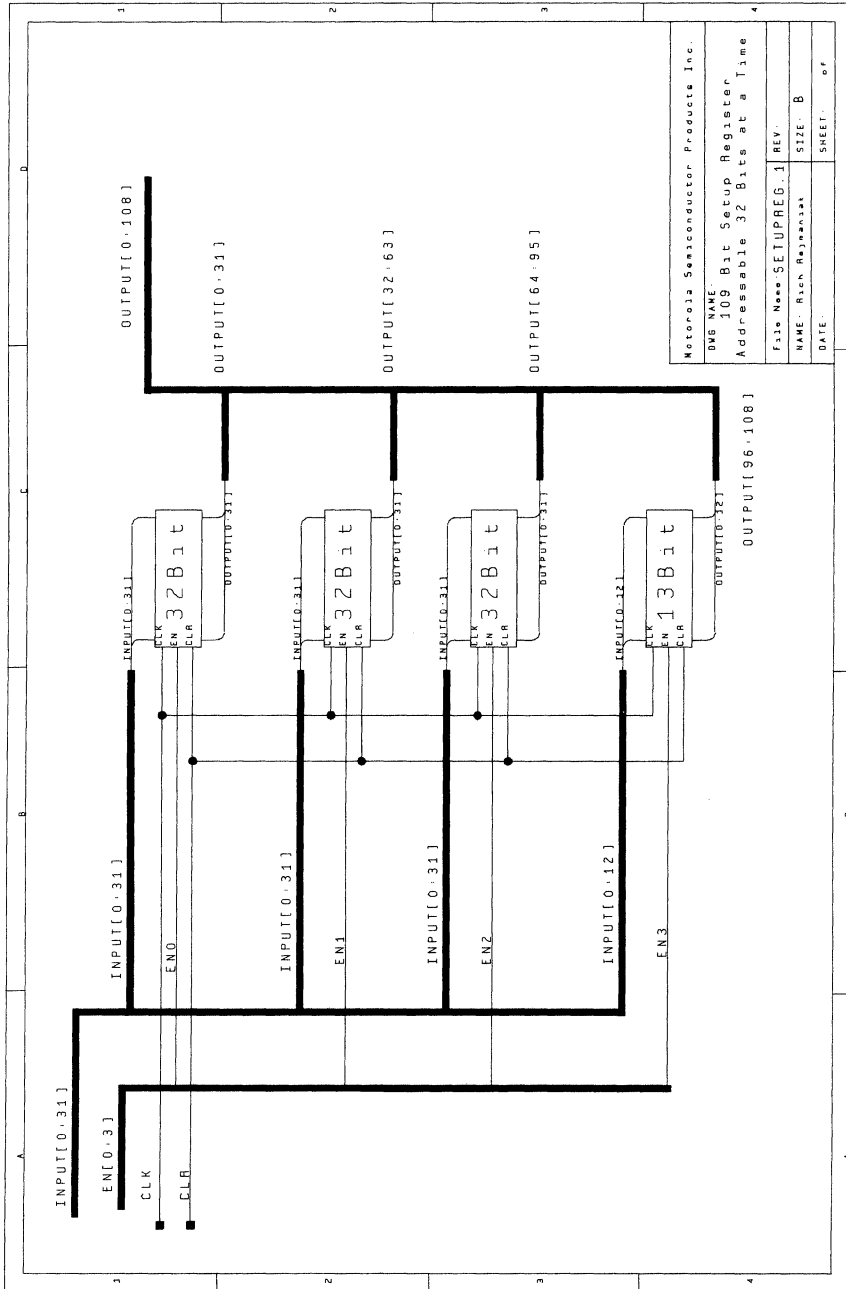


Figure 26. REG_FILE





Motorola Semiconductor Products Inc.	
DSG NAME:	109 Bit Setup Register
Addressable 32 Bits at a Time	
File Name: SETUPREG_1	REV.
NAME: Rajarajasekar	SIZE: B
DATE:	SHEET: of

4

Figure 27. SETUPREG



4

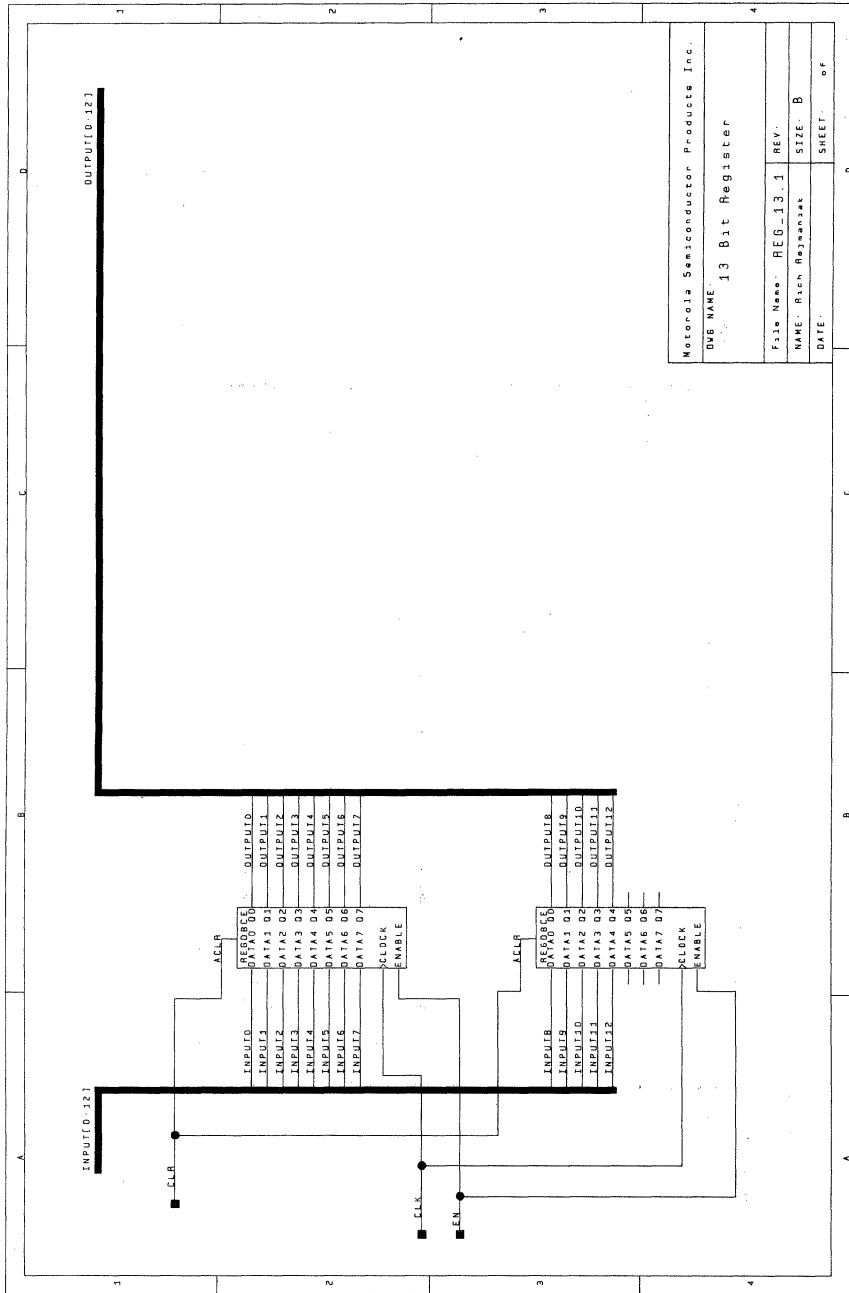
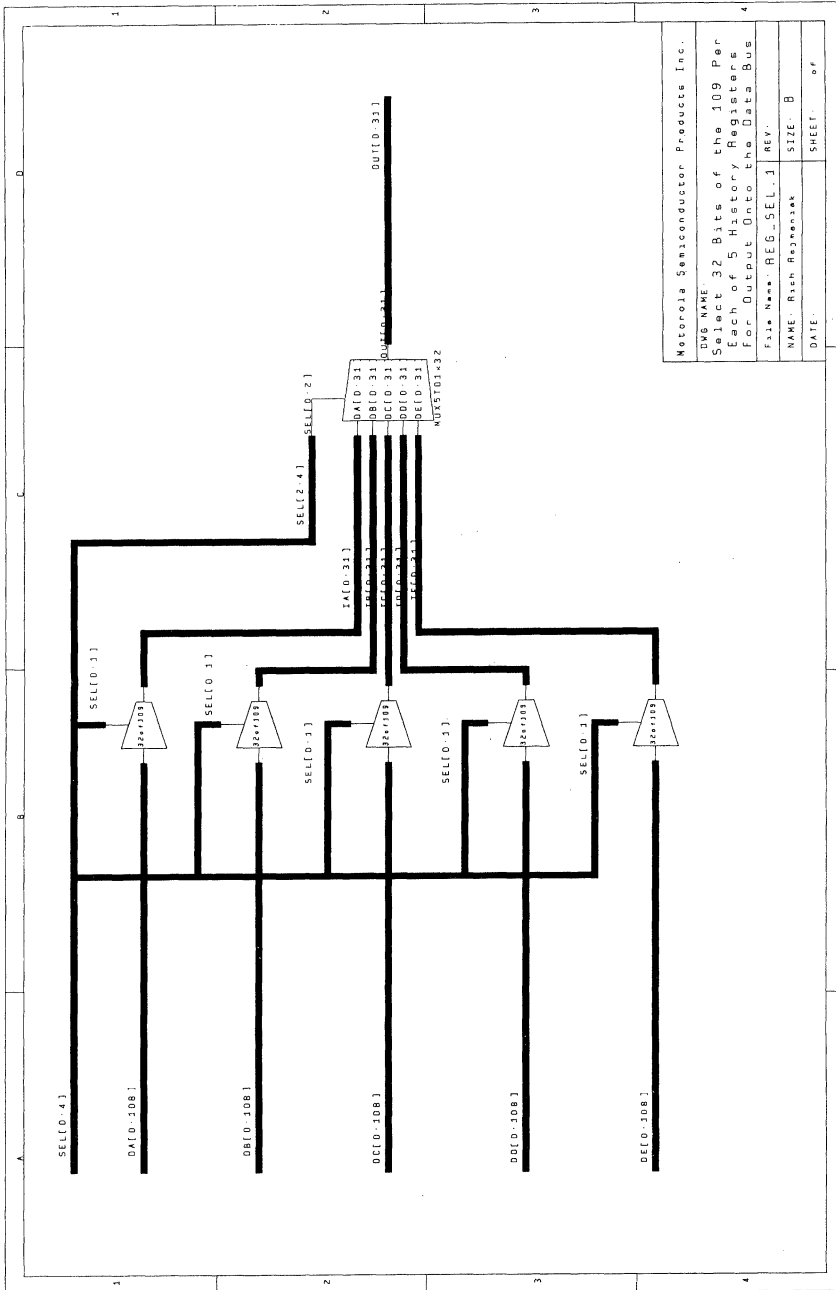


Figure 28. REG_13





Motorola Semiconductor Products Inc.	
DWG NAME: 32 Bits of the 109 Per	
Each of 5 History Registers	
For Output Onto the DATA BUS	
File Name: REG_SEL.J	REV:
NAME: Rich Rajmehak	SIZE: B
DATE:	SHEET: of

Figure 29. REG_SEL



4

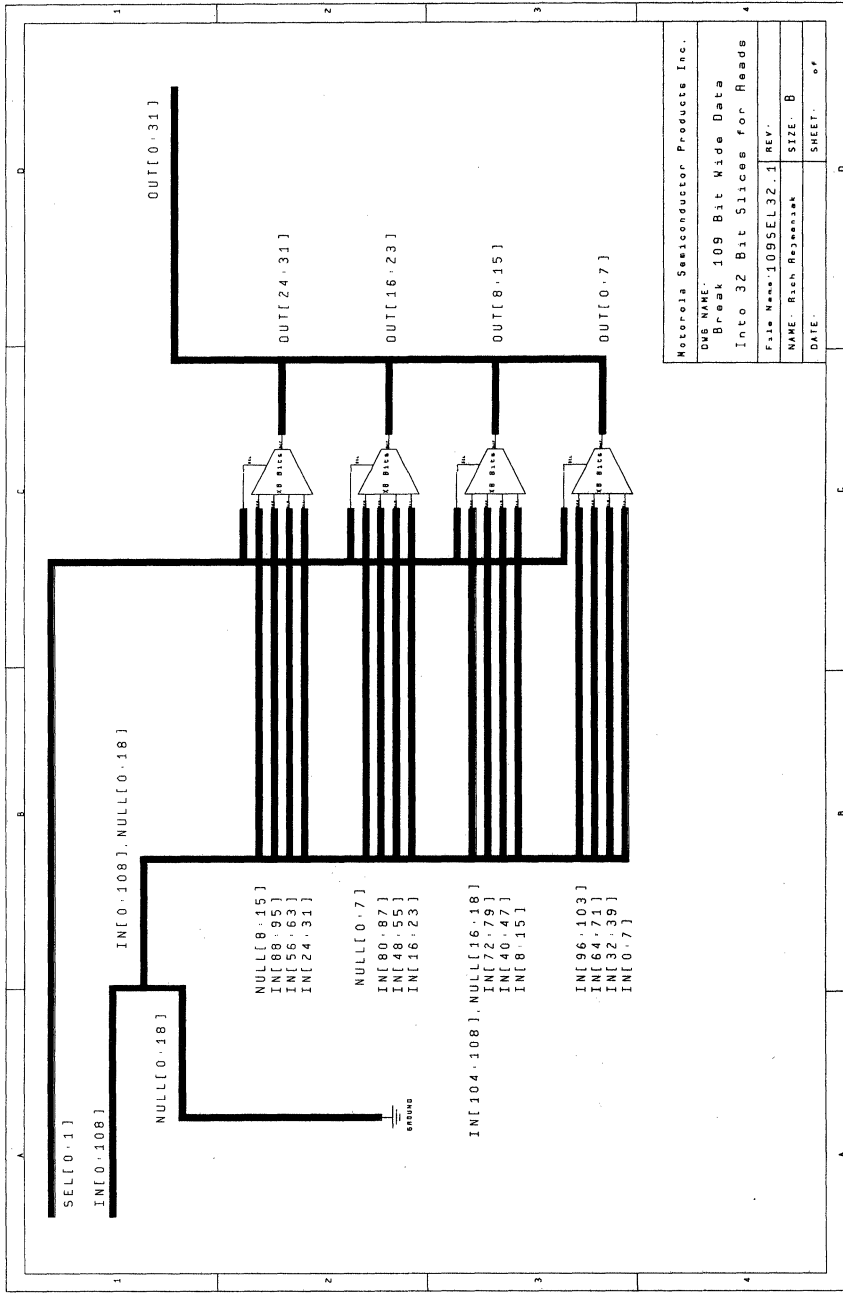
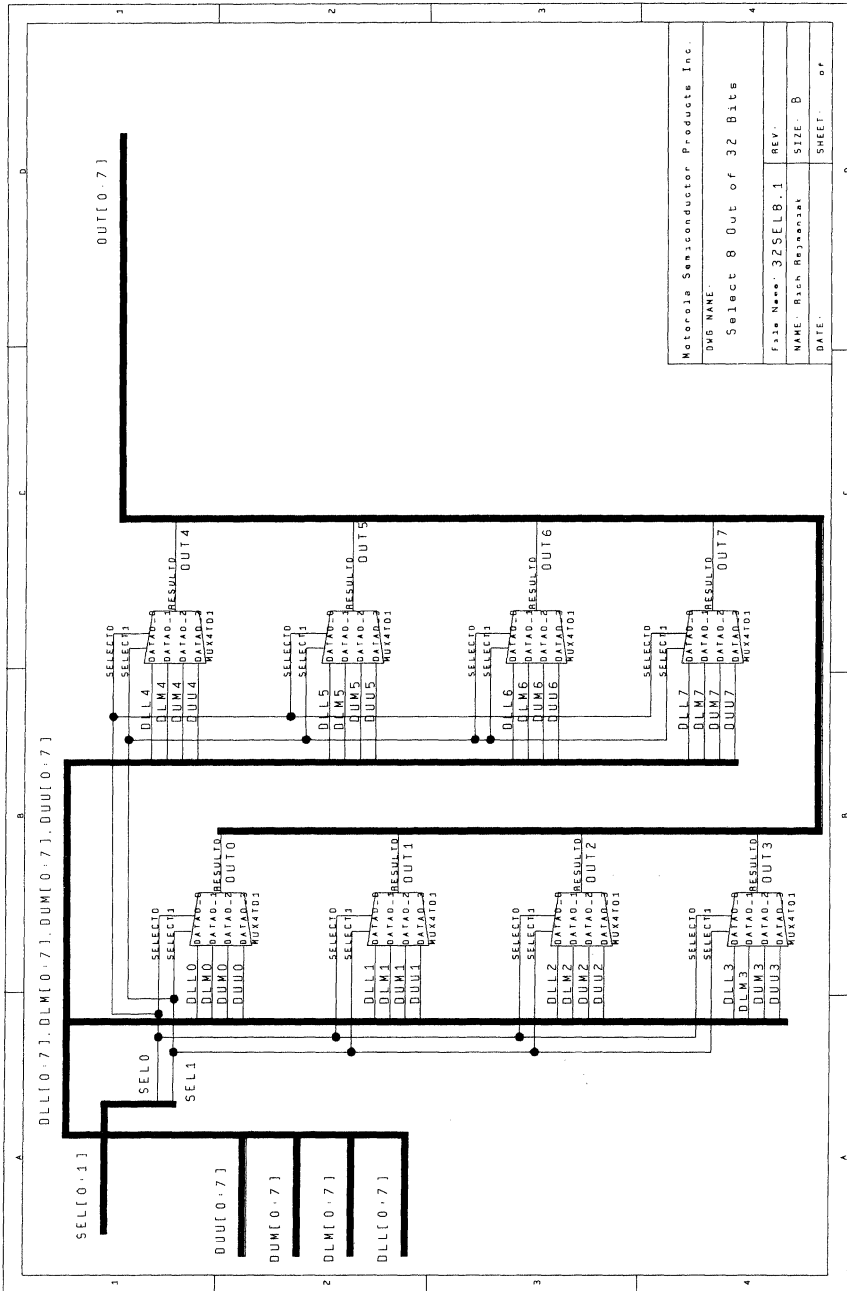


Figure 30. 109SEL32



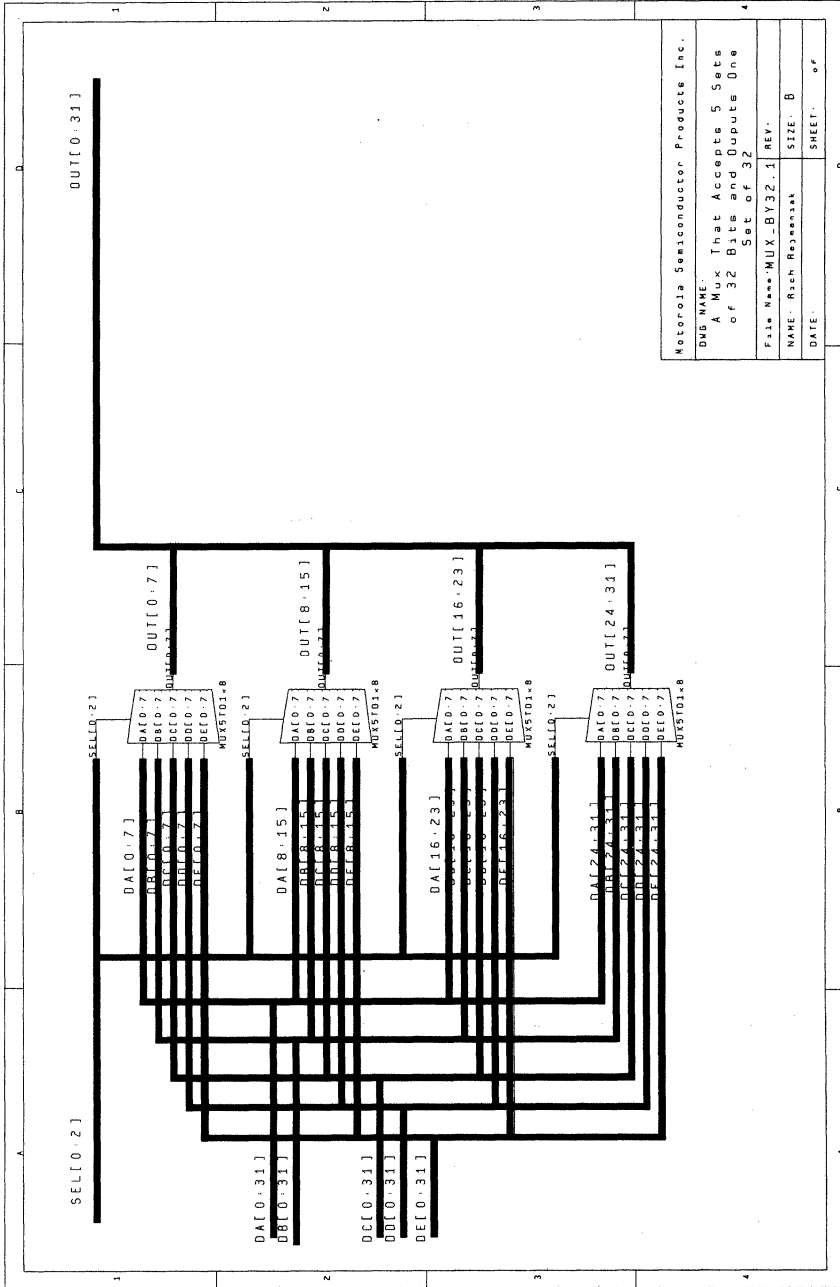


Motorola Semiconductor Products Inc.	
DWG NAME:	
Select 8 Out of 32 Bits	
File Name:	32SEL8.1
NAME:	Bish Rajaraman
DATE:	
REV:	SIZE: B
SHEET:	OF

Figure 31. 32SEL8



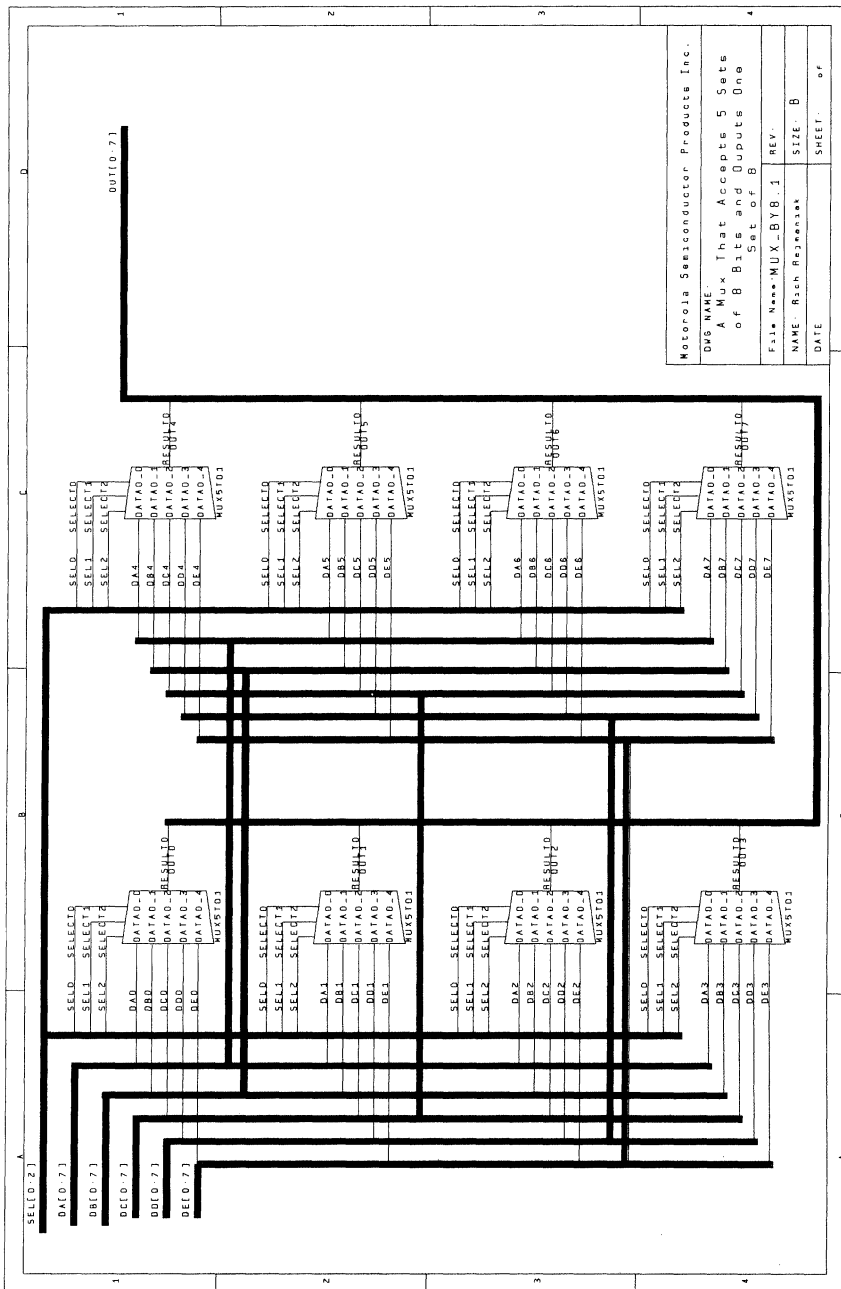
4



Motorola Semiconductor Products Inc.
 DWG NAME: A Mux That Accepts 5 Sets of 32 Bits and Outputs One Set of 32
 FILE NAME: MUX_BY32.1 REV: NAME: Rich Rejzszak SIZE: B DATE: SHEET: of

Figure 32. MUX_BY32





Motorola Semiconductor Products Inc.

DWG NAME: A Mux That Accepts 5 Sets of 8 Bits and Outputs One Set of 8

FILE NAME: MUX_BY8.1 REV: B

NAME: Rich Rejzmannak SIZE: B

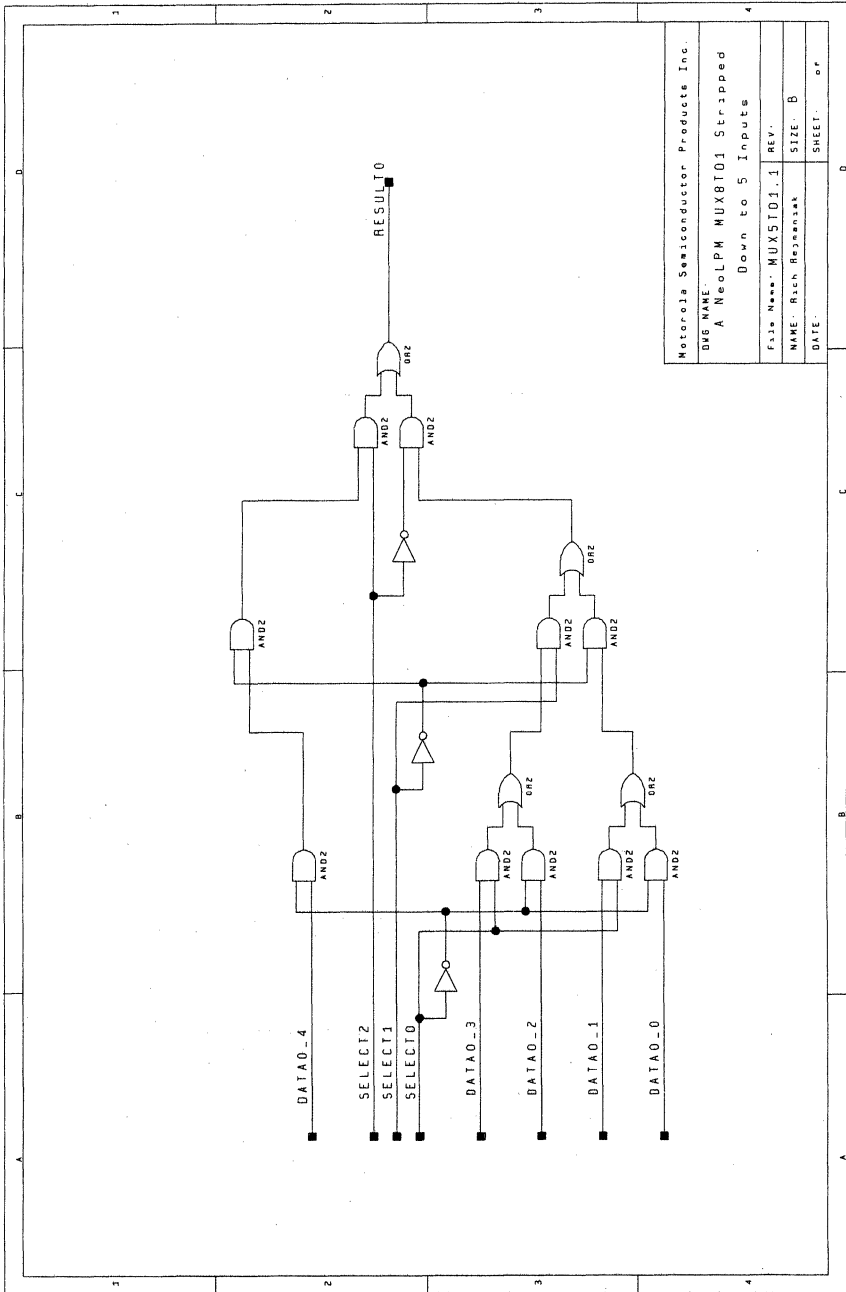
DATE: SHEET: of

4

Figure 33. MUX_BY8



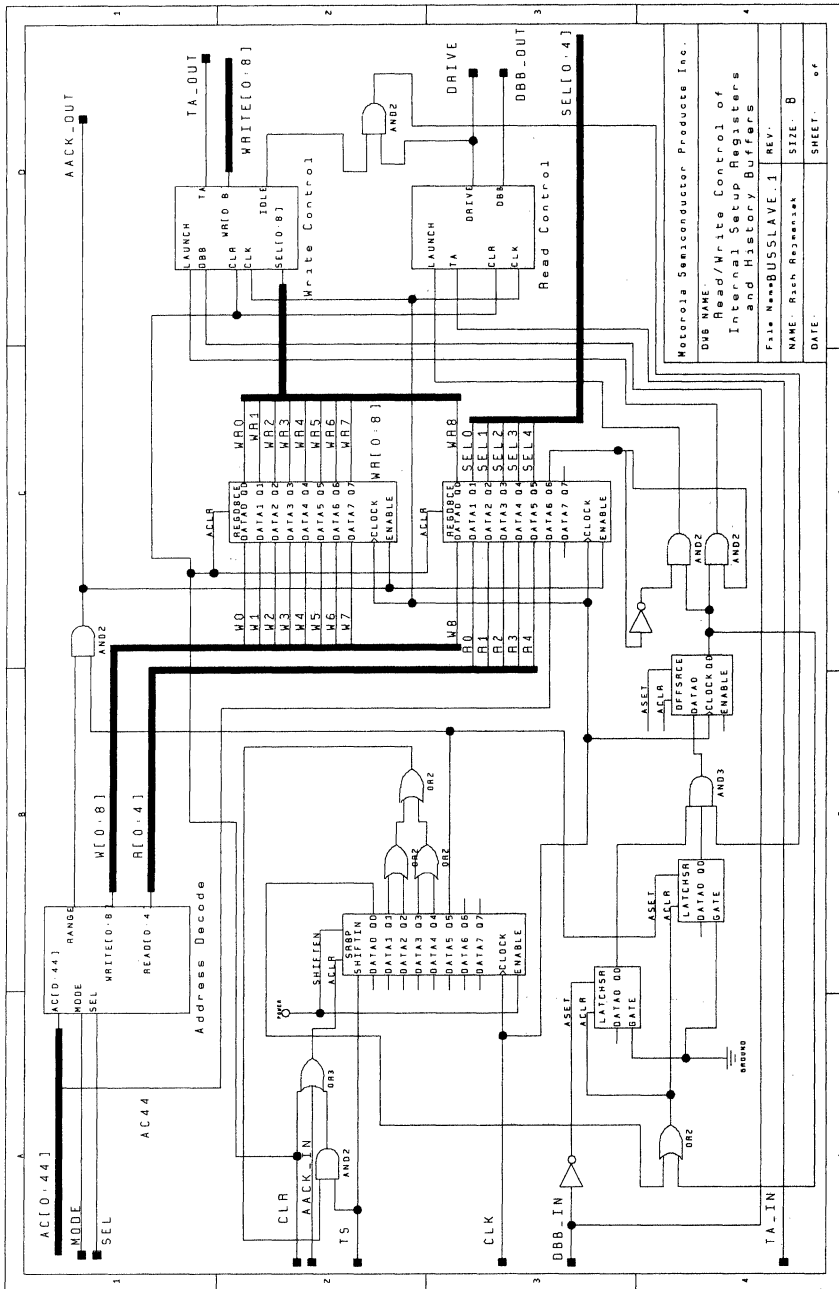
4



Motorola Semiconductor Products Inc.	
BUG NAME:	A NeolPM MUX8T01 Stripped
Down to 5 Inputs	
File Name:	MUX5T01.1 REV.
NAME: Rich Reznick	SIZE: B
DATE:	SHEET: of

Figure 34. MUX5T01



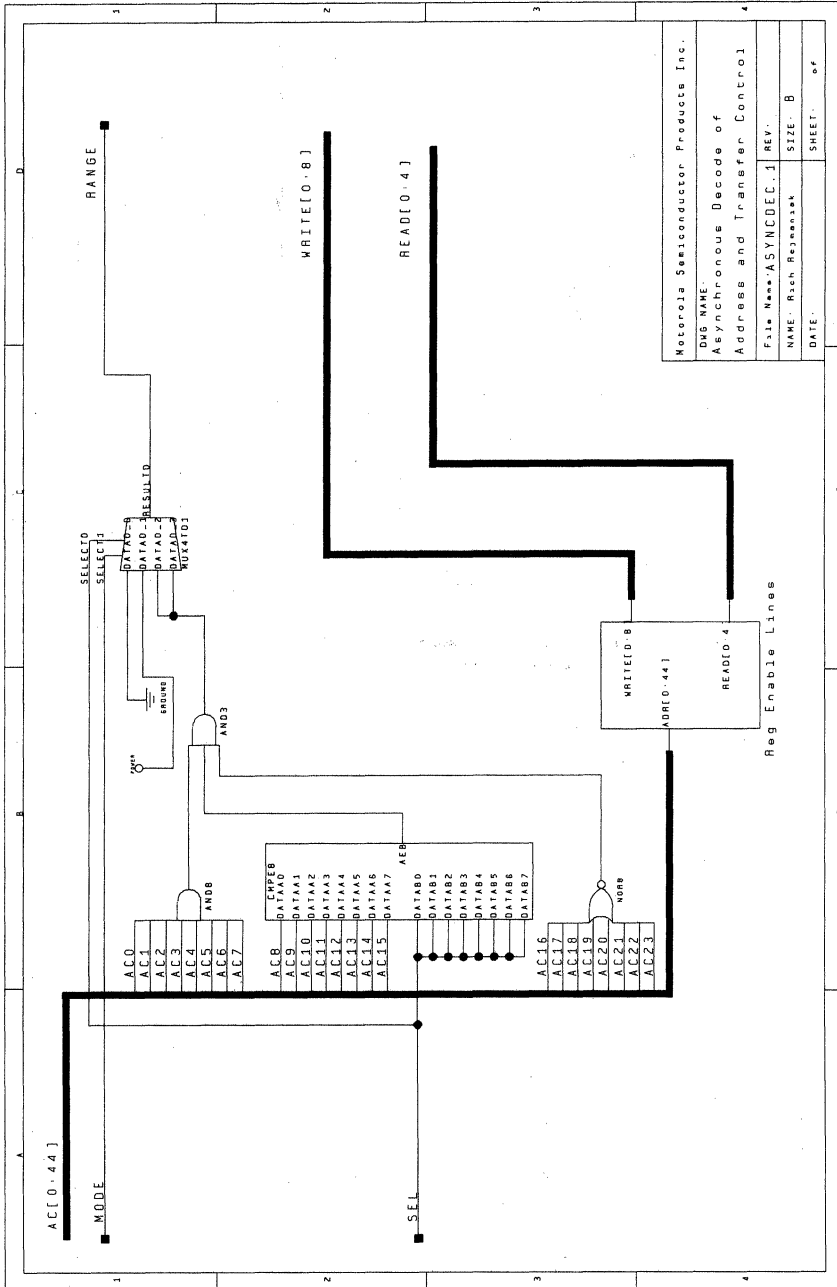


Motorola Semiconductor Products Inc.
 DIB NAME
 Read/Write Control of
 Internal Setup Registers
 and History Buffers
 File Name: BUSSLAVE.1 REV.
 NAME: Bus Slave
 DATE: _____ SHEET: 8 of 8

Figure 35. BUSSLAVE



4

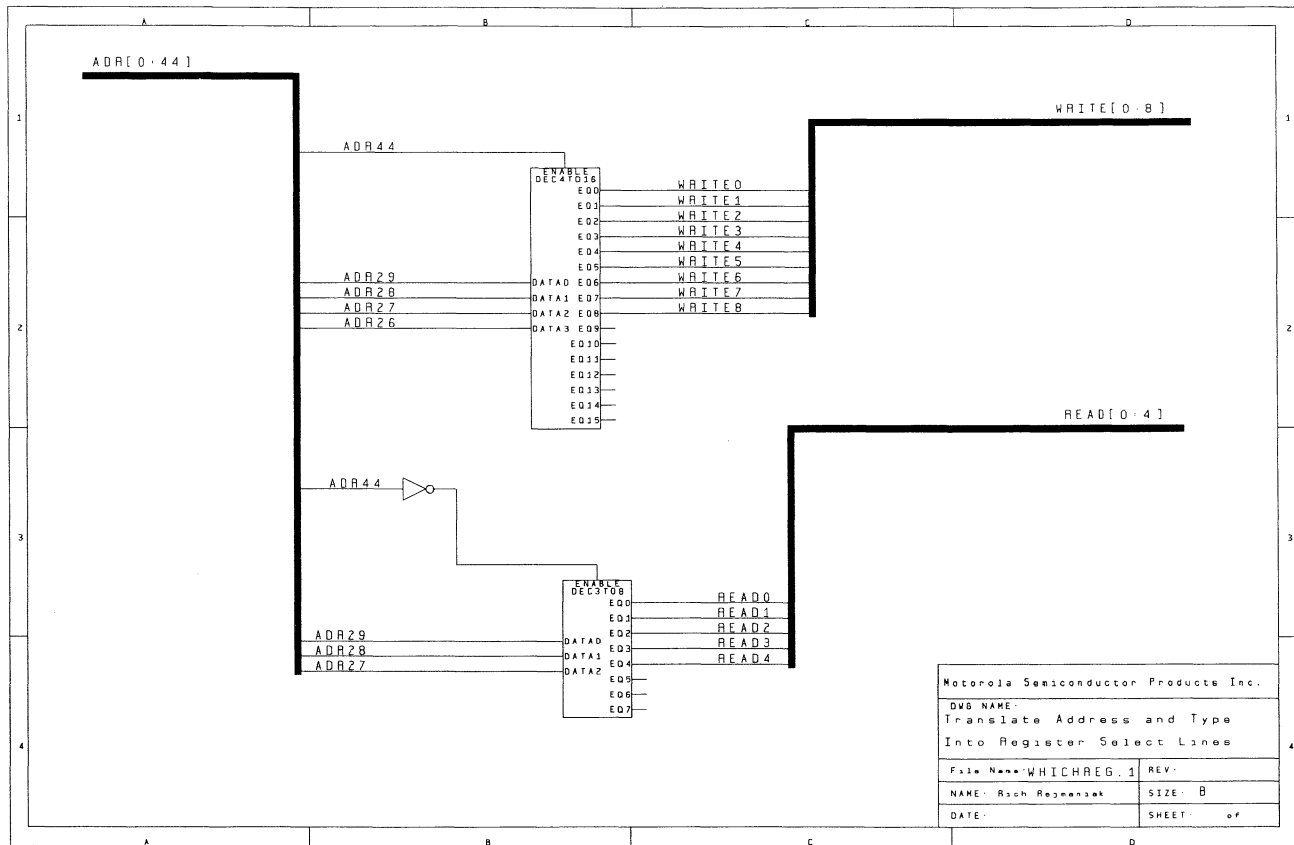


Motorola Semiconductor Products Inc.	
DWG NAME:	Asynchronous Decode of
Address and Transfer Control	
FILE NAME:	ASYNCDEC.1
REV:	1
NAME:	Rsh-Rejmanek
DATE:	
SHEET:	54

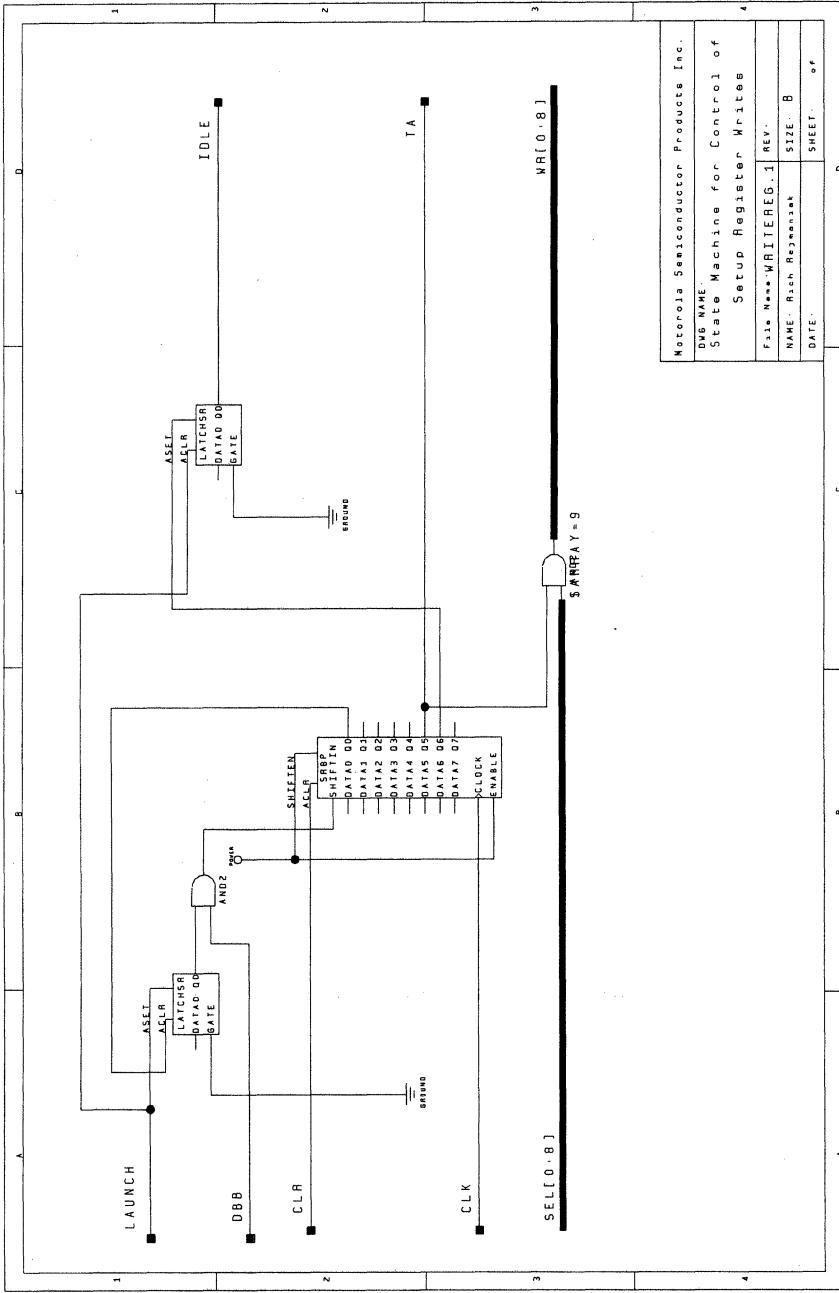
Figure 36. ASYNCDEC



Figure 37. WHICHREG



4



Motorola Semiconductor Products Inc.

DWG NAME: State Machine for Control of Setup Register Writer

File Name: WRITEREG.1 REV. 4

NAME: Bsch Rejznszak	SIZE: B
DATE:	SHEET: of

Figure 38. WRITEREG



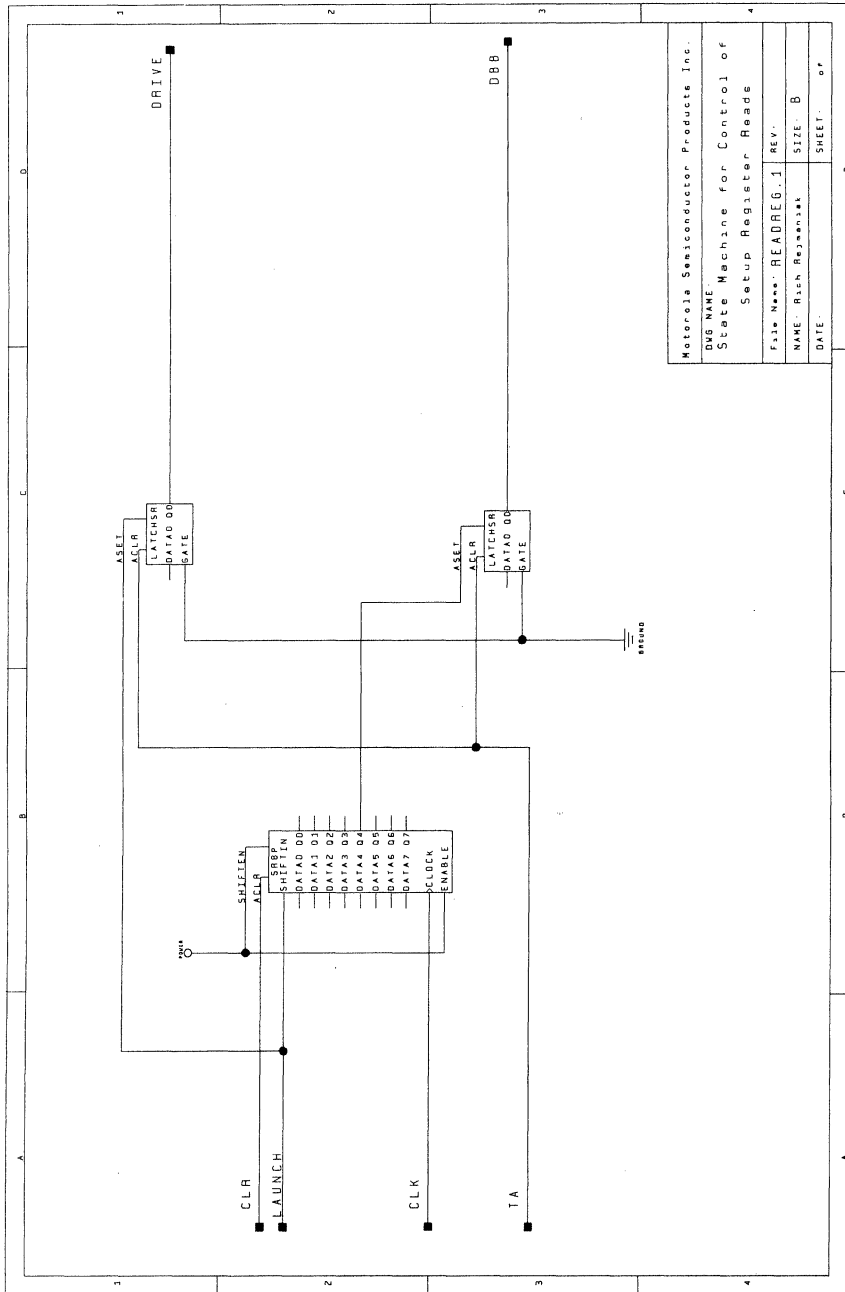
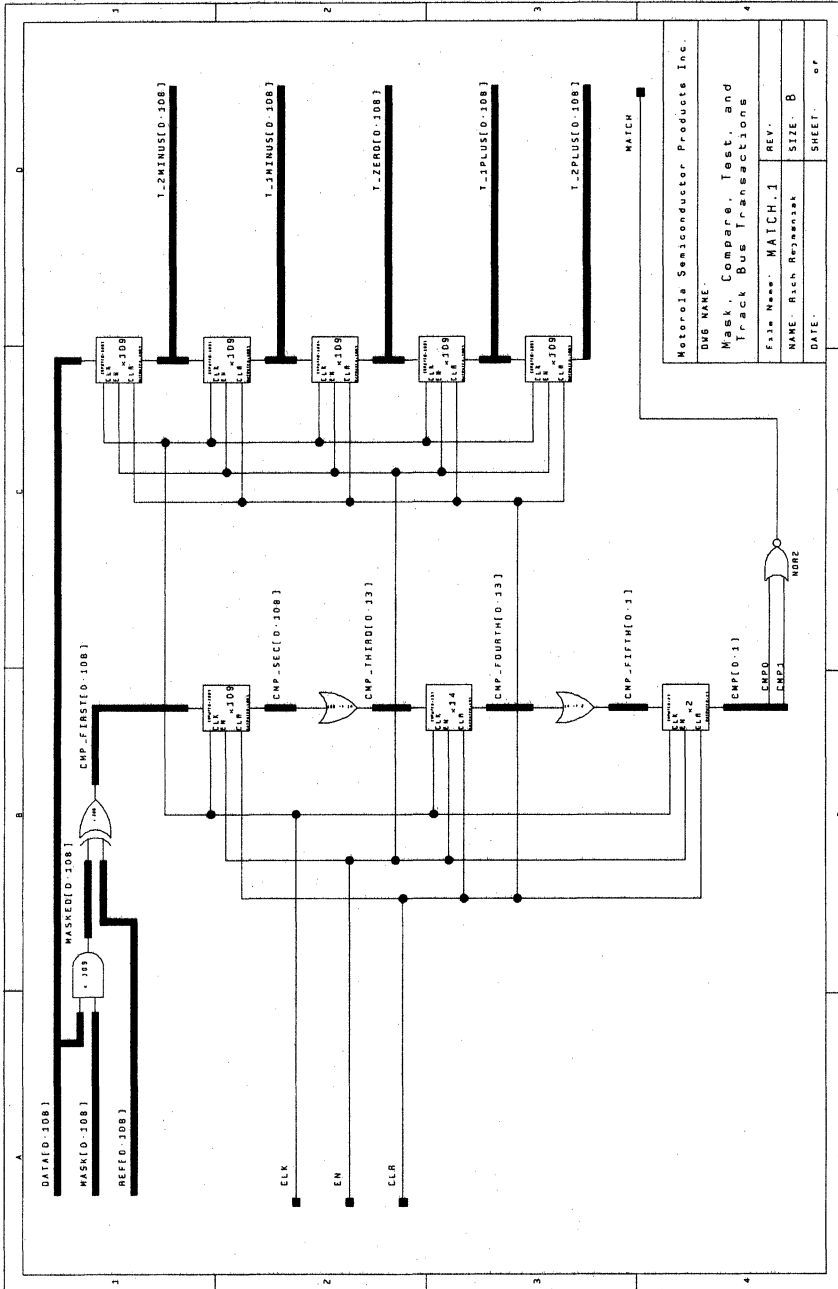


Figure 39. READREG



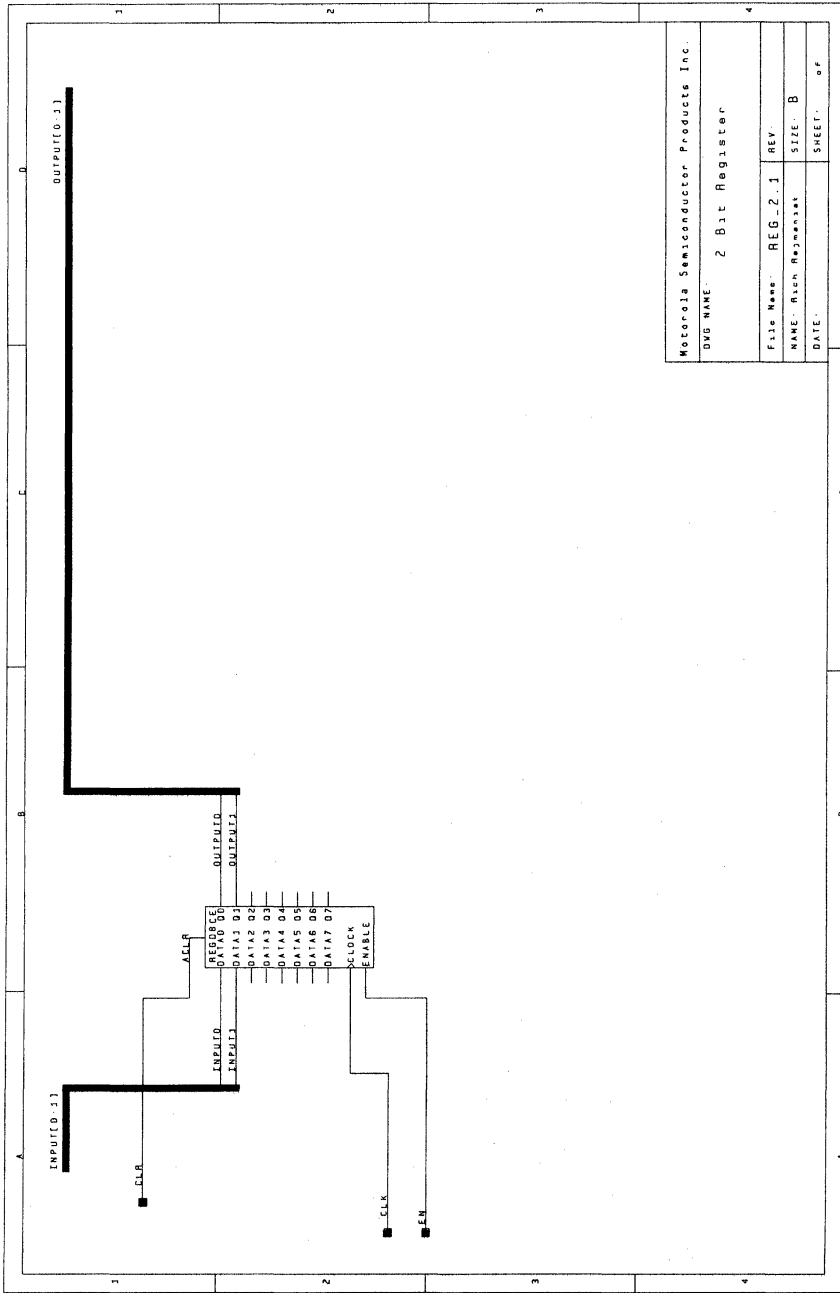
4



Motorola Semiconductor Products Inc.	
DWG NAME:	
Mask, Compare, Test, and Track Bus Transactions	
File Name:	MATCH.1
REV:	REV. B
NAME:	Bus Response
DATE:	
SHEET:	of

Figure 40. MATCH





Motorola Semiconductor Products Inc.			
DWG NAME:	2 Bit Register		
File Name:	REG_2_1	REV:	
NAME - Rich Rajmanna		SIZE:	B
DATE:		SHEET:	of

Figure 41. REG_2



4

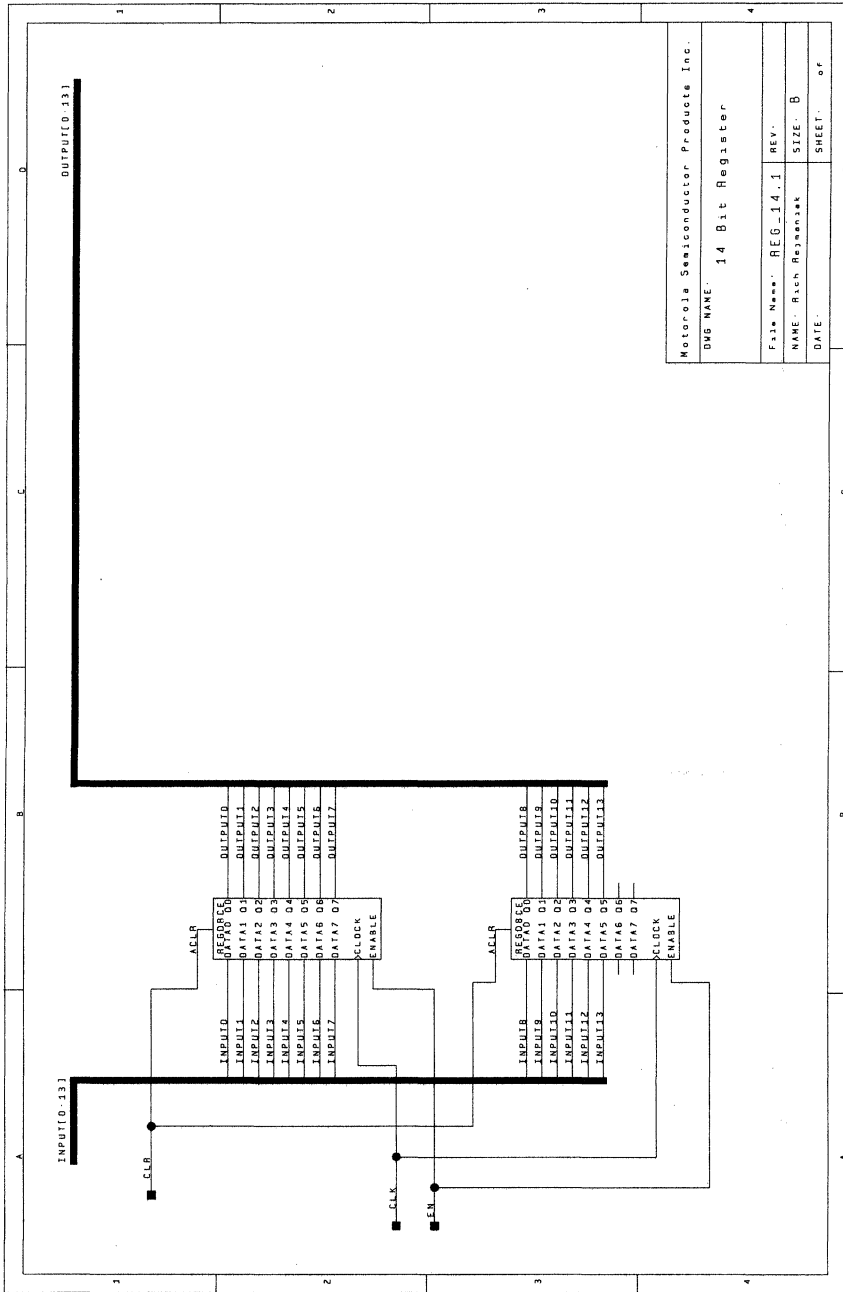
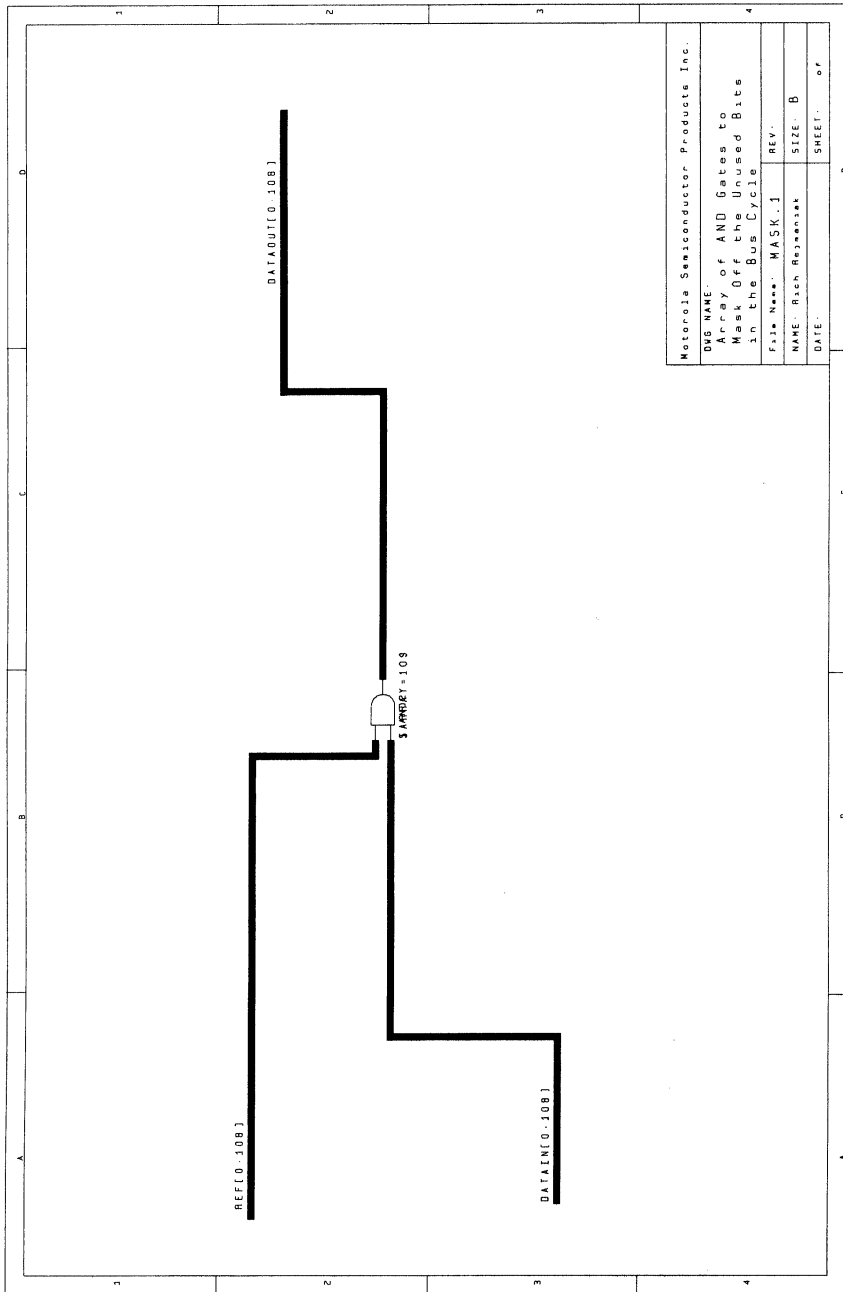


Figure 42. REG_14





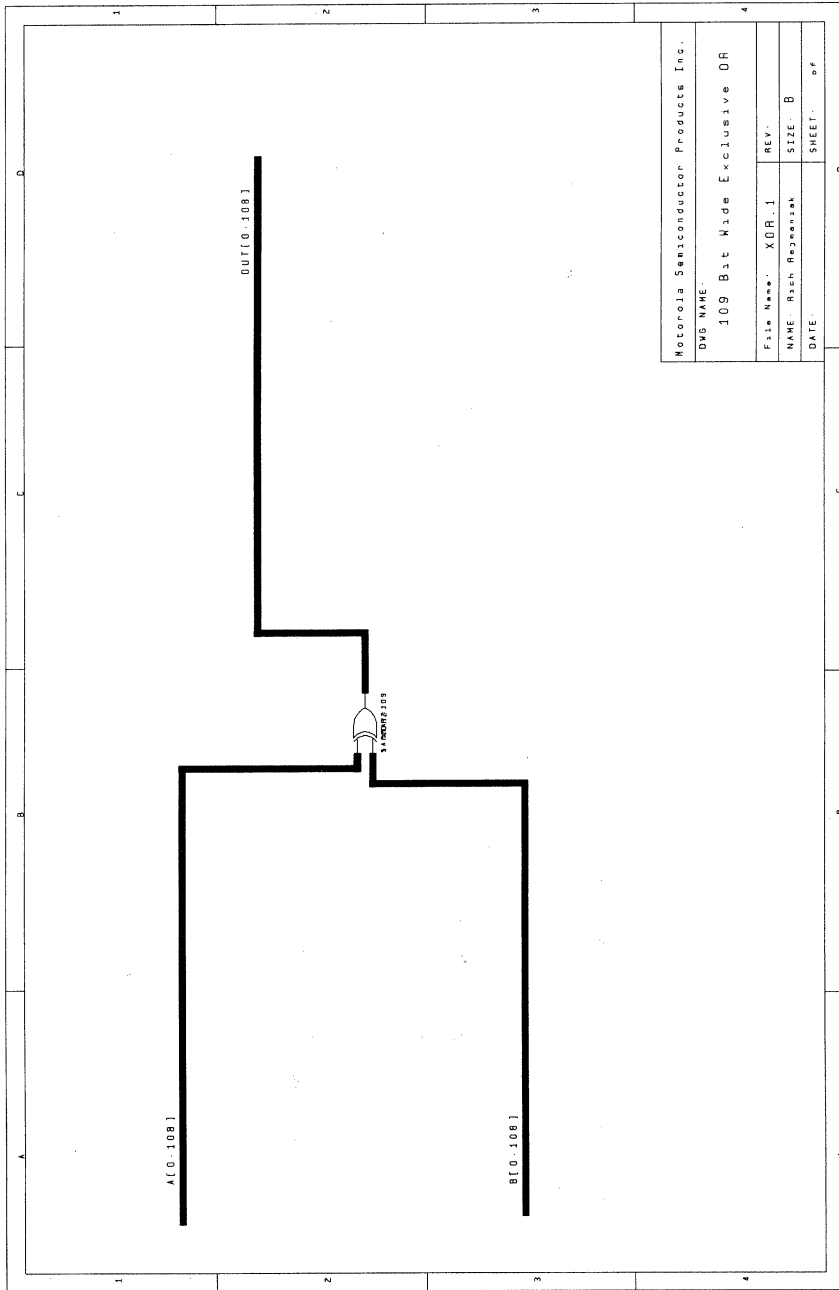
Motorola Semiconductor Products Inc.			
DWS NAME: Array of AND Gates to Mask Off the Unused Bits in the Bus Cycle			
File Name: MASK.1		REV:	
NAME: Rich Rejznszak	SIZE: B	SHEET: of	
DATE:			

4

Figure 43. MASK



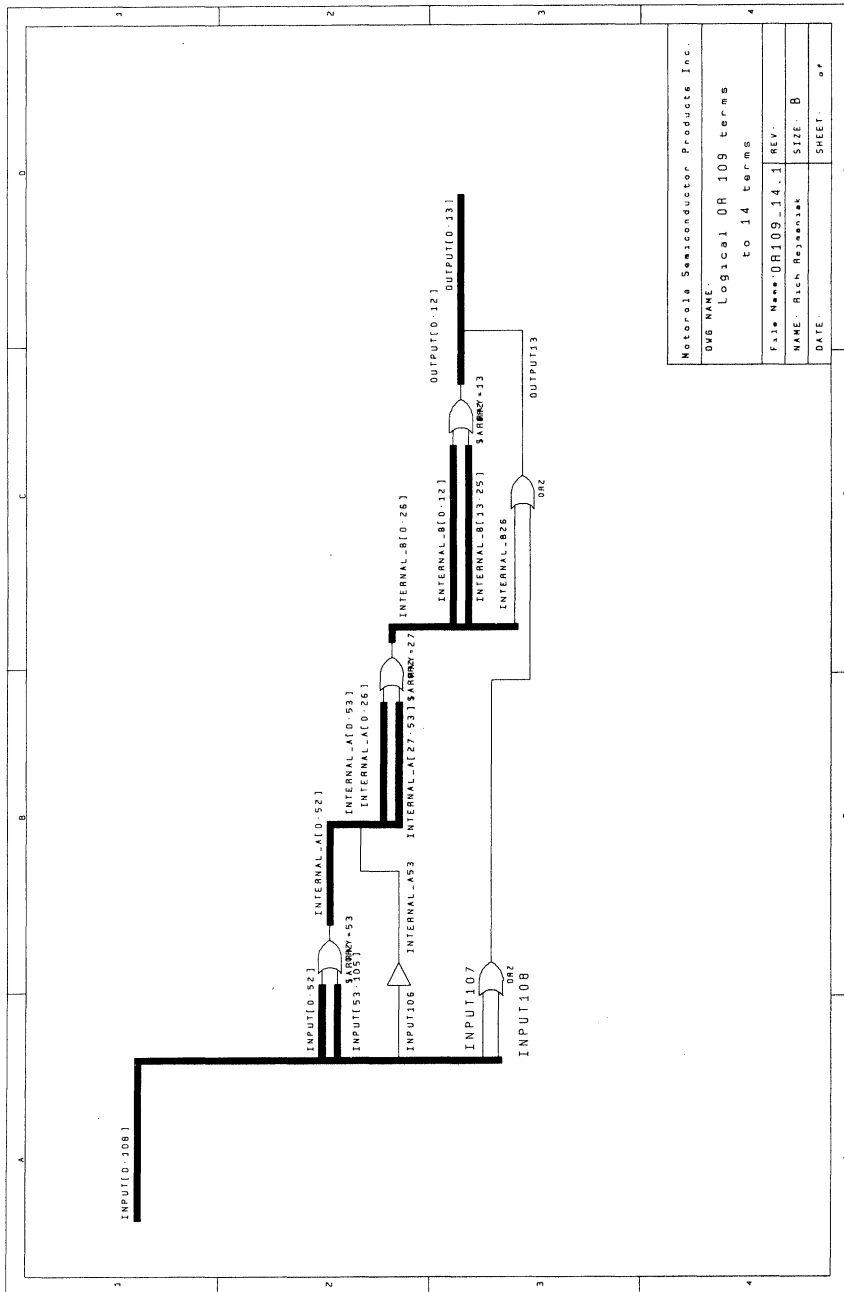
4



Motorola Semiconductor Products Inc.	
DWG NAME:	109 Bit Wide Exclusive OR
FILE NAME:	XOR . I
REV:	
NAME:	Rich Rosenbark
SIZE:	B
DATE:	
SHEET:	of

Figure 44. XOR





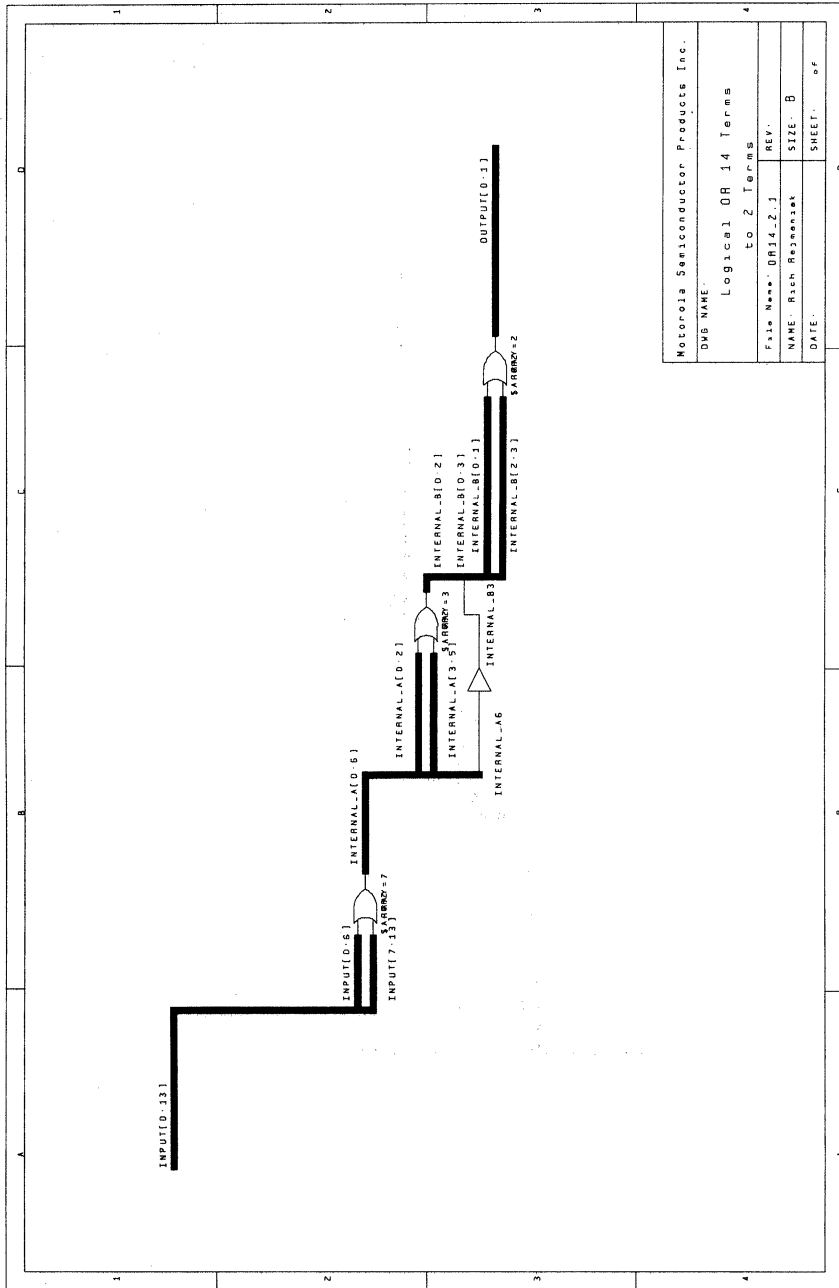
Motorola Semiconductor Products Inc.	
DSG NAME	Logical OR 109 terms to 14 terms
File Name	OR109_14.1
NAME	Rich Rejzba
DATE	REV. B
SHEET	of 1

4

Figure 45. OR109_14



4



Motorola Semiconductor Products Inc.	
DWG NAME:	
Logical OR 14 Terms	
to 2 Terms	
File Name: OR14_2.1	REV:
NAME: Rich Resnick	SIZE: B
DATE:	SHEET: of

Figure 46. OR14_2



Using VIEWlogic's PROSeries 6.1 with the MPA Design System

Prepared by
Douglas M. Shade
Motorola Programmable Logic Products

4



Using VIEWlogic's PROSeries 6.1 with the MPA Design System

Introduction

The Motorola family of MPA devices and supporting software provides system designers with a collection of flexible and powerful tools. This application note focuses on the use of VIEWlogic's PROSeries 6.1 schematic capture and simulation programs as front end design tools for the MPA Design System FPGA place and route software. A basic design flow is introduced followed by a more in depth discussion of parameters for place and route and concludes with a discussion on back annotation and simulation procedures.

Basic Design Flow

In this simplest example of using PROSeries, a straight path is taken from design entry through export to the MPA Design System. More detailed discussions on: place and route parameters, I/O parameters, hardware dependent macros, back annotation and simulation are deferred. The reader is assumed to be familiar with PROSeries and have casual knowledge of the MPA Design System.

Libraries

The EDIF net list reader of the MPA Design System is currently constrained to understand only those components passed to it from the MACROLIB, MICROLIB and IOLIB libraries provided. Only a very few other symbols from the BUILTIN library may be used directly in the schematic. These are: IN, OUT and BI; their usage is explained more fully later. Your VIEWDRAW.INI file must contain lines similar to the following in order to steer PROSeries in the correct direction when adding components to your schematic.

```
DIR [rm] C:\mpa\wvlibs\mpalib\macrolib (macrolib)
DIR [rm] C:\mpa\wvlibs\mpalib\microlib (microlib)
DIR [rm] C:\mpa\wvlibs\mpalib\iolib (iolib)
DIR [rm] C:\mpa\wvlibs\mpalib\builtin (builtin)
```

When adding components to your FPGA schematic, be sure to use only components from these first 3 libraries and only the special hierarchical connectors IN, OUT and BI from the BUILTIN.

Capture

There are just a few unique steps to take during schematic capture to ensure a valid MPA Design System EDIF netlist. The netlist importer of the MPA Design System needs to recognize your design's I/O pins. To accomplish this, you may either create a top level symbol for your completed schematic or you may opt to include VIEWlogic's hierarchical connectors.

If you are going to instantiate your completed FPGA schematic into a larger board or system level schematic then generating a top level symbol is the more appropriate method to use. In order to do this, each of the IOLIB components used must have a named net stub attached to their 'external world' pins. Once this task is completed all that is left is to create a VIEWlogic symbol for the entire FPGA schematic. Each of the pin names on the symbol must match the net-stub names exactly. A pin is required for every I/O net-stub.

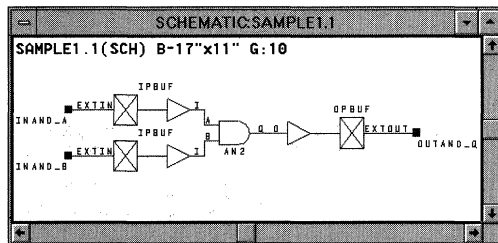


Figure 1. Top Level Schematic, Named Net Stubs

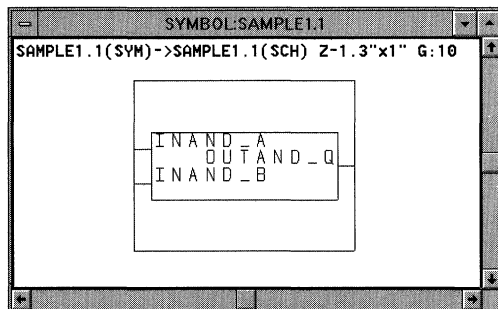


Figure 2. Top Level Schematic's Symbol, Pin Names = Net Stub Names

If on the other hand the schematic you are creating is stand alone for the FPGA, then a short cut method is available to you. As before, place the desired IOLIB components on your schematic. Then from the BUILTIN library select the IN, OUT or BI hierarchical connector as appropriate. Connect this hierarchical connector with to the IOLIB component's 'external world' pin and name the net.

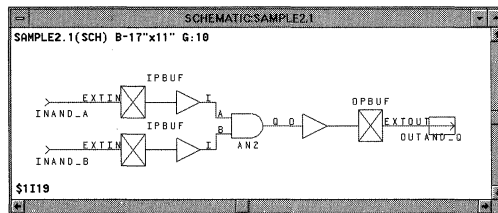


Figure 3. Top Level Schematic Using IN and OUT Symbols from BUILTIN. The OUT Symbol Is Highlighted (Boxed), Instance \$I119.

This name may now be referenced for stimulus/response in the VIEWlogic simulator. Additionally, this net name is passed in the exported EDIF netlist to the MPA Design System place and route tool.



Net List Export

EDIFNETO is the VIEWlogic tool that translates your completed schematic into an EDIF file importable to the MPA Design System. While the PROSeries netlisting tool is generally available from the pull down menus, some of the requisite features of the tool for MPA compatible netlisting are not. So from a DOS session in your current VIEWlogic design directory, run EDIFNETO. The following log shows the correct answers to EDIFNETO's questions. You need to generate a "flattened" netlist, at the level "micro".

```

MS-DOS Prompt
C:\PROSER61\MYDESIGN>edifneto
EDIF U2.0.R. WIRELISTER - U4.1.2; Workview 4.1.2 032992, 3000 Series
© Copyright 1985,1992 by Viewlogic Systems, Inc.
Project Name: sample2
Do you want a flattened netlist? [N] : y
Do you want to evaluate parameterized attributes? [N] :
Level Name: micro
Output File Name [C:\PROSER61\MYDESIGN\SAMPLE2.EDN]:
Button string: Doug Shade
Port/supply type configuration file name [EDIFPTVP.CFG]:
Level string: MICRO
(1 Comp - 3 Nets) Reading complete.
Using EDIFPTVP.CFG for signal name, port/supply type name pairs.
Updating global interface nets.
Update complete.
Processing complete.
0 errors and 0 warnings found.
2 module(s) written to C:\PROSER61\MYDESIGN\SAMPLE2.EDN
C:\PROSER61\MYDESIGN>

```

Figure 4. EDIFNETO DOS Window Session

You now have a .EDN EDIF netlist ready for import to the MPA Design System. Give it a try.

Attributes

The MPA Design System's import process can accept a set of attributes to help the designer tune the layout and routing processes. The system also accepts I/O parameters to specify CMOS/TTL compatibility, I/O drive, package pin assignment and slew rate control. Declaring attributes in the schematic will result in their being passed into the EDIF netlist and then imported into the MPA Design System. Optionally the designer may prefer to include attributes in an external .PAT file of the same name as the design. The designer may choose to use the combination of the two methods, but it should be noted that attributes passed into the MPA Design System in .PAT files will always take precedence if declared in both places.

Place and Route Layout Attributes

The MPA Design System enables the tuning of place and route algorithms in three ways. The first is with the adjustment of the Auto Layout tool options such as annealing temperature, target delays, target zone utilization etc. Additional details on the available options and their use are available in the on-line help facility of the MPA Design System. The second method involves the construction of separate clock files. Here again, additional information is provided in the on-line help system and is not presented in this application note. The third method of influencing place and route results is the inclusion of the following attributes in the schematic, or in an external .PAT file.

Table 1. Valid Attributes

Sch. Component	Place and Route Attributes	I/O Attributes
Net	DPLD_IGNORE_TIMING	
	DPLD_CLUSTER_SEED	
I/O Symbol (instance)	DPLD_IGNORE_TIMING	PULLUP or PULLDOWN
	DPLD_PAD_PLACE	DPLD_OPDRIVE
or Formal Port		DPLD_OPLEVEL
		DPLD OPSLEW
		DPLD_IPLEVEL
		DPLD_PAD_PROPERTIES

DPLD_IGNORE_TIMING

The DPLD_IGNORE_TIMING attribute is used to inform the tool which nets to ignore timing on. It may be set on a symbol (instance), a net or an external pin (formal port). If a net has the attribute set, then all delay paths associated with that net are ignored. If an instance has the attribute set, then all input delay paths driving and all delay paths being driven from that instance are ignored. Assigning the attribute to a formal port has exactly the same effects as assigning it to the I/O instance itself. Once all of the objects to be ignored have been identified, their paths are propagated forward and backward through combinatorial gates until clocked objects (or top level circuit I/O) are reached. The result is that additional segments other than those explicitly specified may be ignored for timing purposes as well.

You are required to use a dummy value with this attribute, but the value stated is otherwise ignored.

Assigning this attribute to a symbol, net or formal port frees the timing driven auto-layout algorithms to more optimally cluster, place and route the speed critical nets.

DPLD_CLUSTER_SEED

The DPLD_CLUSTER_SEED attribute is used to assign a cluster seed to a net. This will cause the clustering to treat all instances that connect to that net specially. The action taken depends on the value of the attribute, as follows:

- 0 ignore this net during clustering. Setting this attribute on a net is likely to cause the net to be implemented in global interconnect.
- 1 default operation
- <1000 weight this net by the given factor in the clustering

DPLD_PLACE_PRIORITY

The DPLD_PLACE_PRIORITY attribute can be applied to a net to force the software to lay out that net in a physically smaller area – in other words, to place the instances connected to that net closer together. The value of DPLD_PLACE_PRIORITY should be an integer in the range 1 to 10 (1 is the default). Higher values of place priority let you prioritize nets relative to each other.

DPLD_PAD_PLACE

DPLD_PAD_PLACE – instructs the I/O pad to be allocated to the package pin number specified. Only one pad may be

4



allocated to any pin. Automatic placement of I/O pads usually results in a better layout, so this attribute should only be added when it is necessary. Example: `DPLD_PAD_PLACE=C2`

I/O Parameter Attributes

DPLD_PUP

DPLD_PUP – attaches to WPUP primitive cells only, to select either or both of the pull-up resistors available in a WPUP cell. Valid values are 1, 2 or BOTH.

PULLUP or PULLDOWN

PULLUP – set this to 1 if you want to enable the pull-up resistor on the external pin of an I/O pad. Default is 0 (resistor disabled). Example: `PULLUP = 0`

PULLDOWN – set this to 1 if you want to enable the pull-down resistor on the external pin of an I/O pad. Default is 0 (resistor disabled). Example: `PULLDOWN = 0`

DPLD_OPDRIVE

DPLD_OPDRIVE – sets the output drive current of an output or bi-directional pad to either 6ma (default) or 12ma.

DPLD_OPLEVEL

DPLD_OPLEVEL – sets the output voltage level of an output or bi-directional pad to either 3v or 5v (default).

DPLD_OPSLEW

DPLD_OPSLEW – sets the output slew rate (transition speed) of an output or bi-directional pad to high (default) or low.

DPLD_IPLEVEL

DPLD_IPLEVEL – sets the input threshold voltage of an input or bi-directional pad to either CMOS or TTL (default).

DPLD_PAD_PROPERTIES

VIEWlogic permits the use of the combined attribute, `DPLD_PAD_PROPERTIES` which combines the following

attributes into a single comma separated list: `PULLUP, PULLDOWN, DPLD_IPLEVEL, DPLD_OPLEVEL, DPLD_OPDRIVE, DPLD_OPSLEW`. This is especially useful when defining a block of I/O pins in an external attribute file. When prompted, answer NO to the “Expand Label?” prompt in VIEWlogic. Example: `DPLD_PAD_PROPERTIES = 0,0,CMOS,5v,12ma,high`

Defaults and Invalid Combinations

The default I/O pad attributes have been selected so that they can connect to either TTL or 5v CMOS without adjustment. The default parameters may not be ideal for every design, and they should be matched to the application in order to achieve the best performance and noise immunity. 3v CMOS users should be especially careful to set `DPLD_OPLEVEL` to 3v, otherwise damage to peripheral IC’s may result.

The following combinations of user attributes are not permitted:

`DPLD_OPDRIVE = 6ma AND DPLD_OPSLEW = low.`
`PULLUP = 1 AND PULLDOWN = 1`

The following combination of user attributes on a single bi-directional pad should be avoided, as it may produce unpredictable results:

`DPLD_IPLEVEL = CMOS AND DPLD_OPLEVEL = 3v`

Assigning Attributes in a Schematic

Any of the attributes listed above can be assigned to pins, nets and macro symbols as appropriate. In PROSeries, the method of assigning attributes is straight forward. Select the desired net or symbol (its color will change or a bounding box will appear respectively, identifying it as being the currently selected object) then from the “Add” pull down menu, select “Object Attribute”, the bottom of the screen (the text input area) will then prompt you with “Attribute Text String”. Type in the attribute and the value (if appropriate) and hit enter. The attributes will then be included in the EDIF netlist once EDIFNETO is run.

4

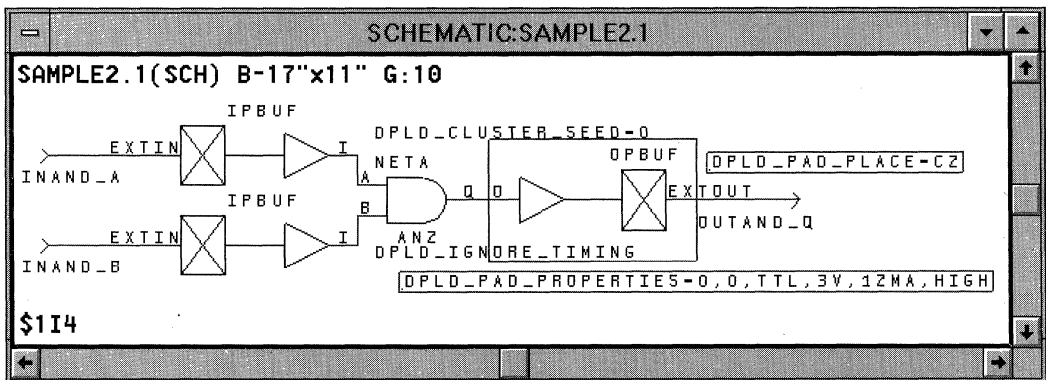


Figure 5. The Net “NETA” Attributed With `DPLD_CLUSTER_SEED=0`. The “B” Input Pin of AN2 Attributed With `DPLD_IGNORE_TIMING`. The Selected Component “OPBUF \$1I4” and its Attached Attributes are Boxed.



Selecting a component pin is just a bit trickier. Left click on the desired component (in this example AN2) then right click on the desired input pin. “Add” – “Object Attribute” as described above.

Assigning Attributes & Instances in an External .PAT File

Assigning attributes to a long series of pins, or a variety of nets in the above manner can be time consuming and may be error prone. The MPA Design System gives the designer the option to enter all the valid attributes in an external .PAT file. Entries in the .PAT file take precedence over any attributes that may have also been instantiated in the EDIF netlist via schematic entry. The .PAT file must have the same name as the EDIF netlist .EDN file, and must reside in the same directory.

The external attributes file supports three main operations:

- 1) Insertion of attributes to specify pin placements and pad characteristics.
- 2) Insertion of special pad cells, IPCLK/IPRST, to drive the primary clock/reset network.
- 3) Insertion of special buffer primitives into a named net.

This has two uses:

- a) Force a named net onto/off the peripheral bus, by inserting the primitives APBUF/PABUF respectively.
- b) Force a named net onto the primary clock/reset network, by inserting the primitives ACLK/ARST respectively.

The external attributes file must exist in the same directory and with the same name as the EDIF netlist, with the file extension .PAT During import, the MPA Design System automatically checks for the existence of a .PAT file and uses it when one is found.

Syntax of the External Attributes (.PAT) File

The external attributes file contains a list of commands, one per line. Each command contains up to five fields, as follows:

```
<object-class> <object-name> <operation> <name> [<value>]
```

where:

<object-class>	is one of port, net, or instance.
<object-name>	is the netlist name of the object (port, net or instance) being operated on.
<operation>	is one of attribute or instance.
<name>	is the name of a definition or an attribute.
<value>	is only used in attribute operations, and is the value to be given to the attribute. This field is only required when an attribute requires a value.

```
// This is a comment
# This is a comment as well
```

```
port sega attribute dpld_pad_place 22
//This results in the IPBUF being placed on the pad associated with package
//pin 22.
```

The following are specific syntax forms of all valid attribute or instance assignments.

port <name> attribute <name> <value>

Attribute <name> with (optional) value <value> is added to the port instance (the instance driven by the formal port). Only works with input and ports.

port <name> instance <name>

The port instance (the instance driven by the formal port) is replaced by an instance of the given definition. The only valid definitions are IPCLK, IPRST. This syntax is limited input ports with 1 input pin and 1 output pin (IPBUF for example).

net <name> attribute <name> <value>

Attribute <name> with (optional) value <value> is added to the net.

net <name> instance <name>

Creates an instance of the given definition and inserts it into the named net. The only valid definitions are ACLK, ARST, APBUF and PABUF.

instance <name> attribute <name> <value>

Attribute <name> with (optional) value <value> is added to the instance

4

Example .PAT File Entries

The following sample .PAT file entries reference the simple schematic shown in Figure 6.

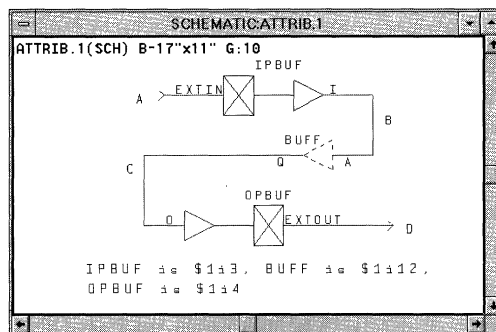


Figure 6. The .PAT File Attributes Are Added to This Simple Schematic. Nets Are Named SEGA, SEGB, SEGC and SEGD.



```

port sega attribute dpld_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the net segment associated with the formal port sega. A dummy
//argument is required for this attribute.

port sega instance ipclk
//This results in the IPBUF being replaced by an IPCLK, forcing the net onto
//a clock network.

net segb attribute dpld_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the entire net "segb". A dummy argument is required for this
//attribute.

net segb attribute dpld_cluster_seed 1
//The dpld_cluster_seed attribute and value shown get assigned to the net B
//for evaluation during place and route.

net segb attribute dpld_place_priority 1
//The dpld_place_priority attribute and value shown get assigned to the net
//B for evaluation during place and route.

net segb instance aclk
//This results in an ACLK buffer being inserted between IPBUF's output and
//BUF's input. BUF is now driven off the resulting clock routing
//resource.

instance $1I14 attribute dpld_opdrive 12ma
//This results in the OPBUF getting 12ma drive capability.

instance $1I12 attribute dpld_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the all nets associated with the instance $1I12. A dummy
//argument is required for this attribute.

```

4

All Valid Combinations of Attributes & Instances in an External .PAT File

The following show all the valid combinations of the attributes in the .PAT file.

port <name> instance (name here can only refer to input instances with one input pin and one output pin)

ipclk
iprst

port <name> attribute

dpld_ignore_timing dummy_arg
dpld_pad_place <value, see data book for package being used>
pullup 110
pulldown 110
dpld_opdrive 6ma12ma
dpld_oplevel 3v5v
dpld_opslew highlow
dpld_ipevel CMOS/TTL
dpld_pad_properties (see above paragraph describing this attribute)

net <name> attribute

dpld_cluster_seed n (where $0 \leq n \leq 1000$, 1 is default, 0 means ignore net)

dpld_place_priority n (where $0 \leq n \leq 10$, 1 is default)

net <name> instance

aclk
apbuf
arst
pabuf

instance <name> attribute

dpld_ignore_timing dummy_arg
dpld_pad_place <value, see data book for package being used>
pullup 110
pulldown 110
dpld_opdrive 6ma12ma
dpld_oplevel 3v5v
dpld_opslew highlow
dpld_ipevel CMOS/TTL
dpld_pad_properties (see above paragraph describing this attribute)

Library Elements Specific to MPA Hardware Features

Most designs can be fit to the desired speed into the MPA family without the designer needing to know much about the details of the device's internal construction. However, to get the most out of the MPA, you should take some time to browse through the on-line help files thoroughly. The help files are rich



in detail regarding the hardware specific macros and routing resources macros inherent in the MPA design system. The following topics are presented as simple examples on most of these hardware specific features. For an exhaustive presentation, please refer to the on-line help facility.

Logic One and Logic Zero

It may at times be necessary to modify the functionality of a macro from the supplied libraries by tying one or more of its inputs to logic high or low. There are two special elements provided in MICROLIB for this purpose. ONE produces a logic high to all input pins tied to it; there are no fan-out restrictions for the signal. The ZERO element provides a logic low. An alternate method is to tie the input pins requiring a logic high to a net named VDD, and tie the output pins requiring a logic low to a net named GND.

For all the above methods, the MPA Design System will recognize the logic as static and eliminate superfluous logic elements wherever possible during the place and route process.

Wired-OR

The MPA family allows for connecting many outputs to a single common signal line. The available macros for this function are: WND2, WINV, WOR2, WBUF. When using these macros, a WPUP component is required. The maximum number of connections to a single signal is given by

$$\frac{1}{2}\sqrt{\# \text{ of core cells.}}$$

Table 2. Maximum Drivers on a Wired-OR Signal

MPA Family Member	Max Drivers
MPA1016	20
MPA1036	30
MPA1064	40
MPA1100	50

If the number of drivers on the Wired-OR net is below half of the maximum allowed, then only a single pull-up resistor is recommended. Adding a second pull-up resistor to a very large net, will help speed things up some, but at the cost of increased power consumption. The allowable values for the DPLD_PUP attribute are 1, 2 or BOTH. Resistors 1 & 2 are of equal value, so it makes no difference which one you select. BOTH ties resistors 1 & 2 in parallel and decreases the low to high transition time, but at the expense of extra power consumption.

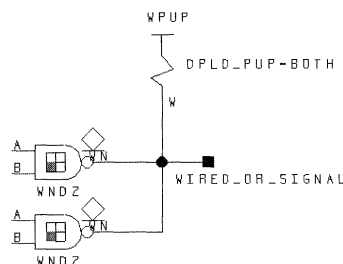


Figure 7. The Diamond Symbol Reminds the User That These Outputs Cannot Source a Logic HIGH, but Are Only Able to Pull the Output to a Logic LOW.

Adding additional wired-or outputs to the net WIRED_OR_SIGNAL will slow it down. Adding additional inputs to the net has little effect on speed. Low to high transitions are typically slower than high to low transitions on a wired-or signal.

Peripheral Bus

The periphery of the MPA die is bordered with an 8 bit wide Peripheral Bus (P-Bus). The P-Bus can be broken at each corner of the array by switches. (Setting of the switches is handled automatically in the MPA Design System software.) Each of the resulting 4 segments has two programmable pull up circuits, one at each end. The P-Bus is ideal for routing common signals to many I/O macros.

It is easy to build a scenario where many I/O pins have a single or several control signals in common. In this instance you would want to place that signal on the P-Bus using a APBUF. Conversely, pulling a signal off the P-Bus back into the array is accomplished using the PABUF.

The ability to construct Wired-OR nets is not limited to signals internal to the array. P-Bus Wired-OR nets can be constructed using APWBUF (or APWINV) and their associated pull-up structure WPUP. However, using these features requires the user to be aware of the natural consequences of adding capacitance and pull-ups to a resistive bus. Adding additional segments of P-Bus (by assigning I/Os to different edges of the die) increases capacitance that effects rise and especially fall times. Also, the somewhat resistive nature of the P-Bus can cause Vol noise margin problems if the active low P-Bus driver is far from the pull-up element and driven element.

The assignment of P-Bus pull-up resistors is automatic. On import, all WPUP instances defined by the designer are removed from the netlist. The tool automatically balances the number of pull-up resistors against the P-Bus loading (incurred by the use of multiple die edges) by automatically inserting additional pull-up capacity for each P-Bus segment used. During autolayout, the MPA Design System re-attaches a single peripheral bus pull-up resistor per occupied die edge, for each unique P-Bus signal. For example, if several I/Os that use a P-Bus Wired-OR signal get split between 'top' and 'right' edges of the die during autoplacement, the tool would assign two pull-up resistors to the net.

4



Low Skew, Clock and Reset Nets

For a high fan out signal or for a signal where you would like to keep the skew limited to less than 1nS, you should consider moving the signal onto a clock network using the ACLK or ARST macro (they both do the same thing).

MPA I/O Structures

The standard I/O cell of the MPA array is very feature rich. The complexity of this structured is apparent in the large number of choices available in the I/O macro library IOLIB. (Here again, the reader is encouraged to invest some time in the on-line help facility, in particular "Help on Libraries – Input and Output Pads" and "Help on Device – Functional Description – I/O Cell". Defining a bookmark for these two particularly useful help sections will provide a short cut for referencing them in the future.)

Each of the macros in the IOLIB fit in a single I/O cell. Couple these relatively complex functions with the availability of P-Bus routing resources (including the P-Bus Wired-OR resources previously mentioned) and it becomes clear that significant functionality can be achieved before ever using the normal internal resources of the array.

In Figure 14, a three bit address decoder was implemented using only the resources available in the complex I/O Cells and the associated P-Bus. No internal logic or routing resources were consumed. The features used unique to the I/O Cell and P-Bus include: input delay to synchronize external data with the buffered clock signal, APBUF used to bus a common enable to signal to several I/O sites, XNOR used to compare external and internal address values, and finally the Wired-OR P-Bus line.

Of course, all of this functionality could be moved internal to the array, using only the simple I/O macros.

4

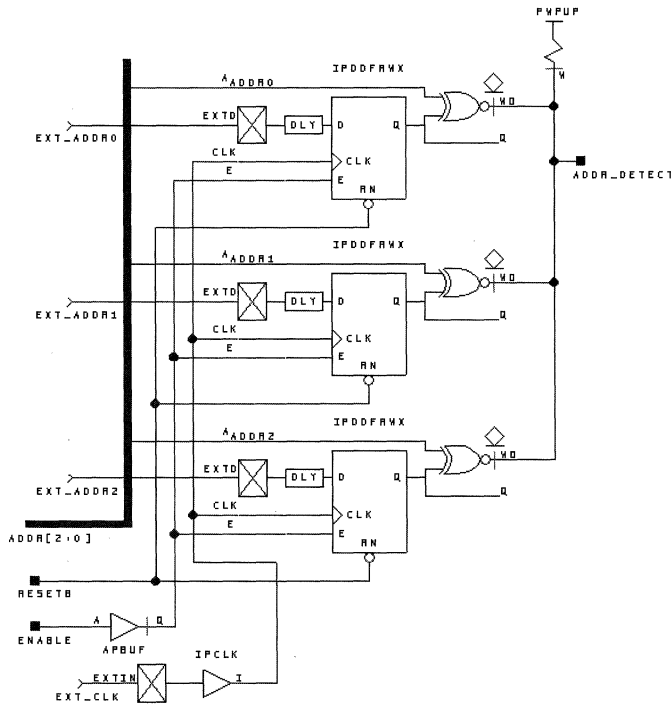


Figure 8. A Three Bit Address Decoder, Only I/O Cell and P-Bus Resources Used

Simple I/O Structures

Your first designs for the MPA will probably only use the above I/O macros, with all the latching, decoding etc. being handled internal to the array. Soon though, you will probably encounter a real world design that constrains you to a specific

clock-to-Q specification or some other requirement that will have you going back to re-examine the components available in the IOLIB. There are many variations of latched, registered, and Wired-OR inputs, outputs and bi-directional macros. Again, please invest some time with the on-line help descriptions of the device and the libraries.



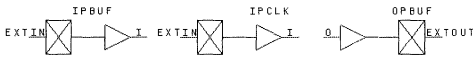


Figure 9. The Three Most Commonly Used I/O Macros

Back Annotation and Simulation

It is assumed that the reader is familiar with the tools and procedures involved in using the VIEWlogic PROSim tools.

The MPA Design System has a facility that provides post place and route back annotation data in a format compatible with the PROSim netlister. Briefly, a VIEWlogic format back annotation data table file (.DTB) can be generated after completion of a place and route. The .DTB file can be read in by VIEWlogic's VSM netlister tool, to provide an accurate simulation netlist of your completed design.

Within the main window of the MPA Design system, select the Tool Options button (or use the pull down menu "Tools – Options"). Select Back Annotation and select "VIEWlogic (.dtb)". Once your design has been imported, placed and routed, the back annotation tool button becomes selectable. Invoke the back annotation tool by pressing its button. A back annotation file will be placed in the same directory as the rest of the MPA Design System's output files. The files name will be the same as your .DSN file (visible in the title bar of the MPA Design System main window). Generally, this file needs to be moved up to your VIEWlogic design directory.

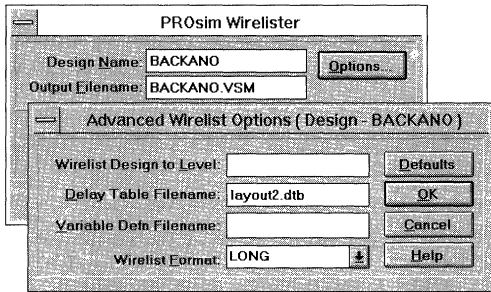


Figure 10. VSM Netlister, and its Options sub-window. "BACKANO" is the design name. The back annotation file name is "layout2.dtb".

Invoking the VSM netlister from the PROCapture is done in the normal way. Using the "Tools – Link to PROSim" pull down menu, select "Options" in the PROSim Wirelister window. Enter the name of the desired back annotation file. (You may have several different valid layouts completed at this point, but you can only merge one back annotation file at a time into the simulation netlist.) In the example shown above, the

schematic's name is "BACKANO" and the post place and route back annotation file is "layout2.dtb". The MPA Design System generated delays will now be included in the simulation net list.

In Figure 12, ANDB goes high at time 45 and ripples through the AND tree (Q1, Q2, Q3) with the final output ANDOUT going high about 14nS later. When ANDB drops, the third AND gate (Q3) is the first to fall because in this implementation it happened to be placed closer to the ANDB input than AND gates Q1 & Q2. With ANDA held high, gate delay is the limiting factor for a rising ANDOUT and path delay is the limiting factor for a falling ANDOUT.

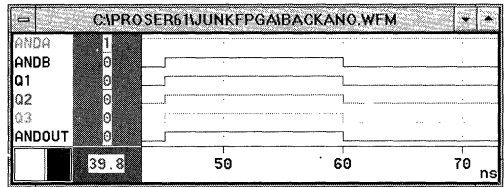


Figure 11. Before back annotation. ANDB going high ripples through all three AND gates and the output instantaneously.

4

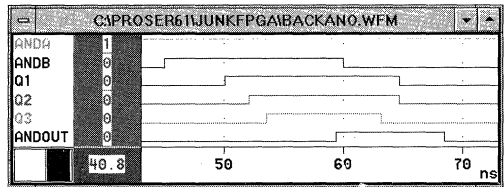


Figure 12. After back annotation of place and route delays. ANDB going high ripples through Q1,2 & 3, but going low happens to take Q3 low first.

Back Annotation and Logic Reduction

The MPA Design System does some "bubble pushing" during the retargeting / fitting of the logical design into the physical implementation. In some cases, redundant gates in the schematic design are eliminated in the physical layout. When this occurs, back annotation proceeds as before with the resulting delays being added only to the input pin of the most downstream logic element not eliminated in the retargeting / fitting process.

In Figure 21, the 4 inverters are eliminated in the retarget / fitting process. The wire delay between the remaining IPBUF and OPBUF gets back annotated to the OPBUF's input pin. The back annotated simulation netlist has 4 zero delay inverters (just as the pre back annotation netlist had) followed by the OPBUF with the real world delay.



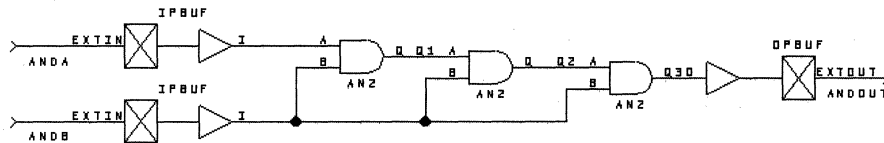


Figure 13. A sample circuit to explore simulation before and after back annotation.

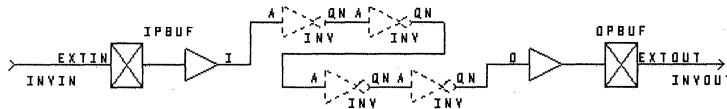


Figure 14. The string of inverters will be removed during place and route.

4



In System Prototyping Using HDLs and FPGAs

Prepared by
Thomas G. Felske
Doug Hergatt
Motorola Programmable Logic Group

4



In System Prototyping Using HDLs and FPGAs

Introduction

This paper describes the rapid prototyping method used in the development of the power control logic for a complex communication system. VHDL and SRAM based FPGAs were used to optimize the HDL code before it was merged into an existing ASIC. This method will prove an effective way to verify the HDL code's functionality while reducing development time.

Application

A large state machine needed to be developed and verified to monitor and control the receiving and transmitting function of a commercial space communication system. The underlying goal of the design was to minimize overall system power consumption by intelligently managing the sequence (state events) of transmit, receive, and power up/down steps inherent to the system. Long system level simulation runs, typically two days per simulation would be required to verify an HDL modification. The use of an FPGA as an in system verification vehicle to model new HDL code gave instant real time feedback of the design performance.

4

Design Environment

To develop the large scale communication system, a high level design environment for signal processing was chosen. The Signal Processing Worksystem (SPW) was used as the behavioral front end model for all logic in the system. The VHDL derived from the SPW tool was then the input to Synopsys for logic synthesis and technology mapping to the

ASIC and FPGA architectures. For the FPGA, the Motorola SRAM based MPA1036 was used. The Motorola FPGA place and route tools provided the bitstream to program an EPROM with the FPGA configuration. System level verification was completed in a test environment as shown in Figure 1, *FPGA Prototyping Application*.

The system evaluation was controlled by the System Test Environment (STE) software that was resident on an HP workstation. The System Test Environment could exercise the desired receive and transmit function, while logic analyzers and current probes monitored the system signals and current levels. With this data displayed graphically, the success of the power control logic was observed immediately.

The initial state machine design and subsequent modifications went through the FPGA Design Flow as shown in Figure 2, *FPGA Design Flow*. The logic synthesis was controlled by user defined scripts and synthesis constraint files to produce an EDIF netlist. The Place and route tools imported the EDIF file and used minimal design constraints with fixed pin placements to generate the FPGA configuration bitstream. Each new bitstream was loaded into a new EPROM that replaced the one on the prototype board. Only the EPROM on the prototype board was replaced with each state machine modification. Retrieving and emulating an earlier design was as easy as replacing the EPROM on the prototype board with an alternate EPROM. In the future, download cable capability from the HP will further reduce the turn around time in configuring the FPGA.

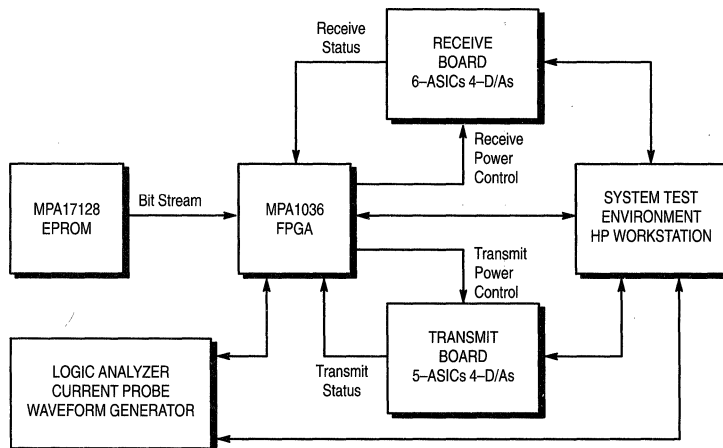


Figure 1. FPGA Prototyping Application



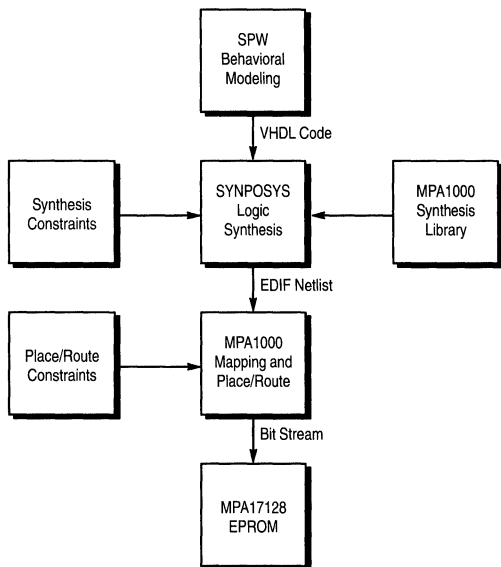


Figure 2. FPGA Design Flow

Summary

The development of the complex state machine that monitors and controls the power of the transmit and receiving modules of the communications system would be extremely time consuming and difficult to verify in a simulation environment. The rapid prototyping and emulation approach described offered a flexible design environment with quick turn-around time. In this case, new RTL code could be synthesized, placed and routed, and a new FPGA configuration bitstream generated in less than two hours. The advantages are the quick turn-around time for design modifications and the real time system emulation and verification. Down loading the FPGA configuration, performing the system emulation, and observing the system signals and current probe readings proved to be the quickest way to evaluate the control logic. The alternative would be to make several system level simulations that take typically two to three days per simulation, then several hours to evaluate the simulation data.

Choosing an SRAM based FPGA proved cost effective by being able to reprogram the existing FPGA. The EPROM that was used to hold the configuration bitstream costs approximately \$7.00. An equivalent antifuse FPGA that would require a new part for every evaluation would cost approximately \$70.00.

For the solution that met the design requirements, the HDL code was then merged into an existing ASIC HDL module without any technology mapping issues.



Tuning the MPA Design System for Speed

Prepared by
Douglas M. Shade
Motorola Programmable Logic Products

4



Tuning the MPA Design System for Speed

Introduction

This application note covers methods for maximizing the clock frequency of a given design using the MPA1000 generation of Motorola MPAs. The discussion is limited to those areas specific to the MPA Design System: MPA library components, pin, net and instance attributes, MPA Design System Tool Options and clock files. Generic fast logic design techniques such as look ahead carry and pipelined logic are not covered in this note. A section which covers the progress monitors of the tool is also included.

Front end techniques discussion is limited to schematic entry however the basic concepts of each of the sections of this application note apply to all design entry methods.

Taking Advantage of the Architecture

A very good investment of your design time is to spend a while reviewing the data book section "MPA1000 Architectural Overview" and the extensive on line documentation included with the MPA Design System. In particular use the pull down menus: "Help > Help on Device", and "Help > Help on Libraries". Read these sections thoroughly.

Clock Resources

Each member of the MPA1000 family has an identical set of clock distribution resources. Two pads on each edge of the die may be dedicated to driving the 8 clock distribution lines. These primary clock distribution lines roughly bisect the die along the horizontal and vertical. From there, the lines branch to form a load balanced distribution comb covering the entire die. Skew is held to within 1nS for the complete clock network.

Using these clock resources to distribute register clock and latch enables is a conventional requirement of most designs. These resources are mentioned in this application note because they may also be used to distribute internal logic (non-clock or reset) signals with high fanout or otherwise tight skew requirements. Speeding up high fanout signals by putting them on one of the 8 available clock networks is accomplished by appending the output of the source driver with an ACLK or ARST buffer.

Similarly, external non-clock and non-reset signals can be distributed throughout the array on the clock network by using the IPCLK or IPRST input buffers at one of the 8 valid pad locations. Using one of these buffers without specifying the pad location will result in the MPA Design System automatically assigning the buffer to one of these 8 valid pad locations.

The MPA Design System treats ACLK and ARST identically. The same holds IPCLK and IPRST. However, each Zone is limited to two unique primary clock and two unique primary reset signals. Since most designs have more clocks than resets, it may be more prudent to use ARST and IPRST macros for routing speed critical high fan out nets.

Taking Advantage of the I/O Cell

The standard I/O cell of the MPA array is very feature rich. The complexity of this structure is apparent in the large number of choices available in the I/O macro library IOLIB. (Here again, the reader is encouraged to invest some time in the on-line help facility, in particular "Help on Libraries > Input and Output Pads" and "Help on Device > Functional Description > I/O Cell". Defining a bookmark for these two particularly useful help sections will provide a short cut for referencing them in the future.)

Each of the macros in the IOLIB fit in a single I/O cell. Couple these relatively complex functions with the availability of P-Bus routing resources and it becomes clear that significant functionality can be achieved before ever having to slow down to use the normal internal resources of the array. Another advantage to using the registers in the I/O cells is guaranteed clock to out timing.

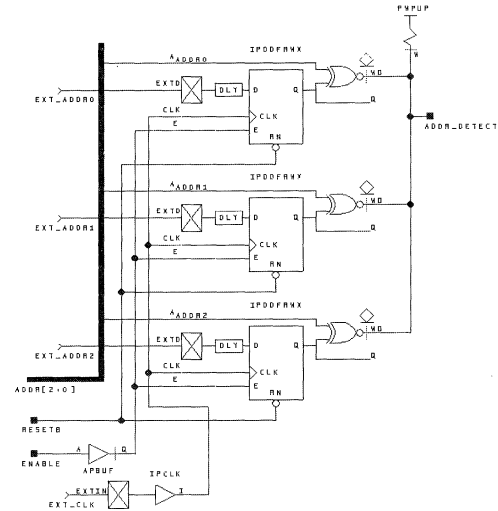


Figure 1. A Three Bit Address Decoder, Only I/O Cell and P-Bus Resources Used

In the example above, a three bit address decoder was implemented using only the resources available in the complex I/O Cells and the associated P-Bus. No internal logic or routing resources were consumed. The features used unique to the I/O Cell and P-Bus include: input delay to synchronize external data with the buffered clock signal, APBUF used to bus a common internal enable to signal to several I/O sites, XNOR used to compare external and internal address values, and finally the Wired-OR P-Bus line. The P-Bus Wired-OR of Figure 1 is shown only for reference. Wired-OR should be avoided on speed critical nets; explained more fully in the next section.

4



Of course, all of this functionality could be moved internal to the array, using only the simple I/O macros, but the design will generally incur a slight speed penalty when doing so.

Wired-OR, Not the best choice

Several elements of the MPA library give you access to the chip's Wired-OR (open drain) structures. While the high to low transition time of such nets is generally acceptable, the low to high 'drive' of such nets is provided only by pull-up resistors. As such, the low to high transition time suffers and so such structures should be avoided on speed critical nets.

The pull-up resistor structure is provided in the WPUP component. The number of parallel pull-up resistors used in the WPUP is under control of the DPLD_PUP attribute. If it becomes necessary to use such a net on a speed critical path, be sure to use the DPLD_PUP attribute set to BOTH.

The available macros for the internal Wired-OR functions are: WND2, WINV, WOR2, WBUF. When using these macros, exactly one WPUP is required per Wired-OR net. The maximum number of drivers of a single Wired-OR signal is given by

$$\frac{1}{2} \sqrt{\# \text{ of core cells.}}$$

4

MPA Family Member	Max Drivers
MPA1016	20
MPA1036	30
MPA1064	40
MPA1100	50

Figure 2. Maximum Drivers on a Wired-OR Signal

If the number of drivers on the Wired-OR net is below half of the maximum allowed, then only a single pull-up resistor is recommended for power conservation. The allowable values for the DPLD_PUP attribute are 1, 2 or BOTH. Resistors 1 & 2 are of equal value, so it makes no difference which one you select. BOTH ties resistors 1 & 2 in parallel and decreases the low to high transition time, but at the expense of extra power consumption.

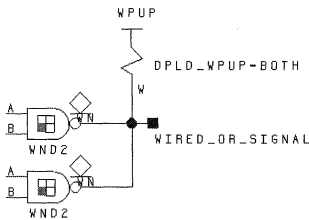


Figure 3. The diamond symbol reminds the user that these outputs can not source a logic high, but are only able to pull the output to logic low.

Adding Wired-OR drivers to the net WIRED_OR_SIGNAL will slow it down. Adding inputs to the net has little effect on speed.

The ability to construct Wired-OR nets is not limited to signals internal to the array. Peripheral Bus (P-Bus) Wired-OR nets can be constructed using APWBUF (or APWINV) and their associated pull-up structure PWPUP. However, using these features requires the user to be aware of the natural consequences of adding capacitance and pull-ups to a resistive bus. Adding additional segments of P-Bus (by assigning I/Os to different edges of the die) increases capacitance that effects fall and especially rise times. Also, the somewhat resistive nature of the P-Bus can cause Vol noise margin problems if the active low P-Bus driver is far from the pull-up element and driven element.

The assignment of P-Bus pull-up resistors is automatic. On import, all PWPUP instances defined by the designer are removed from the netlist. The tool automatically balances the number of pull-up resistors against the P-Bus loading (incurred by the use of multiple die edges) by automatically inserting additional pull-up capacity for each P-Bus segment used. During autolayout, the MPA Design System re-attaches a single peripheral bus pull-up resistor per occupied die edge, for each unique P-Bus signal. For example, if several I/Os that use a P-Bus Wired-OR signal get split between the 'top' and 'right' edges of the die during autoplacement, the tool would assign two pull-up resistors to the net.

The Wired-OR resources are provided to help simplify some logic designs, however, their use should be avoided on speed critical paths.

Guiding Layout with Attributes

The MPA Design System's import process can accept a set of attributes to help the front end designer tune the layout and routing processes. The system also accepts I/O attributes to specify CMOS/TTL compatibility, I/O drive, package pin assignment and slew rate control. Declaring attributes in the schematic will result in their being passed into the EDIF netlist and then imported into the MPA Design System. Optionally the designer may prefer to include attributes in an external .PAT file of the same root file name as the EDIF netlist. The designer may otherwise choose to use the combination of the two methods, but is should be noted that attributes passed into the MPA Design System in .PAT files will always take precedence if declared in both places.

Sch. Comp't	Attached Place and Route Attribute	Attached I/O Attribute
Net	DPLD_IGNORE_TIMING DPLD_CLUSTER_SEED DPLD_PLACE_PRIORITY	
Symbol (instance)	DPLD_IGNORE_TIMING DPLD_PAD_PLACE (I/Os only)	DPLD_PUP PULLUP or PULLDOWN DPLD_OPDRIVE DPLD_OPLEVEL DPLD_OPSLEW DPLD_IPELEVEL DPLD_PAD_PROPERTIES
Formal Port	DPLD_IGNORE_TIMING	

Figure 4. All Valid Attributes. Place and Route Attributes can be used to affect a design speed up.



Place and Route Attributes

Place and Route Attributes can be used to affect a design speed up by providing guidance to the autolayout tool about which are the unimportant nets, and which nets and should be clustered and placed tightly together. Absolute placement of I/Os and relative placement of instances are also used as autolayout guides.

DPLD_IGNORE_TIMING

The DPLD_IGNORE_TIMING attribute is used to inform the tool which nets to ignore timing on. It may be set on a symbol (instance), a net or an external pin (formal port). If a net has the attribute set, then all delay paths associated with that net are ignored. If an instance has the attribute set, then all input delay paths driving and all delay paths being driven from that instance are ignored. Assigning the attribute to a formal port has exactly the same effects as assigning to the I/O instance itself. Once all the objects to be ignored have been identified, their paths are propagated forwards and backwards through combinatorial gates until clocked objects (or top level circuit I/O) are reached. The result is that additional segments other than those explicitly specified may be ignored for timing purposes as well.

You are required to use a dummy value with this attribute, but the value stated is otherwise ignored.

Assigning this attribute to a symbol, net or formal port frees the timing driven autolayout algorithms to more optimally cluster, place and route the speed critical nets.

DPLD_CLUSTER_SEED

Once a netlist import is complete, the first step of autolayout is clustering. During clustering the tool attempts to group related chunks of logic together. This helps simplify the place and route problem by reducing the total number of 'things' the place and route algorithm has to deal with.

The DPLD_CLUSTER_SEED attribute is used to assign a cluster seed to a net. This will cause the clustering to treat all instances that connect to that net specially. The action taken depends on the value of the attribute, as follows:

- 0 ignore this net during clustering. Setting this attribute on a net is likely to cause the net to be implemented in global interconnect.
- 1 default operation
- <1000 weight this net by the given factor in the clustering

Assigning a high value cluster seeds on your most speed critical nets results in a tighter clustering and consequently shorter delays for these nets.

DPLD_PLACE_PRIORITY

The DPLD_PLACE_PRIORITY attribute can be applied to a net to guide the software to lay out that net in a physically smaller area – in other words, to physically place the instances connected to that net closer together. The value of DPLD_PLACE_PRIORITY should be an integer in the range 1 to 10 (1 is the default). Higher values of place priority let you prioritize nets relative to each other.

DPLD_PAD_PLACE

DPLD_PAD_PLACE – instructs the I/O pad to be allocated to the package pin number specified. Only one pad is allocated to any pin. Automatic placement of I/O pads usually results in a better layout, so this attribute should only be added when it is necessary to back fit an existing PCB layout. Example: DPLD_PAD_PLACE=C2

4

Assigning Attributes in a Schematic

In PROSeries, the method of assigning attributes is straight forward. With a left mouse click, select the desired net or symbol (instance). A net's color will change or a bounding box will appear around the instance, identifying it as the currently selected object. Then from the "Add" pull down menu, select "Object Attribute", the bottom of the screen (the text input area) will then prompt you with "Attribute Text String". Type in the attribute and the value (if appropriate) and hit enter. The attributes will then be included in the EDIF netlist once EDIFNETO is run.

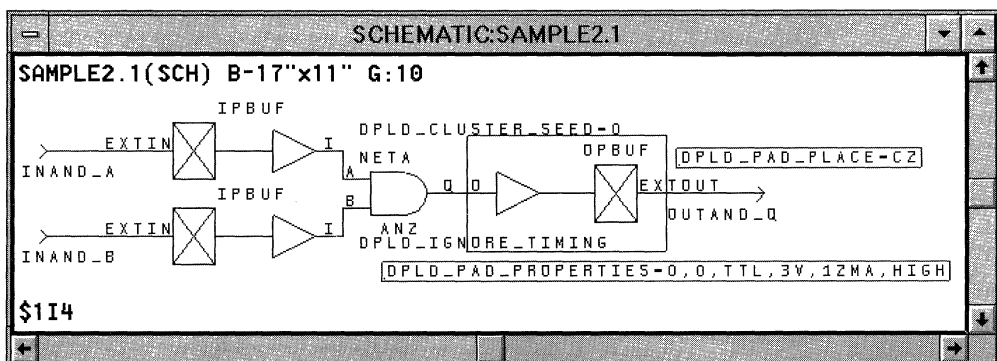


Figure 5. The net "NETA" attributed with DPLD_CLUSTER_SEED=0, The AN2 is attributed with DPLD_IGNORE_TIMING. The selected component "OPBUF \$1I4" and its attached attributes are boxed.



Assigning Attributes & Instances in an External .PAT File

Assigning attributes to a long series of instances, or a variety of nets in the above manner can be time consuming and may be error prone. The MPA Design System gives the designer the option to enter all the valid attributes in an external .PAT file. Entries in the .PAT file take precedence over any attributes that may have also been instantiated in the EDIF netlist via schematic entry.

The external attributes file supports four main operations:

- 4) Insertion of attributes to specify pin placements and pad characteristics.
- 5) Insertion of special pad cells, IPCLK/IPRST, to drive the primary clock/reset network.
- 6) Insertion of special buffer primitives into a named net.

This has two uses:

- c) Force a named net onto/off the peripheral bus, by inserting the primitives APBUF/PABUF respectively.
 - d) Force a named net onto the primary clock/reset network, by inserting the primitives ACLK/ARST respectively.
- 7) Attaching place and route attributes to existing nets, instances and formal ports.

The external attributes file must exist in the same directory and with the same name as the EDIF netlist, with the file extension .PAT. During import, the MPA Design System automatically checks for the existence of a .PAT file and uses it when one is found.

Syntax of the External Attributes (.PAT) File

The external attributes file contains a list of commands, one per line. Each command contains up to five fields, as follows:

```
<object-class> <object-name> <operation> <name> [<value>]
```

where:

<object-class>	is one of port, net, or instance.
<object-name>	is the netlist name of the object (port, net or instance) being operated on.
<operation>	is one of attribute or instance.
<name>	is the name of a definition or an attribute.
<value>	is only used in attribute operations, and is the value to be given to the attribute. This field is only required when an attribute requires a value.

The following are specific syntax forms of all valid attribute or instance assignments.

```
port <name> instance
```

(name here can only refer to input instances with one input pin and one output pin)

```
ipclk
```

```
iprst
```

```
port <name> attribute
```

```
dpld_ignore_timing dummy_arg
```

```
dpld_pad_place <value, see data book for package being used>
```

```
pullup 110
```

```
pulldown 110
```

```
dpld_opdrive 6ma12ma
```

```
dpld_oplevel 3v15v
```

```
dpld_opslew highlow
```

```
dpld_iplevel CMOSITTL
```

```
dpld_pad_properties 011, 011, CMOSITTL, 3v15v, 6ma12ma, highlow
```

```
net <name> attribute
```

```
dpld_cluster_seed n (where, 1 is default, 0 means ignore net)
```

```
dpld_place_priority n (where, 1 is default, 10 is highest priority)
```

```
net <name> instance
```

```
ack
```

```
apbuf
```

```
arst
```

```
pabuf
```

```
instance <name> attribute
```

```
dpld_ignore_timing dummy_arg
```

```
dpld_pad_place <value, see data book for package being used>
```

```
pullup 110
```

```
pulldown 110
```

```
dpld_opdrive 6ma12ma
```

```
dpld_oplevel 3v15v
```

```
dpld_opslew highlow
```

```
dpld_iplevel CMOSITTL
```

```
dpld_pad_properties 011, 011, CMOSITTL, 3v15v, 6ma12ma, highlow
```

The nomenclature of port, net and instance and their use can be a little confusing, I'll attempt to clarify a bit. A 'net' is simply the name of the net of interest. In Figure 6 the valid net names are SEG[0:8]. An 'instance' is the unique designator for the instantiated library macro. In Figure 6, \$1146 is the instance of the OR gate. A 'port' refers to only formal ports (external I/O pins).

Another point of confusion is that the valid attribute sets for "port <name> attribute" and "instance <name> attribute" are identical and each guides the MPA Design System to respond in an identical fashion. You may use either syntax, but for the sake of simplicity, stay consistent in your methodology.

Example .PAT File Entries

The following sample .PAT file entries reference the simple schematic shown in Figure 6. The entries of Figure 7 should be considered one at a time; considering them all to be part of the same .PAT file, all operating on this simple circuit simultaneously is not the intended interpretation.



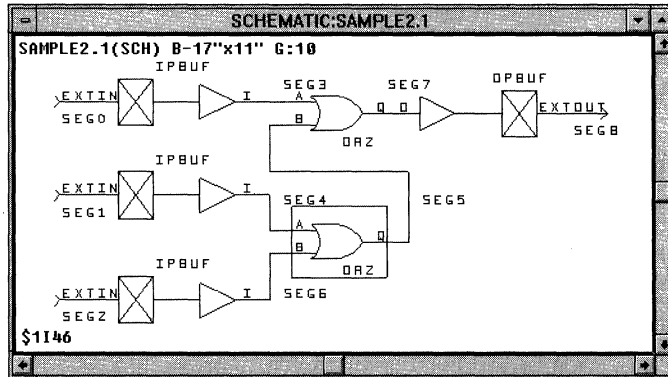


Figure 6. A sample schematic to add .PAT file attributes to. Nets are named SEG[0:8].

```
// This is a comment
# This is a comment as well
port seg0 instance ipclk
//This results in the top IPBUF being replaced by an IPCLK, forcing the
//net SEG3 onto a clock network.
port seg0 attribute dpld_pad_place 22
//This results in the top IPBUF being placed on the pad associated with
//package pin 22.
port seg2 attribute dpld_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the entire delay path driven from the formal port SEG2. A dummy
//argument is required for this attribute.
net seg5 attribute dpld_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the entire delay path associated with the net SEG5. For the design
//of Figure 6, this statement has the same effect as the previous one.
//A dummy argument is required for this attribute.
net seg5 attribute dpld_cluster_seed 500
//During clustering, the MPA Design System will strongly associate
//the top and bottom OR2s. The resulting layout will likely have
//these two instances in the same cluster.
net seg5 attribute dpld_place_priority 10
//With the net SEG5 attributed with a high place priority, the autolayout
//tool will likely place the top and bottom OR2s physically adjacent to one
//another.
net seg5 instance aclk
//This results in an ACLK buffer being inserted after the bottom OR2's output
//and the top OR2's B input. The B input of the top OR2 will now be driven
//off of a primary clock routing resource.
instance $1I23 attribute dpld_opdrive 12ma
//This results in the OPBUF getting 12ma drive capability.
instance $1I46 attribute dpld_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the all nets associated with the instance $1I46. A dummy
//argument is required for this attribute.
```

4

Figure 7. Sample .PAT file entries showing the major syntax variations allowed. Consider each entry individually, this entire figure is not applicable as a .PAT file for the referenced design.



Tool Options – Autolayout

Referring to Figure 8, clicking on the Tool Options button brings you to the Tool Options window of Figure 9.

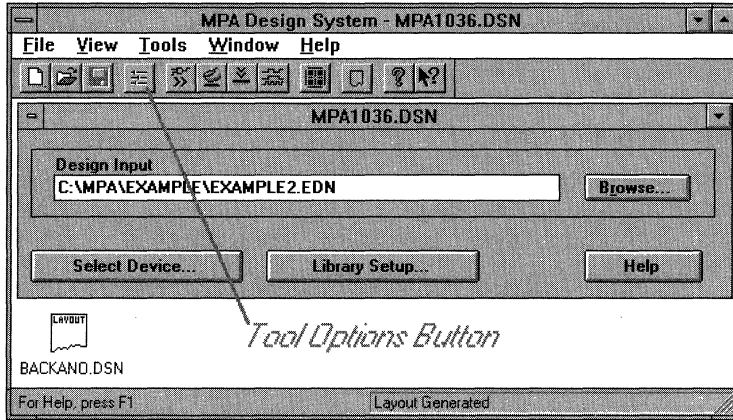


Figure 8. A sample tool context window. In this example, two design (.DSN) files are available.

4

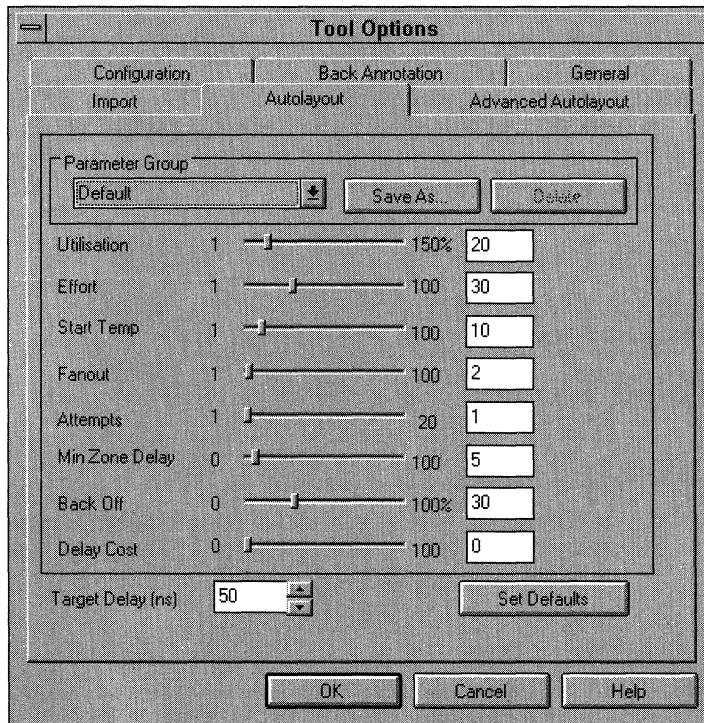


Figure 9. Almost all autolayout parameters are adjusted from this window. 'Seed' is available under the Advanced Autolayout tab.



The 'Parameter Group' roll down menu has a scrollable list of pre-defined tool settings to choose from: Default, High Utilisation, Minimum Delay, Try Harder, and Ignore Timing. Each of these parameter groups was established as a result of studying many sample designs, each with varying design styles and densities. These provided Parameter Groups are generally very good starting points for guiding the MPA Design System through optimization of your designs. If after some experience, you find a new unique set of parameter settings that works better for you, you may save your own custom parameter groups. (The changes are written to a file called PMEL.INI in the Windows installation directory.)

Target Delay

Minimum 1 Maximum 9999 Default 50 (5nS)

Target delay is the most significant guiding parameter of the autolayout process. The 'delay' of a combinatorial network is calculated as the longest path from input to output. For synchronous circuits, path delays are calculated as I/O to clocked element (register), register to I/O and from register to register.

Autolayout attempts to keep the delay of each of these paths types below the target delay value. For a single clock design, you set the target delay to the reciprocal of the desired operating frequency.

The units are 10⁻¹⁰s. For example, 80 yields an 8ns delay target. (A screen shot of Version 2.2.3 of the MPA Design System is shown in Figure 9. The panel shows the units as "ns", this is incorrect and will be corrected on version 2.2.4 and later.)

Utilisation

Minimum 1 Maximum 150 Default 80

The percentage of the recommended maximum number of core cells partitioning will attempt to use in a zone. Partitioning will exceed this number if necessary to complete.

For the MPA1000 family, the recommended number of cells per zone to use is around 50. Setting utilisation to 100% tells the partitioning tool to consume approximately 50 of the 100 possible cell sites per zone. If the utilisation parameter is set too high, then the autolayout tool may spend a lot of effort on a few highly utilized zones, while other zones are left empty or underutilized. If the utilisation parameter is too low, then the autolayout tool will adjust the parameter upward until it is compatible with the design. Increasing the value of the utilisation parameter may increase the operating speed of the circuit, only at the cost of increased tool run time. Values greater than 100% are not recommended.

Effort

Minimum 1 Maximum 100 Default 30

The relative amount of work applied to the partitioning phase. When setting utilisation high, effort should be set high as well.

The two major phases of autolayout that typically consume the most time are partitioning and zone routing. The time taken in partitioning is directly related to effort.

Start Temp

Minimum 1 Maximum 100 Default 10

Partitioning and zone routing are performed using a simulated annealing algorithm. Setting the start temperature higher gives the partitioning and zone routing tools the freedom to make more aggressive moves in the search for the optimal solution. With the start temp set low, the respective algorithms may only be free to find local minimum solutions whereas the best overall solution lies over some other cost hump that the tool was otherwise constrained from traversing. It is usually best to increase start temp in a highly utilized device. If increasing start temp, be sure to increase effort as well.

Fanout

Minimum 1 Maximum 100 Default 2

This is the maximum fanout of a net which is still included in the clustering. All nets which have a fanout greater than the number specified are ignored in the clustering phase. Any nets ignored during clustering are more likely to be routed on global resources (typically a bit slower than medium or zonal routing resources).

Attempts

Minimum 1 Maximum 20 Default 1

The number of runs through partitioning phase. More runs typically yield a better solution, only at the expense of extending tool run time.

Min Zone Delay

Minimum 1 Maximum 9999 Default 5 (0.5nS)

The units are 10⁻¹⁰s. The zone router will attempt to keep the delay of all net segments within the zone being routed to less than the value specified.

Back Off

Minimum 0% Maximum 100% Default 30%

Percentage of relaxation to apply if the target delay was unobtainable. Expressed as a percentage of the previously described Target Delay.

Delay Cost

Minimum 1 Maximum 100 Default 5

This is the weighting given to timing during partitioning. If a particular partition includes a net that is failing to meet its timing target, then the cost of that partition will be artificially raised by an amount proportional to the delay cost. Increasing the delay cost is likely to trade off against achievable utilization. Setting delay cost to zero will result in much reduced tool run times and increased achievable utilization, however this is accomplished at the expense of lowering the design's maximum frequency.

Seed

A 'seed' value may be set in the Advanced Autolayout panel of the Tool Options window. The seed is as a starting point for the autolayout's pseudo-random number generator. Random numbers are used in several of the autolayout algorithms. Because it is not a true random number generator, two layout runs with identical settings will yield identical results.

4



Changing nothing but the seed value may yield significantly different solutions. The PC and Workstation versions of the MPA Design System have different pseudo-random generators and so solutions from each will always differ in spite of identical initial seed values. It is mentioned here only for completeness. Changing the seed value does not guarantee a faster place and route solution, only a different one.

Clock Files

In a simple single clock design, it is sufficient to declare a Target Delay value in the autolayout panel of the tool options window. The autolayout attempts to achieve a place and route solution in which all I/O to clocked element path delays and all clocked element to clocked element path delays are shorter than the target delay.

However, more complex designs may have multiple clocks, each running at unrelated frequencies. In such instances it may be unwise to ask the autolayout tool to constrain every path of the design to the delay target of the fastest path required.

Clock files provide a method of grouping related components of your design into unique timing groups. The timing groups may then each be assigned unique clock specifications, only as restrictive as required. This assists the autolayout tool by guiding it to complete the more speed

critical nets using the more valuable placement, routing and switching resources as required.

Clock files also provide a method of specifying target delays between such timing groups. The syntax presented below can be a bit daunting at first. Please read through to the examples, and it should become a bit clearer on just how to use a clock file.

Clock File Syntax

The following meta-syntax conventions are used in this definition:

::=	introduces a rule
;	terminates a rule
{...}+	Indicate one or more occurrences of the phrase inside the braces
{...}*	Indicates zero or more occurrences of the phrase inside the braces
[...]	Indicates an optional single occurrence of the phrase inside the brackets
	Separates alternative choices

Timing groups identify those portions of the design driven by a particular clock with a particular specification. The syntax for Timing Group Definitions is given as:

```
timingGroupDefinitions ::= (' `timingGroupDefinitions` {timingGroup}+ ');
  where
    timingGroup ::= (' `timingGroup` timingGroupDef [clock]
                    timingGroupInstanceList ');
      where
        timingGroupDef ::= timingGroupName; any string, ie my_timing_group
        clock ::= (' `clock` period [phase] ');
          where
            period ::= time_in_units_10-10_seconds (50 = 5nS), ie '50'
            phase ::= time_in_units_10-10_seconds
        timingGroupInstanceList ::= {allIO|allFlipFlops|netRef|instanceRef}+;
          where
            allIO ::= (' `allIO` ');
            allFlipFlops ::= (' `allFlipFlops` [clockInputSense] ');
              where
                clockInputSense ::= 'INVERTED'|'NONINVERTED'
            netRef ::= (' `netRef` netName [clockInputSense] ');
              where
                netName ::= any string, ie my_net_name
                clockInputSense ::= 'INVERTED'|'NONINVERTED'
            instanceRef ::= (' `instanceRef` instanceName ');
              where
                instanceName ::= ``any_instance ``', ie "$1I19"
```

Intended Target Delay is the method used to limit the delay between the previously defined timing groups.

The syntax for Intended Target Delay is given as:

```
intendedTargetDelay ::= (' `intendedTargetDelay` targetDelay driverGroupRef
                        drivenGroupRef ');
  where
    targetDelay ::= time_in_units_10-10_seconds (50 = 5nS), ie '50'
    driverGroupRef ::= timingGroupName; any string, ie my_timing_group
    drivenGroupRef ::= timingGroupName; any string, ie my_timing_group
```



Clock File Examples

The syntax specification for clock files is harder to read than it is to type, and it wasn't easy to type. So in an effort to clarify the topic some, let's look at some clock file examples.

```
( timingGroupDefinitions
  ( timingGroup My_Flip_Flops
    ( clock 50 )
    ( allFlipFlops )
  )
  ( timingGroup My_IO
    ( allIO )
  )
)
( intendedTargetDelay 10000000 My_Flip_Flops My_IO )
( intendedTargetDelay 10000000 My_IO My_Flip_Flops )
```

In the above example all of the flip-flops are intended to be clocked with a 5nS period clock. I'm further specifying that I don't care how long it takes to get signals in and out of the design by defining a very large intendedTargetDelay between groups and not specifying a target clock period set for "allIO",

```
( timingGroupDefinitions
  ( timingGroup My_CLK1_Group
    ( clock 200 )
    ( netRef CLK1 )
  )
  ( timingGroup My_CLK2_Group
    ( clock 100 )
    ( netRef CLK2 )
  )
)
```

4

In the above example, two clock groups were specified: 'My_CLK1_Group' and 'My_CLK2_Group'. All instances driven with the clock named CLK1 will be placed and routed to meet a 20nS delay criteria, and all instances driven with the clock named CLK2 will accommodate a 10nS clock.

```
( timingGroupDefinitions
  ( timingGroup My_Group
    ( clock 200 )
    ( instanceRef "$1I*" )
  )
)
```

In the above example, the wild card * is used to define the instances that are members of the timing group 'My_Group'.

```
( timingGroupDefinitions
  ( timingGroup My_CLK1_Group
    ( clock 200 )
    ( netRef CLK1 )
  )
  ( timingGroup My_CLK2_Group
    ( clock 100 )
    ( netRef CLK2 )
  )
)
( intendedTargetDelay 100 My_CLK1_Group My_CLK2_Group )
( intendedTargetDelay 100 My_CLK2_Group My_CLK1_Group )
```

In the above example, two clock groups were specified: 'My_CLK1_Group' and 'My_CLK2_Group'. All data paths between these two groups are constrained to meet the more restrictive of the two timing requirements.



Tool Progress Monitoring

As the you gain experience and the density of your designs starts to increase, and our your speed requirements become more restrictive, you may find the MPA Design System working longer on your designs. As such, this section is included to give you a clear explanation of all the status windows and displays the system gives you during the course of design importation and auto layout.

Import

4

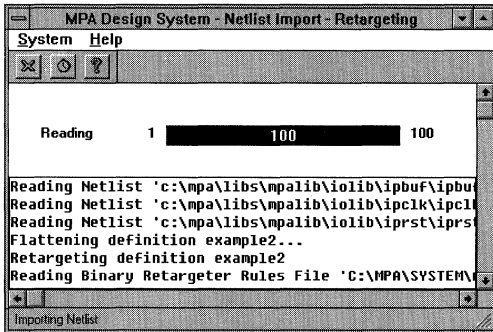


Figure 10. Importing and Retargeting

During the import and retarget phase, the MPA Design System takes the EDIF netlist in, checks it for compatibility and errors. The macro elements of the EDIF netlist are mapped to the cell definitions available to the MPA family. If there are no problems found, the import phase concludes with the output of a .NET file, the native netlist format of the tool.

Autolayout

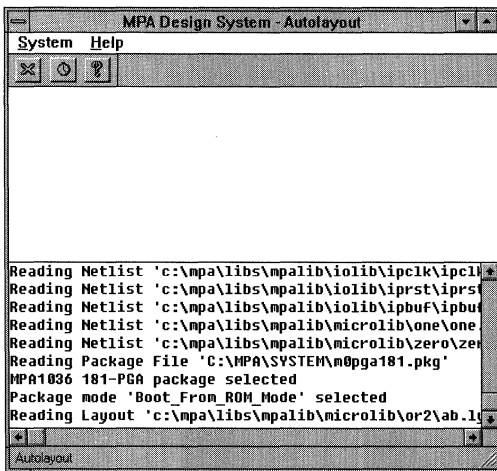


Figure 11. The beginning of Autolayout

During the brief initialization period of the autolayout process, the previously mapped component nets are read in, the package information is read in, I/O pads are counted along with core cells. A figure of utilization is echoed back and the tool proceeds to the clustering phase. No status bars are presented in this step.

Clustering

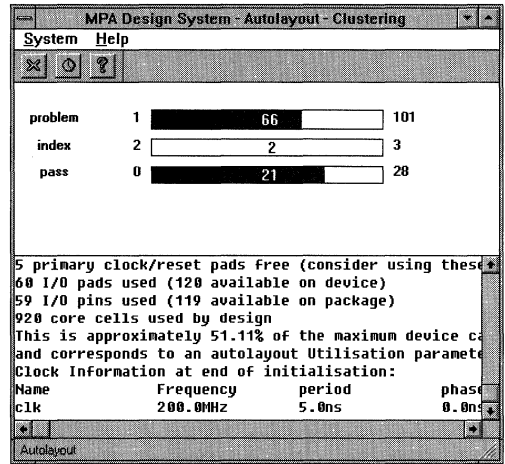


Figure 12. The Clustering Status

Clustering combines groups of related logic together to help break up the place and route problem into more manageable chunks. In Figure 12: **Problem** refers to the number of root level clustering problems to be solved. A root level clustering problem is a cluster containing sub-clusters or and I/O. **Index** refers to the 3 phases of clustering for that problem. Phase 1 is the stage where the clustering algorithm decides which clusters are root level clusters, and is already complete. Therefore the progress bar starts on 2 and finishes on 3. **Pass** refers to the estimated number of items per root level problem.

Partitioning and Pre-Placement

About half of the tool's time is spent in the partitioning phase. During this phase, the MPA Design System places the clusters formed in the previous process into zones. Pre-placement refers to the placement of I/O sites that were optionally fixed via the DPLD_PAD_PLACE attribute.



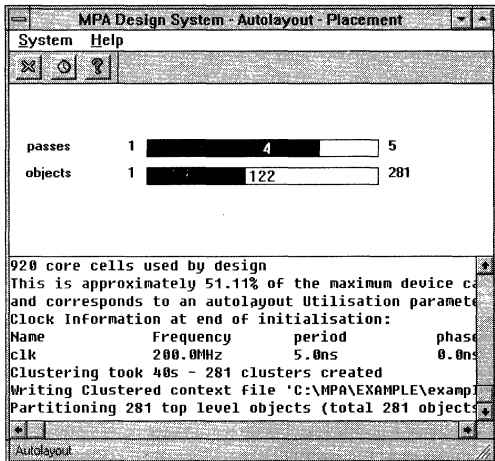


Figure 13. The Initial Partitioning Phase

The first part of the partitioning algorithm completes the initial placement tasks. **Passes** refers to the fixed number of passes at the placement problem. The first pass yield an initial placement of clusters, the remaining 4 passes are refinements of this initial placement. **Objects** refers to the number of clusters created in the previous clustering phase.

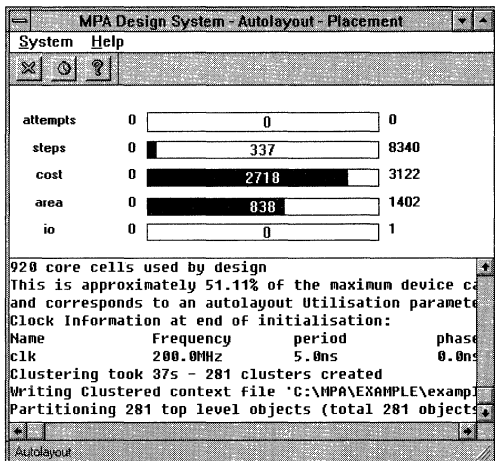


Figure 14. The Second Partitioning Phase

This phase takes the initial placement results and refines it by assigning specific zones and routing ports to the clusters. **Attempts** refers to the number of times this entire phase

will/has run. **Attempts** is set as an autolayout tool option 'Attempts'. Partitioning is the most important phase of the autolayout process. If you are looking for more speed from your design this is a good place to experiment. Increasing 'Attempts' in the autolayout tool options will generally yield improved results, but at the expense of extending the tool's run time of course. **Steps** refers to the number of steps the tool must take to complete the partitioning. The step rate increases as the tool proceeds. This is the indicator to watch to get the best feel for how long partitioning will take. **Cost** refers to the total cost of the design (the sum of the cost of the nets plus the cost of the cost of the parts). Cost generally starts off high and proceeds in a downward trend as a solution is approached. It gives you an indication of the quality of the partition in the current stage, whether or not the tool is converging to a solution, how good that solution is, and how sensitive the design is to changes in the way the clusters are placed. **Area** is proportional to the sum of the number of instances in the zones in excess of the number specified in the utilisation parameter. **IO** refers to number of over congested areas in the IO zones.

Global Routing

4

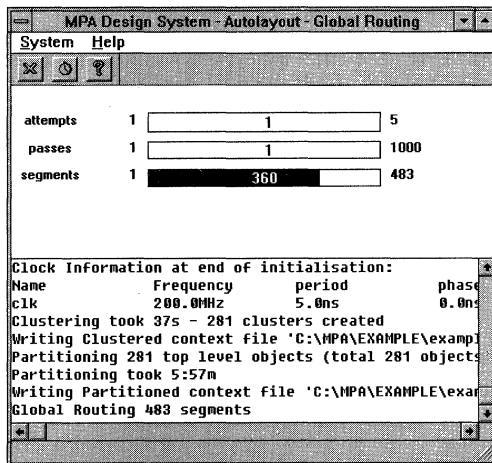


Figure 15. Global Routing Phase

After partitioning has assigned all the clusters to zones, global routing assigns signals to zone port cells and routes these ports to one another as required. **Attempts** refers to the number of attempts to route all segments. If global router fails to complete the routes on the fifth iteration it will give up. **Passes** refers to the number of attempts per segment to route globally (1000 per segment). **Segments** refers to the total number of segments to be routed.



Zone Routing

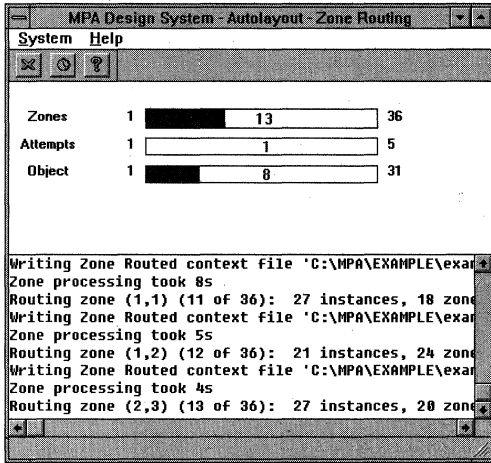


Figure 16. Simple Zone Routing Example

4

Zone routing is the final major step in completing the autolayout problem. The tool can often spend more than half of its time completing this task. Partitioning has placed clusters into zones, and global routing has assigned and routed signals from zone to zone and zone to I/O. Now the zone routing tool must complete the connections at the lowest level of hierarchy. **Zones** refers to which zone is currently being processed. **Attempts** is the number of attempts taken at completing the zone routing. If the zone router fails to route the zone on the fifth iteration, it will give up on the current placement solution and attempt to replace instances within the zone as described in Figure 17. **Object** refers to the total number of objects to be placed and routed in the zone.

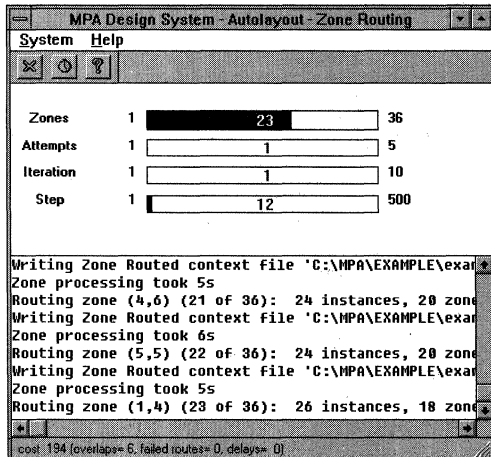


Figure 17. Complex Zone Routing Problem

If the instance placement solution provided by partitioning can not be zone routed, the tool (zone router) gives up on the process described in Figure 16 and moves to a second phase of zone routing. In this phase the tool discards the placer's solution and provides new possible solutions to try. In this case, **Zones** refers to the zone currently being processed. **Attempts** is the number of attempts taken at completing the zone routing. **Iteration** is the number of attempts to take to achieve the targets for zone routing. **Step** refers to the maximum number of steps (per iteration) to obtain the zone routing targets.

Back Annotation

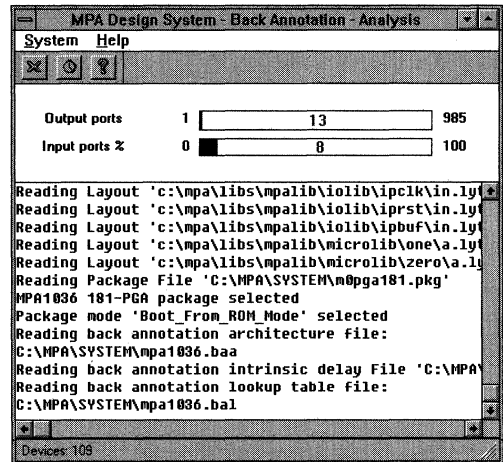


Figure 18. Back Annotation Status

Once you've successfully completed a layout, you may want to generate a back annotation file so timing data can be passed back to your simulator. In this process the estimated RC and routing switch delays are compiled and passed back to a netlist format of your choosing. **Output ports** refers to the every driving pin of the design. **Input ports** (expressed as a percentage) refers to all the inputs that could possibly affect the output port being considered.



Summary

Figure 19 is a matrix that shows how each of the methods discussed in this application note interacts with the various phases of the autolayout process. You can see from this matrix that the most critical phase of the autolayout process is partitioning. If you are having trouble meeting a restrictive timing requirement, this is where you should concentrate your efforts.

The number of different combination of design styles, tool settings and use of attributes is very large and so it is impossible to give exact guidance on how to use the MPA Design System to get the absolute best possible speed solution for your particular design. However, the MPA Design System and the MPA1000 architecture were co-developed and as such should provide you with a total system ready to implement your requirements quickly and easily with no need to adjust the parameters and attributes as described in this note. If however your timing requirements are not met on your first pass through the tool, hopefully you have been provided with enough information in this note to start experimentation in an informed manner.

	Map (import)	Pre-Place	Cluster	Partition	Global	Zonal
Front End Design						
Macro Selection	X					
PIN Attribute						
DPLD_IGNORE_TIMING			X	X	X	X
INSTANCE Attribute						
DPLD_IGNORE_TIMING			X	X	X	X
NET Attributes						
DPLD_IGNORE_TIMING			X	X	X	X
DPLD_CLUSTER_SEED			X			
DPLD_PLACE_PRIORITY					X	
DPLD_PAD_PLACE		X				
Tool Options						
Target Delay			X	X	X	X
Utilization				X		
Effort				X		
Start Temp				X		
Fan Out			X			
Attempts				X		
Min Zone Delay						X
Back Off				X	X	X
Delay Cost				X	X	X
Seed			X	X	X	X

Figure 19. Effected Modules for Macros, Attributes, and Tool Option Settings

4



Estimating Power in the MPA1000 Family

Prepared by
Paul Butler

4



Estimating Power in the MPA1000 Family

This application note is intended to provide the users of Motorola's MPA1000 family a method of quickly estimating the worst case power consumption of a completed design.

MPA1000 power consumption is dependent on the size, physical structure, logical structure and timing behavior of a Customer design. MPA Design System software is used to automatically transform an input design into a physical implementation or layout. From this representation a bitstream is derived which when loaded into a MPA1000 device produces the desired design behavior.

MPA1000 power consumption has four components: internal or array power (P_a), clock network power (P_c), intrinsic I/O power (P_{io}) and output load power (P_{ol}). The major factors affecting MPA1000 power consumption are: array size, number of simultaneous switching cells and I/Os, I/O capacitive loading, supply voltage, and the switching frequency of cells and I/Os.

The switching frequency of a cell or I/O is not the same as the system or applied clock frequency. For synchronous designs each individual cell or I/O in may switch at a rate related to the applied clock frequency, but the actual switching frequency depends on the logical structure and timing behavior of the design. Similarly, the number of simultaneously switching nodes is also related to design structure and timing behavior. Manually calculating switching frequencies of each node in a design is typically not practical. Some commercially available tools utilize static timing analysis techniques coupled with device specific nodal capacitances to make very accurate power calculations.

This application note provides a simplified method for estimating worst case MPA1000 power consumption based on simplifying assumptions concerning switching frequencies and simultaneous switching behavior.

A worst case switching frequency assumption might be to consider all nodes switch once per clock period assuming a single system clock. If any knowledge concerning design structure and timing behavior can be applied a much more realistic power consumption estimate can be obtained, however.

Simultaneous switching estimation rules of thumb used by various ASIC vendors varies from 15% to 25%. Since FPGAs tend to have routing delays which approach logic delays, a significant amount of signal skew can occur. This skew can effectively lower the amount of simultaneous switching one might expect when reviewing the design alone. While applying knowledge about design timing behavior improves accuracy, variable routing delays can significantly alter actual behavior.

Array Power Consumption (P_a)

In MPA1000 devices core cells are used to implement logic functions and are also used to implement routing functions under certain circumstances. In addition, many routing resources are self buffering and also contribute to overall power consumption. To simplify power consumption estimation, typical resource usage has been taken into

account and an effective power consumption for a single logic cell has been derived.

Under the assumption of a single system clock frequency and a single estimate for logic cell simultaneous switching, the worst case internal array power consumption can be estimated by:

$$P_a = \#cells * cellp * ssw\% * f * 10e-6$$

Where:

P_a	Total internal power in watts.
#cells	Total number of logic cells used by the design as reported by the MPA Design System Autolayout tool.
cellp	Effective cell power in micro watts per MHz ($\mu W/MHz$). For the MPA1000 device this has been derived as 33 $\mu W/MHz$ when $V_{dd} = 5.0v$.
ssw%	Estimated percentage of cells which switch simultaneously.
f	Switching frequency in MHz.

For a more accurate internal power estimate, the total number of cells consumed by the design could be partitioned into several groups each with its own switching frequency. The power consumed for each group could then be calculated and the total internal power obtained by summing the results. A very pessimistic result can be obtained by using the input or system clock frequency as the switching frequency. If multiple clocks are used, partitioning and summing can also be used to improve accuracy.

4

Clock Power Consumption – P_c

The MPA1000 has 8 dedicated low skew resources. These resources have a large capacitive load and can contribute significantly to overall power consumption. Power consumption for a single dedicated clock is given by:

$$P_c = clockp * f * 10e-6 * 2$$

Where:

P_c	Clock power in watts.
clockp	Effective clock power in micro watts per MHz ($\mu W/MHz$) MPA1000 Clock power is a function of the device size. Clock power for each MPA1000 device is:
MPA1016	600 $\mu W/MHz$
MPA1036	1050 $\mu W/MHz$
MPA1064	1650 $\mu W/MHz$
MPA1100	2380 $\mu W/MHz$

f Clock frequency in MHz. Note the factor of 2 is required because clock nodes switch twice per clock period. Nodal switching frequency not the clock period determines power consumption.

The power consumed by multiple clocks is calculated by summing the power consumption of each clock used. This



gives the worst case power consumption for all clocks assuming they are related and frequently have common simultaneous transitions.

I/O Power Consumption – P_{ioi} & P_{ol}

Total I/O power consumption is the summation of I/O intrinsic power requirements and power required to drive external loads. When calculating I/O power an average load capacitance can be calculated based on actual output pin loading and applied to all active outputs. Optionally, the power consumed by each I/O could be calculated individually based on specific external loading conditions and summed.

The total intrinsic power consumed is given by:

$$P_{ioi} = \#ios * ioip * ssw\% * f * 10e-6$$

Where:

- P_{ioi} Total I/O intrinsic power in watts.
- #ios The total number of I/O pads used by the design. This is reported by the MPA Design System Autolayout tool.
- ioip Intrinsic I/O pad power in micro watts per MHz (μW/MHz) For the MPA1000 device this is 200μW/MHz.
- ssw% Estimated percentage of I/O pads which switch simultaneously.
- f Switching frequency in MHz.

If outputs are not latched and tied to a common output clock, it is unlikely that they will simultaneously switch due to path skews induced by routing delays. Because I/Os consume significant power very general assumptions of I/O switching frequency and simultaneous switching behavior can make resultant power estimates very inaccurate.

If multiple clocks are used, a closer estimate could be obtained by dividing #ios into groups, calculating power for each group and summing the results.

Output load power consumption is given by:

$$P_{ol} = \#out * ssw\% * Cload * Vdd^2 * f * 10e-6$$

Where:

- P_{ol} Total output load related power in watts.
- #out The total number of outputs used by the design. This is the subset of total I/Os which are used to drive external loads.
- ssw% Estimated percentage of outputs which switch simultaneously.
- Cload Average load capacitance in pf.
- Vdd Supply voltage in volts
- f Output switching frequency in MHz.

Total I/O power is given by:

$$P_{tot} = P_{ioi} + P_{ol}$$

Example Power Consumption Calculation

For an MPA1036 with:

#cells	1500
ssw% (cells)	20%
#clocks	1
f (clock)	70MHz
f (cells) = f (io) = f (clock)	Pessimistic Est.
#io	100
#out	50
ssw% (io)	10%
Cload	50pf

$$P_a = 1500 * 33 * .20 * 70 * 10e-6 = 693mW$$

$$P_c = 1050 * 70 * 10e-6 * 2 = 147mW$$

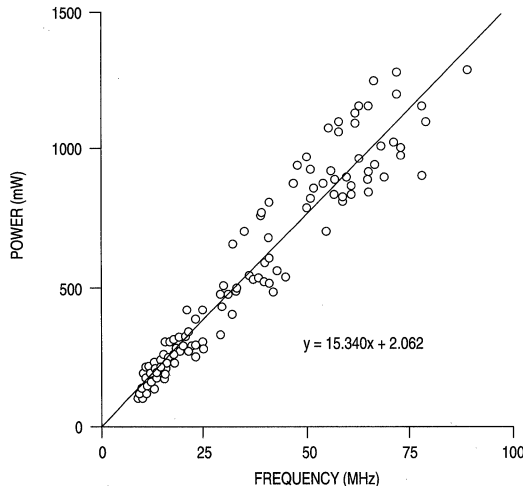
$$P_{ioi} = 100 * 200 * .10 * 70 * 10e-6 = 140mW$$

$$P_{ol} = 50 * .10 * 50 * 25 * 70 * 10e-6 = 437.5mW$$

$$P_{total} = P_a + P_c + P_{ioi} + P_{ol} = 1.42W$$

4

Sample Data



The data from the above graph is a collection of 125 sample designs fit into an MPA1036. P_{ol} is not included in this sample data set. Power consumption for the MPA1036 is roughly 15mW/MHz.



Using Mentor Graphics' Design Architect ver. A3 with the MPA Design System

Prepared by
Claudia Colombini
Motorola Programmable Logic Products

4



Using Mentor Graphics' Design Architect ver. A3 with the MPA Design System

Introduction

The Motorola family of MPA devices and supporting software provides hardware designers with a wide selection of design methodologies. This application note is intended to demonstrate a valid design flow for Motorola MPA series field programmable gate arrays using Mentor Graphics' Design Architect environment.

The reader should be familiar with the Mentor Design Architect tool suite. The reader should also have access to the

Motorola MPA Design System software and the Mentor integration kit libraries supporting the MPA1000 family. This application note was written using version A3 of Design Architect and version 2.2.3 of the MPA Design System.

Schematic Entry Flow

Figure 1 shows the recommended flow for moving your Design Architect schematic through export and into the MPA Design system and Quicksim simulation.

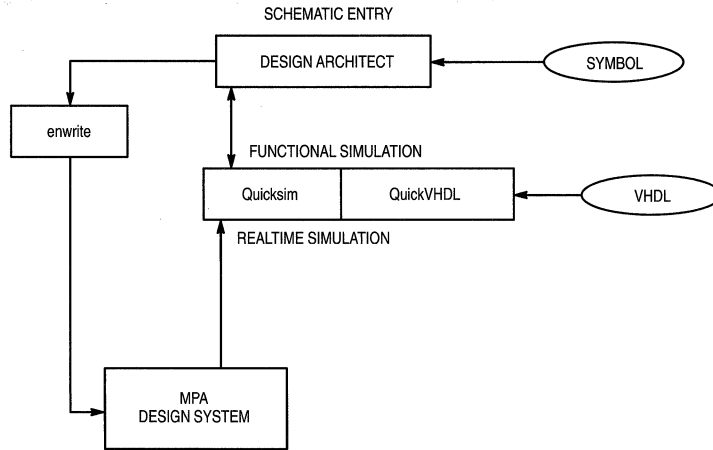


Figure 1. General Design Flow Using Design Architect

The complete functional schematic is moved from Design Architect via the EDIF netlist writer "enwrite". The MPA Design System imports the resulting EDIF netlist.

The implementation into the MPA1000 device is completed using the Motorola MPA Design System place and route tool. After completion of autolayout, a backannotated VHDL gate

level netlist can optionally be generated and exported to the Mentor simulation environment. The VHDL timing representation is used as input for the real time simulation with Quicksim. The whole flow will be explained with the help of a small example. A 4 bit carry look ahead adder with synchronous inputs and outputs shown in Figure 2.

4



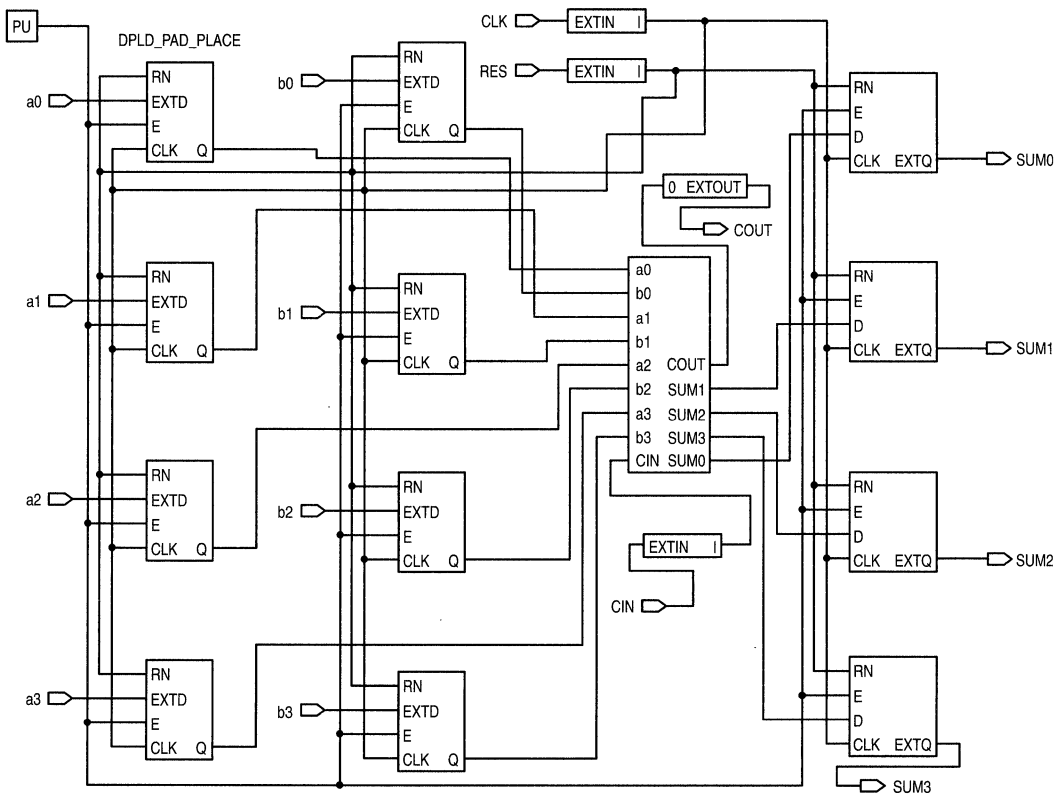


Figure 2. A Simple 4 Bit Adder

Schematic Entry Setup

1. Install the libraries for Motorola FPGA on your workstation
2. Add the following line to your `mgc_location_map` file:

```
$MPA_SYMBOLS -t LIBRARY
<path_to_Motorola_library>/library/symbols
```

By adding this line to the `mgc_location_map` file, the Design Architect will be able to locate the Motorola MPA symbol library and the symbols can be used for drawing schematics.

3. Create a directory for your design:

```
mkdir <design_dir_name>
```

All design related data will be located in this directory.

Schematic Entry Rules

Please refer to Figure 2 for help.

1. Generate a top level sheet containing:

- symbol of the core design
- I/O pads
- Mentor GENLIB portin, portout, portbi symbols (“a0” is a portin, “sum0” is a portout)

2. Use special I/O pads for clock and reset pins:

For external clock signals the I/O macro IPCLK and for external reset signals the I/O macro IPRST should be used. This will force the external clock and reset signals to the clock distribution network during MPA place and route and increase the speed of the design. There are 8 clock/reset pads available in the MPA1000 series.

3. Use I/O macros for buffered I/Os:

In the example all I/O signals are synchronous. Best performance for area and speed in the MPA1000 series can be obtained using the macrocells targeted specifically for the MPA's feature rich I/O pad sites. For the registered I/Os the IPDFR and OPDFR macros are used as inputs and outputs respectively. The use of these special I/O macros in the design entry will force the place and route tool to automatically take the flip-flops available in each I/O cell and not consume internal flip-flop resources for mapping. For more information on the I/O macro elements please refer to the online help of the MPA Design System tool. The Motorola library contains more than 70 different I/O symbols covering all available possibilities.

4



4. Use special macros for Ones and Zeros

In the example schematic a "One" symbol from the \$MPA1000 library was used to tie high the enable inputs of the flops. The library also contains a "Zero" macro for similar use. Do not attempt to tie unused inputs to anything but One or Zero. Do not leave unused macro inputs floating.

5. Fix I/O placement:

To specify the I/O placement, pin numbers are attached to the I/O symbols:

Commands:

- select I/O symbol
- attach attribute: name: DPLD_PAD_PLACE
value: <pin_number>

Refer to the Motorola MPA databook for pin number information for the different packages. For this example the MPA1016 in the 84 PLCC package is used. While assigning pins in this fashion keeps all the design documentation in a single place, it can be tedious and error prone on larger designs.

The MPA Design System gives the designer the option to enter all the valid attributes in an external .PAT file. Entries in the .PAT file take precedence over any attributes that may have also been instantiated in the EDIF netlist via schematic entry. The .PAT files must have the same root file name as the EDIF netlist .EDN files, and must reside in the same directory during import to the MPA Design System. See the addendum on .PAT files for further details.

6. Generate a symbol out of the top level schematic

The symbol can be generated automatically out of the top level schematic in the Design Architect with the generate symbol command.

This symbol is needed for the EDIF netlist writer to recognize the I/O interfaces. Without this symbol the port directions will not be written correctly into the EDIF netlist,

which will cause problems during place and route in the MPA design system tool.

Functional simulation setup:

1. cd <design_dir>
2. qvmap mpa <path_to
Motorola_lib>/library/mpa

This command will copy the Mentor quickvhd.ini file into the <design_dir_name> directory and attach the path to the Motorola VHDL simulation library to the VHDL library name map. The quickvhd.ini file will be accessed during simulation start and simulation.

3. qvlib <qvpro_lib_name>

This creates a library where the qvpro simulator will put the VHDL code for the design. During qvpro startup a VHDL representation will be generated automatically and the compiled entity and architecture will be put into the <qvpro_lib_name> directory.

Functional Simulation with Quicksim

Start the simulator with the command:

```
qvpro <design_sim_toplevel> -lib <qvpro_lib_name>
```

Qvpro is the Mentor simulation environment which allows cosimulation of of VHDL and Quicksim simulation models with Quicksim or QuickVHDL simulator. For schematic entry the Quicksim simulator will allow the probing of the signals which should be traced in the schematic. The qvpro command initializes the design for Quicksim and QuickVHDL cosimulation. The functional simulation can be done using the Quicksim tool as known to the user.

This application note describes the use of qvpro with Quicksim as the master. Prerequisite for this configuration is a schematic as top level simulation view. For other use of the qvpro simulator please refer to the Mentor documentation, Simulating with qvpro.

4



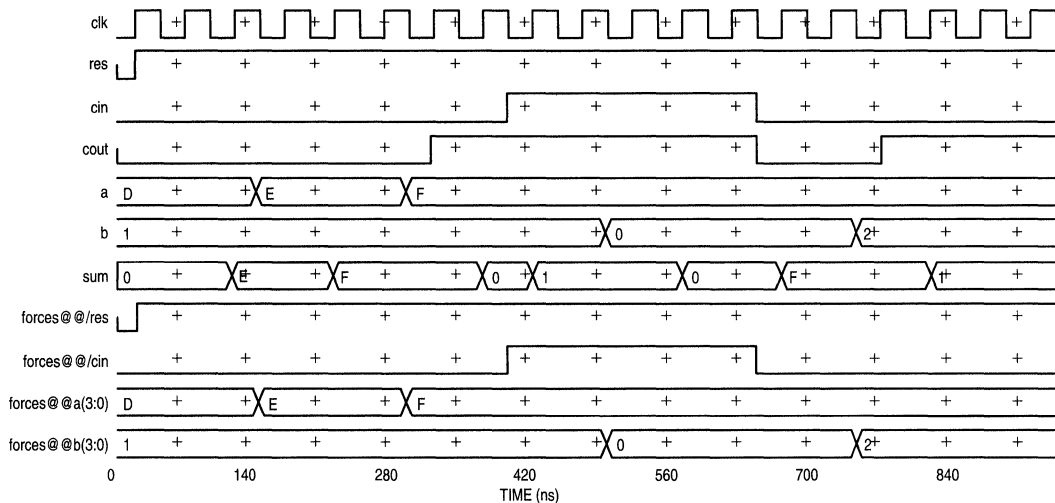


Figure 3. A Functional Simulation Run of the Example Circuit

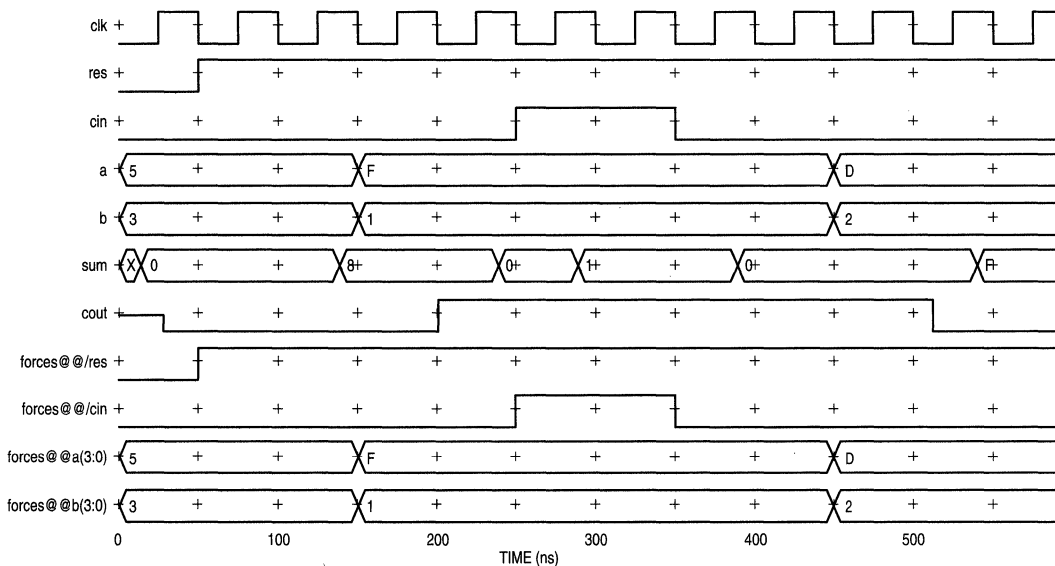


Figure 4. A Simulation Run with Back Annotated Timing Data

4



Interface to the Motorola MPA Place & Route Tool

Input for the MPA Design System software is an EDIF 2.0 netlist which is generated by the Mentor EDIF netlist writer "enwrite". To have all hierarchical information available in the MPA design software for automatic design partitioning, it is recommended to produce a hierarchical netlist.

Command:

```
enwrite <top_level_design> <designname.edn>
```

Implement the Design Into the Motorola FPGA

Figure 4 shows the steps during the implementation of the design into the Motorola MPA using the MPA Design System software.

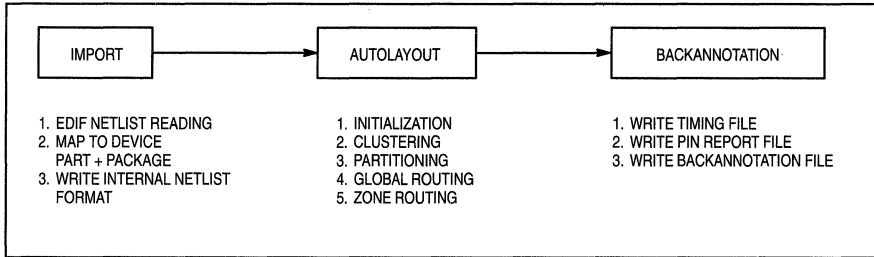


Figure 5. The Basic Place and Route Flow

Implementation of the design in the MPA1000 family device is a straightforward two step process.

4

1. Import:

Reads the EDIF netlist of the design, maps it to the selected device and generates an internal netlist representation (a .NET file).

2. Autolayout:

All steps during autolayout will be reported in the .LOG file. In the .LOG file all information about device usage and estimated maximum frequency after each step is reported.

Using default parameters for import and autolayout will produce satisfying results in most cases. Nevertheless it is possible to modify the results by changing the autolayout options and re-running. For more information please refer to the MPA Design System online help or the application note AN1569 [Tuning the MPA Design System for Speed](#). The necessary backannotation format for timing simulation with Mentor is VHDL. Therefore VHDL must be chosen as backannotation format from the Tools -> Options pull down menu.

Setup for Timing Simulation:

1. Copy the backannotated vhdl file to
<design_dir_name>

2. qvlib <compile_dir_name>

creates a directory where the backannotated vhdl will be compiled to

3. qvmap <compile_lib_name>
<path_to_compile_dir_name>

adds the path to the <compile_dir_name> to the quickvhdl.ini file

4. qvcom <backannotated_vhdl_file_name> -work
<compile_dir_name>

compiles the backannotated vhdl file

5. Create a symbol and instantiate it in a new schematic.

To be able to use the Quicksim environment also for timing simulation, a symbol has to be created out of the backannotated vhdl file and instantiated in a new schematic.

Steps:

1. start Design Architect

2. generate symbol

settings:

- source: entity

- quickvhdl.ini file : <design_dir_name>/quickvhdl.ini

- library logical name: <compile_lib_name>

With these settings in the generate symbol window, the entity and architecture of the vhdl file will be found when clicking on entity and architecture

3. check symbol and save

4. generate new schematic <backannotated_top_level> and instantiate the symbol generated out of the backannotated VHDL.

Start Timing Simulation

Start the qvpro cosimulation environment for simulating with Quicksim:

```
qvpro <backannotated_top_level> -lib  
<qvpro_lib_name>
```

Quicksim simulation can be started.

Results of the timing simulation from the example see Figure 5.

Comment:

The Quickvhdl simulator can also be used for timing simulation. Just compile the backannotated design and start the Quickvhdl simulator: qvsim <compile_dir_name>



Addendum – .PAT External Attribute Files

The external attributes .PAT file supports three main operations:

- 1) Insertion of attributes to specify pin placements and pad characteristics.
- 2) Insertion of special pad cells, IPCLK/IPRST, to drive the primary clock/reset network.
- 3) Insertion of special buffer primitives into a named net.

This has two uses:

- a) Force a named net onto/off the peripheral bus, by inserting the PMeL primitives APBUF/PABUF respectively.
- b) Force a named net onto the primary clock/reset network, by inserting the PMeL primitives ACLK/ARST respectively.

The external attributes file must exist in the same directory and with the same name as the EDIF netlist, with the file extension .PAT. During import, the MPA Design System automatically checks for the existence of a .PAT file and uses it when one is found.

Syntax of the External Attributes (.PAT) File

The external attributes file contains a list of commands, one per line. Each command contains up to five fields, as follows:

`<object-class> <object-name> <operation> <name> [<value>]`

where:

<code><object-class></code>	is one of port, net, or instance.
<code><object-name></code>	is the netlist name of the object (port, net or instance) being operated on.
<code><operation></code>	is one of attribute or instance.

`<name>` is the name of a definition or an attribute.

`<value>` is only used in attribute operations, and is the value to be given to the attribute. This field is only required when an attribute requires a value.

The following are specific syntax forms of all valid attribute or instance assignments.

`port <name> attribute <name> <value>`

Attribute `<name>` with (optional) value `<value>` is added to the port instance (the instance driven by the formal port). Only works with input and ports.

`port <name> instance <name>`

The port instance (the instance driven by the formal port) is replaced by an instance of the given definition. The only valid definitions are IPCLK, IPRST. This syntax is limited to input ports with 1 input pin and 1 output pin (IPBUF for example).

`net <name> attribute <name> <value>`

Attribute `<name>` with (optional) value `<value>` is added to the net.

`net <name> instance <name>`

Creates an instance of the given definition and inserts it into the named net. The only valid definitions are ACLK, ARST, APBUF and PABUF.

`instance <name> attribute <name> <value>`

Attribute `<name>` with (optional) value `<value>` is added to the instance

4



Example .PAT File Entries

```

// This is a comment
# This is a comment as well

port sega attribute dp1d_pad_place 22
//This results in the IPBUF being placed on the pad associated with package //pin 22.

port sega attribute dp1d_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the net segment associated with the formal port sega. A dummy
//argument is required for this attribute.

port sega instance ipclk
//This results in the IPBUF being replaced by an IPCLK, forcing the net onto
//a clock network.

net segb attribute dp1d_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the entire net "segb". A dummy argument is required for this
//attribute.

net segb attribute dp1d_cluster_seed 1
//The dp1d_cluster_seed attribute and value shown get assigned to the net B
//for evaluation during place and route.

net segb attribute dp1d_place_priority 1
//The dp1d_place_priority attribute and value shown get assigned to the net
//B for evaluation during place and route.

net segb instance aclk
//This results in an ACLK buffer being inserted between IPBUF's output and
//BUF's input. BUF is now driven off the resulting clock routing
//resource.

instance $1I4 attribute dp1d_opdrive 12ma
//This results in the OPBUF getting 12ma drive capability.

instance $1I12 attribute dp1d_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the all nets associated with the instance $1I12. A dummy
//argument is required for this attribute.

```

4



All Valid Combinations of Attributes & Instances in an External .PAT File

The following show all the valid combinations of the attributes in the .PAT file.

port <name> instance (name here can only refer to input instances with one input pin and one output pin)

ipclk
iprst

port <name> attribute

dpld_ignore_timing dummy_arg
dpld_pad_place <value, see data book for package being used>
dpld_pup 1|2|BOTH
pullup 1|0
pulldown 1|0
dpld_opdrive 6ma|12ma
dpld_oplevel 3v|5v
dpld_opslew high|low
dpld_iplevel CMOS|TTL
dpld_pad_properties 0|1, 0|1, CMOS|TTL, 3v|5v, 6ma|12ma, high|low

net <name> attribute

dpld_cluster_seed n (where $0 \leq n \leq 1000$, 1 is default, 0 means ignore net)
dpld_place_priority n (where $1 \leq n \leq 10$, 1 is default)

net <name> instance

ack
apbuf
arst
pabuf

instance <name> attribute

dpld_ignore_timing dummy_arg
dpld_pad_place <value, see data book for package being used>
dpld_pup 1|2|BOTH
pullup 1|0
pulldown 1|0
dpld_opdrive 6ma|12ma
dpld_oplevel 3v|5v
dpld_opslew high|low
dpld_iplevel CMOS|TTL
dpld_pad_properties 0|1, 0|1, CMOS|TTL, 3v|5v, 6ma|12ma, high|low

4



Using OrCAD's Capture and Simulate with the MPA Design System

Prepared by
Derrick H.J. Klotz
Motorola Field Applications Engineer

4



Using OrCAD's Capture and Simulate with the MPA Design System

Introduction

The Motorola family of MPA devices and supporting software provides system designers with a collection of flexible and powerful tools. This application note focuses on the use of OrCAD's Capture and Simulate programs as the front end schematic capture and logic simulation design tools for the MPA Design System FPGA place and route software. A general overview of the following topics is provided:

- Libraries
- File Naming Convention
- Schematic Capture
- MPA Software Attributes
- Timing Control
- MPA Hardware Features
- Netlist Export
- Functional Simulation
- EDIF Netlist Import
- MPA Autolayout
- Layout Viewer
- Device Configuration
- Back Annotation and Simulation

This application note is intended to be used as an overall MPA design reference. Much of the information here is also available via the on-line help but has been included here in order to be complete.

The reader is assumed to be familiar with OrCAD Capture and Simulate and have casual knowledge of the MPA Design System. More detailed coverage of the above topics can be found in the appropriate documentation and on-line help.

Libraries

The EDIF netlist reader of the MPA Design System is currently constrained to understand only those components passed to it from the MACROLIB, MICROLIB and MPA1000 libraries provided. It is recommended that these library files be located within an "MPA" subdirectory under OrCAD's Capture library and Simulate library directories, as shown in Figure 1.

Only components from these three libraries can be used for MPA schematic design. The only other symbols which may be utilized directly in the schematic are the hierarchical ports and off-page connectors found in OrCAD's CAPSYM library. Refer to the appropriate OrCAD reference documentation regarding the proper use of these components.

File Naming Convention

One important consideration that the designer must be aware of is the file and directory naming conventions which are used by OrCAD and the MPA Design System. By default, OrCAD Capture uses the ".dsn" extension for schematic design files. This is the same extension used for MPA design context files.

In order to avoid confusion, a recommended user's file naming convention is illustrated in Figure 2. All user provided files are named "project.xxx" and are located in a subdirectory called "example". The OrCAD Capture schematic design file, "project.dsn", and VHDL netlist file used by OrCAD Simulate, "project.vhd", are not directly required by the MPA Design System and, hence, don't need to be in the "example" subdirectory, but are shown here as a matter of convenience.

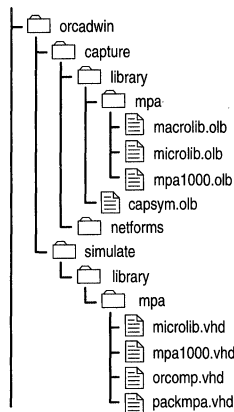


Figure 1. MPA Library Files

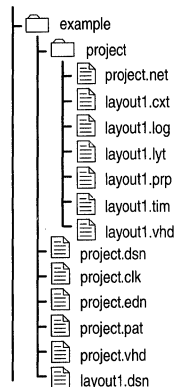


Figure 2. MPA Design Files

4



A brief description of each file follows. More details regarding each file can be found in the MPA Design System on-line help facility. Note that the "project" subdirectory and "project.net" file are not provided by the user but are created by the MPA Design System.

Files provided by the designer:

- project.dsn (optional) OrCAD Schematic
- project.clk (optional) MPA-DS Timing Control
- project.edn EDIF Netlist (from OrCAD to MPA-DS)
- project.pat (optional) MPA-DS External Attributes
- project.vhd (optional) OrCAD VHDL (simulation)

Files and directories created by the MPA Design System:

- layout1.dsn Design Context
- project Autolayout Results Subdirectory
- project.net MPA-DS Netlist
- layout1.cxt Context
- layout1.log Log
- layout1.lyt Layout
- layout1.prp Port Report
- layout1.tim Timing Report
- layout1.vhd VHDL Back Annotation

4

Schematic Capture

There are just a few unique steps to take during schematic capture to ensure a valid MPA Design System EDIF netlist. Components placed in a schematic will have their OrCAD Primitive property set to "Default". To assure that netlist generation descends properly to the MPA primitive level, hierarchy for parts as well as hierarchical blocks must be set to "Nonprimitive". This is achieved in OrCAD Capture via the Hierarchy tab on the Design Template dialog box, which is accessed by choosing Design Properties from the design manager's Option menu.

The power and ground power connectors supplied with OrCAD must not be used in MPA schematics. Instead, "ONE" must be used for logical one connections and "ZERO" must be used for logic zero connections. For resistive pull-ups on wired-or nets, "WPU" is employed. These components are found in the MPA MICROLIB library. "PWPUP" is put to use as a resistive pull-up on wired-or peripheral bus nets, and can be found in the MPA1000 library.

The provided macro library elements must not be changed. If they are altered, the elements may not perform as described in the documentation help for the MPA Design System. Library elements are marked "LIBRARY COPY - DO NOT MODIFY" for identification. If accidentally modified, a library element can be retrieved by re-installing the MPA Design System libraries.

The netlist importer of the MPA Design System needs to recognize the design's I/O pins. To accomplish this, hierarchical ports must be added to the MPA1000 I/O components, as shown in Figure 3. The hierarchical port type must also be of the correct type (i.e., "input", "output", etc.). Running the Design Rules Check will verify conformance to the electrical rules required for netlist generation.

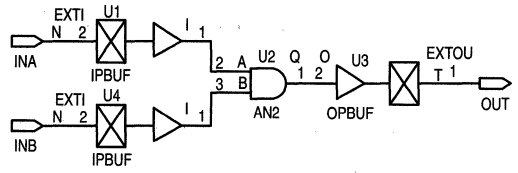


Figure 3. I/O Pin Hierarchical Port Example

MPA Software Attributes

The MPA Design System's import process can accept a set of attributes to help the designer tune the layout and routing processes. The system also accepts I/O parameters to specify CMOS/TTL compatibility, I/O drive, package pin assignments and slew rate control. Declaring attributes in the schematic will result in their being passed into the EDIF netlist and then imported into the MPA Design System. Optionally, the designer may prefer to include attributes in an External Attributes file (.pat) of the same name as the design. The designer may choose to use the combination of the two methods, but it should be noted that attributes passed into the MPA Design System in ".pat" files will always take precedence if declared in both places.

The MPA Design System, for historical reasons, will accept attribute names with or without a "DPLD_" prefix. For example, IGNORE_TIMING can also be written as DPLD_IGNORE_TIMING.

Place and Route Layout Attributes

The MPA Design System enables the tuning of place and route algorithms in three ways. The first is with the adjustment of the Auto Layout tool options such as annealing temperature, target delays, target zone utilization, etc. Additional details on the available options and their use are available in the on-line help facility of the MPA Design System.

The second method involves the construction of separate clock files. Here again, additional information is provided in the on-line help system and is not presented in this application note. The third method of influencing place and route results is the inclusion of the following attributes in the schematic, or in an external ".pat" file.

Table 1. Valid Attributes

Schematic Component	Attached Place and Route Attribute	Attached I/O Attribute
Net	DPLD_IGNORE_TIMING DPLD_CLUSTER_SEED DPLD_PLACE_PRIORITY	
Symbol	DPLD_IGNORE_TIMING DPLD_PAD_PLACE	PULLUP or PULLDOWN DPLD_OPDRIVE DPLD_OPLEVEL DPLD OPSLEW DPLD_IPLLEVEL DPLD_PAD_PROPERTIES
Formal Port	IGNORE_TIMING	



IGNORE_TIMING

The IGNORE_TIMING attribute is used to inform the tool which nets to ignore timing on. It may be set on a symbol (instance), a net or an external pin (formal port). If a net has the attribute set, then all delay paths associated with that net are ignored. If an instance has the attribute set, then all input delay paths driving and all delay paths being driven from that instance are ignored. Assigning the attribute to a formal port has exactly the same effects as assigning to the I/O instance itself. Once all the objects to be ignored have been identified, their paths are propagated forwards and backwards through combinatorial gates until clocked objects (or top level circuit I/O) are reached. The result is that additional segments other than those explicitly specified may be ignored for timing purposes as well.

You are required to use a dummy value with this attribute, but the value stated is otherwise ignored.

Assigning this attribute to a symbol, net or formal port frees the timing driven autolayout algorithms to more optimally cluster, place and route the speed critical nets.

CLUSTER_SEED

CLUSTER_SEED is used to assign a cluster seed to a net. This will cause the autolayout clustering algorithm to treat all instances that connect to that net specially. The action taken depends on the value of the attribute, as follows:

- 0 Ignore this net during clustering. Setting this attribute on a net is likely to cause the net to be implemented in global interconnect.
- 1 Default operation
- >1 Weight this net by the given factor in the clustering.

The maximum value for the CLUSTER_SEED attribute is 1000. Example: CLUSTER_SEED = 2.

PLACE_PRIORITY

The PLACE_PRIORITY attribute can be applied to a net to increase the chance that the autolayout algorithm will route the net more efficiently and with less delay in a physically smaller area, at the possible expense of surrounding nets. The value assigned to PLACE_PRIORITY should be an integer in the range of 1 (default) to 10 inclusive. Higher values of place priority allow the designer to prioritize nets relative to each other. Example: PLACE_PRIORITY = 2.

PUP

PUP attaches to WPUP primitive cells only, to select either or both of the pull-up resistors available in a WPUP cell. Valid values are 1, 2, or BOTH. Example: PUP = BOTH.

I/O Parameter Attributes

The following attributes can be applied to I/O cells only:

PAD_PLACE

PAD_PLACE instructs the I/O pad to be allocated to the package pin number specified. Only one pad may be allocated to any pin. Automatic placement of I/O pads usually results in a better layout, so this attribute should only be added when it is necessary. Example: PAD_PLACE = 1.

PULLUP

Setting PULLUP to 1 will enable the pull-up resistor on the external pin of the I/O pad. Default is 0 (resistor disabled). Example: PULLUP = 1.

PULLDOWN

Setting PULLDOWN to 1 will enable the pulldown resistor on the external pin of the I/O pad. Default is 0 (resistor disabled).

Example: PULLDOWN = 1.

DPLD_OPDRIVE

DPLD_OPDRIVE sets the output drive current of an output or bi-directional pad to either 6mA (default) or 12mA.

Example: DPLD_OPDRIVE = 12mA.

DPLD_OPLEVEL

DPLD_OPLEVEL sets the output voltage level of an output or bi-directional pad to either 5v (default) or 3v.

Example: DPLD_OPLEVEL = 3v.

DPLD OPSLEW

DPLD OPSLEW sets the output slew rate (transition speed) of an output of bi-directional pad to high (default, slew rate limiting off) or low (slew rate limiting on).

Example: DPLD OPSLEW = high.

DPLD IPLEVEL

DPLD IPLEVEL sets the input threshold voltage of an input or bi-directional pad to either TTL (default) or CMOS. Example: DPLD IPLEVEL = CMOS.

DPLD_PAD_PROPERTIES

OrCAD Capture allows the use of a combined attribute, DPLD_PAD_PROPERTIES, that combines the following attributes into a single comma separated list: PULLUP, PULLDOWN, DPLD IPLEVEL, DPLD_OPLEVEL, DPLD_OPDRIVE, DPLD OPSLEW. Example: DPLD_PAD_PROPERTIES = 0,0,CMOS,5v,12mA,high.

Defaults and Invalid Combinations

The default I/O pad attributes have been selected so that they can be connected to either TTL or 5v CMOS without adjustment. The default parameters may not be ideal for every design, and they should be matched to the application in order to achieve the best performance and noise immunity. 3v CMOS users should be especially careful to set OPLEVEL to 3v, otherwise damage to peripheral IC's may result.

The following combinations of user attributes are not permitted:

```
DPLD_OPDRIVE = 6mA
and
DPLD OPSLEW = low.
PULLUP = 1 and PULLDOWN = 1.
```

The following combination of user attributes on a single bi-directional pad should be avoided, as it may produce unpredictable results:

```
DPLD IPLEVEL = CMOS
and
DPLD_OPLEVEL = 3v.
```

Assigning Attributes in a Schematic

Any of the attributes listed above can be assigned to pins, nets and macro symbols as appropriate. With OrCAD

4



Capture, the method of assigning attributes is straight forward by adding user-defined properties.

The Edit Part dialog appears after double clicking on the object with the mouse, or by selecting the object and choosing Properties from the Edit menu (or "Ctrl+E"). Choosing the User Properties button and, in the User Properties dialog box that displays, picking the New button activates the New Property dialog box. The attribute name and its value can then be entered (as above, case insensitive). Choosing OK completes the task and the attribute will be included in the EDIF netlist. OrCAD Capture also allows the attribute to be visible if desired.

Multiple objects can be edited simultaneously by using the spreadsheet editor in OrCAD's Capture. In this case, the desired objects are all selected together and Properties is chosen from the Edit menu (or "Ctrl+E"). The spreadsheet editor eases the process of assigning the same user property to all selected objects. Make sure these objects are all the same type (i.e., nets, symbols, etc.).

After a property has been assigned, it can be modified later with either of the above methods. Viewing and changing the same property across multiple objects is greatly simplified and accelerated by using the spreadsheet editor.

4

Assigning Attributes in an External ".pat" File

Assigning attributes to a long series of pins, or a variety of nets directly to the schematic can be time consuming and may be error prone. The MPA Design System gives the designer the option to enter all the valid attributes in an external attributes text file (extension ".pat"). Entries in the ".pat" file take precedence over any attributes that may have also been instantiated in the EDIT netlist via schematic entry.

The external attributes file supports three main operations:

- 1) Insertion of attributes to specify pin placements and pad characteristics.
- 2) Insertion of special pad cells, IPCLK/IPRST, to drive the primary clock/reset network.
- 3) Insertion of special buffer primitives into a named net.

This has two uses:

- a) Force a named net onto/off the peripheral bus, by inserting the primitives APBUF/PABUF respectively.
- b) Force a named net onto the primary clock/reset network, by inserting the primitives ACLK/ARST respectively.

The external attributes file must exist in the same directory and with the same name as the corresponding EDIF netlist, with the file extension ".pat". During import, the system automatically checks for the existence of a ".pat" file and uses it when one is found.

Syntax of the External Attributes (".pat") File

The external attributes file contains a list of commands, one per line. Each command contains up to five fields, as follows:

<object-class> <object-name> <operation> <name> [<value>]

where:

<object-class>	is one of port, net or instance.
<object-name>	is the netlist name of the object (port, net or instance) being operated on.
<operation>	is one of attribute or instance.
<name>	is the name of a definition or an attribute.
<value>	is only used in attribute operations, and is the value to be given to the attribute. This field is only required when an attribute requires a value.

The following are specific syntax forms of all valid attributes or instance assignments:

port <name> attribute <name> [<value>]

Attribute <name> with (optional) value <value> is added to the port instance (the instance driven by the formal port). Only works with input and ports.

port <name> instance <name>

The port instance (the instance driven by the formal port) is replaced by an instance of the given definition. The only valid definitions are IPCLK and IPRST. This syntax is limited to input ports with 1 input pin and 1 output pin (IPBUF for example).

net <name> attribute <name> [<value>]

Attribute <name> with (optional) value <value> is added to the net.

net <name> instance <name>

Creates an instance of the given definition and inserts it into the named net. The only valid definitions are ACLK, ARST, APBUF, and PABUF.

instance <name> attribute <name> [<value>]

Attribute <name> with (optional) value <value> is added to the instance.

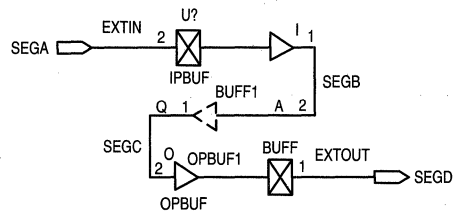


Figure 4. The ".pat" File Attributes are Added to this Simple Schematic

Example ".pat" File Entries

The following sample ".pat" file entries reference the simple schematic shown in Figure 4.



```
// This is a comment
# This is a comment as well
port sega attribute pad_place 22
// This results in the IPBUF being placed on the pad associated with package
// pin 22.
port sega attribute ignore_timing dummy_arg
// For place and route purposes, the design's timing parameters are ignored
// for the net segment associated with the formal port sega. A dummy
// argument is required for this attribute.
port sega instance ipclk
// This results in the IPBUF being replaced by an IPCLK, forcing the net onto
// a clock network.
net segb attribute cluster_seed 1
// The cluster_seed attribute and value shown get assigned to the net B for
// evaluation during place and route.
net segb attribute place_priority 1
// The place_priority attribute and value shown get assigned to the net B for
// evaluation during place and route.
net segb instance aclk
// This results in an ACLK buffer being inserted between IPBUF's output and
// BUFF's input. BUFF is now driven off the resulting clock routing resource.
instance OPBUF1 attribute opdrive 12mA
// This results in the OPBUF getting 12mA drive capability.
instance BUFF1 attribute ignore_timing dummy_arg
// For place and route purposes, the design's timing parameters are ignored
// for all nets associated with the instance BUFF1. A dummy argument is
// required for this attribute.
```

4

All Valid Combinations of Attributes & Instances in an External ".pat" File

The following show all the valid combinations of the attributes in the .PAT file.

port <name> instance (name here can only refer to input instances with one input pin and one output pin)

ipclk
iprst

port <name> attribute

ignore_timing dummy_arg
dpld_pad_place <value> (refer to data book for package being used)
pullup 110
pulldown 110
dpld_opdrive 6mA112mA
dpld_oplevel 3v15v
dpld_opslew highlow
dpld_iplevel CMOS/TTL
dpld_pad_properties (see above paragraph describing this attribute)

net <name> attribute

dpld_cluster_seed n (where $0 \leq n \leq 1000$, 1 is default, 0 means ignore net)
dpld_place_priority n (where $0 \leq n \leq 10$, 1 is default)

net <name> instance

aclk
apbuf
arst
pabuf

instance <name> attribute

dpld_ignore_timing dummy_arg
dpld_pad_place <value> (refer to data book for package being used)
pullup 110
pulldown 110
dpld_opdrive 6mA112mA
dpld_oplevel 3v15v
dpld_opslew highlow
dpld_iplevel CMOS/TTL
dpld_pad_properties (see above paragraph describing this attribute)

Timing Control

The MPA clock frequency determines or is determined by the performance of the design. Any signal starting at an input or flip-flop, propagating through some combinatorial logic, and arriving at an output of a flip-flop, must arrive within one clock cycle.

There are two clocking strategies employed by the MPA Design System. Design critical clocking signals should be implemented via the primary clocking network for guaranteed low skew. This network encourages the use of synchronous design techniques and, combined with the flip-flop rich architecture, eases the inclusion of internal scan paths in the design. Note that the I/O flip-flop clocks and I/O latch enables can only be driven by the primary clock network.

Secondary clocks are those signals not implemented via the primary clocking structure and will be routed as a tree structure so as to minimize any skew between the clock inputs driven by the net.

More control over individual clock net speeds can be obtained by specifying a Clock File (".clk"). A Clock File (also referred to as a "Timing Group File") must have one Timing



Group Definition and may have one or more Intended Target Delays. The Timing Group Definition must have at least one Timing Group.

Each Timing Group must have a unique name and may be a clocked timing group. A Timing Group Instance List consists of individual reference statements defining the instances assigned to each timing group. Each statement is one of either "instanceRef", "netRef", "allFlipFlop", or "allIO". There is no limit to how many statements can be assigned to each Timing Group.

```
( timingGroupDefinitions
  ( timingGroup timingGroupDef
    [( clock [phase] period )]
    ( instanceRef instanceName )
    | ( netRef netName [clockInputSense] )
    | ( allFlipFlops [clockInputSense] )
    | ( allIO )
  )
)
[( intendedTargetDelay targetDelay driverGroupRef drivenGroupRef )]...
```

- where,
- timingGroupDef* - is a text string identifying a timing group.
 - phase* - is an integer defining the clock phase shift in units of 100ps.
 - period* - is an integer defining the clock period in units of 100ps.
 - instanceName* - is a text string identifying an instance or a group of instances.
 - netName* - is a text string identifying a net or a group of nets.
 - clockInputSense* - is either "NONINVERTING" or "INVERTING".
 - targetDelay* - is an integer defining the intended target delay in units of 100ps.
 - driverGroupRef* - is a text string identifying a source timing group.
 - drivenGroupRef* - is a text string identifying a destination timing group.

convention explanation:

- normal - indicates keywords to be entered as shown.
- italics* - indicates variable information specified by the user.
- [] - square brackets enclose an optional symbol or symbols.
- | - a vertical bar indicates a choice between alternatives.
- ... - an ellipsis indicates a repetitive structure.

4

An Intended Target Delay is a guide to the approximate delay intended by the user between two timing groups. If both timing groups are clocked, the MPA Design System will adjust this value to take into account skew. Otherwise, the intended value will be the actual value.

The Clock File syntax is summarized in the following listing. Please refer to the available on-line help facility for a more in-depth description.

MPA Hardware Features

Most designs can be fit to the desired speed into the MPA family without the designer needing to know much about the details of the device's internal construction. However, to get the most out of the MPA, some time should be taken to browse through the on-line help files thoroughly. The help files are rich in detail regarding the hardware specific macros and routing resources macros inherent in the MPA Design System. The following topics are presented as simple examples on most of these hardware specific features. For an exhaustive presentation, please refer to the on-line help facility.

Logic One and Logic Zero

It may at times be necessary to modify the functionality of a macro from the supplied libraries by tying one or more of its inputs to logic high or low. There are two special elements provided in MICROLIB for this purpose. ONE produces a logic high to all input pins tied to it. The ZERO element provides a logic low. There are no fan-out restrictions for these signals. An alternate method is to tie the input pins requiring a logic high to a net named VDD, and tie the input pins requiring a logic low to a net named GND.

For all the above methods, the MPA Design System will recognize the logic as static and eliminate superfluous logic elements wherever possible during the place and route process.

Wired-OR

The MPA family allows for connecting many outputs to a single common signal line. The available macros for this function are: WND2, WINV, WOR2, and WBUF. When using these macros, a WPUP component is required. The maximum number of connections to a single signal is given by:

$$\frac{1}{2} \sqrt{\# \text{ of core cells.}}$$

Table 2. Maximum Drivers on a Wired-OR Signal

MPA Family Member	Max. Drivers
MPA1016	20
MPA1036	30
MPA1064	40
MPA1100	50

If the number of drivers on the Wired-OR net is below half of the maximum allowed, then only a single WPUP element is



recommended. Adding a second WPUP to a very large net, will help speed things up some, but at the cost of increased power consumption. The allowable values for the WPUP attribute are 1, 2, or BOTH. Resistors 1 & 2 are of equal value, so it makes no difference which one you select. BOTH ties resistors 1 & 2 in parallel and decreases the low to high transition time, but at the expense of extra power consumption.

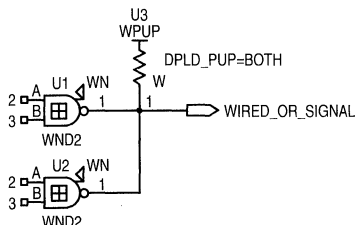


Figure 5. Simple Wired-OR Net

Adding additional Wired-OR outputs to the net WIRED_OR_SIGNAL will slow it down. Adding additional inputs to the net has little effect on speed. Low to high transitions are typically slower than high to low transitions on a Wired-OR signal.

Peripheral Bus

The periphery of the MPA die is bordered with an 8 bit wide Peripheral Bus (P-Bus). The P-Bus can be broken at each corner of the array by switches. (The setting of these switches is handled automatically in the MPA Design System software.) Each of the resulting 4 segments has two programmable pull up circuits, one at each end. The P-Bus is ideal for routing common signals to many I/O macros.

It is easy to build a scenario where many I/O pins have a single or several control signals in common. In this instance, that signal can be placed on the P-Bus by using an APBUF. Conversely, pulling a signal off of the P-Bus back into the array is accomplished by using the PABUF.

The ability to construct Wired-OR nets is not limited to signals internal to the array. P-Bus Wired-OR nets can be constructed using APWBUF (or APWINV) and their associated pull-up structure PWPUP. However, using these features requires the user to be aware of the natural consequences of adding capacitance and pull-ups to a resistive bus. Adding additional segments of P-Bus (by assigning I/Os to different edges of the die) increases capacitance that effects rise and especially fall times. Also, the somewhat resistive nature of the P-Bus can cause Vol noise margin problems if the active low P-Bus driver is far from the

pull-up element and driven element. The assignment of P-Bus pull-up resistors is automatic.

On import, all PWPUP instances defined by the designer are removed from the netlist. The tool automatically balances the number of pull-up resistors against the P-Bus loading (incurred by the use of multiple die edges) by automatically inserting additional pull-up capacity for each P-Bus segment used. During autolayout, the MPA Design System re-attaches a single peripheral bus pull-up resistor per occupied die edge, for each unique P-Bus signal. For example, if several I/Os that use a P-Bus Wired-OR signal get split between the 'top' and 'right' edges of the die during autoplacement, the tool would assign two pull-up resistors to the net.

The Wired-OR resources are provided to help simplify some logic designs, however, their use should be avoided on speed critical paths.

Low Skew, Clock, and Reset Nets

For a high fan out signal or for a signal with a desired skew limited to less than 1ns, the signal can be moved onto a clock network using the ACLK or ARST macro (they both do the same thing).

MPA I/O Structures

The standard I/O cell of the MPA array is very feature rich. The complexity of this structure is apparent in the large number of choices available in the I/O macro library MPA1000. (Here again, the reader is encouraged to invest some time in the on-line help facility, in particular "Help on Libraries - Input and Output Pads" and "Help on Device - Functional Description - I/O Cell". Defining a bookmark for these two particularly useful help sections will provide a short cut for referencing them in the future.)

Each of the macros in the MPA1000 library fit in a single I/O cell. Couple these relatively complex functions with the availability of P-Bus routing resources (including the P-Bus Wired-OR resources previously mentioned) and it becomes clear that significant functionality can be achieved before ever using the normal internal resources of the array.

In Figure 6, a three bit address decoder was implemented using only the resources available in the complex I/O Cells and the associated P-Bus. No internal logic or routing resources were consumed. The features used which are unique to the I/O Cell and P-Bus include: input delay to synchronize external data with the buffered clock signal, APBUF used to bus a common enable to signal to several I/O sites, XNOR used to compare external and internal address values, and finally the Wired-OR P-Bus line.

Of course, all of this functionality could be moved internal to the array, using only the simple I/O macros.

4



4

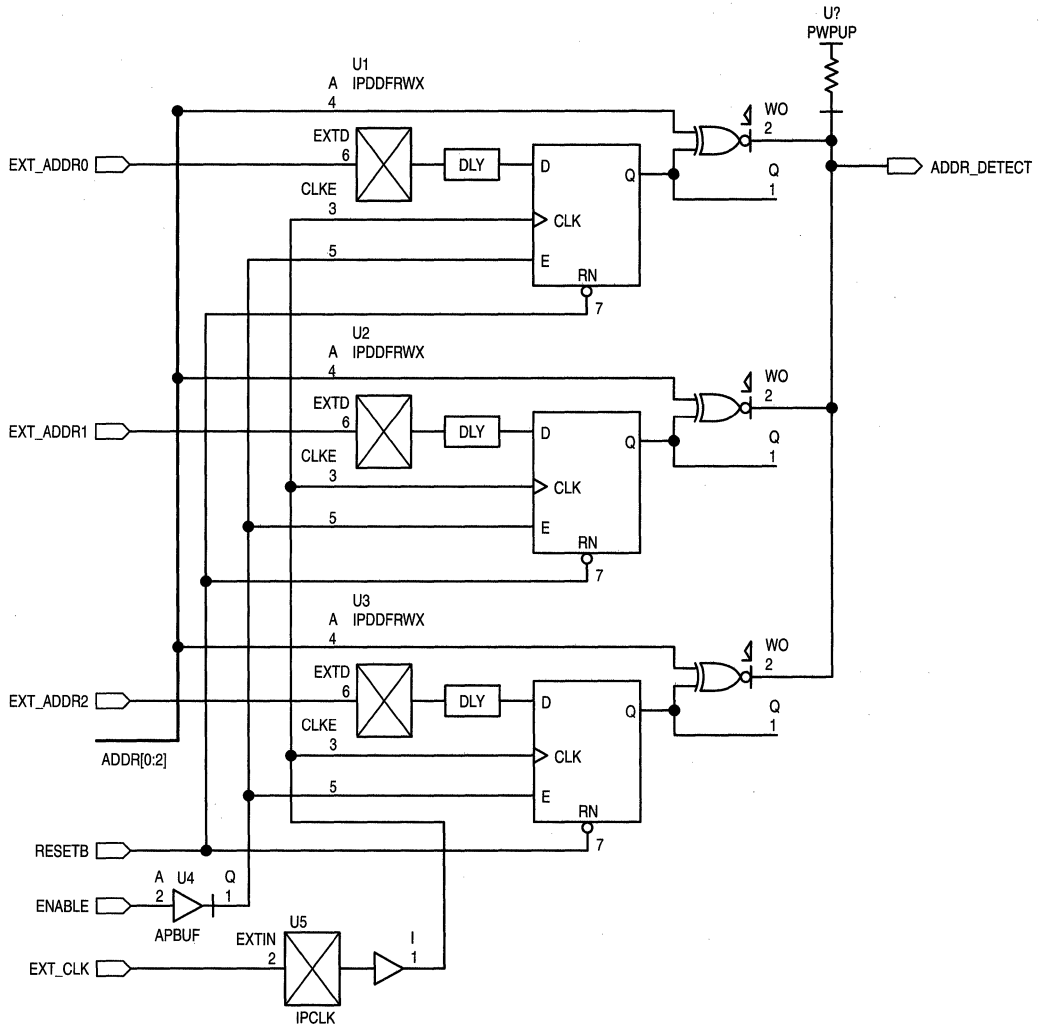


Figure 6. A Three Bit Address Decoder, Only I/O Cell and P-Bus Resources Used

Net List Export

The OrCAD Create Netlist tool in the design manager's Tool menu is used to translate a completed schematic into an EDIF file ("*.edn") which is then importable to the MPA Design System. Unlike some other netlist formats available and supported by OrCAD, creating an EDIF 2 0 0 netlist will work properly in either logical or physical view, regardless of whether the design is flat or hierarchical.

Selecting the EDIF 2 0 0 tab in the Create Netlist dialog box presents the user with various options. Part Value must be specified as {Value} (the default) and PCB Footprint is inconsequential. "Allow nonEDIF characters" must not be

enabled as the MPA Design System Import tool will only accept EDIF characters. All other options should be enabled to allow any special attributes to be included in the netlist. The selected options are:

- Output pin names (instead of pin numbers)
- Do not create "external" library declaration
- Output net properties
- Output part properties
- Output pin properties

Successful execution of the netlist tool will generate a file with an ".edn" extension; for example: "project.edn". This is the file that gets imported into the MPA Design System.



Functional Simulation

Prior to creating an EDIF netlist for importing to the MPA Design System, the designer should verify circuit operation. Generating a VHDL netlist (".vhd") is required for functional simulation with OrCAD's Simulate. Both netlists will describe the circuit as required and can be produced at the same time in the design flow if desired. Please refer to the appropriate documentation for a details.

MPA Netlist Import

Netlist import is the process used by the MPA Design System to import a design produced from a front-end design capture package into the Autolayout Tool. The Netlist Import tool runs in a standard Tool Execution Window. Any errors encountered during netlist import will be output to this window.

When a netlist is first imported into the system, it is processed through the following phases to prepare the circuit for autolayout:

- LPM Instantiation
- Addition of External Attributes
- Netlist Checking
- Retargeting
- Wired-or Net Splitting

A more detailed description of each phase is provided in the MPA Design System on-line help facility. When comparing the original schematic with the final chip layout, the designer should be aware that the MPA Design System does some "bubble pushing" during the retargeting/fitting of the logical design into the physical implementation. In some cases, redundant gates in the schematic design are eliminated in the physical layout.

Once the netlist has been imported into the system, Autolayout can be applied to the netlist repeatedly without having to re-import.

MPA Autolayout

The Autolayout Tool performs fully automatic timing driven layout for the current design. The Autolayout Tool runs in a Tool Execution Window. Any errors encountered during layout will be output to this window.

Control over the autolayout process is provided through the autolayout parameters and attributes that can be added to the netlist. Before the autolayout process begins the layout, the system performs definition checking to make sure that the definition is suitable for autolayout.

The autolayout process goes through several distinct phases and generates a context file after each phase is complete. When autolayout is complete, the following files are generated:

- A timing report file (".trm") to allow for further analysis of the timing aspects of the layout,
- A port report file (".prp") is also generated giving the pinout of the resulting autolayout,
- A back-annotation file, giving delay information for the layout suitable for input to a logic simulator, and, of course,
- A layout file (".lyt"), containing the final layout. This layout is used as the input to the configuration

generator to create a configuration bit-stream for the target device.

The area of the chip the layout will use is determined by the circuit size and the autolayout parameters specified for the design.

Autolayout Options

Autolayout options are set using the Autolayout pane of the Tool Options dialog. The choices available are outlined here:

Parameter Groups

Autolayout Parameter groups are used to reference a particular set of parameter settings. Any number of Autolayout Parameter Groups can be set up for easy access to commonly used parameter settings. These parameter groups can be selected from the Autolayout options dialog.

Utilization

This parameter controls the number of functional cells that the partitioning operation will attempt to place in a 100 core cell area (i.e., a zone). Its value is the percentage of functional core cells in the imported design against the recommended maximum number of usable core cells (= 50). Autolayout will automatically exceed this limit if it has to.

Minimum:	1
Maximum:	150
Default:	80

4

Effort

This parameter controls the amount of work applied to the partitioning operation. A larger value will generally produce a better result in the partitioning for high utilization circuits. The time taken for partitioning is directly related to Effort. Effort must be balanced by the Start Temp parameter for optimum results.

Minimum:	1
Maximum:	100
Default:	30

Start Temp

Partitioning is performed using a simulated annealing technique. The Start Temp parameter controls the start temperature for that process. A higher temperature allows the average cost of a move during partitioning to be greater. Increasing the start temperature may result in a better partition, especially in high Utilization designs. Note that a high value of Start Temp may result in a bad partition if not used in conjunction with a larger value for the Effort parameter.

Minimum:	1
Maximum:	100
Default:	10

Simulated annealing is a software technique used for combinatorial optimization. It simulates the relaxation of stresses during the repeated heating and cooling of materials, where the temperature that the material is heated to each time is gradually reduced. Simulated annealing is a stochastic technique, and will produce different results for different sets of starting conditions (such as layout parameters, seed values, etc.). Simulated annealing is used by the MPA Design



System during the Partitioning and Zone Routing phases of autolayout.

Attempts

This parameter controls the number of runs through the partitioning phase. More runs should deliver a better partition.

Minimum:	1
Maximum:	20
Default:	1

Backoff

This parameter controls the percentage relaxation of the target delay if the initial target is not obtainable. This is expressed as a percentage of the previous target delay.

Minimum:	0%
Maximum:	100%
Default:	30%

Increasing this parameter makes the next attempt at autolayout use a new Delay Target further away from the maximum possible delay thus making it easier to complete. This is expressed as a percentage increase in the optimum target delay.

4

Delay Cost

This parameter is the weighting given to timing during partitioning. If a particular partition includes a net that is failing to meet its timing target, then the cost of that partition will be artificially raised by an amount proportional to the delay cost. Increasing the delay cost is likely to trade off against achievable Utilization.

Minimum:	0
Maximum:	100
Default:	5

Delay Target

This parameter is the desired maximum delay target for the autolayout process to achieve, in steps of 0.1ns. For combinatorial circuits, the maximum delay is calculated from the longest path from input to output. For synchronous circuits, the maximum delay is calculated from the longest path between flip-flops, or from I/O to flip-flop, whichever is the greater. This parameter is only relevant when you are using the standard timing model. If a clock file is specified, then the Delay Target parameter will be ignored.

Minimum:	0
Maximum:	9999
Default:	50 (5ns)

Fanout

This parameter is the maximum fan-out of a net which is included in the clustering. All the nets which have a fan-out greater than this value are ignored during clustering. Any nets ignored during clustering are more likely to have to be globally routed.

Minimum:	1
Maximum:	100
Default:	2

Min Zone Delay

This parameter forces the delay of segments within a zone to be within a certain value. The delay is specified in unit of 0.1ns. The smaller the delay is, the harder the zone routing algorithm will work to reduce delays within the zones.

Minimum:	0
Maximum:	9999
Default:	5 (0.5ns)

Seed

Setting the seed for the autolayout process lets you control the repeatability of the autolayout. If you run the autolayout with a given seed value, you can reproduce the same autolayout from the same circuit simply by setting the same seed value.

Minimum:	0
Maximum:	9999
Default:	1

Several of the autolayout algorithms use stochastic techniques similar to simulated annealing that require a pseudo-random number generator. The Seed is used to set the random number generator to a known point in the sequence.

Layout Viewer

The MPA Design System provides a Layout Viewer tool which gives a graphical view of the completed layout. This view shows the instances of primitives in the design and the routing that connects them together. Aspects of the device architecture are also shown.

Multiple graphical and browser views may be created. Moving about in the graphical view is made possible by zooming and panning. Instances, nets and ports can be selected. The mouse cursor changes according to the design object under it, and the status bar gives more information about the design object beneath the cursor.

Note that no changes to the layout are possible with the Layout Viewer. Please refer to the on-line help facility for more details.

Device Configuration

Device configuration is the process of translating a completed layout into a stream of 1's and 0's used to program the actual device. Pressing the device configuration button launches this tool. Please refer to the on-line help facility for more details.

Back Annotation and Simulation

The MPA Design System has a facility that provides post place and route back annotation data in a VHDL format (".vhd"). This file has some regular VHDL constructs which unfortunately are not currently supported by OrCAD's Simulate. Back annotation with OrCAD's Simulate is planned for support in the near future.



Using VIEWlogic's Workview Office 7.0 with the MPA Design System

Prepared by
Marten L. Smith
Motorola Programmable Logic

4



Using VIEWlogic's Workview Office 7.0 with the MPA Design System

Introduction

The Motorola family of MPA devices and supporting software provide system designers with a collection of flexible and powerful tools. This application note focuses on the use of VIEWlogic's Workview Office 7.0 schematic capture and simulation programs as front end design tools for the MPA Design System's FPGA place and route software. A basic design flow is introduced followed by more in depth discussion of parameters for place and route and concludes with a discussion on back annotation and simulation procedures.

Basic Design Flow

In this simplest example of using Workview Office, a straight path is taken from design entry through export to the MPA Design System. More detailed discussions on: place and route parameters, I/O parameters, hardware dependent macros, back annotation and simulation are deferred. The reader is assumed to be familiar with Workview Office and have casual knowledge of the MPA Design System.

Libraries

The EDIF net list reader of the MPA Design System is currently constrained to understand only those instances passed to it from the MACROLIB, MICROLIB and IOLIB libraries provided. Only a very few other symbols from the BUILTIN library may be used directly in the schematic. These are: IN, OUT and BI; their usage is explained more fully later. Your VIEWDRAW.INI file must have the following lines appended to the end of the file in order to steer Workview Office in the correct direction when adding instances to your schematic. Typically the VIEWDRAW.INI file will be located at C:\vwoffice\standard\viewdraw.ini.

```
DIR [rm] C:\mpa\wvlibs\mpalib\macrolib (macrolib)
DIR [rm] C:\mpa\wvlibs\mpalib\microlib (microlib)
DIR [rm] C:\mpa\wvlibs\mpalib\iolib (iolib)
DIR [rm] C:\mpa\wvlibs\mpalib\builtin (builtin)
```

When adding instances to your FPGA schematic, be sure to use only instances from these first 3 libraries and only the special hierarchical connectors IN, OUT and BI from the BUILTIN library.

Starting a New Project

Before starting the schematic capture process you must first set up a project with the Viewlogic Project Manager. First pull down the Viewlogic Project Manager "File" menu and select "New" for a new project. Then by selecting the "Browse" button you can point Workview Office to the primary directory in which you will be placing your project in (see Figure 1).

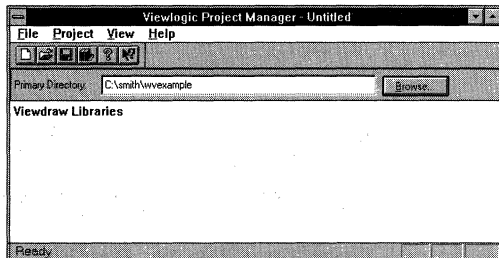


Figure 1. Viewlogic Project Manager Window with Project Path

The next step is to select the MPA libraries for the project. This can be done in two ways. One method is to pull down the "Project" menu, select "Libraries", and manually select the paths for the MPA libraries using the "Browse" and then "Add" buttons. The other method is to pull down the "Project" menu and select "Import Existing Searchorder." This will give you a window that asks for an .INI file name. Put in the full path name of the VIEWDRAW.INI file, that you modified earlier, in the "File Name:" space. This will automatically bring in the MPA libraries' path names for your project. The resulting Viewlogic Project Manager window should have the MPA libraries as shown in Figure 2.

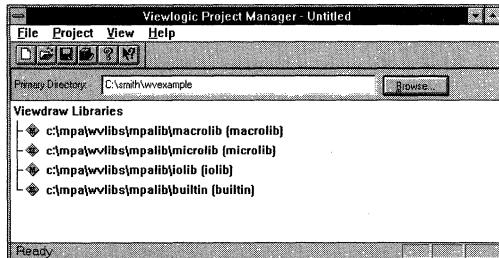


Figure 2. MPA Libraries Selected

After selecting the libraries, the final project step is to save the new project. At this point you will be asked to type in the name of the Viewlogic project file in the "File Name" space. The name of the file will be <your filename> with the .VPJ file extension (see Figure 3). After pressing the "OK" button Viewlogic Project Manager saves the .VPJ file to your project directory. Viewlogic Project Manager also saves a copy of the VIEWDRAW.INI file in your project directory.



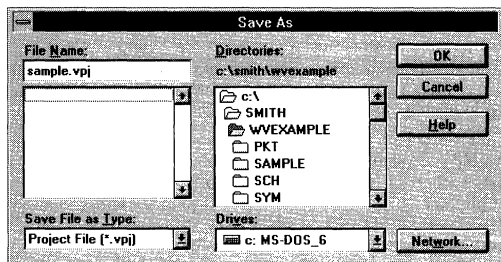


Figure 3. Saving the .VPJ File

Capture

There are just a few unique steps to take during schematic capture to ensure a valid MPA Design System EDIF netlist. The netlist importer of the MPA Design System needs to recognize your design's I/O ports. To accomplish this, you may either create a top level symbol for your completed schematic or you may opt to include VIEWlogic's hierarchical connectors.

If you are going to instantiate your completed FPGA schematic into a larger board or system level schematic then generating a top level symbol is the more appropriate method to use. In order to do this, each of the IOLIB instances used must have a named net stub attached to their 'external world' pins (see Figure 4). Once this task is completed all that is left is to create a VIEWlogic symbol for the entire FPGA schematic. Each of the pin names on the symbol must match the net-stub names exactly. A pin is required for every I/O net-stub (see Figure 5).

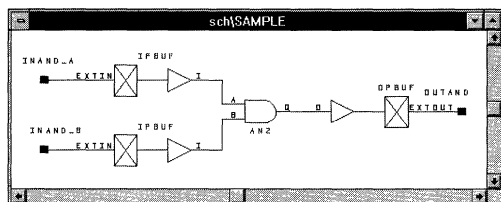


Figure 4. Top Level Schematic, Named Net Stubs

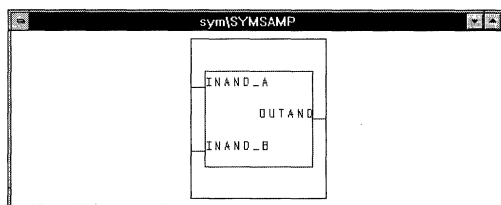


Figure 5. Top Level Schematic's Symbol, Pin Names = Net Stub Names

If on the other hand the schematic you are creating is stand alone for the FPGA, then a short cut method is available to you. As before, place the desired IOLIB instances on your schematic. Then from the BUILTIN library select the IN, OUT or BI hierarchical connector as appropriate. Connect this

hierarchical connector to the IOLIB instance's 'external world' pin and name the net (see Figure 6). This name may now be referenced for stimulus/response in the VIEWlogic simulator. Additionally, this net name is passed in the exported EDIF netlist to the MPA Design System place and route tool.

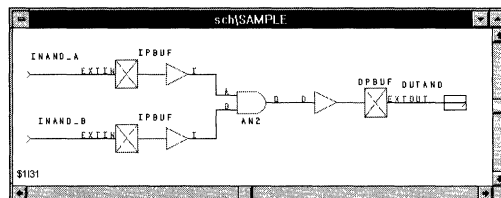


Figure 6. Top Level Schematic Using IN and OUT Symbols from BUILTIN. The OUT Symbol is Highlighted (boxed), Instance \$I131

The final step before ending your ViewDraw session is to pull down the "File" menu and select "Save + Check." This not only saves the current version of your schematic, but it also checks for minor connectivity violations as well as creates a file in the WIR directory which will be used later to create a simulation netlist.

Net List Export

The EDIF netlist writer is the VIEWlogic tool that translates your completed schematic into an EDIF file importable to the MPA Design System. The Workview Office netlisting tool is called EDIFGRAF.EXE and it can be put into your Workview Office toolbar by using Workview Office's "customize toolbar" option.

When you execute Workview Office's EDIF program you will have the choices shown in Figure 7. Select "EDIF Netlist Writer" and press the "OK" button.

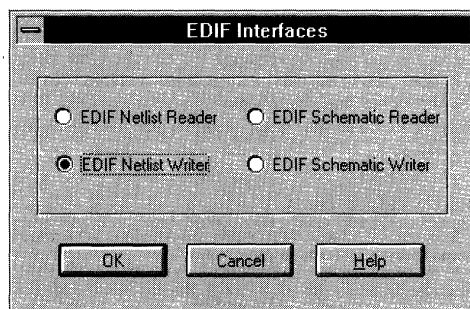


Figure 7. EDIF Interfaces Window

This will give you the "EDIF Netlist Writer" window (see Figure 8). Use the "Browse" button to select the schematic file that you want to generate the EDIF netlist for. Type "micro" in the Level box and select "Flatten without evaluating attributes." Press the "OK" button and the EDIF Netlist Writer will generate a .EDN EDIF netlist file ready for import to the MPA Design System.

4



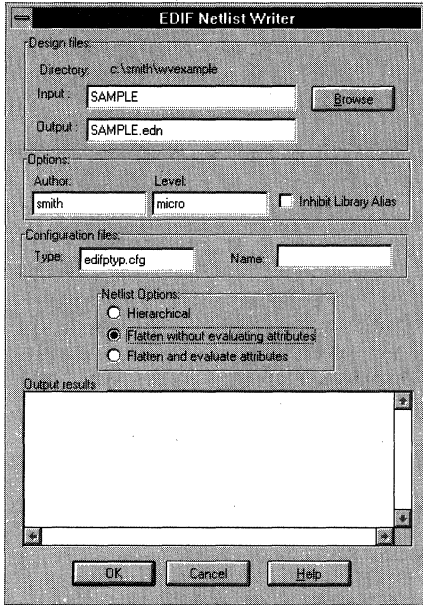


Figure 8. EDIF Netlist Writer Window

4

Attributes

The MPA Design System's import process can accept a set of attributes to help the designer tune the layout and routing processes. The system also accepts I/O parameters to specify CMOS/TTL compatibility, I/O drive, package pin assignment and slew rate control. Declaring attributes in the schematic will result in their being passed into the EDIF netlist and then imported into the MPA Design System. Optionally the designer may prefer to include attributes in an external .PAT file of the same root file name as the design's EDIF netlist. The designer may choose to use the combination of the two methods, but it should be noted that attributes passed into the MPA Design System in .PAT files will always take precedence if declared in both places.

Table 1. Valid Attributes

Sch. Component	Attached Place and Route Attribute	Attached I/O Attribute
Net	DPLD_IGNORE_TIMING DPLD_CLUSTER_SEED DPLD_PLACE_PRIORITY	
I/O Symbol (Instance) or Formal Port	DPLD_IGNORE_TIMING DPLD_PAD_PLACE	DPLD_PAD_PROPERTIES PULLUP or PULLDOWN DPLD_OPDRIVE DPLD_OPLEVEL DPLD OPSLEW DPLD_IPLEVEL

Place and Route Layout Attributes

The MPA Design System enables the tuning of place and route algorithms in three ways. The first is with the adjustment of the Auto Layout tool options such as start temperature, target delays, utilization etc. Additional details on the available options and their use are available in the on-line help facility of the MPA Design System. The second method involves the construction of a separate clock file. Here again, additional information is provided in the on-line help system and is not presented in this application note. The third method of influencing place and route results is the inclusion of the following attributes in the schematic or in an external .PAT file.

DPLD_IGNORE_TIMING

The DPLD_IGNORE_TIMING attribute is used to ignore certain paths for timing purposes. It may be set on an instance, a formal port, or a net. If a net has the attribute set, then the timing on all segments within that net are ignored. If an instance has the attribute set, then the timing on all inputs and output net segments from that instance are ignored. Assigning the attribute to a formal port has the same effect as assigning it to the associated I/O instance.

The effect of ignoring a net, a formal port, or an instance for timing purposes is to cause the autolayout to ignore all paths between clocked objects which the net, formal port, or instance lies on. Once all the objects to be ignored have been identified, their paths are propagated forwards and backwards through combinatorial gates until clocked objects are reached. The result is that additional segments other than those explicitly specified will be ignored for timing purposes as well.

You are required to use a dummy value with this attribute, but the value stated is otherwise ignored. Also, assigning this attribute to an instance or a net frees the timing driven autolayout algorithms to more optimally cluster, place, and route the speed critical nets.

DPLD_CLUSTER_SEED

The DPLD_CLUSTER_SEED attribute is used to assign a cluster seed to a net. This will cause the clustering to treat all instances that connect to that net differently and it increases the chance that the net and surrounding nets will be implemented in local or medium interconnect.

The action taken depends on the value of the attribute as follows:

- 0 ignore this net during clustering. Setting this attribute on a net is likely to cause the net to be implemented in global interconnect.
- 1 default operation
- <1000 weight this net by the given factor in the clustering

DPLD_PLACE_PRIORITY

The DPLD_PLACE_PRIORITY attribute can be applied to a net to force the software to lay out that net in a physically smaller area, in other words, to place the instances connected to that net closer together. The value of DPLD_PLACE_PRIORITY should be an integer in the range 1 to 10 (1 is the default). Higher values of place priority let you prioritize nets relative to each other.



DPLD_PAD_PLACE

DPLD_PAD_PLACE – instructs the I/O pad to be allocated to the package pin number specified. Only one pad may be allocated to any pin. Automatic placement of I/O pads usually results in a better layout, so this attribute should only be added when it is necessary. Example: DPLD_PAD_PLACE=C2

I/O Parameter Attributes**DPLD_PUP**

DPLD_PUP – attaches to WPUP primitive cells only, to select either one or both of the pull-up resistors available in a WPUP cell. Valid values are 1, 2 or BOTH.

PULLUP or PULLDOWN

PULLUP – set this to 1 if you want to enable the pull-up resistor on the external pin of an I/O pad. Default is 0 (resistor disabled). Example: PULLUP = 0

PULLDOWN – set this to 1 if you want to enable the pull-down resistor on the external pin of an I/O pad. Default is 0 (resistor disabled). Example: PULLDOWN = 0

DPLD_OPDRIVE

DPLD_OPDRIVE – sets the output drive current of an output or bi-directional pad to either 6ma (default) or 12ma.

DPLD_OPLEVEL

DPLD_OPLEVEL – sets the output voltage level of an output or bi-directional pad to either 3V or 5V (default).

DPLD_OPSLEW

DPLD_OPSLEW – sets the output slew rate (transition speed) of an output or bi-directional pad to high (default) or low.

DPLD_IPLEVEL

DPLD_IPLEVEL – sets the input threshold voltage of an input or bi-directional pad to either CMOS or TTL (default).

DPLD_PAD_PROPERTIES

Workview Office permits the use of the combined attribute, DPLD_PAD_PROPERTIES which combines the following

attributes into a single comma separated list: PULLUP, PULLDOWN, DPLD_IPLEVEL, DPLD_OPLEVEL, DPLD_OPDRIVE, DPLD_OPSLEW. This is especially useful when defining a block of I/O pins in an external attribute file such as a .PAT file. Example: DPLD_PAD_PROPERTIES = 0,0,CMOS,5V,12MA,HIGH (Also, see Figure 11 for an example).

Defaults and Invalid Combinations

The default I/O pad attributes have been selected so that they can connect to either TTL or 5V CMOS without adjustment. The default parameters may not be ideal for every design, and they should be matched to the application in order to achieve the best performance and noise immunity. 3V CMOS users should be especially careful to set DPLD_OPLEVEL to 3V, otherwise damage to peripheral IC's may result.

The following combinations of user attributes are not permitted:

DPLD_OPDRIVE = 6ma AND DPLD_OPSLEW = low.
PULLUP = 1 AND PULLDOWN = 1

The following combination of user attributes on a single bi-directional pad should be avoided, as it may produce unpredictable results:

DPLD_IPLEVEL = CMOS AND DPLD_OPLEVEL = 3V

Assigning Attributes in a Schematic

The attributes listed above can be assigned to nets and macro symbols (also referred to as, "instances" or components) as appropriate. In Workview Office, the method of assigning attributes is straight forward.

To assign an attribute to a net, select that desired net (its color will change identifying it as being the currently selected net) and then double-click on that net (or you can just double-click to begin with). The Net window will appear. Select the "Attributes" tab and then type the attribute in the "Name" box. Finally, press the "Set" button and the "OK" button (see Figure 9). The attributes will then be included in the EDIF netlist once the EDIF Netlist Writer is run.

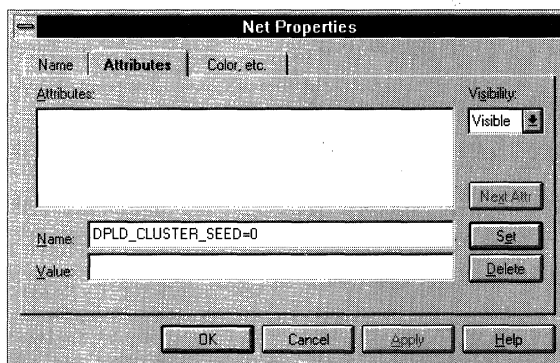


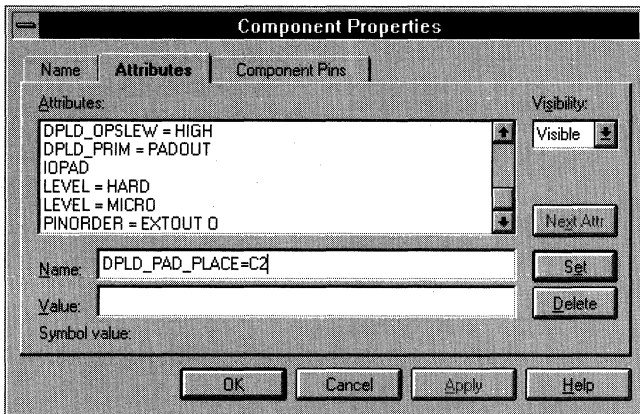
Figure 9. The Net Properties Box, Used to Assign Net Attributes

4



To assign an attribute to an instance, select that desired instance (a bounding box will appear identifying it as being the currently selected instance) and then double-click on that instance (or you can just double-click to begin with). The Component Properties window will appear. Select the

“Attributes” tab and then type the attribute in the “Name” box. Finally, press the “Set” button and the “OK” button (see Figure 10). The attributes will then be included in the EDIF netlist once the EDIF Netlist Writer is run.



4

Figure 10. The Component Properties Box, Used Here to Assign Instance Attributes

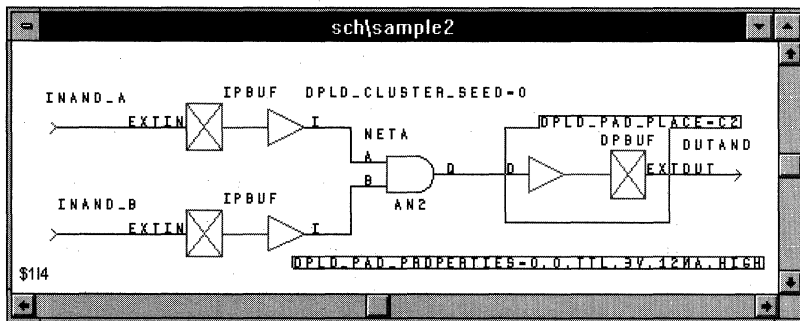


Figure 11. The Net “NETA” Attributed with DPLD_CLUSTER_SEED=0. The Selected Instance “OPBUF \$114” and its Attached Attributes are Boxed.

Assigning Attributes & Instances in an External .PAT File

Assigning attributes to a long series of instances, ports, or nets in the above manner can be time consuming and may be error prone. The MPA Design System gives the designer the option to enter all the valid attributes in an external .PAT file. Entries in the .PAT file take precedence over any attributes that may have also been instantiated in the EDIF netlist via schematic entry. The .PAT file must have the same root file name as the EDIF netlist .EDN file, and must reside in the same directory.

The external attributes file supports three main operations:

- 4) Insertion of attributes to specify pin placements and pad characteristics.
- 5) Insertion of special pad cells, IPCLK/IPRST, to drive the primary clock/reset network.

- 6) Insertion of special buffer primitives into a named net.

This has two uses:

- c) Force a named net onto/off the peripheral bus, by inserting the primitives APBUF/PABUF respectively.
- d) Force a named net onto the primary clock/reset network, by inserting the primitives ACLK/ARST respectively.

The external attributes file must exist in the same directory and with the same name as the EDIF netlist, with the file extension .PAT. During import, the MPA Design System automatically checks for the existence of a .PAT file and uses it when one is found.



Syntax of the External Attributes (.PAT) File

The external attributes file contains a list of commands, one per line. Each command contains up to five fields, as follows:

```
<object-class> <object-name> <operation> <name> [<value>]
```

where:

<object-class> is one of port, net, or instance.
<object-name> is the netlist name of the object (port, net or instance) being operated on.
<operation> is one of attribute or instance.
<name> is the name of a definition or an attribute.
<value> is only used in attribute operations, and is the value to be given to the attribute. This field is only required when an attribute requires a value.

The following are specific syntax forms of all valid attribute or instance assignments.

```
port <name> attribute <name> <value>
```

Attribute <name> with (optional) value <value> is added to the port instance (the instance driven by the formal port). Only works with input and output ports.

```
port <name> instance <name>
```

The port instance (the instance driven by the formal port) is replaced by an instance of the given definition. The only valid definitions are IPCLK, IPRST. This syntax is limited to input ports with 1 input pin and 1 output pin (IPBUF for example).

```
net <name> attribute <name> <value>
```

```
// This is a comment
# This is a comment as well

port sega attribute dp1d_pad_place 22
//This results in the IPBUF being placed on the pad associated with package
//pin 22.

port sega attribute dp1d_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the instance associated with the formal port sega. A dummy
//argument is required for this attribute.

port sega instance ipclk
//This results in the IPBUF being replaced by an IPCLK, forcing the net onto
//a clock network.

net segb attribute dp1d_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the entire net "segb". A dummy argument is required for this
//attribute.

net segb attribute dp1d_cluster_seed 1
//The dp1d_cluster_seed attribute and value shown get assigned to the net B
//for evaluation during place and route.

net segb attribute dp1d_place_priority 1
//The dp1d_place_priority attribute and value shown get assigned to the net
//B for evaluation during place and route.
```

Attribute <name> with (optional) value <value> is added to the net.

```
net <name> instance <name>
```

Creates an instance of the given definition and inserts it into the named net. The only valid definitions are ACLK, ARST, APBUF and PABUF.

```
instance <name> attribute <name> <value>
```

Attribute <name> with (optional) value <value> is added to the instance

Example .PAT File Entries

The following sample .PAT file entries reference the simple schematic shown in Figure 12.

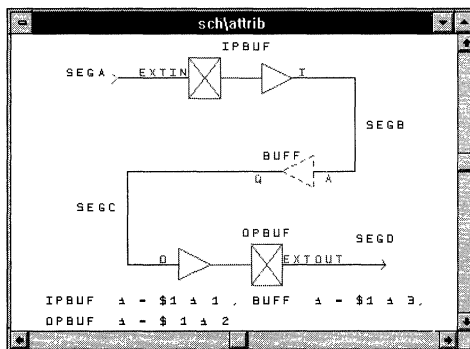


Figure 12. The .PAT File Attributes Are Added to This Simple Schematic. Nets Are Named SEGA, SEGB, SEGC and SEGD.

4




```

net segb instance aclk
//This results in an ACLK buffer being inserted between IPBUF's output and
//BUFF's input. BUFF is now driven off the resulting clock routing
//resource.

instance $II4 attribute dpld_opdrive 12ma
//This results in the OPBUF getting 12ma drive capability.

instance $II12 attribute dpld_ignore_timing dummy_arg
//For place and route purposes, the design's timing parameters are ignored
//for the all nets associated with the instance $II12. A dummy
//argument is required for this attribute.
    
```

All Valid Combinations of Attributes & Instances in an External .PAT File

The following show all the valid combinations of the attributes in the .PAT file.

port <name> instance (name here can only refer to input instances with one input pin and one output pin)

ipclk
iprst

port <name> attribute

dpld_ignore_timing dummy_arg
dpld_pad_place <value, see data book for package being used>
pullup 110
pulldown 110
dpld_opdrive 6ma12ma
dpld_oplevel 3v15v
dpld_opslew highlow
dpld_iplevel CMOSITTL
dpld_pad_properties (see above paragraph describing this attribute)

net <name> attribute

dpld_cluster_seed n (where $0 \leq n \leq 1000$, 1 is default, 0 means ignore net)
dpld_place_priority n (where $1 \leq n \leq 10$, 1 is default)
dpld_ignore_timing dummy_arg

net <name> instance

aclk
apbuf
arst
pabuf

instance <name> attribute

dpld_ignore_timing dummy_arg
dpld_pad_place <value, see data book for package being used>
pullup 110
pulldown 110
dpld_opdrive 6ma12ma
dpld_oplevel 3v15v
dpld_opslew highlow
dpld_iplevel CMOSITTL
dpld_pad_properties (see above paragraph describing this attribute)

Library Elements Specific to MPA Hardware Features

Most designs can be fit to the desired speed into the MPA family without the designer needing to know much about the details of the device's internal construction. However, to get the most out of the MPA, you should take some time to browse through the on-line help files thoroughly. The help files are rich in detail regarding the hardware specific macros and routing resources macros inherent in the MPA design system. The following topics are presented as simple examples on most of these hardware specific features. For an exhaustive presentation, please refer to the on-line help facility.

Logic One and Logic Zero

It may at times be necessary to modify the functionality of a macro from the supplied libraries by tying one or more of its inputs to logic high or low. There are two special elements provided in MICROLIB for this purpose. The ONE element produces a logic high to all input pins tied to it. The ZERO element provides a logic low. There are no fan-out restrictions for either of these elements. An alternate method is to tie the input pins requiring a logic high to a net named VDD, and tie the output pins requiring a logic low to a net named GND.

For all the above methods, the MPA Design System will recognize the logic as static and eliminate superfluous logic elements wherever possible during the place and route process.

Wired-OR

The MPA family allows for connecting many outputs to a single common signal line. The available macros for this function are: WND2, WINV, WOR2, WBUF. When using these macros, a WPUP instance is required. The maximum number of connections to a single signal is given by

$$\frac{1}{2} \sqrt{\# \text{ of core cells.}}$$

Table 2. Maximum Drivers on a Wired-OR Signal

MPA Family Member	Max. Drivers on a Wired-OR Signal
MPA1016	20
MPA1036	30
MPA1064	40
MPA1100	50

If the number of drivers on the Wired-OR net is below half of the maximum allowed, then only a pull-up resistor is recommended. Adding a second WPUP to a very large net,



will help speed things up some, but at the cost of increased power consumption. The allowable values for the DPLD_PUP attribute are 1, 2 or BOTH. Resistors 1 & 2 are of equal value, so it makes no difference which one you select. BOTH ties resistors 1 & 2 in parallel and decreases the low to high transition time, but at the expense of extra power consumption.

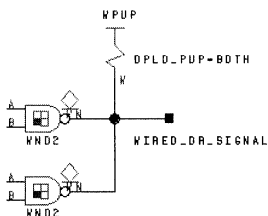


Figure 13. The Diamond Symbol Reminds the User That These Outputs Cannot Source a Logic HIGH, but Are Only Able to Pull the Output to a Logic LOW.

Adding additional wired-or outputs to the net WIRED_OR_SIGNAL will slow it down. Adding additional inputs to the net has little effect on speed. Low to high transitions are typically slower than high to low transitions on a wired-or signal.

Peripheral Bus

The periphery of the MPA is bordered with an 8 bit wide Peripheral Bus (P-Bus). The P-Bus can be broken at each corner of the array by switches. (Setting of the switches is handled automatically in the MPA Design System software.) Each of the resulting 32 segments has two programmable pull up circuits, one at each end. The P-Bus is ideal for routing common signals to many I/O macros.

It is easy to build a scenario where many I/O pins have a single or several control signals in common. In this instance you would want to place that signal on the P-Bus using the APBUF. Conversely, pulling a signal off the P-Bus back into the array is accomplished using the PABUF.

The ability to construct Wired-OR nets is not limited to signals internal to the array. P-Bus Wired-OR nets can be

constructed using APWBUF (or APWINV) and their associated pull-up structure PWPUP. However, using these features require the user to be aware of the natural consequences of adding capacitance and pull-ups to a resistive bus. Adding additional segments of P-Bus (by assigning I/Os to different edges of the die) increases capacitance that effects rise and especially fall times. Also, the somewhat resistive nature of the P-Bus can cause Vol noise margin problems if the active low P-Bus driver is far from the pull-up element.

The assignment of P-Bus pull-up resistors is automatic. On import, all PWPUP instances defined by the designer are removed from the netlist. The tool automatically balances the number of pull-up resistors against the P-Bus loading (incurred by the use of multiple die edges) by automatically inserting additional pull-up capacity for each P-Bus segment used. During autolayout the MPA Design System re-attaches a single peripheral bus pull-up, per occupied die edge, for each unique P-Bus signal. For example, if several I/Os that use a P-Bus Wired-OR signal get split between the 'top' and 'right' edges of the die during autoplacement, the tool would assign two pull-up resistors to the net.

Low Skew, Clock and Reset Nets

For a high fan out signal or for a signal where you would like to keep the skew limited to less than 1nS, you should consider moving the signal onto a clock network using the ACLK or ARST macro (they both do the same thing).

MPA I/O Structures

The standard I/O cell of the MPA array is very feature rich. The complexity of this structure is apparent in the large number of choices available in the I/O macro library IOLIB (Here again, the reader is encouraged to invest some time in the on-line help facility, in particular "Help on Libraries - Input and Output Pads" and "Help on Device - Functional Description - I/O Cell". Defining a bookmark for these two particularly useful help sections will provide a short cut for referencing them in the future.)

Each of the macros in the IOLIB fit in a single I/O cell. Couple these relatively complex functions with the availability of P-Bus routing resources (including the P-Bus Wired-OR resources previously mentioned) and it becomes clear that significant functionality can be achieved before ever using the normal internal resources of the array.

4



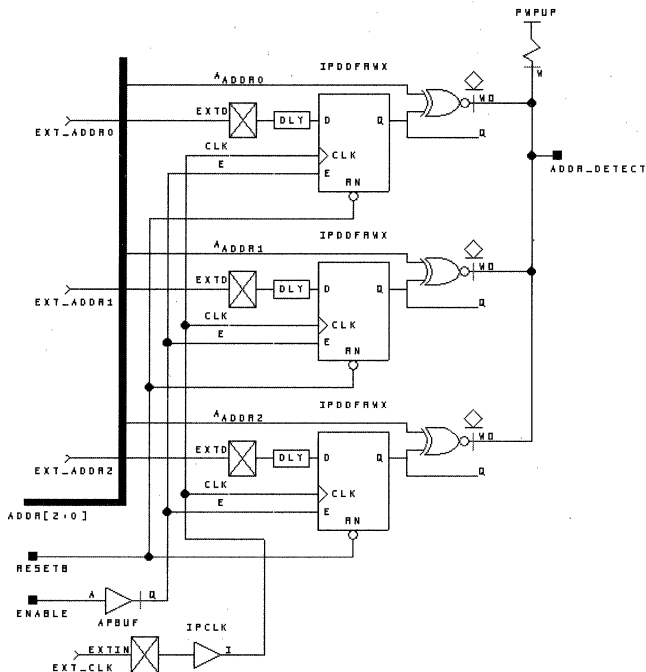


Figure 14. A Three Bit Address Decoder, Only I/O Cell and P-Bus Resources Used

In Figure 14, a three bit address decoder was implemented using only the resources available in the complex I/O Cells and the associated P-Bus. No internal logic or routing resources were consumed. The features that were used which are unique to the I/O Cell and P-Bus include: input delay to synchronize external data with the buffered clock signal, APBUF used to bus a common enable signal to several I/O sites, XNOR used to compare external and internal address values, and finally the Wired-OR P-Bus line. Of course, all of this functionality could be moved internal to the array, using only the simple I/O macros.

Simple I/O Structures

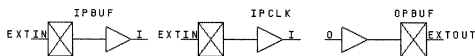


Figure 15. The Three Most Commonly Used I/O Macros

Your first designs for the MPA will probably only use the above I/O macros, with all the latching, decoding etc. being handled internal to the array. Soon though, you will probably encounter a real world design that constrains you to a specific clock-to-Q specification or some other requirement that will have you going back to re-examine the instances available in the IOLIB. There are many variations of latched, registered, and Wired-OR inputs, outputs and bi-directional macros.

Again, please invest some time with the on-line help descriptions of the device and the libraries.

Back Annotation and Simulation

It is assumed that the reader is familiar with the tools and procedures involved in using the Workview Office ViewSim tools. As mentioned earlier, when the schematic capture (ViewDraw) session is complete you should create a wirelist file in the WIR directory. You do this, in ViewDraw, by pulling down the "File" menu and selecting "Save + Check." Having created the wirelist file, the next step is to create a simulation netlist. This is done with VIEWlogic's VSM netlist tool. Running VSM from one of VIEWlogic's window-based programs only gives you a functional simulation without any timing data.

The MPA Design System has a facility that provides post place and route timing data in a format compatible with the VIEWlogic VSM netlist tool. The VIEWlogic format back annotation data table file (.DTB) can be generated after completion of a place and route. The .DTB file can be read in by the VIEWlogic VSM netlist tool to provide an accurate simulation netlist of your completed design.

After the MPA Design System generates the .DTB file, it will need to be moved up one level to your Workview Office design directory. In order to use the timing data in the .DTB file the VSM netlist program must be run from a DOS command prompt. It is run by entering the command line: vsm <schematic name> -d <.DTB filename>.

4



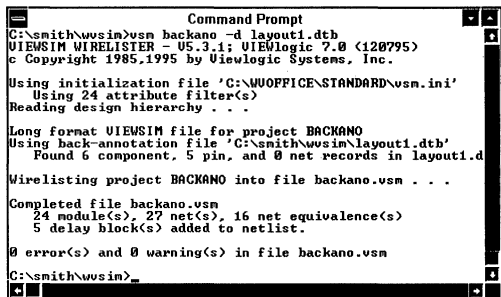


Figure 16. VSM Netlist Tool. The Schematic Name is "backano" and the back annotation file name is "layout1.dtb."

By running VSM, a ".VSM" file is created. The ".VSM" file must be loaded into the ViewSim program. This is done in

ViewSim, by pulling down the "Find" menu and selecting "Load ViewSim Netlist". Then enter the name of the desired ".VSM" file (see Figure 17). In this example, the ViewSim netlist file is called "backano.vsm." The MPA Design System generated delays will now be included in the simulation net list.

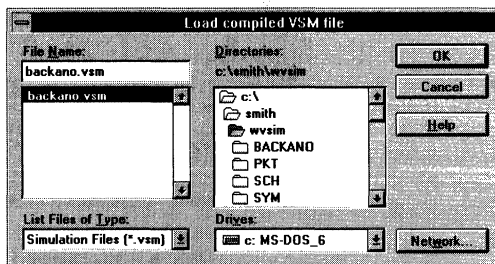
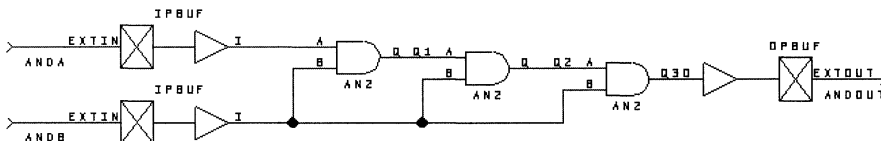


Figure 17. Load ViewSim Netlist Menu.



4

Figure 18. The Sample Circuit, "backano" Used to Explore Simulation Before and After Back Annotation.

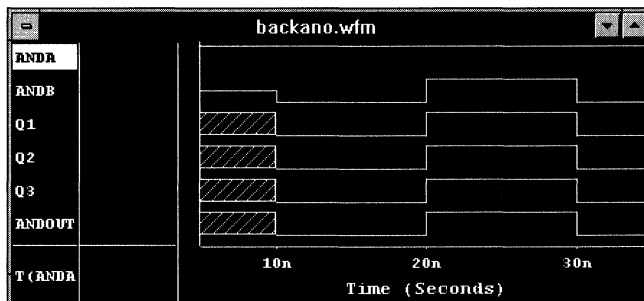


Figure 19. Before Back Annotation. ANDB Going High Ripples Through All Three AND Gates and the Output Instantaneously.



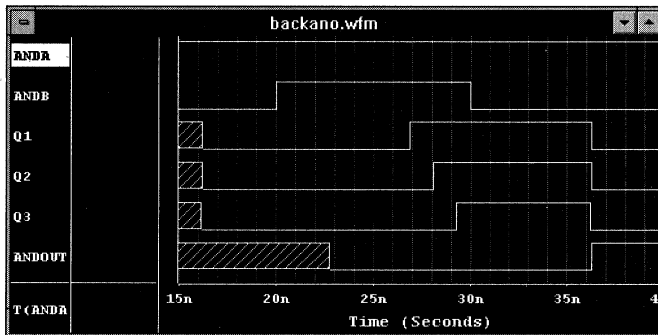


Figure 20. After Back Annotation of Place and Route Delays. ANDB Going High Ripples Through Q1,2 & 3 with Delays, and When Going Low, Takes Q3 Low First.

In Figure 20, ANDB goes high at time 20 and ripples through the AND tree (Q1, Q2, Q3) with the final output ANDOUT going high about 16nS later. When ANDB drops, the third AND gate (Q3) is the first to fall because in this implementation it happened to be placed closer to the ANDB input than AND gates Q1 & Q2. With ANDA held high, gate delay is the limiting factor for a rising ANDOUT and path delay is the limiting factor for a falling ANDOUT.

Back Annotation and Logic Reduction

The MPA Design System does some “bubble pushing” during the retargeting / fitting of the logical design into the physical implementation. In some cases, redundant gates in the schematic design are eliminated in the physical layout. When this occurs, back annotation proceeds as before with the resulting delays being added only to the input pin of the most downstream logic element not eliminated in the retargeting / fitting process.

4

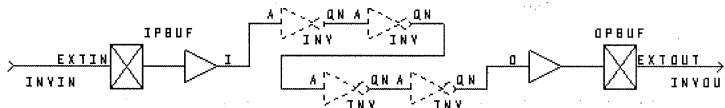


Figure 21. The String of Inverters will be Removed During Place and Route.

In the example above, the 4 inverters are eliminated in the retarget / fitting process. The wire delay between the remaining IPBUF and OPBUF gets back annotated to the

OPBUF's input pin. The back annotated simulation netlist has 4 zero delay inverters (just as the pre-back annotation netlist had) followed by the OPBUF with the real world delay.



Programming Large Configuration Files into Smaller Serial PROMs

Prepared by
Douglas M. Shade
Motorola Programmable Logic Products

4



Programming Large Configuration Files into Smaller Serial EPROMs

Introduction

This application note describes methods for programming large configuration files into lower capacity serial PROMs. Three source file formats are demonstrated along with three target PROM programmers. BP Microsystems BP-1200 programmer with version 3.19 software, Data IO Unisite at software version 5.20, and Logical Devices ALLPRO-88 programmer at version 2.70 are all demonstrated. While your programmer and software version numbers may be different, the concepts demonstrated here are generally applicable. The reader is assumed to have some familiarity with the programmer of interest.

An MPA1064KE was selected as the target FPGA device. Configuration files in Intel Hex format (1064.IHD), Motorola S Record format (1064.MS1) and a generic ASCII Hex format (1064.AHF) were all generated from the source design. Serial PROMs have the characteristic of reading out the bits of a byte in the opposite order from which they were programmed in. To accommodate this behavior, it is required that you select "Serial PROM (reverse bit ordering)" in the Tools -> Options -> Configuration panel of the MPA Design System Software, Figure 1.

The MPA17128 Serial PROM was selected as the target PROM. The MPA1064 was selected as the target array and requires a total of 202072 bits for a complete configuration image. Since an MPA17128 has a capacity of only 128k bits, two will be required. For more details on the data formats and device sizes, please reference the Motorola Programmable Array data book (DL201), the section MPA1000 Configuration Data Format.

Using the BP Microsystems BP-1200

Once you select the desired PROM using the "Select" pull-down menu, in this case an MPA17128 in a DIP package, you should set the programmable reset polarity to the desired value as shown in Figure 2. Use the Device -> ResetPol menu.

Using the Device -> Options menu as shown in Figure 3, set the "Data path width" to 1, the "Number of banks" to 2, and the "Programming mode" to SET.

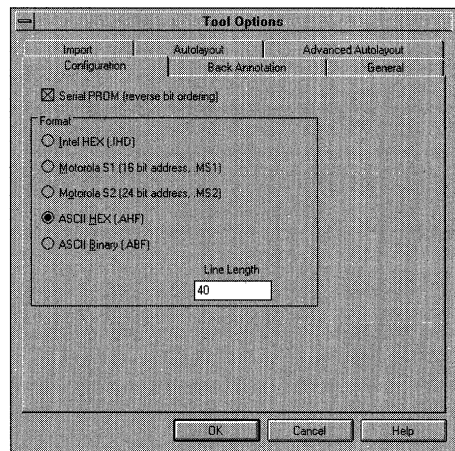
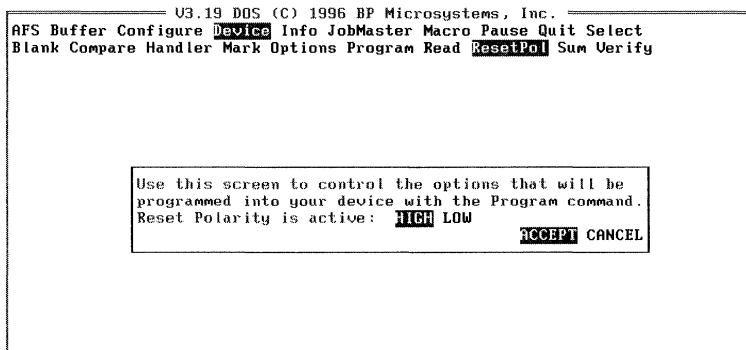


Figure 1. Select "Serial PROM" When Building a Configuration File for the MPA or Other Families of Serial PROMs

4



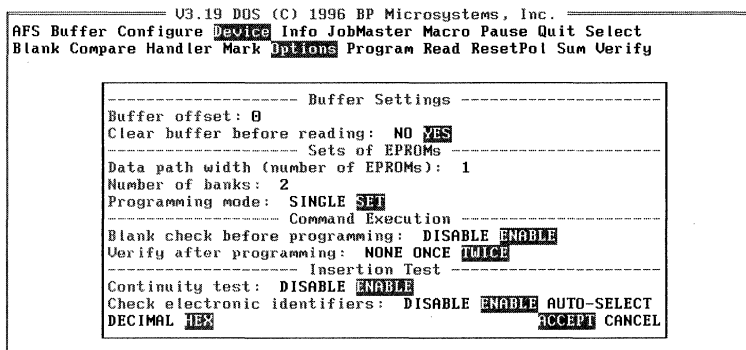


```

Buffer: Empty
Device: Motorola MPA17128          Size: 4000x8H  Pins: 8
Config: BP-1200/32/SM48D LPT1 Test-Twice Blank-Check Verify-Twice Check-IDs
      F1 Help  Enter when done  Tab Next field  Esc To cancel
    
```

Figure 2. Selecting the Programmable Reset Polarity to Be "High"

4



```

Buffer: Empty
Device: Motorola MPA17128          Size: 4000x8H  Pins: 8
Config: BP-1200/32/SM48D LPT1 Test-Twice Blank-Check Verify-Twice Check-IDs
      F1 Help  Enter when done  Tab Next field  Esc To cancel
    
```

Figure 3. Setting the "Buffer Settings" to Correctly Handle Multiple Serial PROMs

The next step is to read in the source configuration file. For this example the ASCII Hex format file was selected, 1064.AHF. The final step is to program as normal. The programmer will prompt you to insert a blank device '0'. After successful programming and verification of the first PROM, the programmer will then prompt for device '1'.

That's all there is to it. The BP-1200 handles the chore easily. If this is something you intend to do on a regular basis with your programmer, you might consider writing a macro using the Macro menu. A sample macro for this procedure is shown in Figure 4.




```

;Macro file V3.19 for BP-1200 generated 10/08/96 11:08:11.
;
/Select
  Device selector: Motorola MPA17128
  Package type: DIP
  Family shown: All
  : ACCEPT
/Device/ResetPol
  Reset Polarity is active: HIGH
  : ACCEPT
/Device/Options
  Buffer offset: 0x0
  Clear buffer before reading: YES
  Data path width (number of EPROMs): 0x1
  Number of banks: 0x2
  Programming mode: SET
  Blank check before programming: ENABLE
  Verify after programming: TWICE
  Continuity test: ENABLE
  Check electronic identifiers: ENABLE
  : HEX
  : ACCEPT
/Buffer/Load
  Directory: c:\sch\twosite\top\*.ahf
  File to load: 1064.AHF
  Type: STRAIGHT
  Clear buffer before loading: YES
  Lowest address to load: 0x0
  Highest address to load: 0x7fffff
  Load address in buffer: 0x0
  : HEX
  : ACCEPT
/Device/Program
  : ACCEPT
  : ACCEPT
  : ACCEPT
;End of file

```

Figure 4. TWOPROM.PGM, A Sample BP-1200 Macro File

The BP-1200 software does a good job of automatically detecting the configuration file formats automatically once a file has been selected. There are some limitations to be aware of however. The MPA Design System allows you to specify "Line Length" when using the ASCII Hex file format. A line length of 0 causes an .AHF file to be written out with no "newline" characters inserted anywhere in the file. This fools the BP-1200 software into thinking it is looking at a "binary" file type, which is incorrect. Setting the line length option to 40, results in an .AHF file in which 40 configuration bytes are represented on each line of the file (80 ASCII characters followed with a "newline" character). This setting seems to work well with the BP-1200's automatic file type detection (and most text editors as well).

Using Logical Devices ALLPRO-88

Using the ALLPRO-88 to perform this task is only slightly

more complicated. Select the device and the source file type and name as you normally would. In this example an MPA17128 DIP was used. The source file selected was an Intel Hex format file, 1064.IHD. After selecting the source file, the ALLPRO prompts you for additional input into the "Download Parameter Editor" window, Figure 5. Unlike the BP Microsystems software there is no direct entry for specifying "Banks". One method then of splitting the configuration file between two PROMs is to read in the source file once, ignoring all data extending past the upper physical address range of the PROM and programming the device; then read in the source file a second time, this time skipping past the previously programmed data.

Figure 5 shows the appropriate settings for "start file address" and "end file address for the bank'0' serial PROM. Figure 6 likewise shows the settings for the second or bank'1' serial PROM.



Once the entire configuration file has been read in, program the PROM as you normally would. The only out of the ordinary message you should see in Figure 9 is that the PROM was

recognized as receiving only a "partial set". The "User data size" exceeds the "Device block size".

```

FILENAME:                RAM AVAIL: 1024 OF 1024KB  REV: 5.20  5.20  EM1
MANUFACTURER:Motorola   PART #: 17128          FAMILY/PIN CODE: 206 / 226
I/O FORMAT: Motorola S3
| OPERATION COMPLETE. Partial Set Sumcheck = 00005EE5 ( 8-bit)
MAIN MENU                PROGRAM MEMORY DEVICE    (non-default)
Select device           Source (RAM,Disk)          R
                                                                Set auto-increment  N
Quick copy
Data word width        8
Load device            Next device          1
Total set size         2
Program device         User data size       62AB
Next operation begins at 0
Verify device
More commands
Device block size      4004
Return: Execute        PF4: Select mode/options
PF1: Main menu         PF2: Prev menu       PF3 or ?: Help
                                                                Num      Bin

```

Figure 9. After Programming the First (Bank '0') PROM, the Unisite Returns a "Partial Set Sumcheck" Message; There Is More Data Than the PROM Can Hold

The next step after successful programming of the bank '0' PROM is to read in the source configuration file once again; this time using an "I/O addr offset" as shown in Figure 10. In this case, the offset used was equal to the capacity of the previously programmed serial PROM, 4000x byte locations. (The Unisite reports that the "Device block size" as being

4004. The first 4000 locations of an MPA17128 are available for user data, with the last 4 bytes reserved for programming the reset polarity.) After the second read of the configuration file, the Unisite recognizes that less data was read in than the selected PROM has room for and issues an appropriate warning to the user "Partial or no transfer performed".

4

```

FILENAME:                RAM AVAIL: 1024 OF 1024KB  REV: 5.20  5.20  EM1
MANUFACTURER:Motorola   PART #: 17128          FAMILY/PIN CODE: 206 / 226
I/O FORMAT: Motorola S3
- WARNING: Partial or no transfer performed. Data sum = 000063EB (8-bit)
TRANSFER MENU           DOWNLOAD DATA FROM HOST
Download data           Source (Remote,Terminal)    T
Upload data             Destination (RAM, Disk)      R
Compare data
Format select           I/O Translation Format      95
Input from disk         I/O addr offset            4000
Output to disk          Memory begin address        0
Serial output           User data size              22AB
Download host command
transfer c:\promfile\1064.ms1
PF1: Main menu         PF2: Prev menu       PF3 or ?: Help
                                                                Num      Bin

```

Figure 10. Setting "I/O Addr Offset", After Reading in the Larger File, the Unisite Warns You About a "Partial...Transfer"

Now that the second half of the configuration data set has been read in, you're almost ready to program the bank '1' PROM. Note that the "PROGRAM MEMORY DEVICE" window of Figure 11 shows the "User data size" to be less than

the "Device block size". Attempting to program with these parameters will result in the "OPERATION FAILED" error shown Figure 12.



```

FILENAME:                RAM AVAIL: 1024 OF 1024KB  REV: 5.20  5.20  EM1
MANUFACTURER:Motorola   PART #: 17128             FAMILY/PIN CODE: 206 / 226
I/O FORMAT: Motorola S3
/
MAIN MENU                PROGRAM MEMORY DEVICE  (non-default)
Select device           Source (RAM,Disk)         R
                                     Set auto-increment  N

Quick copy
Data word width        8
Load device            Next device          1
Total set size         1
Program device         User data size       22AB
Next operation begins at 0

Verify device
More commands
Device block size     4004
Return: Execute       PF4: Select mode/options
PF1: Main menu        PF2: Prev menu      PF3 or ?: Help
Num                   Bin

```

Figure 11. After Reading in the Partial (Second Part) Configuration File, Note the "User Data Size" Is Reported to Be Something Less Than the Capacity of the PROM

4

```

FILENAME:                RAM AVAIL: 1024 OF 1024KB  REV: 5.20  5.20  EM1
MANUFACTURER:Motorola   PART #: 17128             FAMILY/PIN CODE: 206 / 226
I/O FORMAT: Motorola S3
\ OPERATION FAILED: Partial device operation is not allowed
MAIN MENU                PROGRAM MEMORY DEVICE  (non-default)
Select device           Source (RAM,Disk)         R
                                     Set auto-increment  N

Quick copy
Data word width        8
Load device            Next device          1
Total set size         1
Program device         User data size       22AB
Next operation begins at 0

Verify device
More commands
Device block size     4004
Return: Execute       PF4: Select mode/options
PF1: Main menu        PF2: Prev menu      PF3 or ?: Help
Num                   Bin

```

Figure 12. Attempting to Program the Device With the "User Data Size" Less Than the "Device Block Size" Results in an "OPERATION FAILED". The Unisite Will Normally Refuse to Program Just a Portion of a Device



```

FILENAME:                RAM AVAIL: 1024 OF 1024KB  REV: 5.20  5.20  EM1
MANUFACTURER:Motorola   PART #: 17128             FAMILY/PIN CODE:  206 / 226
I/O FORMAT: Motorola S3
\ OPERATION COMPLETE: Sumcheck = 000010B4 ( 8-bit)
MAIN MENU                PROGRAM MEMORY DEVICE    (non-default)
Select device           Source (RAM,Disk)          R
                                                                Set auto-increment  N

Quick copy
Data word width        8
Load device            Next device          1
                                                                Total set size      1
Program device         User data size      4004
                                                                Next operation begins at  0

Verify device
More commands
Device block size     4004
Return: Execute       PF4: Select mode/options
PF1: Main menu        PF2: Prev menu      PF3 or ?: Help
                                                                Num          Bin
    
```

Figure 13. Hand Editing the “User Data Size” to Match the “Device Block Size” is a Simple Work Around; Allowing You to Program the Bank ‘1’ or Second PROM

Figure 13 shows the completion of a successful programming of the bank ‘1’ PROM; after the hand edit of the “User data size” to match the “Device block size”.

The configuration data for the second (bank ‘1’) PROM only occupied the first 22AB locations. Hand editing the “User data size” to 4004 results in programming the addresses 22AB and above with whatever happened to be in the Unisite’s RAM at the time. This will cause no problems for the MPA at configuration time, so long as addresses 4000–4003 inclusive

are set appropriately.

As always, the user should be cautious about how the programmable reset polarity is being set prior to programming both the bank ‘0’ and ‘1’ devices. When using the Unisite, the user is required to go in and manually edit the locations appropriate to the particular PROM. For the MPA17128, the programmable reset polarity is controlled by the contents of 4000–4003 as shown in Figure 14.

4

```

CURSOR AT LOCATION: 00004005      8 BIT ADDRESSING
                                HEXADECIMAL
                                ASCII
ADDRESS  -0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -A -B -C -D -E -F 0123456789ABCDEF
. 00004000 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004020 00 00 00 00 00 00 00 00 0C 00 00 00 00 00 00 00 .....
. 00004030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 00004090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 000040A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 000040B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 000040C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 000040D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 000040E0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
. 000040F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
^P: Prev block      ^E: Exchange          with
^N: Next block      ^F: Search pattern      ^U: Restore block
^D: Delete byte     ^B: Jump to address 4000  PF2: Exit editor
^T: Start/stop     insert  Tab: Toggles between Hex/ASCII mode
                                                                Num          Bin
    
```

Figure 14. Programming the Locations 4000–4003 Inclusive With FF sSelects Active High Reset Polarity. Setting These Locations to 00 Would Yield Active Low Reset Polarity



The Unisite's user interface is a bit more cumbersome than the others described, but it is an extremely robust and capable machine. With the help of this application note you should be able to work your way through programming a large configuration file into smaller serial PROM devices with the Unisite or any other programmer using any one of the file formats available to you from the MPA Design System.

Appendix – Sample File Formats

The heads and tails of all the available configuration file formats from the MPA Design System version 2.3 are presented in this appendix. The target device for each of these files was the MPA1064 array. "Serial PROM (reverse bit ordering)" was selected for each of these formats. An excellent detailed reference for these and other generic file formats can be found in the documentation for the Data I/O Unisite programmer.

Intel Hex (.IHD)

```
:020000020000FC
:20000000C8C902B8000000000000000000000000000000000000000000000000000095
:20002000000000000000000000000000000000000000000000000000000000000000C0
:200040000000000000000000000000000000000000000000000000000000000000003C
[center of file deleted]
:20626000000000000000000000000000000000000000000000000000000000000000DA
:20628000000000000000000000000000000000000000000000000000000000000000FE
:0B62A00000000000000000000000000000000000000000000000000000000000000059E
:00000001FF
```

Looking at the first line of the .IHD format file:

```
:020000020000FC
: <- is the start character
02 <- is the byte count
0000 <- is the address for the data
02 <- is the record type of Extended Address
0000 <- is the offset address
FC <- is the line's checksum
```

Looking at the second line:

```
:20000000C8C902B8000000000000000000000000000000000000000000000000000095
: <- is the start character
20 <- is the byte count
0000 <- is the address for the data
00 <- is the record type of Data
C8C902B8 <- represents the MPA1064's JTAG ID (bit order reversed)
00 <- is the data type record for the following config data
`95' is the line's checksum -> 95
```

Looking at the last line:

```
:00000001FF
: <- is the start character
00 <- is the byte count
0000 <- is the address for the data
01 <- is the record type of End Record
FF <- is the line's checksum
```

Motorola S1 (16 bit address, .MS1)

```
S1230000C8C902B8000000000000000000000000000000000000000000000000000091
S1230020000000000000000000000000000000000000000000000000000000000000BC
S123004000000000000000000000000000000000000000000000000000000000000038
[center of file deleted]
S1236260000000000000000000000000000000000000000000000000000000000000D6
S1236280000000000000000000000000000000000000000000000000000000000000FA
S10B62A000000000000000000000000000000000000000000000000000000000000059A
S9030000FC
```

Looking at the first line of the .MS1 format file:

```
S1230000C8C902B8000000000000000000000000000000000000000000000000000091
S1 <- is the start character specifying the address field has 4 characters (16 bits)
23 <- is the byte count (35 decimal) for data, address and line's checksum
0000 <- is the address for the data
C8C902B8 <- represents the MPA1064's JTAG ID (bit order reversed)
00 <- is the data type for the following data (00 = Sequential)
`91' is the line's checksum -> 91
```

4



Using Exemplar Logic's Galileo with the MPA Design System

Prepared by
Philip J. Rauba
Motorola Field Applications Engineer

4



Using Exemplar Logic's Galileo with the MPA Design System

Purpose

The goal of this application note is to guide a Field Programmable Gate Array (FPGA) designer on the use of Exemplar Logic's Galileo synthesis tool in conjunction with the Motorola Programmable Array Design System (MPADS) when targeting a FPGA project to the Motorola Programmable Array (MPA) family. Exemplar Logic's Galileo Logic Explorer is a powerful synthesis tool that supports the Verilog and VHDL Hardware Description Languages (HDL). The advantages of using HDL design entry along with a synthesis tool such as Galileo, as opposed to using schematic capture based tools, include faster time to market, portability to other technologies, and reuseability of designs. The Motorola fine grain architecture is very synthesis friendly and takes advantage of the same type of optimizing algorithms used for gate arrays, yielding designs with higher utilization and faster speeds as opposed to coarse grain architectures.

Prior knowledge of coding with Verilog or VHDL HDLs and of using the MPADS place and route tools is assumed. Note that file and directory syntax mentioned within this application note are in reference to a DOS based PC platform. Reference to directory paths assume that Galileo and MPADS was installed at the default directories of c:\exemplar and c:\mpa respectively. Within the DOS environment file names are limited to eight characters with a three character file extension identifying the file type. The HDL design file names mentioned in this article are recommendations and are named using the design convention that Verilog, VHDL, and EDIF source filenames match the Verilog module name or VHDL entity name instantiated within the design's file. Text enclosed within brackets like <design>.v indicates that the user should provide a name to substitute for the text included within the brackets, but not including the brackets themselves.

HDL Design Environment

Figure 1 shows the HDL design flow for the Galileo synthesis tool and the MPADS place and route tool. Designs typically start with the development of HDL source code with the suggested filenames designated as design.v for Verilog or design.vhd for VHDL. As the source code grows in size, it is often necessary to verify that the code is logically performing as intended through the use of a Verilog or VHDL simulator. Simulators are highly recommended and are readily available from third party vendors, such as the Model Technology Incorporated V-System VHDL simulator from Exemplar Logic. When verifying HDL code with a simulator, test bench code is commonly developed that can provide system clock sources with test vectors and is designated with the suggested filename stimulus.v for Verilog or stimulus.vhd for VHDL. The HDL test bench code will include the instantiation of the HDL source code under test with the instance name of "u1" and module (Verilog) or entity name (VHDL) of "design". Please refer to the section entitled Verilog Simulation for more clarification.

Once the source code has been verified through simulation, it can be synthesized with Galileo producing an EDIF 2.0.0 netlist with the filename designated as design.edf. Galileo

processes the design through three steps including: synthesis, architecture specific optimization, and technology mapping. MPADS is then used for importing the EDIF netlist, automatic timing driven place and route, bitstream generation (configuration), and back annotation to a structural Verilog (layout1.v) or VHDL 1076 (layout1.vhd) netlist.

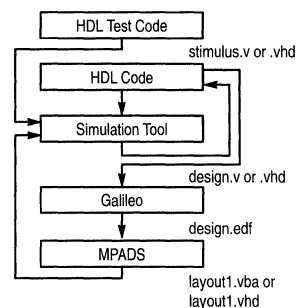


Figure 1. HDL Design Flow

Structural netlists can be used by a simulation tool to post simulate the final placed and routed design and include the intrinsic gate and path delays of the MPA device. Post simulations will show the true timing of the device, which are based upon highly accurate RC spice models.

Hierarchical Designs

When starting a new Verilog or VHDL design, the designer has a choice of using a flat or a hierarchical design methodology. If the design is relatively small, it may be easier to enter the source as one flat design file; however, as a design gets larger, one should consider using a hierarchical approach.

The advantages of using a hierarchical design methodology are: One, the design will be more manageable in terms of file and directory structures. Two, the reuse of commonly used pieces of code such as modules in Verilog and entities in VHDL, will result in code efficient design. Three, the designer has control on how each module of the code will be synthesized. Four, the synthesis of individual modules of the design will put less demand on your computer system memory, speeding up compile times. Five, design changes will require only the modules that changed to be resynthesized, instead of the whole design, again speeding up compile times.

Essentially the top module or entity of a design instantiates the individual pieces of the design with all of the appropriate interconnections. The submodules will be referenced in the top module and contain empty code sections with just declarations of inputs and outputs. These submodules will then be redeclared within individual files that contain the actual code of the design. File names must be the same names as the modules or entities.

When working with hierarchical designs, the designer should not use vector or bus syntax for input or output signals

4



between modules. If a bus or vectors are used, Galileo will separate and rename them into individual signals, causing possible interconnection problems, when imported into the MPADS place and route tools.

The hierarchical design methodology as described above requires the top module and each submodule of the design to be individually synthesized, producing multiple EDIF netlists. The Motorola place and route tool requires the type EDIF 2.0.0 netlist and has multiple netlist reader capability. For designs with multiple netlists the MPADS reader requires that each EDIF file name matches the module name and will search the design directory for modules which are referenced within the top module's EDIF netlist. If you are going to use multiple EDIF netlists for your design, you will need to set up MPADS to enable this feature. By using your favorite editor, open the DOS text file:

```
c:/windows/pm.el.ini
```

Change the line from:

```
#DPLD_MULTI=TRUE
```

To:

```
DPLD_MULTI=TRUE
```

4

Note that if you are not going to use a hierarchical methodology, you will not have to modify the pm.el.ini file, since after installation the MPADS tool defaults to read in a single flat EDIF netlist.

Galileo MPA Library Setup

After Galileo and the MPA Design System (MPADS) has been installed, Galileo will need to be setup before synthesis is invoked. First, copy the MPADS supplied synthesis libraries from c:/mpa/exemplar/p_mpa20.syn and p_mpa23.syn to c:/exemplar/lib. Second, create a file called c:/exemplar/data/exemplar.rc which includes the line:

```
Technology: IO p_mpa none none Motorola MPA1000
```

You do have the option of adding this line to Galileo's supplied file called c:/exemplar/data/master.rc, but this file will be overwritten whenever a new version of Galileo is installed.

Galileo synthesizes a design using the gate primitives that are referenced within the Motorola supplied libraries. Descriptions of the MPA gate primitives are available within MPADS by invoking Help on Libraries. If Galileo's graphical user interface (GUI) is being used to invoke the synthesizer, the exemplar.rc file is used for enabling the listing of Motorola's MPA library in the menu of the available target technologies.

Galileo Control File

Galileo has the capability for the designer to guide the synthesis through an optional control file designated as <input_file>.ctr. When synthesis is invoked, Galileo will search for the control file with the same filename as the input design (by default) and will execute the commands contained within the file.

The control file is particularly useful for assigning clocks and resets to the eight low skew networks that are available in

the MPA family. Clock and reset pad inputs can be assigned to the low skew network with the ipclk (input pad clock) and iprst (input pad reset) MPA primitives using the control file commands:

```
buffer_sig ipclk <clock_name>
```

```
buffer_sig iprst <reset_name>
```

Assignment of internally generated clocks and resets may also be connected to the low skew network with the ackl (array clock) and arst (array reset) MPA primitives using the control file commands:

```
buffer_sig ackl <clock_net>
```

```
buffer_sig arst <reset_net>
```

Note that these assignments will use up one of the MPA clock I/O pads for each command.

Another resource that is available in the MPA architecture is an eight bit peripheral bus, that runs around the outside I/O boundary. The peripheral bus is a resistive path, slower than the low skew network, but useful for assigning tristate I/O buffer output enable signals. Connections to the bus are made through the apbuf (array to peripheral bus buffer) primitive and can be assigned using the control file command:

```
buffer_sig apbuf <oe_net>
```

A sample control file is available after MPADS installation, which can be used as a template for new designs, and is called c:/mpa/exemplar/control.txt.

It is clearly important to assign clocks and resets to the low skew network in order to achieve a high speed design, but it may be unclear why the control file is needed. When writing HDL source code, the inclusion of complex I/O, clock buffers, or reset buffers specific to one vendor is not often desirable, since it makes the code lose portability. The Galileo control file is an attractive mechanism for instantiating the vendor specific primitives into the design without touching the original source code; thus, maintaining portability.

The Galileo control file, however, is not the only method of configuring the clocks and resets within a design. MPADS has a similar capability through a pin attribute file that is designated as <design>.pat and will be discussed in detail later. The decision on using the Galileo control file, the MPADS attribute file, or possibly both, rests upon the designer. When a design is synthesized, signal names are commonly replaced by cryptic names such as N\$1000. To make sure clock and reset primitives get instantiated and hooked up correctly when undergoing signal name changes during synthesis, the control file is recommended. In cases of internally generated resets or clocks, the designer may find that a signal name has changed because of optimization with no way of identifying the correct signal name within a control file. Here the designer should use the MPADS pin attribute file.

Please refer to the Galileo Reference manual for additional control file commands.

Galileo Synthesis Recommendations

A designer has the option of running Galileo from its graphical user interface (GUI) or from a DOS batch command



file. For those unfamiliar to DOS batch commands, this method requires a file to be created with the file extension .bat, such as synth.bat, that contains a list of DOS commands that the user wishes to execute. When double clicking on the batch command file icon from the Windows Explorer, the list of commands within the file are executed. Note that running Galileo from a batch command is no longer supported on Windows 3.X and only available to Windows 95 and Windows NT users. The advantages of using a DOS batch command file over a GUI when invoking Galileo are: One, the user can execute multiple synthesize commands. Two, the user has a documented step by step record on how a design is synthesized and what Galileo synthesis options were used for a particular design or module. Three, the user can save time during design iteration by not having to set up the GUI each time synthesis is required.

HDL designers should be aware of the type of gates that are being generated by the synthesizer from the source code they have written. This is particularly important to users who are trying to optimize their design for speed. For example, in the Boolean equation $d = a \bullet (b \bullet c)$, the input a is specified as the last gating term in order to meet timing requirements. During synthesis, Galileo's optimization algorithms, if used, may juggle the equation to $d = (a \bullet b) \bullet c$. Logically the equation is the same, but timing wise it is not. To let the designer see the type of gates that are generated and how they are connected, it is helpful to generate structural Verilog or VHDL netlists in addition to the EDIF netlist, needed by MPADS. These structural netlists are far easier to read than an EDIF netlist and review of them will give the designer insight on the critical paths and structures of his circuit. Examples of the EDIF, Verilog, and VHDL netlists are shown in Appendix A, B, and C respectively.

Instead of reviewing netlists for critical paths, the user may prefer to use the graphical timing analysis tools available in Galileo. Exemplar Logic's Galileo Time Explorer includes timing analysis, schematic viewing, and back annotation. A flat pre-place and route MPA EDIF netlist can be analyzed by Timescope and the critical timing paths can be graphically reviewed by the designer using the Netscope schematic generator and viewer.

Galileo Synthesis

Galileo requires a Verilog or VHDL source file designated with the file name extensions .v or .vhd, respectively and an optional control file with the file name extension .ctr. For ease of discussion, the use of Galileo will be described from the DOS batch command file perspective, instead of the Galileo graphical user interface. The Verilog (line one) and VHDL (line two) synthesis commands for the top module of a hierarchical or flat HDL source are:

```
gc design.v net.v -target=p_mpa -save
-edif_file=design.edf

gc design.vhd net.vhd -target=p_mpa -save
-edif_file=design.edf.
```

The commands generate Verilog (net.v) or VHDL (net.vhd) and EDIF (design.edf) netlists using eleven optimization algorithms for the MPA family. The save option stores the output of each of the eleven passes and records the results in a log file designated as design.log. A sample UART design file is provided during the installation of Galileo designated as c:/exemplar/demo/uart.v and when synthesized produced the following log:

Pass	Area (Gates)	Delay (ns)	CPU min:sec	File
1	176	14.6	00:32	uart_1.vg
2	198	13.6	00:29	uart_2.vg
3	180	13.6	00:27	uart_3.vg
4	180	13.6	00:30	uart_4.vg
5	178	13.6	00:25	uart_5.vg
6	177	14.6	00:24	uart_6.vg
7	180	13.6	00:25	uart_7.vg
8	180	13.6	00:26	uart_8.vg
9	179	13.6	00:35	uart_9.vg
10	208	13.4	00:29	uart_10.vg
11	201	21.8	00:29	uart_11.vg

4

Note that the area is the number of MPA core cells and delay is the worst case path core cell delay, not including the worst case net delays. It is recommended to use the save option in order to select the desired output based upon the design's needs. When optimizing a design for speed, it is not true that selecting the optimization pass that has the smallest delay will result in the fastest design. In most cases the opposite is true in that the smallest area will produce the fastest design. The smallest area will have fewer interconnections, thus less net delays. In FPGA architectures the net delays have a far greater effect on the speed of a design than the gate delays. So select the optimization pass that has the smallest area, in this case pass one will be the best candidate with a core cell area of 176. The designer may find themselves selecting several of the lowest area design outputs, trying them through place and route, and picking the best at the end.

Getting back to the original synthesis commands, Galileo uses default options when compiling a design targeting the MPA family. The descriptions of these are explained in detail in the Options and Switches section of Galileo's Reference Manual, but are listed here for completeness.

```
-area
-chip
-complex_ios
-control=<input_filename>.ctr
-effort=standard
-latch_detect
-report=slack_table
-wire_tree=worst
```

If you are involved in a hierarchical design, the synthesis of individual submodules is needed and can be accomplished with the following commands:



```
gc sub.v subnet.v -target=p_mpa -save -macro
```

```
-edif_file=sub.edf
```

```
gc sub.vhd subnet.vhd -target=p_mpa -save -macro
```

```
-edif_file=design.edf.
```

The primary difference with these commands versus the top module synthesis commands, listed earlier, is that the `-macro` option is used to specify that this module or entity is part of a hierarchical design. It implies that inputs and outputs of the module or entity are not I/Os and are replaced by internal buffers. Please note that the name "sub" is only a suggestion and the naming of file with module or entity names should be maintained.

As discussed earlier in synthesis recommendations, it was stated that control over critical timing paths may be needed. In these cases, consider the use of the remap option capabilities of Galileo. This option does not use optimization in the synthesis of the design, but merely maps the circuitry into MPA primitives as specified within the HDL source code. Thus the timing critical specification of the Boolean equation $d = a \bullet (b \bullet c)$, will be retained. Again, emphasizing the fact that the designer has to be aware of the output of the synthesizer and take into consideration that careful processing of a design is needed for speed critical design. When using the remap option, Galileo will ignore the save option if it is also specified. With synthesis complete, we proceed onto place and route setup.

4

MPADS .pat File Setup

The EDIF netlist produced by Galileo HDL development will typically not include pin attributes for configuring the electrical characteristics of the I/O or timing attributes for optimizing the design for speed. MPADS provides a method for doing these via an attributes file designated as `design.pat`. During the import of the EDIF netlists, MPADS will search for the `.pat` file for preprocessing of the design. Attributes assigned in the `.pat` file will be inserted into an internally generated netlist designated as `design.net` and will override any attributes contained in the EDIF netlist. The attributes that can be assigned to a design are listed in Table 1.

Don't be overwhelmed by all of these attributes, they are simply provided to give you control over your design. Take a few moments to get acquainted with the MPADS design environment, invoking the graphical user interface. One of the features of MPADS is its extensive on line help utility. Launch help from the top help menu and select index. Follow the help linking sequence: Reference Topic: Libraries, Input and Output Pads, I/O Pads: Attributes, and then Attributes: Attribute Glossary. Get familiar with these and then read on.

Table 1. Valid Attributes

Design Component	Attached Place and Route Attributes	Attached I/O Attributes
Net	IGNORE_TIMING CLUSTER_SEED PLACE_PRIORITY	
Symbol or Formal Port	IGNORE_TIMING	DPLD_PAD_PLACE PULLUP or PULLDOWN DPLD_OPDRIVE DPLD_OPLEVEL DPLD OPSLEW DPLD_IPELVE DPLD_PAD_PROPERTIES

MPADS .pat File Setup – I/O Setup

Using your favorite text editor or MPADS View Text File editor, we are going to start configuring your I/O. Open your HDL top module design file and select all of your input and output declarations and copy them into a new file called `<design>.pat`, which must be located in the same file directory as your EDIF netlists `<design>.edf`. Start assigning your input clocks and resets to `ipclk` and `iprst` instance primitives respectively, only if you have not already assigned them through Galileo's control file, by adding the `.pat` commands:

```
port <clock> instance ipclk
port <reset> instance iprst
```

If you mistakenly assigned `ipclk` and `iprst` primitives through both the `.ctr` and `.pat` files, double assignment may occur, adding an extra buffer delay within your design. Remember that the `ipclk` and `iprst` primitives assign clocks and resets from their input pads to the low skew networks within the MPA family.

Assignment of inputs in the MPA family are on an individual basis as TTL or CMOS level, so assign them accordingly adding the `.pat` commands:

```
port <clock> attribute dpld_iplevel cmos
port <reset> attribute dpld_iplevel ttl
```

Note that TTL input level is the default input pin attribute, but really should be specified for completeness.

Assignment of outputs, including bi-directional I/O, in the MPA family are also on an individual basis and can be configured with the attributes `pullup`, `pulldown`, `iplevel` (cmos or ttl input level), `oplevel` (3V or 5V output level), `opdrive` (6mA or 12mA output drive), and `opslew` (low or high slew output for 12mA drive only). It is recommended to use the pin attribute `pad_properties` to assign all of the above mentioned properties all at once with the following statement:

```
port <signal> attribute dpld_pad_properties
0,0,cmos,5V,12mA,high
```

The above command specifies that the output named `signal` has the attributes of no pullup, no pulldown, cmos input level, 5V output level, 12mA source/sink, and high slew rate. For



good design technique, configure all of your I/O by assigning a pin attribute command for each one of your pad signals.

User attribute assignments of the following are not permitted:

`opdrive 6ma` and `opslew low`.

`pullup 1` and `pulldown 1`.

The `opdrive 6ma` and `opslew low` selection is not permitted, because slew control is only available on 12mA drive.

The following combination of user attributes on a single bi-directional pad should be avoided, as it may produce unpredictable results:

`lplevel cmos` and `oplevel 3V`.

`Pad_place` is another attribute that can be used for configuring the I/O to specify the pin assignment of a signal. For new designs it is recommended that the `pad_place` attribute should not be used and let MPADS select the pin assignments during place and route, allowing for an optimal design output. `Pad_place` attributes can be used for production designs that require minor changes by copying these attributes from the `layout.ppr` file that is created by MPADS.

MPADS .pat File Setup – Timing Control

MPADS is a timing driven automatic place and route tool that includes capabilities for the user to specify how a design will perform in terms of the frequency of operation. This is based on the maximum worst case delay within the design. For combinatorial logic the worst delay is the longest path from an input pad to an output pad. For synchronous logic the worst case delay is the longest path, which may be from flip-flop to flip-flop, flip-flop to I/O, or I/O to flip-flop. The user can guide the timing driven algorithms of MPADS in three ways.

The first is with the adjustment of Auto Layout tool options, through the MPADS graphical user interface, with controls such as annealing temperature, target delays, target zone utilization, etc. The target delay function allows the user to specify the overall frequency of operation in terms of its period with resolution in increments of 0.1ns. Additional details on the available options and their use are available in the on-line help facility of the MPA Design System.

The second method involves the use of a separate clock file, designated as `design.clk` and is recommended for advanced users. Here again, additional information is provided in the on-line help system and is not presented in this application note.

The third method of influencing the timing driven place and route algorithms is through the use of the attributes file. Start off assigning your internally generated clocks and resets to the low skew networks with the `ack` and `arst` instance primitives respectively, by adding the `.pat` commands:

`net <int_clock> instance ack`

`net <int_reset> instance arst`

Note that you will probably need to review your structural netlists in order to correctly identify the nets that need to be

assigned to the low skew networks and that you are limited to eight for the entire design. You are now ready to assign timing attributes to nets to control the MPADS timing driven algorithms. There are basically three attributes that you will need to review:

The **ignore_timing** attribute is used to inform the tool which nets to ignore timing on. If during place and route a signal is identified as causing the worst case delay within a design (from the MPADS timing report layout.tim) and its timing is non critical, it should be assigned the attribute `ignore_timing`. The attribute may be set on an symbol (instance), a net, or an external pin (formal port). If a net has the attribute set, then all delay paths associated with that net are ignored. If an instance has the attribute set, then all input and output delay paths from that instance are ignored. Assigning the attribute to a formal port has exactly the same effects as assigning it to the I/O instance itself. Once all the objects to be ignored have been identified, their paths are propagated forwards and backwards through combinatorial gates until clocked objects (or top level circuit I/O) are reached. The result is that additional segments other than those explicitly specified may be ignored for timing purposes as well.

You are required to use a dummy value with this attribute, but the value stated is otherwise ignored. Assigning this attribute to a symbol, net, or formal port frees the timing driven autolayout algorithms to more optimally cluster, place, and route the speed critical nets. Examples include:

`net <signal> attribute ignore_timing dummy_arg`

`port <signal> attribute ignore_timing dummy_arg`

`instance <inst> attribute ignore_timing dummy_arg`

The **cluster_seed** attribute is used to assign a cluster seed to a net. This will cause the autolayout clustering algorithm to treat all instances that connect to that net specially. The action taken depends on the value of the attribute, as follows:

- 0 ignore this net during clustering. Setting this attribute on a net is likely to cause the net to be implemented in global interconnect.
- 1 default operation
- >1 weight this net by the given factor in the clustering

The maximum value for the `cluster_seed` attribute is 1000. Example:

`net <signal> attribute cluster_seed 2`

The **place_priority** attribute can be applied to a net to increase the chance that the autolayout algorithm will route the net more efficiently and with less delay in a physically smaller area, at the possible expense of surrounding nets. The value assigned to `place_priority` should be an integer in the range of 1 (default) to 10 inclusive. Higher values of place priority allow the designer to prioritize nets relative to each other. Example:

`net <signal> attribute place_priority 2`

It is really only through experience that MPADS users will learn how to fine tune their designs through the timing driven algorithms with the three timing control methods provided by the tools. A suggested methodology for high speed design is

4



to first assign critical clocks to the low skew networks by using the Galileo .ctr or MPADS .pat control files that instantiate the ipclk and ack primitives within the design. Try running the design through MPADS and if the design is not meeting the target speed goals, start assigning the ignore_timing attributes within the .pat file to those nets or pads that are not critical timing paths. Run MPADS again and if timing is not met, proceed with assigning critical logic sections using the cluster_seed attribute and critical nets using the place_priority attribute. These attributes will place and route these logic sections as high priority with critical timing algorithms. If more control over critical nets and instances is needed, proceed to the use of the MPADS .clk control file.

MPADS

The Motorola Programmable Array Design System (MPADS) is a push button automatic timing driven place and route tool set for MPA development. MPADS provides the capabilities to import multiple EDIF netlists, to place and route the design, to generate a configuration file for device programming, to back annotate a structural netlist for post simulation, and to view the layout of a completed chip design. The design files generated by a HDL designer include:

design.clk	MPADS timing control (optional)
design.edf	Top EDIF netlist (Galileo)
sub.edf	Hierarchical submodule EDIF netlist
design.pat	MPADS attributes file (optional)
design.v	Verilog source code
sub.v	Hierarchical Verilog submodule
test.v	Verilog test bench code (simulation)
design.vhd	VHDL source code
sub.vhd	Hierarchical VHDL sub-entity
test.vhd	VHDL test bench code (simulation)

Note that the filenames are recommendations, but the extensions are required. If a hierarchical design is used, the name "sub" should be replaced with the module or entity's name.

The files and directories created by MPADS include:

layout1.dsn	Design context
design	Autolayout results subdirectory
design.net	MPA-DS netlist
layout1.log	MPADS design log
layout1.cxt	Context
layout1.lyt	Layout
layout1.prp	Port report
layout1.tim	Timing report
layout1.vba	Verilog back annotation
layout1.vhd	VHDL back annotation

Note that the name "layout1.xxx" is only a suggestion and is only used as an example within this article for consistency.

MPADS Import

Netlist import is the process used by MPADS to import a single flat EDIF netlist or multiple EDIF netlists produced from Galileo into the Autolayout Tool. The Netlist Import tool runs in a standard Tool Execution Window and any errors

encountered during netlist import will be output to this window and to the c:/design/design/layout.log file. The import performs a reader function where it will open up the design.edf file and, if other unresolved modules are referenced in this netlist and the multiple reader option is enabled, import will search and open the other required EDIF files.

Once the netlist is read, import will proceed to search and read the attributes file design.pat. It will instantiate ipclk, iprst, ackl, arst, apbuf, and pabuf primitives if required and assign the additional attributes to nets, ports, and instances as specified in the .pat file.

The next retargeting stage involves checking for unused inputs and if any are detected the tool will fail import. The user must go back to the original HDL source code and remove the unused inputs from the design and resynthesize. Unused output signals are then searched, resulting in them being stripped from the design with all the associated unused logic. Bubble pushing is then executed where INV inverter primitives are stripped out, assigning them into the programmable input and output inverters within the MPA core cells.

Wired-OR splitting occurs next, where output drivers are separated from the input loads by the assignment of an intermediate buffer preparing the design for place and route.

The import process concludes by creating an intermediate netlist designated with the file name design.net. If errors are encountered during import, the design.net file name will not be generated. If you make changes to the .pat file after running import, you will need to rerun import to make sure that the changes get incorporated into the intermediate netlist.

MPADS Autolayout

The Autolayout Tool performs fully automatic timing driven layout for the current design. The Autolayout Tool runs in a Tool Execution Window and any errors that are encountered during layout will be output to this window and to the file c:/design/design/layout.log. Control over the autolayout process is provided through the autolayout option switches and attributes assigned through the .pat file.

The autolayout process first performs **definition checking** to make sure that the design definition is suitable for autolayout. It proceeds to a **clustering** phase, where sections of interconnected logic are grouped together, dividing the design into smaller pieces called clusters. **Partitioning** is then executed to assign clusters of logic into specific zones within the chip. **Placement** is executed assigning I/O to specific pin assignments automatically or to specific pin assignments specified by the user through the .pat file. **Global routing** is executed routing the global nets to the ports cells of the zones and routing the low skew network clocks. **Zone routing** is then executed routing each zone using local and medium bus interconnections for its cluster assignments. The autolayout process concludes with the generation of the following files:

layout.tim	— A timing report file which includes a list of the critical path nets, allowing the user to analyze the timing aspects of the layout
layout.prp	— A port report file which contains the pin out assignments of the resulting layout,
layout.lyt	— A layout file containing the final layout. This layout is used as the input to the configuration generator to create a



configuration bit-stream for the target device.
 layout.cxt – A context file which contains the complete data base generated from each phase of autolayout.

MPADS Autolayout Options

Autolayout options are set using the MPADS graphical user interface Tool Options button. The choices available are outlined here:

Autolayout: Parameter Group is used to reference a particular set of parameter settings. Any number of Autolayout Parameter Groups can be set up and saved under user specified group names for easy access to commonly used parameter settings. These parameter groups can be selected from the Autolayout options dialog. It is recommended for the new MPADS designer to start using the predefined parameter groups that are available. For high speed designs the Minimum Delay parameter group should be selected.

Autolayout: Utilization controls the number of functional cells that the partitioning phase will attempt to place within a zone, that consist of a 100 core cell area. Its value is the percentage of functional core cells in the imported design against the recommended maximum number of 50 usable core cells. Autolayout will automatically exceed this limit if it is required.

Minimum: 1 (0.5 % or 1 core cell)
 Maximum: 150 (75% or 75 core cells)
 Default: 80 (40% or 40 core cells)

Autolayout: Effort controls the amount of work applied to the partitioning operation. A larger value will generally produce a better result in the partitioning for high utilization circuits. The time taken for partitioning is directly related to Effort. Effort must be balanced by the Start Temp parameter for optimum results.

Minimum: 1
 Maximum: 100
 Default: 30

Autolayout: Start Temp controls the start temperature for the partitioning process, using a simulated annealing technique. A higher temperature allows the average cost of a move during partitioning to be greater. Increasing the start temperature may result in a better partition, especially in high utilization designs. Note that a high value of Start Temp may result in a bad partition if not used in conjunction with a larger value for the Effort parameter.

Minimum: 1
 Maximum: 100
 Default: 10

Simulated annealing is a software technique used for combinatorial optimization. It simulates the relaxation of stresses during the repeated heating and cooling of materials, where the temperature that the material is heated to each time is gradually reduced. Simulated annealing is a stochastic technique, and will produce different results for different sets of starting conditions (such as layout parameters, seed values, etc.). Simulated annealing is used by MPADS during the Partitioning and Zone Routing phases of autolayout.

Autolayout: Attempts controls the number of runs through the partitioning phase. More runs should deliver a better partition.

Minimum: 1
 Maximum: 20
 Default: 1

Autolayout: Backoff controls the percentage relaxation of the target delay if the initial target is not obtainable. This is expressed as a percentage of the previous target delay.

Minimum: 0%
 Maximum: 100%
 Default: 30%

Increasing this parameter makes the next attempt at autolayout to use a new Delay Target further away from the maximum possible delay thus making it easier to complete. This is expressed as a percentage increase in the optimum target delay.

Autolayout: Delay Cost controls the weighting given to timing during partitioning. If a particular partition includes a net that is failing to meet its timing target, then the cost of that partition will be artificially raised by an amount proportional to the delay cost. Increasing the delay cost is likely to trade off against achievable utilization.

Minimum: 0
 Maximum: 100
 Default: 5

Autolayout: Delay Target controls the desired maximum delay target for the autolayout process to achieve, in steps of 0.1ns. For combinatorial circuits, the maximum delay is calculated from the longest path from input to output. For synchronous circuits, the maximum delay is calculated from the longest path between flip-flops, or from I/O to flip-flop, whichever is the greater. This parameter is only relevant when you are using the standard timing model. If a clock file is specified, then the Delay Target parameter will be ignored.

Minimum: 0
 Maximum: 9999
 Default: 50 (5ns)

Autolayout: Fanout controls the maximum fan-out of a net which is included in the clustering. All the nets which have a fan-out greater than this value are ignored during clustering. Any nets ignored during clustering are more likely to have to be globally routed.

Minimum: 1
 Maximum: 100
 Default: 2

Autolayout: Min Zone Delay controls the delay of segments within a zone to be within a certain value. The delay is specified in unit of 0.1ns. The smaller the delay is, the harder the zone routing algorithm will work to reduce delays within the zones.

Minimum: 0
 Maximum: 9999
 Default: 5 (0.5ns)

Advanced Autolayout: Seed Setting the seed for the autolayout process lets you control the repeatability of the autolayout. If you run the autolayout with a given seed value,

4



you can reproduce the same autolayout from the same circuit simply by setting the same seed value.

```
Minimum: 0
Maximum: 9999
Default: 1
```

Several of the autolayout algorithms use stochastic techniques similar to simulated annealing that require a pseudo-random number generator. The Seed is used to set the random number generator to a known point in the sequence.

MPADS Layout Viewer

The MPA Design System provides a Layout Viewer tool which gives a graphical view of the completed layout. This view shows the instances of primitives in the design and the routing that connects them together. Aspects of the device architecture can also be viewed.

Multiple graphical and browser views may be created. Moving about in the graphical view is made possible by zooming and panning. Instances, nets and ports can be selected. The mouse cursor changes according to the design object under it, and the status bar gives more information about the design object beneath the cursor.

Note that no changes to the layout are possible with the Layout Viewer. Please refer to the on-line help facility for more details.

4

MPADS Device Configuration

Device configuration is the process of translating a completed layout into a stream of 1's and 0's used to program the actual device. Pressing the device configuration button launches this tool. Please refer to the on-line help facility for more details.

MPADS Back Annotation

The MPA Design System has a back annotation tool that incorporates the post place/route gate and net delay timing into a structural netlist in Verilog (layout.vba) and for VHDL 1076 (layout.vhd) formats. These netlists are useful for verifying worst case timing of a design with third party simulators.

The Standard Delay Format (SDF) will be supported in future releases of MPADS.

Verilog Simulation

The structural Verilog file that is produced by MPADS assumes that you will be using a stimulus file (test bench), and that the module name of your stimuli is stimulus. It is also assumed that you instantiate your design within your stimulus file with an instance name of u1, for example:

```
// Stimulus file, contains stimuli for circuit DESIGN
// the module name is stimulus
module stimulus;
reg IN, CLK, RESET;
// the instance name is u1
DESIGN u1(OUTPUT,IN,CLK,RESET);
<rest of declarations and stimuli>
```

In the Verilog back annotation file layout1.vba, the design is instantiated with the name of the EDIF file that was passed into the MPA Design System. If a stimulus file is not used (eg. if the design is the root of the hierarchy), or if you choose to use different names for your stimulus and instance names, then edit the layout1.vba file and replace the term stimulus.u1 in the file that says:

```
'define module_name stimulus.u1
```

with your design name. For the example above, you would enter:

```
'define module_name DESIGN
```

The back annotation information that is produced by the autolayout tool (as a by-product of the autolayout process) is generated by a different method from the data generated by the more accurate back annotation tool. Some values are different for the two back annotation files (even though they will both be called layout1.vba). It is recommended that you post simulate the layout1.vba file generated by the back annotation tool to simulate the true timing of the placed and routed design. Please enter the line to your stimulus file when you simulate files from the autolayout tool:

```
'define source_autolayout
```

If you are performing a back annotation tool simulation, then please enter the line into your stimulus file:

```
'define source_back_annotation
```

Note that running back annotation will overwrite the layout1.vba file with data from the back annotation tool, but this is fine, since you should be post simulating from the back annotation netlist.

To allow for the best cross-platform capability and flexibility, MPADS has been designed not to be case sensitive. The Verilog hardware description language, however, is case sensitive. If a design is named in lower case, or a mixture of lower case and upper case, then the name of the design will be converted to all capitals in the Verilog back annotation file. This will require you to change your signal names and module name from lower to upper case within your test bench file, stimulus.v.

Please note that it is not possible to infer the correct ordering of a bus from the information contained in an EDIF netlist and MPADS will arbitrarily order busses in big-endian style (for example input [15:0]MYBUS;) in the input/output declarations section of the Verilog back annotation file. If you are writing a new stimulus file (or test bench) for use with the MPA Design System, please use big-endian style. If you are using the MPA Design System with a preexisting stimulus file that has busses ordered in little-endian style (for example input [0:15]MYBUS;) then you must re-order the bus declarations either in the stimulus file or in the Verilog back annotation file.

Leaving inputs unconnected (with no driver) is not a supported method of dealing with unused inputs. It is recommended that pre-layout simulation is performed to detect any inputs unintentionally left unconnected. Post-layout simulation may not show up unconnected inputs. In the post-layout simulation, unconnected inputs may be connected to logic high for simulation purposes.



There are some differences between Verilog simulators. If you are having difficulty with the library, try disabling the line of the library (`c:/mpa/verilog/library.v`) that reads:

```
'define XL_comp
```

and enabling the line that reads:

```
// 'define OVI_comp
```

The conditions under which you may need to do this are explained more fully in the library itself.

VHDL Simulation

The VHDL Simulation Library is equivalent to the `microlib` and `iolib` Technology Schematic and has been separated into two files:

1. **Technology Cell Simulation Models** (see `c:/mpa/vhdl/modelmpa.vhd`) Each model has a generic map to provide a mechanism for passing timing parameters from MPADS back annotation facility. The functionality of each Technology Cell is described using in house behavioral models. The VHDL Initiative Towards ASIC Libraries (VITAL) is not yet supported.
2. **Technology Cell Component Declarations** (see `c:/mpa/vhdl/packmpa.vhd`) provide the Component Declarations necessary for you to instantiate Technology Cells into your source designs and it is also used by the MPADS back annotation tool. The file is split into `microlib` and `iolib` MPA1000 technology libraries. To make the libraries visible within your VHDL design code include the statements:

```
library mpa; -- for Motorola MPA1000
use mpa.microlib.all;
```

```
use mpa.iolib.all;
```

The VHDL Simulation Library has been extensively tested using Model Technology V–System v4.2e simulator, although it should be compatible with all other comprehensive VHDL Simulators. Both files need to be compiled into a Specific Technology Library:

```
library mpa; -- for Motorola MPA1000
```

The VHDL Simulation Library requires visibility to the following packages:

```
use Std.Standard.all;
use IEEE.Std_logic_1164.all;
```

The MPADS Back Annotation tool provides a mechanism for performing Post–Layout simulation of a synthesized/structural VHDL design. The term back annotation is used loosely with reference to the MPADS tool, which generates a structural VHDL representation of the whole design including:

1. Any Technology Cell optimizations which were made during import.
2. Technology Cell intrinsic timing delays
3. Net routing timing delays

VHDL Post–layout Simulation can be applied to designs which were not captured in VHDL. The VITAL Standard Delay File Format (SDF) is not currently supported by the MPA Design System.

A known problem exists with Bi–Directional Entity Ports, where the Mode of a Entity Port Declaration is not set to 'inout' if the port connects to the External port (device pin) of a Bi–directional I/O Cell. The fix is to manually change port mode from 'out' to 'inout'

4



Appendix A – EDIF Netlist Example

```

(edif REFRESH
(edifVersion 2 0 0)
(edifLevel 0)
(keywordMap (keywordLevel 0))
(external extlib (edifLevel 0)
(technology (numberDefinition))
(cell INV (cellType GENERIC)
(view netlist (viewType NETLIST)
(interface
(port A (direction INPUT))
(port QN (direction OUTPUT))))))
(cell AN2 (cellType GENERIC)
(view netlist (viewType NETLIST)
(interface
(port B (direction INPUT))
(port A (direction INPUT))
(port Q (direction OUTPUT))))))
(cell OPBUF (cellType GENERIC)
(view netlist (viewType NETLIST)
(interface
(port O (direction INPUT))
(port EXTOUT (direction OUTPUT))))))
(cell IPBUF (cellType GENERIC)
(view netlist (viewType NETLIST)
(interface
(port EXTIN (direction INPUT))
(port I (direction OUTPUT))))))
(cell DFE (cellType GENERIC)
(view netlist (viewType NETLIST)
(interface
(port E (direction INPUT))
(port D (direction INPUT))
(port CLK (direction INPUT))
(port Q (direction OUTPUT))))))
(cell DF (cellType GENERIC)
(view netlist (viewType NETLIST)
(interface
(port D (direction INPUT))
(port CLK (direction INPUT))
(port Q (direction OUTPUT))))))
(library REFRESH
(edifLevel 0)
(technology (numberDefinition))
(cell REFRESH (cellType GENERIC)
(view Synthesis (viewType NETLIST)
(interface
(port s_25mhz (direction INPUT))
(port timerclk (direction OUTPUT))))
(contents
(instance (rename i1 "i315") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i2 "i320") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i3 "i325") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i4 "i330") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i5 "i335") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i6 "i340") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i7 "i341") (viewRef netlist (cellRef AN2 (libraryRef extlib))))
(instance (rename i8 "i336") (viewRef netlist (cellRef AN2 (libraryRef extlib))))
(instance (rename i9 "i331") (viewRef netlist (cellRef AN2 (libraryRef extlib))))
(instance (rename i10 "i326") (viewRef netlist (cellRef AN2 (libraryRef extlib))))

```

4



```

(instance (rename i11 "i321") (viewRef netlist (cellRef AN2 (libraryRef extlib))))
(instance (rename i12 "i316") (viewRef netlist (cellRef AN2 (libraryRef extlib))))
(instance (rename i13 "i345") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i0 "temp_0__0") (viewRef netlist (cellRef INV (libraryRef extlib))))
(instance (rename i14 "i180") (viewRef netlist (cellRef OPBUF (libraryRef extlib))))
(instance (rename i15 "IPBUF_s_25mhz_int") (viewRef netlist (cellRef IPBUF (libraryRef
extlib))))
(instance XMPLR_INST_29 (viewRef netlist (cellRef DFE (libraryRef extlib))))
(instance XMPLR_INST_31 (viewRef netlist (cellRef DFE (libraryRef extlib))))
(instance XMPLR_INST_33 (viewRef netlist (cellRef DFE (libraryRef extlib))))
(instance XMPLR_INST_35 (viewRef netlist (cellRef DFE (libraryRef extlib))))
(instance XMPLR_INST_37 (viewRef netlist (cellRef DFE (libraryRef extlib))))
(instance XMPLR_INST_39 (viewRef netlist (cellRef DFE (libraryRef extlib))))
(instance XMPLR_INST_41 (viewRef netlist (cellRef DFE (libraryRef extlib))))
(instance XMPLR_INST_43 (viewRef netlist (cellRef DF (libraryRef extlib))))
(net s_25mhz
  (joined
    (portRef s_25mhz)
    (portRef EXTIN (instanceRef i15))))
(net s_25mhz_int
  (joined
    (portRef I (instanceRef i15))
    (portRef CLK (instanceRef XMPLR_INST_29))
    (portRef CLK (instanceRef XMPLR_INST_31))
    (portRef CLK (instanceRef XMPLR_INST_33))
    (portRef CLK (instanceRef XMPLR_INST_35))
    (portRef CLK (instanceRef XMPLR_INST_37))
    (portRef CLK (instanceRef XMPLR_INST_39))
    (portRef CLK (instanceRef XMPLR_INST_41))
    (portRef CLK (instanceRef XMPLR_INST_43))))
(net (rename n0 "timerclk")
  (joined
    (portRef EXTOUT (instanceRef i14))
    (portRef timerclk)))
(net timerclk_int
  (joined
    (portRef Q (instanceRef XMPLR_INST_29))
    (portRef A (instanceRef i1))
    (portRef O (instanceRef i14))))
(net (rename ref_6 "ref_6_")
  (joined
    (portRef Q (instanceRef XMPLR_INST_31))
    (portRef A (instanceRef i2))
    (portRef A (instanceRef i12))))
(net (rename ref_5 "ref_5_")
  (joined
    (portRef Q (instanceRef XMPLR_INST_33))
    (portRef A (instanceRef i3))
    (portRef A (instanceRef i11))))
(net (rename ref_4 "ref_4_")
  (joined
    (portRef Q (instanceRef XMPLR_INST_35))
    (portRef A (instanceRef I4))
    (portRef A (instanceRef i10))))
(net (rename ref_3 "ref_3_")
  (joined
    (portRef Q (instanceRef XMPLR_INST_37))
    (portRef A (instanceRef i5))
    (portRef A (instanceRef i9))))
(net (rename ref_2 "ref_2_")

```



```

(joined
  (portRef Q (instanceRef XMPLR_INST_39))
  (portRef A (instanceRef i6))
  (portRef A (instanceRef i8))))
(net (rename ref_1 "ref_1_")
  (joined
    (portRef Q (instanceRef XMPLR_INST_41))
    (portRef B (instanceRef i7))
    (portRef A (instanceRef i13))))
(net (rename ref_0 "ref_0_")
  (joined
    (portRef Q (instanceRef XMPLR_INST_43))
    (portRef A (instanceRef i7))
    (portRef E (instanceRef XMPLR_INST_41))
    (portRef A (instanceRef i0))))
(net (rename n1 "n315")
  (joined
    (portRef QN (instanceRef i1))
    (portRef D (instanceRef XMPLR_INST_29))))
(net (rename n2 "n320")
  (joined
    (portRef QN (instanceRef i2))
    (portRef D (instanceRef XMPLR_INST_31))))
(net (rename n3 "n325")
  (joined
    (portRef QN (instanceRef i3))
    (portRef D (instanceRef XMPLR_INST_33))))
(net (rename n4 "n330")
  (joined
    (portRef QN (instanceRef i4))
    (portRef D (instanceRef XMPLR_INST_35))))
(net (rename n5 "n335")
  (joined
    (portRef QN (instanceRef i5))
    (portRef D (instanceRef XMPLR_INST_37))))
(net (rename n6 "n340")
  (joined
    (portRef QN (instanceRef i6))
    (portRef D (instanceRef XMPLR_INST_39))))
(net (rename n7 "n341")
  (joined
    (portRef Q (instanceRef i7))
    (portRef B (instanceRef i8))
    (portRef E (instanceRef XMPLR_INST_39))))
(net (rename n8 "n336")
  (joined
    (portRef Q (instanceRef i8))
    (portRef B (instanceRef i9))
    (portRef E (instanceRef XMPLR_INST_37))))
(net (rename n9 "n331")
  (joined
    (portRef Q (instanceRef i9))
    (portRef B (instanceRef i10))
    (portRef E (instanceRef XMPLR_INST_35))))
(net (rename n10 "n326")
  (joined
    (portRef Q (instanceRef i10))
    (portRef B (instanceRef i11))
    (portRef E (instanceRef XMPLR_INST_33))))
(net (rename n11 "n321")

```

4



```

(joined
  (portRef Q (instanceRef i11))
  (portRef B (instanceRef i12))
  (portRef E (instanceRef XMPLR_INST_31))))
(net (rename n12 "n316")
  (joined
    (portRef Q (instanceRef i12))
    (portRef E (instanceRef XMPLR_INST_29))))
(net (rename n13 "n345")
  (joined
    (portRef QN (instanceRef i13))
    (portRef D (instanceRef XMPLR_INST_41))))
(net (rename temp_0 "temp_0_")
  (joined
    (portRef QN (instanceRef i0))
    (portRef D (instanceRef XMPLR_INST_43))))))
(design REFRESH
  (cellRef REFRESH
    (libraryRef REFRESH)))

```

Appendix B – Verilog Back Annotated Netlist Example

```

// Verilog delay model produced by MPA Design System 2.3.0
// on Thu Dec 05 08:04:30 1996
// prologue.vba start
`timescale 100ps/100ps
// setup timescale to correct value for back annotated values
`define post_layout
// controls the functionality of the library.v file to indicate
// that a post-layout simulation is being performed
`define module_name stimulus.ul
// this assumes you are using a stimulus file, where the module
// name of the stimuli is stimulus and you have instantiated
// your design with an instance name of ul.
`default_nettype wire
// implicitly declared nets are of type wire
// `define source_autolayout
`define source_back_annotation
// If you have not already made one of these defines in your
// stimulus file, please enable one of the above statements.
`ifdef source_autolayout
// If this file was written by the autolayout software, please
// define source_autolayout in your stimulus file, otherwise
// the values for back annotation will be used.
// The values written by the autolayout software for the
// parameters below are ignored in favour of the values below.
`define TPHLCQ 15
`define TPLHCQ 15
`define TPHLRQ 15
`define TPLHRQ 15
`define TPPLSQ 15
`define TPLHSQ 15
`define TPLHDQ 35
`define TPLHDQ 35
`define TPHLEQ 25
`define TPLHEQ 25
`define TSUD 15
  `ifdef source_back_annotation
    module display;
      initial begin
$display("Please use only one of `define source_autolayout");
$display("or `define source_back_annotation, not both.");

```

4




```

        end
    endmodule

    `endif
    module display_mode;
initial $display("Autolayout values have been selected.");
    endmodule

`else
// If this file was written by the back annotation software, please
// define source_back_annotation in your stimulus file.
// The values written by the back annotation software for the
// parameters below are ignored in favour of the values below.
`define TPHLCQ 13
`define TPLHCQ 13
`define TPHLRQ 13
`define TPLHRQ 13
`define TPHLSQ 13
`define TPLHSQ 13
`define TPHLDQ 13
`define TPLHDQ 13
`define TPHLEQ 13
`define TPLHEQ 13
`define TSUD 14
    module display_mode;
initial $display("Back Annotation values have been selected");
    endmodule
    `ifdef source_back_annotation
    `else
        module display;
            initial begin
                $display("The simulator does not know if your data has come from");
                $display("the AutoLayout software or the Back Annotation software.");
                $display("Back Annotation software values have been used as default.");
                $display("Please enter the statement `define source_autolayout or");
                $display("`define source_back_annotation in your stimulus file as");
                $display("appropriate.");
            end
        endmodule
    `endif
`endif

`define tCLKMARK 15
// minimum time from CLK ^ to CLK v
`define tCLKSPACE 15
// minimum time from CLK v to CLK ^
`define tRECOVERY 15
// minimum time from RN ^ or SN ^ to CLK ^
`define tRESETHOLD 15
// minimum time from RN v to RN ^
`define tSETHOLD 15
// minimum time from SN v to SN ^
`define tEMARK 15
// minimum time from E ^ to E v
`define tESPACE 15
// minimum time from E v to E ^
`define tETOCCLK 3
// minimum time from E v or E ^ to CLK ^
`define tCLKTOE 15
// minimum time from CLK ^ to E v
`define tSUD `TSUD
// minimum time from D ^ or D v to E v (latches) or CLK ^ (FlipFlops)
// There is no minimum hold time. Some of the above apply only to

```

4



```

// latches, and some apply only to FlipFlops.
// Infringements of the above minimums should generate warning messages
// in most simulators, even though the simulation may perform correctly.
// It is unlikely that the real device will behave the same way as the
// simulation if any of the above warnings are encountered during
// post-layout simulation.
// prologue.vba end
module REFRESH(S_25MHZ, TIMERCLK);
    input S_25MHZ;
    output TIMERCLK;
    wire S_25MHZ;
    wire TIMERCLK;
    wire N315;
    wire N316;
    wire N320;
    wire N321;
    wire N325;
    wire N326;
    wire N330;
    wire N331;
    wire N335;
    wire N336;
    wire N340;
    wire N341;
    wire N345;
    wire S_25MHZ_INT;
    wire TEMP_0_;
    IPCLK #(0, 0, 0, 0) IPBUF_S_25MHZ_INT
        (.I(S_25MHZ_INT) ,
         .EXTIN(S_25MHZ) );
    OPBUF #(0, 0, 21, 21) I180
        (.EXTOUT(TIMERCLK) ,
         .O(N315) );
    AN2 #(0, 0, 33, 29, 18, 14) I316
        (.Q(N316) ,
         .A(N320) ,
         .B(N321) );
    AN2 #(0, 0, 40, 36, 12, 8) I321
        (.Q(N321) ,
         .A(N325) ,
         .B(N326) );
    AN2 #(0, 0, 46, 43, 14, 9) I326
        (.Q(N326) ,
         .A(N330) ,
         .B(N331) );
    AN2 #(0, 0, 13, 8, 12, 8) I331
        (.Q(N331) ,
         .A(N335) ,
         .B(N336) );
    AN2 #(0, 0, 26, 21, 23, 17) I336
        (.Q(N336) ,
         .A(N340) ,
         .B(N341) );
    AN2 #(0, 0, 13, 9, 13, 8) I341
        (.Q(N341) ,
         .A(TEMP_0_) ,
         .B(N345) );
    DFE #(0, 0, 21, 21, 24, 24, 6, 4) XMPLR_INST_29
        (.Q(N315) ,
         .CLK(S_25MHZ_INT) ,

```



```

        .D(~N315) ,
        .E(N316) );
DFE #(0, 0, 21, 21, 23, 24, 22, 20) XMPLR_INST_31
    (.Q(N320) ,
     .CLK(S_25MHZ_INT) ,
     .D(~N320) ,
     .E(N321) );
DFE #(0, 0, 21, 21, 21, 6, 4) XMPLR_INST_33
    (.Q(N325) ,
     .CLK(S_25MHZ_INT) ,
     .D(~N325) ,
     .E(N326) );
DFE #(0, 0, 21, 21, 23, 24, 19, 16) XMPLR_INST_35
    (.Q(N330) ,
     .CLK(S_25MHZ_INT) ,
     .D(~N330) ,
     .E(N331) );
DFE #(0, 0, 21, 21, 21, 33, 31) XMPLR_INST_37
    (.Q(N335) ,
     .CLK(S_25MHZ_INT) ,
     .D(~N335) ,
     .E(N336) );
DFE #(0, 0, 21, 21, 24, 24, 23, 19) XMPLR_INST_39
    (.Q(N340) ,
     .CLK(S_25MHZ_INT) ,
     .D(~N340) ,
     .E(N341) );
DFE #(0, 0, 21, 21, 21, 7, 5) XMPLR_INST_41
    (.Q(N345) ,
     .CLK(S_25MHZ_INT) ,
     .D(~N345) ,
     .E(TEMP_0_) );
DF #(0, 0, 21, 21, 21, 22) XMPLR_INST_43
    (.Q(TEMP_0_) ,
     .CLK(S_25MHZ_INT) ,
     .D(~TEMP_0_) );

```

```
endmodule
```

```
module annotate;
```

```
defparam
```

```

`module_name.IPBUF_S_25MHZ_INT.TPLHIN = 19,
`module_name.IPBUF_S_25MHZ_INT.TPHLIN = 19,
`module_name.I180.TPLHOUT = 50,
`module_name.I180.TPHLOUT = 46,
`module_name.I316.TPLH = 0,
`module_name.I316.TPHL = 0,
`module_name.I321.TPLH = 0,
`module_name.I321.TPHL = 0,
`module_name.I326.TPLH = 0,
`module_name.I326.TPHL = 0,
`module_name.I331.TPLH = 0,
`module_name.I331.TPHL = 0,
`module_name.I336.TPLH = 0,
`module_name.I336.TPHL = 0,
`module_name.I341.TPLH = 0,
`module_name.I341.TPHL = 0,
`module_name.XMPLR_INST_29.TSUD = 14,
`module_name.XMPLR_INST_29.TPLHCQ = 13,
`module_name.XMPLR_INST_29.TPHLCQ = 13,
`module_name.XMPLR_INST_31.TSUD = 14,
`module_name.XMPLR_INST_31.TPLHCQ = 13,

```

4



```

'module_name.XMPLR_INST_31.TPHLCQ = 13,
'module_name.XMPLR_INST_33.TSUD = 14,
'module_name.XMPLR_INST_33.TPLHCQ = 13,
'module_name.XMPLR_INST_33.TPHLCQ = 13,
'module_name.XMPLR_INST_35.TSUD = 14,
'module_name.XMPLR_INST_35.TPLHCQ = 13,
'module_name.XMPLR_INST_35.TPHLCQ = 13,
'module_name.XMPLR_INST_37.TSUD = 14,
'module_name.XMPLR_INST_37.TPLHCQ = 13,
'module_name.XMPLR_INST_37.TPHLCQ = 13,
'module_name.XMPLR_INST_39.TSUD = 14,
'module_name.XMPLR_INST_39.TPLHCQ = 13,
'module_name.XMPLR_INST_39.TPHLCQ = 13,
'module_name.XMPLR_INST_41.TSUD = 14,
'module_name.XMPLR_INST_41.TPLHCQ = 13,
'module_name.XMPLR_INST_41.TPHLCQ = 13,
'module_name.XMPLR_INST_43.TSUD = 14,
'module_name.XMPLR_INST_43.TPLHCQ = 13,
'module_name.XMPLR_INST_43.TPHLCQ = 13;
endmodule

```

Appendix C – VHDL Back Annotated Netlist Example

```

-- VHDL delay model produced by MPA Design System 2.3.0
-- on Wed Dec 04 15:13:17 1996
library ieee;
use ieee.std_logic_1164.all;
-- Motorola MPA1000 FPGA Family
library mpa;
use mpa.iolib.all;
use mpa.microlib.all;
--
entity REFRESH is
  port (
    S_25MHZ : in std_logic;
    TIMERCLK : out std_logic);
end REFRESH;
architecture LAYOUT1 of REFRESH is
  signal
    N315,
    nN315,
    N316,
    N320,
    nN320,
    N321,
    N325,
    nN325,
    N326,
    N330,
    nN330,
    N331,
    N335,
    nN335,
    N336,
    N340,
    nN340,
    N341,
    N345,
    nN345,
    S_25MHZ_INT,
    TEMP_0,
    nTEMP_0 : std_logic;

```

4



AN1604

begin

```

IPBUF_S_25MHZ_INT : IPCLK
    generic map (
        TPLHIN => 1.9 ns,
        TPHLIN => 1.9 ns)
    port map (
        I => S_25MHZ_INT,
        EXTIN => S_25MHZ);

I180 : OPBUF
    generic map (
        TPLHOUT => 5.0 ns,
        TPHLOUT => 4.6 ns,
        TPLHO => 2.1 ns,
        TPHLO => 2.1 ns)
    port map (
        EXTOUT => TIMERCLK,
        O => N315);

I316 : AN2
    generic map (
        TPLH => 0.0 ns,
        TPHL => 0.0 ns,
        TPLHA => 3.3 ns,
        TPHLA => 2.9 ns,
        TPLHB => 1.8 ns,
        TPHLB => 1.4 ns)
    port map (
        Q => N316,
        A => N320,
        B => N321);

I321 : AN2
    generic map (
        TPLH => 0.0 ns,
        TPHL => 0.0 ns,
        TPLHA => 4.0 ns,
        TPHLA => 3.6 ns,
        TPLHB => 1.2 ns,
        TPHLB => 0.8 ns)
    port map (
        Q => N321,
        A => N325,
        B => N326);

I326 : AN2
    generic map (
        TPLH => 0.0 ns,
        TPHL => 0.0 ns,
        TPLHA => 4.6 ns,
        TPHLA => 4.3 ns,
        TPLHB => 1.4 ns,
        TPHLB => 0.9 ns)
    port map (
        Q => N326,
        A => N330,
        B => N331);

I331 : AN2
    generic map (
        TPLH => 0.0 ns,
        TPHL => 0.0 ns,
        TPLHA => 1.3 ns,
        TPHLA => 0.8 ns,
        TPLHB => 1.2 ns,

```

4



```

        TPHLB => 0.8 ns)
port map (
    Q => N331,
    A => N335,
    B => N336);

I336 : AN2

generic map (
    TPLH => 0.0 ns,
    TPHL => 0.0 ns,
    TPLHA => 2.6 ns,
    TPHLA => 2.1 ns,
    TPLHB => 2.3 ns,
    TPHLB => 1.7 ns)
port map (
    Q => N336,
    A => N340,
    B => N341);

I341 : AN2

generic map (
    TPLH => 0.0 ns,
    TPHL => 0.0 ns,
    TPLHA => 1.3 ns,
    TPHLA => 0.9 ns,
    TPLHB => 1.3 ns,
    TPHLB => 0.8 ns)
port map (
    Q => N341,
    A => TEMP_0,
    B => N345);

XMPLR_INST_29 : DFE

generic map (
    TSUD => 1.4 ns,
    TPLHCQ => 1.3 ns,
    TPFLCQ => 1.3 ns,
    TPLHCLK => 2.1 ns,
    TPFLCLK => 2.1 ns,
    TPLHD => 2.4 ns,
    TPFLD => 2.4 ns,
    TPLHE => 0.6 ns,
    TPFLHE => 0.4 ns)
port map (
    Q => N315,
    CLK => S_25MHZ_INT,
    D => nN315,
    E => N316);

XMPLR_INST_31 : DFE

generic map (
    TSUD => 1.4 ns,
    TPLHCQ => 1.3 ns,
    TPFLCQ => 1.3 ns,
    TPLHCLK => 2.1 ns,
    TPFLCLK => 2.1 ns,
    TPLHD => 2.3 ns,
    TPFLD => 2.4 ns,
    TPLHE => 2.2 ns,
    TPFLHE => 2.0 ns)
port map (
    Q => N320,
    CLK => S_25MHZ_INT,
    D => nN320,

```



```

        E => N321);
XMPLR_INST_33 : DFE
    generic map (
        TSUD => 1.4 ns,
        TPLHCQ => 1.3 ns,
        TPHLCQ => 1.3 ns,
        TPLHCLK => 2.1 ns,
        TPHLCLK => 2.1 ns,
        TPLHD => 2.1 ns,
        TPHLD => 2.1 ns,
        TPLHE => 0.6 ns,
        TPHE => 0.4 ns)
    port map (
        Q => N325,
        CLK => S_25MHZ_INT,
        D => nN325,
        E => N326);
XMPLR_INST_35 : DFE
    generic map (
        TSUD => 1.4 ns,
        TPLHCQ => 1.3 ns,
        TPHLCQ => 1.3 ns,
        TPLHCLK => 2.1 ns,
        TPHLCLK => 2.1 ns,
        TPLHD => 2.3 ns,
        TPHLD => 2.4 ns,
        TPLHE => 1.9 ns,
        TPHE => 1.6 ns)
    port map (
        Q => N330,
        CLK => S_25MHZ_INT,
        D => nN330,
        E => N331);
XMPLR_INST_37 : DFE
    generic map (
        TSUD => 1.4 ns,
        TPLHCQ => 1.3 ns,
        TPHLCQ => 1.3 ns,
        TPLHCLK => 2.1 ns,
        TPHLCLK => 2.1 ns,
        TPLHD => 2.1 ns,
        TPHLD => 2.1 ns,
        TPLHE => 3.3 ns,
        TPHE => 3.1 ns)
    port map (
        Q => N335,
        CLK => S_25MHZ_INT,
        D => nN335,
        E => N336);
XMPLR_INST_39 : DFE
    generic map (
        TSUD => 1.4 ns,
        TPLHCQ => 1.3 ns,
        TPHLCQ => 1.3 ns,
        TPLHCLK => 2.1 ns,
        TPHLCLK => 2.1 ns,
        TPLHD => 2.4 ns,
        TPHLD => 2.4 ns,
        TPLHE => 2.3 ns,
        TPHE => 1.9 ns)

```

4



```

    port map (
        Q => N340,
        CLK => S_25MHZ_INT,
        D => nN340,
        E => N341);
XEMPLR_INST_41 : DFE
    generic map (
        TSUD => 1.4 ns,
        TPLHCQ => 1.3 ns,
        TPPLCQ => 1.3 ns,
        TPLHCLK => 2.1 ns,
        TPPLCLK => 2.1 ns,
        TPLHD => 2.1 ns,
        TPPLD => 2.1 ns,
        TPLHE => 0.7 ns,
        TPPLHE => 0.5 ns)
    port map (
        Q => N345,
        CLK => S_25MHZ_INT,
        D => nN345,
        E => TEMP_0);
XEMPLR_INST_43 : DF
    generic map (
        TSUD => 1.4 ns,
        TPLHCQ => 1.3 ns,
        TPPLCQ => 1.3 ns,
        TPLHCLK => 2.1 ns,
        TPPLCLK => 2.1 ns,
        TPLHD => 2.1 ns,
        TPPLD => 2.2 ns)
    port map (
        Q => TEMP_0,
        CLK => S_25MHZ_INT,
        D => nTEMP_0);
-- Inverted Sense Nets
nN315 <= not N315;
nN320 <= not N320;
nN325 <= not N325;
nN330 <= not N330;
nN335 <= not N335;
nN340 <= not N340;
nN345 <= not N345;
nTEMP_0 <= not TEMP_0;
end LAYOUT1;

```



Integrating Schematic Capture and Verilog Synthesis When Designing with the MPA

Prepared by
Rich Rejmaniak
Motorola Field Applications Engineer

4



Integrating Schematic Capture and Verilog Synthesis When Designing with the Motorola Programmable Array

Introduction

The FPGA tool environment is currently moving in a direction where synthesis techniques are supplanting schematic capture for design entry. While tool vendors in almost all cases provide the means for complete development within their respective environments, most engineering departments must deal with legacy circuits developed using a variety of tools as well as engineers with different levels of expertise in various tool sets. The result is a situation where the integration of different design methodologies can be important to the success of a project. This application note details the integration of a Verilog module, compiled with Exemplar's Galileo, instantiated into a schematic capture using Viewlogic's Workview Office. The resulting design is placed into a Motorola Programmable Array compatible with the Motorola MPA demo board revision 2.

Example Circuit

The circuit used as an example in this document is a simple counter circuit. It uses the 2 MHz oscillator on the MPA demo board as a clock source. The clock is divided down into the visual range (one to four Hz) and drives a counter. The counter mode can be set to binary or BCD. The result drives a seven segment display.

The topmost level is entered using schematic capture, as is the primary clock divider and the display driver section. The counter, frequency selection, and mode control are implemented in a compiled Verilog module.

Top level schematic

The overall design is contained in the schematic shown in Schematic 1. It is composed of three modules, two of which are schematic based. It also defines all of the I/O for the design.

I/O cells

I/O cells are defined throughout the design to specify the type and functionality of each signal entry and exit point. While all I/O signals terminate at the top level schematic, only two of the signals have the I/O circuitry defined at this level. The {CLK}, {DEBUG}, and {RADIX} input signals have their I/O cells defined here. The {CLK} uses an "IPCLK" buffer that specifies that this signal should be placed on the low skew clock distribution network within the Motorola FPGA. The {RADIX} signal (which selection binary vs. BCD mode) and the {DEBUG} signal (which disables the prescaler and allows faster simulation) enter through an "IPBUF" cell. The IPBUF cell is a generic input buffer used for general signal input. The {RESET-L} signal initializes the sequential elements of the system. This signal is brought through an "IPRST" buffer that places the signal on the high fanout clock distribution network within the device. It should be noted that specifying I/O is probably the most common use of schematic capture in mixed tool designs that otherwise rely heavily on synthesis.

Input signals {SWITCH0} and {SWITCH1} are used to select the frequency of the counter. While they terminate at this level, their I/O characteristics are specified within the Verilog source code. This method of I/O instantiation is used in completely synthesized designs, as well as designs that use a synthesized module at the topmost level of the project (a reversal of the design methodology used in this example).

Outputs {A} through {G} have their I/O cells specified using schematic capture below the top of the hierarchy. Just as the inputs {SWITCH0} and {SWITCH1} are specified in a sub module, these outputs terminate at the top level schematic.

Clock divide

The clock divider in Schematic 2 is a 19 bit counter that generates a one clock period wide enable pulse at terminal count. With an input of 2.0 MHz, the output pulse will be 500 ns wide every 262 ms, corresponding to a frequency of 3.8 Hz. This division is disabled by driving the {DEBUG} signal high. The purpose of the signal is to allow simulation at full speed without the need to simulate millions of clock cycles to arrive at a change in the system's output.

Display driver

The display driver section in Schematic 3 drives the four bit binary value generated by the counter out to the display device on the demo board. The device is a seven segment numerical display that will show the counter value from 0 to F. The value is decoded using the Motorola macro library element "HEX4-7S". The outputs are inverted for the common anode display. Notice that the inverters are drawn with dashed outlines, this is because inversion is a virtual function in the Motorola FPGA, occurring at the location where the signal terminates. After the inverters, the signals are driven off of the device by I/O cells of type "OPBUF". These are generic output drives from the Motorola I/O library. While the signals are fully defined to the output of the device at this point, they are still propagated up the hierarchy to be shown as outputs on the top schematic page.

Verilog code

The counter circuit that generates the signals used as the display information source as well as the frequency and mode controls are implemented in a Verilog source file named "control.v", shown in [Listing 1].

Modules

The following code modules are used to create the control circuit:

The "control" module is the top level module that passes the input enable pulses through the "speed" module to the "count" module. The input signals are routed to their appropriate modules, the modules are connected together and output is retrieved at this level. The "control" module is the top level module that accepts all input from and pass all output to the top page schematic. It is in this module that the input buffers for the speed selection switches are instantiated. The IPBUF

4



functions specify the same input buffers as for the {RADIX} input signal at the top level schematic. This is the mechanism that would be used for all I/O if the top level of the design was synthesized and contained all I/O specifications.

The "speed" module accepts the input clock, 3.8 Hz enable pulses, and the speed selection switch inputs. The input signal is divided by one, two, three, or four as specified by the switches and a 500 ns wide enable pulse is passed back as the output. This circuit could be constructed with no means of initialization, causing the initial state to be indeterminate. However, after the first cycle through the divider (four pulses maximum) the circuit would stabilize. This would pose a problem with the simulator, which propagates and retains "unknown" states.

The "count" module accepts the output of the "speed" module and the {RADIX} input. It produces the four bit vector "count" that is passed out to the top level schematic as {RESULT_0} through {RESULT_3}.

Compiling to EDIF

The control.v file is compiled to the control.edf file using the Exemplar Galileo compiler. The text shown in [Listing 2] contains the results of the Galileo run, as well as all compiler options as they were set for this file.

At this point the Verilog code has been reduced to a flat EDIF file. The EDIF file is a basic text file specifying each instance of the library primitives for the FPGA and their interconnections. This can be accepted as input to most front end tools as a library module, which is what we will do in our next step.

Instantiating Verilog module

Like most front end tools, Workview Office uses an internally defined file structure to maintain the schematics for a particular project. Each module, schematic element, or schematic (excluding device library components) is comprised of one or more of the following files: WIR, SYM, and SCH.

The WIR file is the internal netlist for a particular circuit. It can be considered the internal representation of an EDIF file. The WIR files are hierarchical and track the schematic structure. When a [Save & Check] is performed on a schematic, a WIR file is generated. This file contains a list of all of the modules on the schematic page and their interconnections. It doesn't contain the internal connections within a module. The WIR file for each module contains its particular internal connections. At the lowest level, all of the modules referenced in a WIR file are library elements. To use the result of our Galileo compile, we will need to convert the compiled EDIF file into a WIR file using a Viewlogic utility.

The SYM file is the symbol that is used as a "handle" to manipulate a module and place it into a schematic. A SYM file must exist for the EDIF file that we plan to place into our schematic. It is through this file that the schematic editor knows how to draw the visual image of the module on the screen and what ports ("pins") are available for the connection of nets and busses. The SYM file can be hand drawn or it can be generated directly from EDIF file by a Viewlogic utility, Viewgen, depending on the users Viewlogic license restrictions. If it is generated by Viewgen, it can be edited to

change its appearance. In this application note we will generate the symbol manually using the schematic editor.

The SCH file is the schematic file. It is from this file that the WIR file is generated. While a WIR file contains a list of interconnections, the SCH file also contains the information to graphically draw the schematic on the screen. This information allows the schematic editor to remember where the components are placed, as well as the visual routing of the nets and busses as edited by the designer. An SCH file of a particular module isn't necessary to place that module into the final schematic. In this example, no SCH file will exist for the module created by the Verilog compile. As a result, we will be unable to "push" down into the internals of our control module from the schematic editor. Again, depending on the users Viewlogic license restrictions, the Viewgen utility can create an SCH file from an EDIF file. If this is done, the user can then push down and view the schematic as generated by the Verilog compiler. This feature can be very helpful when learning synthesis, as the user can see exactly what circuit is generated from the source code. The only shortcoming with this feature is that the schematic generated is composed entirely of primitive gates (AND, NAND, OR, NOR, XOR, XNOR, INV, etc.) connected in a visually arbitrary maze. For small circuits this isn't a problem, for large or complex circuits it basically appears as a sea of gates connected by uncountable (and untraceable) nets.

It should be noted at this point that all of the files for a given module have the same name, and are differentiated only by their location in the project file structure. For example: the schematic, wire, and symbol files for the display driver module are all named "display.1". (The ".1" indicates page number one of that module. All of the modules in this design, and in most hierarchical designs have only one page per module.) If these files are moved to the same directory, they will overwrite each other. If they are moved around and placed back into the wrong subdirectories, errors will result when the files are accessed by any of the Viewlogic tools.

Importing EDIF

In order to use our Verilog generated control module we must have a WIR and an SYM file for the module. The EDIF import utility will create the WIR file for us. We will then create the SYM file using the schematic editor. The Viewgen symbol and schematic generating utilities, if used, obtain their information from the WIR file. Therefore the WIR file generation is the first and necessary step in using our control module.

In this application note, it is assumed that the user is using the GUI interface for all tools and utilities. These tools also have a command line interface for batch processing that is covered in the documentation for each tool.

To begin, the user starts the "Edif interfaces" program from the Viewlogic tool suite. (Note: If your display resolution is less than 800x600, the entire "Edif interfaces" dialog box may not fit on your screen.) The "EDIF Netlist Reader" tab is selected in the dialog box. The "Browse" button is used to locate the input EDIF file. This process will bring to light an issue with the naming of EDIF files. Viewlogic is looking for EDIF files to have the extension of ".edn", whereas Galileo generates EDIF files with the extension of ".edf". This means the file "control.edf"

4



won't appear in the input file window when the proper directory is located. The file type option must be changed to "*" to view the file, which also fills the window with all files in the directory. An alternative is to just type the full file and path name into the main dialog box without browsing. When using the command line version of the tools in a batch file, the user specifies the correct extension in the input file specification.

When selecting the output directory, the user **MUST** specify the WIR subdirectory in the Viewlogic project directory structure. If not, then the user must move or copy the file into this directory for it to be usable by Viewlogic. It should be noted that the EDIF netlist reader does **NOT** point to this directory by default, but instead starts out in the main project directory, one level above the WIR subdirectory.

The "Apply" button is pressed and the tool performs its magic, providing status in the dialog box window.

Creating the Symbol

At this point we have a WIR file for our Verilog module so Viewlogic can now use the information to generate an overall netlist. However, we need to create a SYM file so that we have a symbol containing pins for the connection of nets on our schematic page.

The symbol is created using the Viewdraw schematic editor. "New" is selected from the "File" menu. When the dialog box appears the "Symbol" option is selected (as opposed to the default "Schematic" option). A name is specified, in this case it is "control.1". The name of the symbol **MUST** be exactly the same as the name of the WIR file.

A symbol is created from lines, rectangles, arcs, etc. All of these objects are used to create the visual appearance of the symbol on the schematic. The only functional elements on the symbol are the pins (placed using, you guessed it, the pin tool from the tool bar). There must be one pin for each I/O connection in the parameter list of the topmost Verilog module. The names of the pins **MUST** exactly match the names of the top module's parameters, the only exception is that case is ignored. When vectors are specified, as in the "result" parameter in our control.v file, Galileo expands them to a series of scalar connections. Each of these scalar values has the name of the vector followed by an underscore and the bit position. In our example, the vector "result" that was defined as {{0:3}result} will expand to {RESULT_0} through {RESULT_3}. Four pins were placed on the "control" symbol with these names. The file is then saved. Viewdraw will create a text window defining the properties of the symbol. This text window and the editor window can be closed, assuming there are no errors at this point.

Instantiating the Verilog module into the schematic

Our control module, as defined by the Verilog code, can now be manipulated the same as any of the other schematically drawn modules. The only exception to manipulating this module is that the internal logic can't be viewed by pushing down to the schematic level.

To place the module in our design, the user selects the option of adding a component and chooses "control.1" from the list of project modules. In our case, there is one instantiation of this module. In other cases, such modules can

be placed as many times as necessary into as many different schematic pages as the user wants. It should be noted that this results in a complete duplication of this module into the FPGA for each instantiation.

The top schematic page is then saved using the "Save & check" option. The resulting WIR file of the topmost schematic now contains a reference to the "control" WIR file and it's connections on that schematic page. Connections within the "control" module are not in the topmost schematic file, but are specified in the control module's WIR file.

Generating the final net list

The final netlist used to present the design to the MPADS tool for place and route into the Motorola FPGA must be generated using the Viewlogic "Edif interfaces" utility. At this stage the entire design is spread across a collection of files under the control of the Viewlogic project manager. Just as the Verilog code was distilled down to an EDIF file for processing by Viewlogic, the entire design must now be reduced to a single EDIF file for import into the Motorola back-end tool.

Again, the Viewlogic "Edif interfaces" utility is started. This time the "EDIF Netlist Writer" tab is selected. Use the "browse" button to locate the WIR file for the topmost schematic. In this example the top schematic file is "top.1". (Curiously enough, when browsing for a WIR file for EDIF output, the tool does default to the WIR subdirectory.) The output file will default to the input filename with the ".edn" extension. (The Motorola place and route tool will accept EDIF files with either ".edn" or ".edf" file extensions.) In our case the output file defaults to "top.edn". The resulting EDIF file is placed one level above the WIR subdirectory, into the main project directory. The fact that the input and output files reside in different directories isn't obvious from the dialog box.

The contents of the "Author" field will be placed into the EDIF file as a comment, and is optional. The "Level" field **MUST** be specified as "micro". This tells the tool to specify only the micro gate level library in the resulting EDIF file. With this option set to micro, the final EDIF file will not contain a reference to the "HEX4-7S" used to drive the seven segment display, but will instead specify the primitive gates that make up this library element.

Leave "Inhibit Library Alias" unchecked. Allow the "Configuration files" to default. The "Netlist options" must be set to "Flatten without evaluation attributes". This last option causes the netlist tool to recursively descend the hierarchy whenever a module is encountered until the primitive gate level is reached. The final EDIF netlist will not contain any hierarchy, but will instead be composed entirely of basic gates and their interconnections. Pressing the "Apply" button generates the final netlist.

MPADS interface

The EDIF file (top.edn) that we have created contains all of the design information except for the FPGA pin number specification for the I/O signals. There are three methods of resolving this issue:

First, we could allow the back-end tool to select our pins for us. This is the easiest option. To download this design into the Motorola demo board, the signals must match the LED and switch connections already designated on that board. Even if

4



this is acceptable in an actual engineering project, it is only acceptable during the initial design. Thereafter all design modifications must track the original pin specifications.

The second option is to specify the pin numbers as attributes to the I/O signals in the front-end tool. This is probably the most common method in schematic capture environments.

The third, and the one used in this application note is to use an external pattern file that is imported by the Motorola back-end tool.

All three of these methods can be used on a particular design. Unspecified pins are numbered automatically during autoroute by MPADS. If a pin number is specified in the front-end tool and in a pattern file, the pattern file takes precedence.

Pattern file

The pattern file used in this design is shown in [Listing 3]. If a pattern file is to be used, it **MUST** have the same name as the input EDIF file with the extension ".pat". In addition, it **MUST** also reside in the same directory as the input EDIF file. If either of these conditions is not met, then the MPADS tool will assume that there is no pattern file and will quietly perform pin selection on its own. It is therefore important that the designer check the port report file that is output by MPADS to confirm that the required pin placement information was processed.

Import results

[Listing 4] shows the results of the MPADS import of this design. Just as Viewlogic had to read the EDIF file generated

by Galileo into a WIR file for its own use, MPADS has to read in the final EDIF file into a ".net" file for its use. In addition, the import process checks the specified EDIF file for errors or design inconsistencies that would prevent the autorouter from processing the netlist.

Autoroute results

The results of the autoroute are shown in [Listing 5]. At this point we can check the port report file ("prp") shown in [Listing 6] to confirm proper pin placement.

Updating the design

The above process may be repeated many times in the normal engineering design cycle. The opportunities to introduce mistakes, such as recompiling the Verilog code but forgetting to import the new EDIF file before netlisting the schematic can waste time and energy. The most productive method is use a batch file that runs the proper tools in sequence, specifying the proper tool options and filenames

Conclusion

Dealing with the myriad number of front-end tools can be an advantage if, by integrating them, you make use of their individual strengths. The trick to doing this effectively is to understand what each tool requires in order to pass information to and from other tools. In most cases this isn't a very difficult procedure, but simply one in which there is little direct documentation. It is usually possible to extrapolate the information provided for the integration of front-end and back-end tools to the task of multiple tool sets.

4



[Listing 1]

```

// This file is the control circuit that interprets the dip switches
// on the demo board and produces the results for the display circuit
//
// Engineer      Rich Rejmaniak / Phil Rauba
// Company       Motorola SPS
// Date          01/06/96
// Revised       03/27/97 Rauba
// Revised       04/03/97 Rejmaniak
//-----
module control (clk, enable, radix, reset_l, switch_0, switch_1, result);
    input  clk;                // Main system clock
    input  enable;            // Main system clock enable
    input  radix;             // Select count mode to hex or BCD
    input  reset_l;          // Initialization signal (active low)
    input  switch_0;         // Clock divider control switches
    input  switch_1;
    output [3:0]result;      // Final four bit result of the count
    // Signal 'tick' connects the output of the prescale module to the
    // input enable of the counting module
    wire  tick;
    wire  [1:0]scale;
    wire  reset;              // Reset is active high internally
    assign reset = ~reset_l;
    // The inputs that specify the scaling divisor for the 'speed'
    // module have their I/O buffers specified here. The 'switch'
    // variable connected to the .extin ports will appear as I/O
    // to the final compiled module, just as the parameters specified
    // in the list to this module.
    ipbuf i0 (.extin(switch_0), .i(scale[0]));
    ipbuf i1 (.extin(switch_1), .i(scale[1]));
    // Divide the incoming clock enable frequency by one, two, three, or four
    speed u0 (clk, enable, reset, scale, tick);
    // One count for every 'tick' enable clock pulse
    count ul (clk, tick, radix, reset, result);
endmodule
//-----
// This is an isochronous module that will advance the output 'count'
// on every enables clock pulse
module count (clk, enable, radix, reset, count);
    // 'radix' determines if the counter will terminate at '9' or 'f'
    input  clk, enable, radix, reset;
    output [3:0]count;
    reg   [3:0]count;
    always @ (posedge clk or posedge reset)
    begin
        if (reset)
            begin
                count <= 0;
            end
            // Reset condition
        else
            begin
                // non-reset or potential count event
                // It's only a count event if enable is true (isochronous vs.
synchronous)

                count <= enable ? (count + 1) : count;
                // When in BCD mode.....
                // Reset to zero on the next enabled clock pulse after '9'
                if ((count == 9) && (radix == 0) && enable)
                    begin
                        count <= 0;
                    end
            end
        end
    end
endmodule

```

4



```

                end
            end // non-reset condition
        end // always block
endmodule
//-----
// Divide the clock enable frequency by 1, 2, 3, or 4
// as specified by the external 'switch' inputs
module speed (clk, enable_in, reset, scale, enable_out);
    input  clk, enable_in, reset;

    // The scaling signals entering through I/O buffers specified in the
    // top module 'control'
    input  [0:1]scale;
    output enable_out;
    reg    enable_out;
    reg    [0:1]divide;
    always @ (posedge clk or posedge reset)
    begin
        if (reset)
            begin // reset condition
                divide <= 0;
                enable_out <= 0;
            end // reset condition
        else
            begin // non-reset condition

                if (enable_in) // only respond to enabled clock pulses
                    begin
                        if (divide == scale)
                            begin // Pass this enable out of the module
                                enable_out <= 1;
                                divide <= 0; // Restart dividing counter
                            end
                        else // Count and block this enable
                            begin
                                enable_out <= 0;

                                // Block this enable
                                divide <= divide + 1;

                                // But count it toward the enable count
                                end
                            end
                        else // input clock pulse isn't enabled to begin with
                            begin
                                enable_out <= 0; // non-enabled clock pulse
                            end
                        end // non-reset condition
                    end // always block
            end
        end
endmodule
//-----
module ipbuf (extin, i); // MPA library element
    input extin;
    output i;
endmodule

```

4



[Listing 2]

```
Galileo - V3.2.5
Copyright 1990-1994, Exemplar Logic, Inc. All rights reserved.
gcw -COM=D:\TEMP\xmplr2 D:\Projects\DemoBrd\Verilog\control.v
D:\Projects\DemoBrd\Verilog\control.edf

      gcw Options:
Option      Value
Input       =      D:\Projects\DemoBrd\Verilog\control.v
Output      =      D:\Projects\DemoBrd\Verilog\control.edf
pass        =      1
parallel_case
encoding    =      ONEHOT
modgen_library =      generic
report      =      DEVICE_UTIL
tristate_map
transformation
wire_tree   =      WORST
macro
effort      =      Standard
area
output_format =      EDIF
input_format =      Verilog
target      =      P_mpa
nocontrol
cwd         =      D:\EXEMPLAR\DEMO
com         =      D:\TEMP\xmplr2
```

The license on this copy of galileo will expire in 177 days.
 If this is a newly licensed copy of galileo, please be sure to
 install the long-term license before your temporary license expires.
 If this is a permanent license, your maintenance contract has expired.
 Please contact (510)337-3703 or license@exemplar.com to renew maintenance.

```
Reading library file 'D:\EXEMPLAR\lib\gcprim23.syn'...
-- Reading file 'D:\Projects\DemoBrd\Verilog\control.v'...
-- Loading module ipbuf
-- Loading module speed
-- Loading module count
-- Loading module control
-- Compiling root module 'control'
Warning, module ipbuf is empty.
-- Compiling module speed
-- Compiling module count
-- Verilog source successfully analyzed.
-- Info, performing high-level optimization...
-- Info, saving design to database..
Warning, Unknown gate 'ipbuf' (instance i0) found. Treating as 'noopt'.
Warning, Unknown gate 'ipbuf' (instance i1) found. Treating as 'noopt'.
Info, inferring module generator MODGEN_EQ(SIZE=>2,SIGNED=>FALSE) for instance u0/modgen_0
Info, inferring module generator MODGEN_INC(SIZE=>2,SIGNED=>FALSE) for instance u0/modgen_1
Info, inferring module generator MODGEN_INC(SIZE=>5,SIGNED=>FALSE) for instance u1/modgen_2
-- Starting module generation
-- Reading module generator description from file D:\EXEMPLAR\data\modgen\generic.vhd
-- Modgen File generic.vhd Version 3.7
Reading library file 'D:\EXEMPLAR\lib\p_mpa23.syn'...
-- Resolving module generator MODGEN_INC(SIZE=>5,SIGNED=>FALSE) from file generic.vhd
-- Resolving module generator MODGEN_INC(SIZE=>2,SIGNED=>FALSE) from file generic.vhd
-- Resolving module generator MODGEN_EQ(SIZE=>2,SIGNED=>FALSE) from file generic.vhd
Library version = 2.10
Delays assume: Temp= 25.0 C Voltage=5.00 V Process=typical
Exemplar Logic's Galileo Fri Apr 18 17:47:25 1997
```

4



AN1613

```
Pass          Area          Delay --CPU--
              (Gates)         (ns) min:sec
1             29             13.2  00:01
              Resource Use Estimate
Design: control
Technology: mpa
File: D:\Projects\DemoBrd\Verilog\control.v
Area: 29.0
Critical Path: 13.2 ns
gcw run complete.
```

[Listing 3]

```
# I/O Pin pattern specification file
# Rich Rejmaniak, Motorola Semiconductor
# 04/18/97
# Jumper specifications for clock divider
# Jumper positions J7-8 (USER0 = switch0) &
# J7-6 (USER1 = switch1) on the demo board
port switch0      attribute pad_place 16
port switch1      attribute pad_place 15
# Jumper controlling BCD vs. binary count
# Jumper position J7-4 (USER2 = radix)
port radix         attribute pad_place 14
# Jumper controlling debug vs. normal operation
# Jumper position J7-2 (USER3 = debug)
port radix         attribute pad_place 13
# Clock input pin uses the onboard 2MHz osc.
port clk attribute pad_place 84
# Seven segment display
port a             attribute pad_place 70
port b             attribute pad_place 69
port c             attribute pad_place 68
port d             attribute pad_place 73
port e             attribute pad_place 72
port f             attribute pad_place 71
port g             attribute pad_place 66
^Z
```

4

[Listing 4]

```
MPA Design System - Starting Import run on Fri May 02 12:09:55 1997
Logfile e:\Projects\DemoBrd\Top\board.log
Reading Device Architecture 'g:\mpads\SYSTEM\mpa1016.dev'
Reading Libraries....
Reading Library 'e:\Projects\DemoBrd'
Reading Library 'g:\mpads\LIBS\MPALIB\MPA1016'
Reading Library 'g:\mpads\LIBS\MPALIB\IOLIB'
Reading Library 'g:\mpads\LIBS\MPALIB\MICROLIB'
Done reading libraries
Reading EDIF netlist 'e:\Projects\DemoBrd\Top.edn'
Written at 19:24:0 9/2/1997 by VIEWlogic's edifnet Version 5.00
Creating temporary definition inv...
...definition inv is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition dfer...
...definition dfer is already in library microlib
Written at 16:6:30 9/2/1996 by
```



```

Creating temporary definition an2...
  ..definition an2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition zero...
  ..definition zero is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition one...
  ..definition one is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition opbuf...
  ..definition opbuf is already in library iolib
Written at 16:6:36 9/2/1996 by
Creating temporary definition nr2...
  ..definition nr2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition or2...
  ..definition or2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition xr2...
  ..definition xr2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition an2...
  ..definition an2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition nd2...
  ..definition nd2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition xn2...
  ..definition xn2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition or2...
  ..definition or2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition nr2...
  ..definition nr2 is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition inv...
  ..definition inv is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition ipbuf...
  ..definition ipbuf is already in library iolib
Written at 16:6:36 9/2/1996 by
Creating temporary definition dfe...
  ..definition dfe is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition df...
  ..definition df is already in library microlib
Written at 16:6:30 9/2/1996 by
Creating temporary definition ipbuf...
  ..definition ipbuf is already in library iolib
Written at 16:6:36 9/2/1996 by
Creating temporary definition ipclk...
  ..definition ipclk is already in library iolib
Written at 16:6:36 9/2/1996 by
Creating      new definition top...
Written at 15:12:58 9/2/1997 by
Design root at cell top
EDIF 200 Level 0 File top(top)
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\inv\inv.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\dfer\dfer.net'

```



```

Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\an2\an2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\zero\zero.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\one\one.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\IOLIB\opbuf\opbuf.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\nr2\nr2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\or2\or2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\xr2\xr2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\nd2\nd2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\xn2\xn2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\IOLIB\ipbuf\ipbuf.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\dfe\dfe.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\df\df.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\IOLIB\ipclk\ipclk.net'
Flattening definition top...
Retargeter : Reading attribute file e:\Projects\DemoBrd\Top.pat
Retargeting definition top
Reading Binary Retargeter Rules File 'g:\mpads\SYSTEM\mpal000.rrf'
Netlist Statistics before Retargeting:
  13 instances of nr2
   1 instances of one
  14 instances of or2
   2 instances of xn2
   4 instances of xr2
   1 instances of zero
  35 instances of inv
  51 instances of an2
   5 instances of df
   2 instances of dfe
  19 instances of dfer
   3 instances of ipbuf
   1 instances of ipclk
   7 instances of opbuf
   3 instances of nd2
Total of 161 instances
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\dfr\dfr.net'
  37 instances removed
  36 nets removed
  20 instances added
  19 nets added
Netlist Statistics after Retargeting:
  13 instances of nr2
  14 instances of or2
   2 instances of xn2
   4 instances of xr2
  19 instances of zero
  50 instances of an2
   5 instances of df
   2 instances of dfe
  18 instances of dfer
   1 instances of dfr
   3 instances of ipbuf
   1 instances of ipclk
   7 instances of opbuf
   3 instances of nd2
Total of 142 instances
Retargeter took 4s
Flattening definition top...
Splitting wired-or nets...
Checking netlist...
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\dfer\ed.lyt'

```

4



```

Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\an2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\IOLIB\opbuf\out.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\nr2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\or2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\xr2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\nd2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\xn2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\IOLIB\ipbuf\in.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\dfe\ed.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\df\d.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\IOLIB\ipclk\in.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\dfr\d.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\zero\a.lyt'
Writing Netlist 'e:\Projects\DemoBrd\Top\Top.net'
Import took 7s
MPA Design System - End of Import run on Fri May 02 12:10:03 1997

```

[Listing 5]

```

MPA Design System - Starting Autolayout run on Fri May 02 12:10:31 1997
Logfile e:\Projects\DemoBrd\Top\board.log
Reading Device Architecture 'g:\mpads\SYSTEM\mpa1016.dev'
Reading Libraries...
Reading Library 'e:\Projects\DemoBrd'
Reading Library 'g:\mpads\LIBS\MPALIB\MPA1016'
Reading Library 'g:\mpads\LIBS\MPALIB\IOLIB'
Reading Library 'g:\mpads\LIBS\MPALIB\MICROLIB'
Done reading libraries
Reading Netlist 'e:\Projects\DemoBrd\top\top.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\IOLIB\ipbuf\ipbuf.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\IOLIB\ipclk\ipclk.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\dfr\dfr.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\dfer\dfer.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\an2\an2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\or2\or2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\nr2\nr2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\IOLIB\opbuf\opbuf.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\nd2\nd2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\xn2\xn2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\xr2\xr2.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\dfe\dfe.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\df\df.net'
Reading Netlist 'g:\mpads\LIBS\MPALIB\MICROLIB\zero\zero.net'
Reading Package File 'g:\mpads\SYSTEM\m2plcc84.pkg'
MPA1016 84 - PLCC package selected
Package mode 'Boot_From_ROM_Mode' selected
Reading Layout 'g:\mpads\LIBS\MPALIB\IOLIB\ipbuf\in.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\IOLIB\ipclk\in.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\dfr\d.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\dfer\ed.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\an2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\or2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\nr2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\IOLIB\opbuf\out.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\nd2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\xn2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\xr2\ab.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\dfe\ed.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\df\d.lyt'
Reading Layout 'g:\mpads\LIBS\MPALIB\MICROLIB\zero\a.lyt'

```

4



AN1613

Context arguments

Min Zone Delay 0.5ns
Back Off 30
Fanout 2

Context arguments done

Autolayout arguments

Start Temp 10
Effort 30
Attempts 1
Utilisation 80
Delay Cost 0.05

Autolayout arguments done

Clocks, Resets and Wired-or Nets:

Net \$1n15 is a primary clock driven by instance \$1i4
Net has 27 clock ports
Net ntop_1 is a tertiary reset driven by instance top_1(zero)
Net has 1 reset port, 1 normal port
Net ntop_2 is a tertiary reset driven by instance top_3(zero)
Net has 1 reset port, 1 normal port
Net ntop_3 is a tertiary reset driven by instance top_5(zero)
Net has 1 reset port, 1 normal port
Net ntop_4 is a tertiary reset driven by instance top_7(zero)
Net has 1 reset port, 1 normal port
Net ntop_5 is a tertiary reset driven by instance top_9(zero)
Net has 1 reset port, 1 normal port
Net ntop_6 is a tertiary reset driven by instance top_11(zero)
Net has 1 reset port, 1 normal port
Net ntop_7 is a tertiary reset driven by instance top_13(zero)
Net has 1 reset port, 1 normal port
Net ntop_8 is a tertiary reset driven by instance top_15(zero)
Net has 1 reset port, 1 normal port
Net ntop_9 is a tertiary reset driven by instance top_17(zero)
Net has 1 reset port, 1 normal port
Net ntop_10 is a tertiary reset driven by instance top_19(zero)
Net has 1 reset port, 1 normal port
Net ntop_11 is a tertiary reset driven by instance top_21(zero)
Net has 1 reset port, 1 normal port
Net ntop_12 is a tertiary reset driven by instance top_23(zero)
Net has 1 reset port, 1 normal port
Net ntop_13 is a tertiary reset driven by instance top_25(zero)
Net has 1 reset port, 1 normal port
Net ntop_14 is a tertiary reset driven by instance top_27(zero)
Net has 1 reset port, 1 normal port
Net ntop_15 is a tertiary reset driven by instance top_29(zero)
Net has 1 reset port, 1 normal port
Net ntop_16 is a tertiary reset driven by instance top_31(zero)
Net has 1 reset port, 1 normal port
Net ntop_17 is a tertiary reset driven by instance top_33(zero)
Net has 1 reset port, 1 normal port
Net ntop_18 is a tertiary reset driven by instance top_35(zero)
Net has 1 reset port, 1 normal port
Net ntop_19 is a tertiary reset driven by instance top_37(zero)
Net has 1 reset port, 1 normal port

7 primary clock/reset pads free (consider using these for high fanout nets)

Context summary

11 I/O pads used (80 available on device)
11 I/O pins used (60 available on package)

End of Context summary

131 core cells used by design

This is approximately 16.38203f the maximum device capacity

4



and corresponds to an autolayout Utilisation parameter of 17.

Clock Information at end of initialisation:

Name	Frequency	period	phase
clk	2.0MHz	500.0ns	0.0ns

Clustering took 1s - 48 clusters created

Writing Clustered context file 'e:\Projects\DemoBrd\top\board.cxt'

Partitioning 48 top level objects (total 48 objects)

Partitioning took 7s

Writing Partitioned context file 'e:\Projects\DemoBrd\top\board.cxt'

Global Routing 52 segments

Global Routing took 2s

Writing Global Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone Routing 8 zones

Routing zone (4,4) (1 of 8): 33 instances, 14 zone ports, 74 segments of 39 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 6s

Routing zone (4,3) (2 of 8): 32 instances, 14 zone ports, 72 segments of 38 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 8s

Routing zone (3,2) (3 of 8): 15 instances, 13 zone ports, 36 segments of 17 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 1s

Routing zone (2,1) (4 of 8): 14 instances, 12 zone ports, 33 segments of 15 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 0s

Routing zone (3,3) (5 of 8): 12 instances, 11 zone ports, 29 segments of 14 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 0s

Routing zone (3,4) (6 of 8): 13 instances, 9 zone ports, 31 segments of 18 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 1s

Routing zone (2,2) (7 of 8): 6 instances, 7 zone ports, 15 segments of 8 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 0s

Routing zone (2,4) (8 of 8): 6 instances, 7 zone ports, 15 segments of 8 nets

Writing Zone Routed context file 'e:\Projects\DemoBrd\top\board.cxt'

Zone processing took 0s

Zone Routing took 18s - 8 zones processed (0 failures)

Clock Information at end of Zone Routing:

Name	Frequency	period	phase
clk	14.9MHz	67.1ns	0.0ns

All nets are complete

Writing Timing File 'e:\Projects\DemoBrd\top\board.tim'

Writing layout file 'e:\Projects\DemoBrd\top\board.lyt'

Writing back annotation file to 'e:\Projects\DemoBrd\top\board.dtb'

New usage 'ntop_18' clashes with existing use 'ntop_3'

Writing port report file 'e:\Projects\DemoBrd\top\board.prp'

MPA Design System - End of Autolayout run on Fri May 02 12:11:08 1997

4

[Listing 6]

```
# I/O Pin report file
# This file is suitable for use as an External Attributes (.pat) file
# Definition: top
# Layout: board
# Port      switch0, Net      switch0, Position ( 0, 43), Package pin 16
port switch0 attribute pad_place 16
# Port      switch1, Net      switch1, Position ( 0, 45), Package pin 15
port switch1 attribute pad_place 15
# Port      radix, Net       radix, Position ( 0, 47), Package pin 14
```



AN1613

port radix attribute pad_place 14	
# Port clk, Net	clk, Position (29, 54), Package pin 84
port clk attribute pad_place 84	
# Port g, Net	g, Position (54, 28), Package pin 66
port g attribute pad_place 66	
# Port c, Net	c, Position (54, 36), Package pin 68
port c attribute pad_place 68	
# Port b, Net	b, Position (54, 40), Package pin 69
port b attribute pad_place 69	
# Port a, Net	a, Position (54, 42), Package pin 70
port a attribute pad_place 70	
# Port f, Net	f, Position (54, 44), Package pin 71
port f attribute pad_place 71	
# Port e, Net	e, Position (54, 46), Package pin 72
port e attribute pad_place 72	
# Port d, Net	d, Position (54, 48), Package pin 73
port d attribute pad_place 73	

4



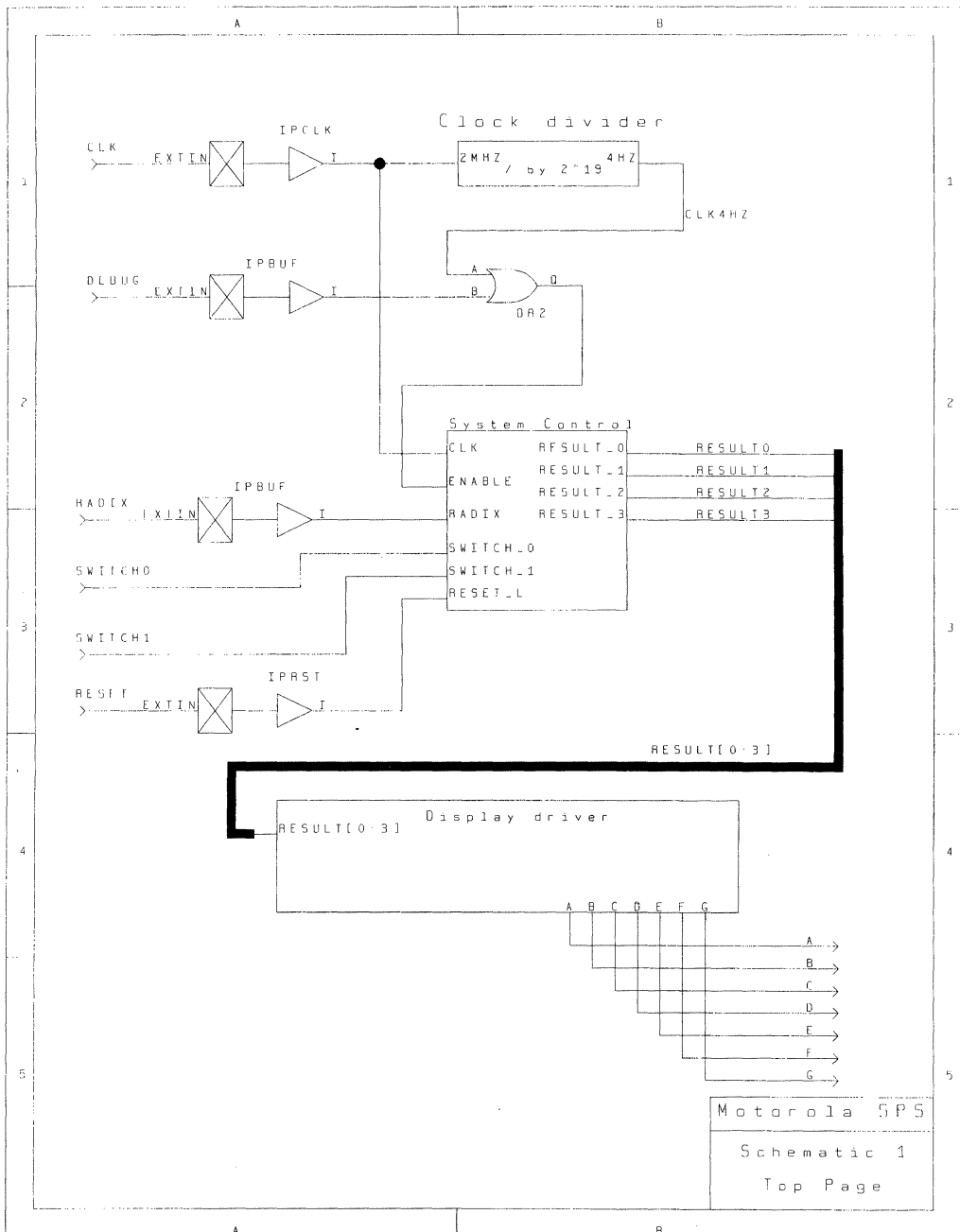


Figure 1. Top Page



4

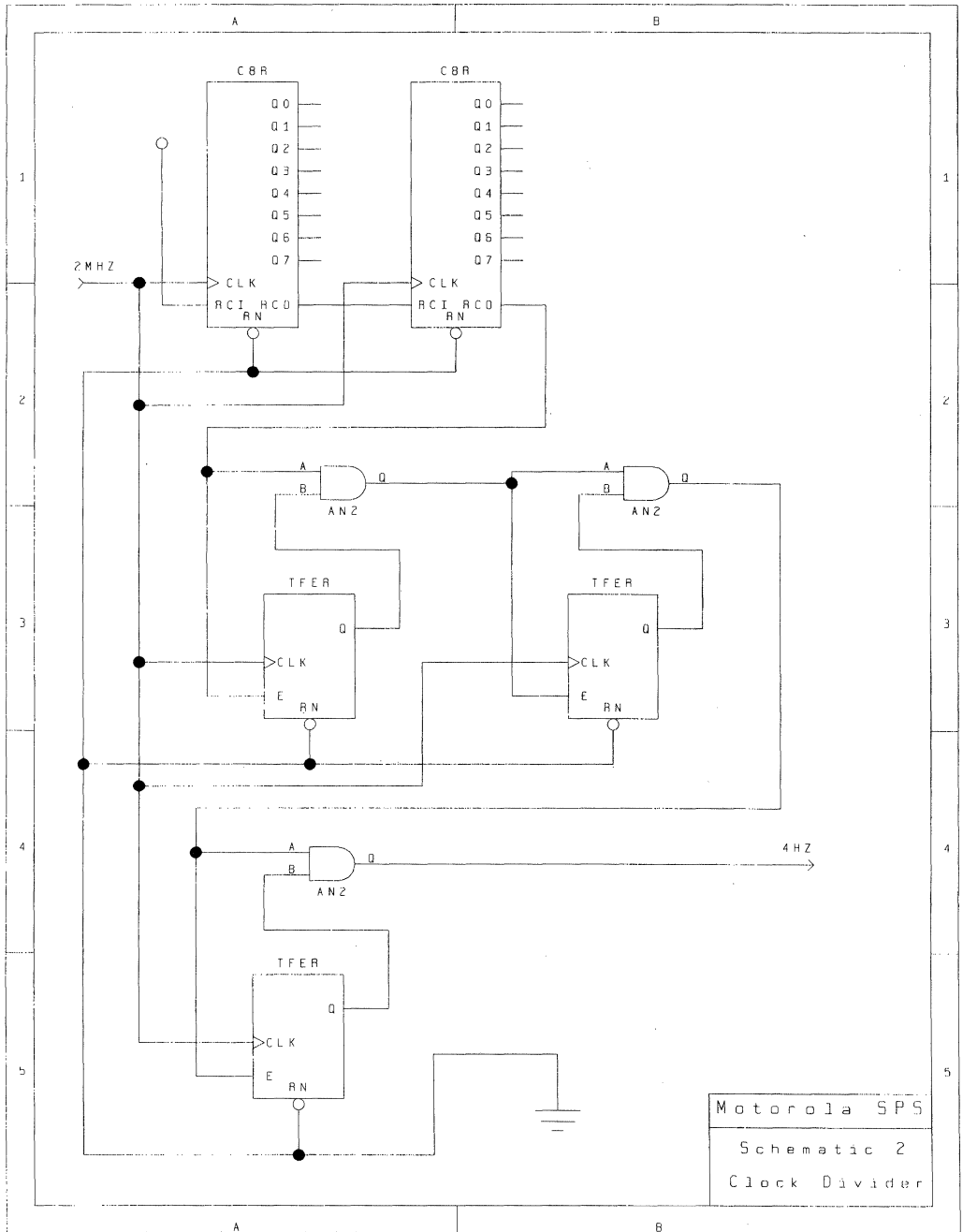


Figure 2. Clock Divider



4

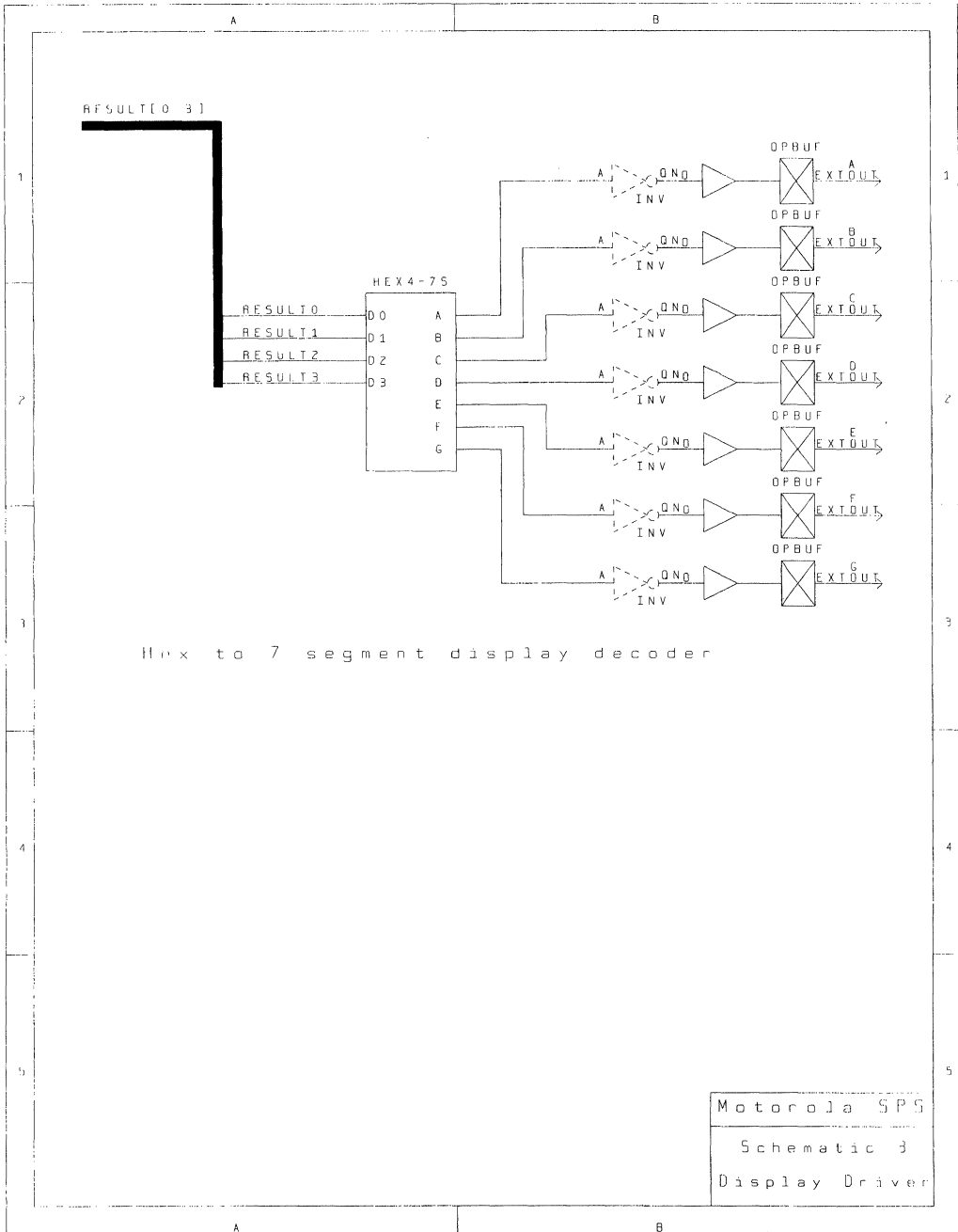


Figure 3. Display Driver



Optimizing VHDL/Verilog Designs for Speed for the MPA Family Using Exemplar Galileo Synthesis

Prepared by
Claudia Colombini
Motorola Field Applications Engineer

4



Optimizing VHDL/Verilog Design for Speed for Motorola MPA1000 Family Using Exemplar Galileo Synthesis

This Application Note is intended to show an easy way of producing good synthesis results in terms of speed for MPA1000 family when using Exemplar Galileo for synthesis.

1. Galileo Options setting and results:

During synthesis the Galileo default module generation libraries (modgen) will be used. To select the kind of modgen library that will be used for the design select the Input Options from the Galileo Logic Explorer menu. For MPA1000 the Module generation library should be set to default.

Select the kind of modgen library. There are 5 possibilities:

- Auto
- Smallest
- Small
- Fast
- Fastest

On this small example of a 16-bit accumulator the different results for these 5 possibilities are shown. These results will help you make the right switches on Galileo Synthesis to produce the required results for your design.

```
-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
entity accu16 is
    port ( clk,rst :in std_logic;
          d :      in std_logic_vector(15 downto 0);
          accu :   out std_logic_vector(15 downto 0)
        );
end accu16;
architecture arch of accu16 is
    signal accu_int : std_logic_vector(15 downto 0);
begin
process (clk, rst)
begin
    if rst = '0' then
        accu_int <= ( others => '0');
    elsif clk'event and clk = '1' then
        accu_int <= accu_int + d;
    end if;
end process;
    accu <= accu_int;
end arch;
-----
```

The best results will always be obtained when the Effort is set to exhaustive. This will cause the tool to run longer especially on PC but the results will be the best. For delay optimization the Optimize option is set to Delay. For synthesis options: Mode is set to Chip – this will automatically insert IO-Buffers. None of the 11 paths is selected, this will make the synthesis run through all 11 paths and then the tool will come up with different results from which the best speed can be selected. All other settings are set to default. See the settings for all runs, the option changing will be select_modgen.

```
-----
gc /home/claudia/exemplar/demo/accu16.vhd /home/claudia/exemplar/demo/accu16.edf -nocontrol
-input_format=VHDL -target=p_mpa -output_format=EDIF -delay -effort=Exhaustive -chip
-wire_tree=Worst -report=slack_table -report=cell_usage -report=device_util -encoding=OneHot
-VHDL_93 -modgen_library=generic -select_modgen=Smallest -status_pipe=8
gc Options:
Option      Value
Input       = /home/claudia/exemplar/demo/accu16.vhd
Output      = /home/claudia/exemplar/demo/accu16.edf
status_pipe =      8
select_modgen =
modgen_library =      generic
-----
```

4



```

vhdl_93
encoding =      OneHot
report  =      device_util
wire_tree =      Worst
chip
effort  =      Exhaustive
delay
output_format =      EDIF
target  =      p_mpa
input_format =      VHDL
nocontrol
    
```

1.1. select_modgen is set to smallest:

Here are the 11 paths with the different results, the last 2 paths do another speedup of the fastest paths:

Pass	Area (Gates)	Delay --CPU-- (ns) min:sec
1	150	38.6 00:03
2	150	38.6 01:58
3	150	38.6 00:03
4	150	38.6 00:03
5	150	38.6 00:03
6	150	38.6 00:03
7	150	38.6 00:03
8	150	38.6 00:03
9	168	94.0 00:05
10	252	51.8 00:30
11	255	69.8 00:09
1Spd	151	38.6 00:06
2Spd	151	38.6 00:05

Galileo will automatically choose the fastest implementation and save the edif file, but it is also possible to save other paths if wanted.

```

Resource Use Estimate
Design: ACCU16
Technology: mpa
File: /home/claudia/exemplar/demo/accu16.vhd
Area: 150.0
Critical Path: 38.6 ns
gc run complete.
    
```

4



1.2. select_modgen is set to small:

Here are the 11 paths with the different results.

```
-----
Pass          Area      Delay --CPU--
              (Gates)   (ns)  min:sec
1             150       38.6   00:04
2             150       38.6   02:01
3             150       38.6   00:03
4             150       38.6   00:03
5             150       38.6   00:03
6             150       38.6   00:03
7             150       38.6   00:03
8             150       38.6   00:03
9             168       94.0   00:05
10            252       51.8   00:32
11            255       69.8   00:09
1Spd         151       38.6   00:06
2Spd         151       38.6   00:05
-----
```

For this small example the result is not varying much to the option smallest.
This is the selected result:

```
-----
Resource Use Estimate
Design: ACCU16
Technology: mpa
File: /home/claudia/exemplar/demo/accu16.vhd
Area: 150.0
Critical Path: 38.6 ns
gc run complete.
-----
```

4

1.3. select_modgen is set to fast:

Here are the 11 paths with the different results, the last 2 paths do another speedup of the fastest paths and include the automatically selected result for the option fast:

```
-----
Pass          Area      Delay --CPU--
              (Gates)   (ns)  min:sec
1             162       33.8   00:05
2             188       37.4   00:04
3             192       33.8   00:04
4             186       37.4   00:04
5             162       33.8   00:04
6             188       37.4   00:04
7             192       33.8   00:04
8             186       37.4   00:04
9             185       95.2   00:06
10            159       82.0   00:05
11            177       79.6   00:04
1Spd         162       31.4   00:08
2Spd         162       33.8   00:06
-----
```

```
Resource Use Estimate
Design: ACCU16
Technology: mpa
File: /home/claudia/exemplar/demo/accu16.vhd
```



Area: 162.0
 Critical Path: 31.4 ns

1.4. Modgen_select is set to fastest:

Here are the 11 paths with the different results, the last 2 paths do another speedup of the fastest paths and includes the automatically selected result for the option fastest:

Pass	Area (Gates)	Delay --CPU-- (ns) min:sec
1	168	26.6 00:05
2	194	30.2 00:05
3	198	26.6 00:04
4	192	30.2 00:05
5	168	26.6 00:04
6	194	30.2 00:04
7	198	26.6 00:05
8	192	30.2 00:05
9	191	96.4 00:06
10	165	86.8 00:05
11	182	83.2 00:05
1Spd	168	24.2 00:08
2Spd	168	26.6 00:07

Resource Use Estimate

Design: ACCU16
 Technology: mpa
 File: /home/claudia/exemplar/demo/accu16.vhd
 Area: 168.0
 Critical Path: 24.2 ns

1.5. select_modgen is set to auto:

This option will cause the software to select the modgen library due to the optimization constraints. In this example the optimization is set to delay therefore the Modgen_select is set to fast and the run will produce the same result as for the option fast.

2. MPADS implementation:

The MPA1000 family consists of 4 different arrays. For this example the MPA1016, the smallest device with about 3500 equivalent gates is chosen in the 84 PLCC package with the Boot from ROM programming mode. Before the IMPORT can be started, the clock and reset signals of the design need to be tied to the primary clock network of the device. This will be done in the External Attributes file, the .pat file.

Settings in the .pat file:

```
port CLK instance IPCLK
port RST instance IPRST
```

The settings for the autolayout is done in the Options → Autolayout menu. These settings can also influence the results of the implementation.

For Autolayout Options the parameter group "Minimum Delay" is chosen. The settings in this parameter group are set to produce the best implementation in terms of design speed. Important for this result is also the value which is put into the target delay. For this example the target delay is set to 200 which means 20ns, a target frequency of 50MHz. In the advanced autolayout options panel, be sure to enable tertiary clocks.



The results of the implementations of the different synthesis results of the 16-bit accumulator can be seen below.

Explanations of names:

accu_s: select_modgen is set to small
 accu_sm: select_modgen is set to smallest
 accu_f: select_modgen is set to fast
 accu_fa: select_modgen is set to fastest

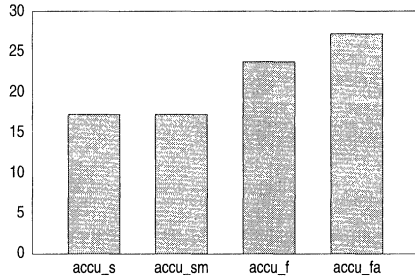


Figure 1. results accu 16bit

Name	Post Autolayout Result (MHz)
accu_s	17.30
accu_sm	17.30
accu_f	23.80
accu_fa	27.30

As these results show the selection of the select_modgen influences the performance of the design. The best implementation in terms of speed will be produced by choosing the modgen library fastest. The 16 Bit Accumulator is able to be implemented with a maximum frequency of 27.3 Mhz. This performance can be reached with a simple VHDL description and without any use of special macros. Due to this it can be said that this performance of an 16 bit accumulator can also be reached within a big design where the accumulator is just a part of it when the whole design is described using VHDL.

4



An FPGA Primer for PLD Users

Prepared by
Terry Schaul
Motorola Technical Resource Manager

4



An FPGA Primer for PLD Users

Introduction

Welcome to the exciting world of Field Programmable Arrays (FPGA). As the title suggests, the goal of this application note is to familiarize the current Programmable Logic Device (PLD) user with FPGA architecture, while also introducing Motorola's exciting new MPA (Motorola Programmable Array). Digital system design is becoming more complex as system requirements continue to push the envelope of semiconductor technology. These demanding requirements call for a digital logic solution optimized to meet capacity, performance, cost and time-to-market constraints. An FPGA solution meets these challenging requirements.

Programmable Logic Devices

As Figure 1 shows, logic devices can be classified into five general types: standard logic, custom and fixed function logic, PLD, Application Specific Integrated Circuit (ASIC), and Microcontrollers.

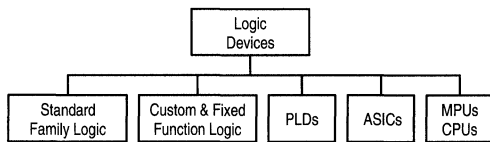


Figure 1. Logic Device Types

Of these five classifications, the PLD segment is the most dynamic of these markets. The advantages of PLD's are clear: increased integration, improved reliability, lower cost, added flexibility, and accelerated time to market. PLD's can be further classified into the following families: Program- mable Array Logic (PAL), simple PLD (SPLD), complex PLD (CPLD) and Field Programmable Gate Arrays (FPGA). Figure 2 displays this classification along with the competing technologies.

PLD's differ in structure from the traditional standard logic devices. PLD's allow the designer to program a specific logic function into an uncommitted logic array after the device has been fabricated. The uncommitted logic array differentiates the PLD from the ASIC family. The size and complexity of the logic targeted for a PLD differs among the various classifications, and the actual dividing line between the classifications is fuzzy.

PAL and SPLD are one in the same. They are both array-oriented devices that have an AND-OR structure with a wide-input AND gate feeding a narrower OR gate. This forms a sum-of-products structure suitable to implementing reduced Boolean equations. A limited number of registers are made available at the the output of selected OR gates. This architecture tends to be "logic rich" because the ratio of logic gates to registers is on the order of 5 to 1. FPGA's, on the other hand, tend to be "register rich", with a logic-to-register ratio closer to 2 to 1.

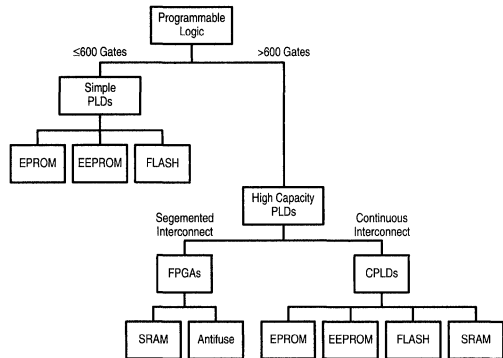


Figure 2. Programmable Logic Market

CPLD's are sometimes referred to as "super PAL's". This device typically consists of multiple optimized PAL blocks interconnected by a programmable switching matrix. The density and complexity of these devices can accommodate significant logic, but not the density and complexity attainable by FPGA's.

FPGA's consist of four key elements: logic cells, logic cell interconnection, programming elements, and I/O modules. Figure 3 shows the configuration of a generic FPGA. Note that each FPGA supplier packages these elements together differently, and these elements will be examined in detail later.

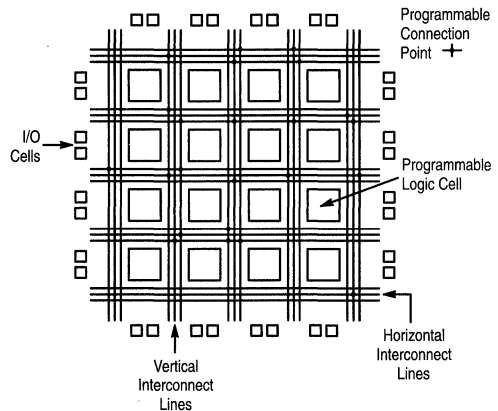


Figure 3. Generic FPGA Architecture

CPLD and FPGA differ in their interconnect structures. Typically, CPLD's use a continuous interconnect structure, while FPGA's use a segmented interconnect structure. The CPLD's fixed interconnect structure implies that the designer knows the delay between logic cells before any design work is done. For FPGA's, however, these delays can only be

4



estimated until the place and route is complete. After place and route, all delays inside an FPGA are known.

Besides the physical structure, the different PLD families also vary greatly in the density of the devices. PAL and SPLD contain fewer than 600 usable gates, and are technologically limited in their growth potential. CPLD's typically contain between 500 and 10,000 usable gates, while FPGA's offer the most dense devices ranging from 2,000 to 50,000 usable gates and beyond. Only the FPGA architecture will allow for continuous density growth.

Development tools play a very important role in designing with PLD's. The development tool strategy differs among the PLD families and their suppliers. PAL's and SPLD's are less complex and dense, and they usually require the use of a single industry standard development tool such as ABEL or PALASM to complete the PLD design cycle. CPLD's and FPGA's, however, with their increased complexity and density, require more sophisticated development tools. CPLD's and FPGA's require both front- and back-end development tools. Design capture can be in the form of schematic capture or a Hardware Descriptive Language (HDL). Some suppliers have a flexible front-end tool strategy that allows the use of popular third party tools for design capture and netlist generation. Still other suppliers offer their own proprietary front-end development tools. Regardless of the front-end tool strategy, all devices require a specific back-end tool that fits or places and routes the design into the specific device. These back-end tools are offered by the CPLD/FPGA suppliers for their specific devices.

4

FPGA Attributes

The previous section summarized the different classifications of PLD's. The key elements of the the FPGA elements that will now be examined are: physical structure, granularity, interconnectivity and development tools.

Physical Structure

The physical structure of the FPGA memory element comes in two flavors: SRAM and Anti-fuse. SRAM-based FPGA's consist of programmable logic cells surrounded by a programmable interconnect matrix where the function and routing that is programmed into the device via the SRAM is volatile. At each power-up, the program is downloaded into the device from an off-chip, non-volatile memory or a microprocessor. The reprogrammable capability is an obvious benefit for the SRAM-based solution. Since FPGA designs tend to be complex, several design iterations are usually required to complete the design. The reprogrammable FPGA can significantly reduce prototype costs. More importantly, reprogrammability adds flexibility to a system by providing easy field upgrades or on-the-fly reconfigurations by simply changing the downloaded program. The volatility of SRAM-based FPGA's requires some additional overhead for power-up downloading. This is most often accomplished with industry standard serial boot PROM's. These devices connect directly to the FPGA with a very simple 3-wire interface (chip enable, clock and data).

SRAM-based architectures integrate well with CMOS technology. This provides a roadmap for the technology-

savvy supplier to combine FPGA modules with other popular CMOS cores such as: DSP, SRAM and MCUs.

Total system reliability is another benefit of SRAM-based FPGA's. SRAM-based devices are fully tested before shipping, so they require no pre-inspection. Since they are volatile and programmed in-circuit, they can be placed directly into the target system, avoiding additional pre-programming, inventory, and testing that non SRAM-based devices require.

Anti-fuse, also referred to as one-time programmable device, is a physical structure whereby the interconnect between logic units is an irreversible fusible link. Configuring the device is accomplished by placing a programming voltage between two metal plates to cause an electric field. This electric field then causes metal to melt and rush in, filling a pin hole in the insulation. The filling of this pin-hole with metal completes the circuit, hence the name "anti-fuse".

While anti-fuse devices typically implement the same register intensive logic functions as SRAM-based FPGA's, they cannot be reprogrammed. Anti-fuse devices also require higher voltages and longer times to program the fuse. As mentioned earlier, FPGA designs often require several design iterations to complete, and an anti-fuse FPGA solution can potentially increase engineering time and prototyping device cost.

Reliability issues must be addressed with anti-fuse devices. Since it is impossible to determine the resistance at which the metal will melt during programming, test vectors must be generated to prove programming accuracy. Even with test vectors, the devices are not 100% testable. The extra handling that occurs to program the devices before they are placed in the target system introduces additional reliability concerns. Additionally, anti-fuse technology has historically proven to be a difficult process to control. Consistent fuse formation and prevention of "healing" have been some of the troublesome past issues. Recently at least one major vendor has abandoned its anti-fuse efforts.

The anti-fuse structure provides few advantages over the SRAM-based structure. While the anti-fuse itself is typically a smaller element than a single SRAM cell, there is significant overhead circuitry associated with high voltage programming and testability features that largely negate this supposed advantage. Anti-fuse devices are generally more secure than SRAM-based devices. Anti-fuse devices provide a mechanism to prevent the "read-back" of programming information. This feature, coupled with the fact that there is no external programming bitstream to intercept, makes anti-fuse devices a natural choice for systems where security of the processing algorithm is critical.

Granularity

FPGA's, by definition, contain logic cells, and granularity refers to logical capacity of these individual FPGA logic cells. Granularity is generally referred to as "fine grain" or "coarse grain". Again, there is no clear dividing line between fine and coarse grain, as FPGA suppliers offer nearly a continuous spectrum of logic cell capacity. Fine grain indicates fewer logic gates per logic cell. Coarse grain indicates greater logic gates per logic cell. As a rule of thumb, fine grain logic cells contain 1-8 logic gates per logic cell, whereas coarse grain logic cells contain 9-25 logic gates per logic cell.



Granularity affects the designer both logically and physically. Logically, when fitting a design into the logic cells of a fine grain device, the design is less abstracted than when placed into a coarse grain device. This lesser degree of abstraction makes design adjustments easier. Physically, the designer must understand the difference between raw gates and usable gates. The actual capacity of the device is described by raw gates. Raw gates per logic cell is equal to the total raw gates divided by number of logic cells. Raw gates is an idealistic number, and the coarser the granularity of the device the less raw gates resembles usable gates. Usable gates refers to the number of logic gates actually used in a design. Usable gates depends heavily on the architecture of the routing resources, on the device granularity, and on the logic design fitted into the device. The disparity between raw gates and usable gates comes from the fact that those logic cell gates not used in mapping a product term into a particular logic cell are wasted, or used for routing. Generally, a coarser logic cell decreases the logic gate utilization efficiency. To improve this logic gate utilization many FPGA suppliers are migrating towards fine grain architectures.

Interconnectivity

Another key attribute of the FPGA architecture is the interconnectivity between the logic cells. Channels of wire segments are provided between the rows and columns of the logic cells. These wire segments are connected at various points throughout this wire matrix. While sufficient wiring segments must be provided for good routability, excess wiring segments may waste valuable chip area.

One of the most important aspects of an FPGA architecture is the ability of the development tool to route a design into the selected device without the designer having to spend time manually placing and routing. Most anti-fuse devices achieve maximum routability by providing a fully connected crossbar. SRAM-based devices simply cannot afford to provide a fully connected crossbar for die size reasons. Thus SRAM-based devices typically provide a partially connected crossbar as shown in Figure 4. Since SRAM based arrays often use elements of otherwise unoccupied logic cells to participate in routing, this apparent disadvantage is minimized.

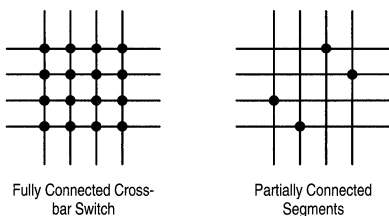


Figure 4. Switch Connections

The segmentation of the routing tracks plays an important role in the interconnectivity of an FPGA. The wire segments span along the FPGA in routing channels between the logic cells. The length of this wire segment span varies among FPGA devices. FPGA segmentation allows for incremental delays. As the number of interconnect segments increases,

the interconnect delay also increases. Since the number of segments required to interconnect signals is neither constant nor exactly predictable, the routing delays can not be determined precisely until place and routing has been completed. Fine grained FPGA inherently use more logic cells. This places a heavy burden on the interconnect structure. A robust interconnect structure is required by fine grain architectures to avoid a routing bottleneck. Varied length or hierarchal segmentation approaches provide adequate resources to avoid such bottlenecks.

Development Tools

Last, but certainly not least, an FPGA primer would not be complete without examining the critical issue of FPGA development tools. This is a very complex subject, and the intent of this application note is to introduce the reader to various FPGA tool strategies. No matter how efficient an FPGA architecture is, its efficiency lies in the ability of the development software to optimize for that specific architecture. The development tools comprise the software system that enables the designer to define the logic, capture it, verify it and implement the logic into the FPGA device. These tools are typically bundled as front-end tools and back-end tools. Some software venders and FPGA suppliers offer both front- and back-end tools as a complete toolset, while others just offer a portion of the toolset.

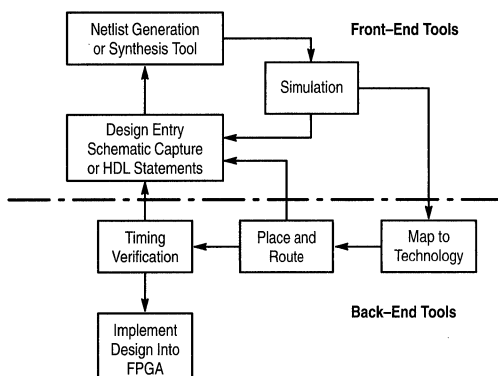


Figure 5. Front & Back End Tools

Front-end Tools

As shown in Figure 5, front-end tools facilitate design entry, primary design verification and netlist generation. These front-end tools allow for design entry via schematic capture or a Hardware Description Language (HDL). Schematic capture requires the designer to enter the design in the form of gates, logic elements from libraries of primitive functions and macros. VHDL is one such behavioral language that describes the function and attributes of primitive components and macros in the design. VHDL and Verilog are both gaining momentum as the front-end tools of choice for several reasons. Once a designer is comfortable with an HDL, the time to enter a design is generally reduced versus schematic capture, especially for very large designs. HDL's are technology



independent, allowing for a more efficient design. However, gate counts may increase over schematic captured designs when downloading an HDL design into an FPGA. Some popular third party Computer Aided Engineering (CAE) development tools include: Viewlogic, Mentor, Synopsys, Orcad, and Veribest.

Simulation takes place in two key places in the design flow. Logical simulation just after netlist generation verifies the functionality of the design. At this point the design has not been fitted into a specific device, thus only estimated or unit wire delays are used. The second place where simulation can take place is following the place and routing of the design. This is a functional simulation of the design and uses actual back annotated routing delays.

Back-end Tools

As shown in Figure 5, a back-end tool maps the design into the specific FPGA device, provides timing verification, and configures the downloadable program (a.k.a. configuration bitstream). Mapping tools convert original design elements in the netlist into logic elements available to the specific FPGA. Place and route tools interconnect the logic cells to each other and the I/O blocks. Design requirements such as speed, delay and skew are realized during this phase. Meeting these design requirements depends heavily on the efficiency and robustness of the place and route tool as well as the device architecture. Timing verification can be performed with a timing analyzer or by functional simulation. Finally, the back-end tool provides the configuration bitstream to download the logic into the FPGA device. This step is called bit stream generation. Back-end tools are specific for each FPGA device and thus offered by the individual FPGA suppliers.

WHY MPA

Motorola enters the exciting FPGA market with its MPA1000 family of Programmable Arrays. The MPA1000 products are SRAM-based, fine grained, high performance, low cost solutions for reconfigurable logic applications.

The SRAM-based physical structure of the MPA1000 offers significant advantages. First and foremost is the reprogrammability that SRAM-based products provide over the one-time programmable anti-fuse products. With today's iterative and complex design flows, reprogrammability allows a flexible, cost-effective solution. Additionally, the most recent logic system design methodologies depend on the in-system-reprogrammability of the SRAM based FPGA's to accommodate on-the-fly functionality changes, built-in-self-test (BIST), field upgrades and other flexible system features. FPGA's are no longer just another vehicle to implement a fixed logic function. Their reprogrammability has become a foundation for many new system designs.

Motorola is a broad-based semiconductor innovator, and brings technology leadership unmatched by its FPGA competitors. As mentioned earlier, SRAM-based structures integrate well with CMOS technology. Motorola will lead the effort in integrating a programmable array module with other popular CMOS cores. This will allow Motorola to differentiate itself from the crowded FPGA and processor markets by offering exciting new products.

The MPA utilizes a hierarchical routing scheme which takes advantage of the fine grained structure while avoiding the routing bottlenecks. Each MPA1000 family member is a partitioned array of logic cells. At the highest level of hierarchy each device is partitioned into 4 equal size quadrants. I/O cells surround this core of quadrants. Each quadrant is further subdivided into zones. A zone consists of a 10x10 array of fine grained logic cells, 20 port cells, and a clock distribution cell. Each logic cell provides a NAND gate or a secondary function (Wired-Or, Exclusive-Or, D-Flip/Flop or latch) depending on the position of the logic cell. The MPA interconnect structure is partitioned into 3 hierarchical levels: global, medium and local. Hierarchical routing resources provide routing flexibility that enables the MPA devices to overcome bottlenecks associated with other fine and course grain architectures.

Motorola's development tool strategy provides flexibility for the designer. Motorola accepts both schematic capture and HDL design inputs from over ten front-end CAE development tool vendors. Motorola provides the Motorola Programmable Array Design System back-end toolset at a nominal cost (free for low-end devices). Motorola Programmable Array Design System automatically analyzes, optimizes, transforms, and places and routes a design into an MPA device. Full incremental design support as well as complete on-line help reduces design time in a user friendly environment.

In summary, Motorola's MPA products are a high density, high performance, low cost solution for reconfigurable logic needs. Fine grain structure, abundant hierarchical interconnection resources and automatic, timing driven, tools work together to quickly provide design implementations that meet timing constraints without sacrificing device utilization.

CONCLUSION

This FPGA primer highlighted PLD families, provided a glimpse into the competing FPGA architectures, and examined the MPA solution. The key FPGA attributes of physical structure, granularity, interconnectivity and development tools have been elegantly addressed by the MPA devices. Today's digital engineers face such critical design issues as schedule, product requirements (performance, size, cost, etc.) and tool compatibility. To ensure design success, these issues need to be considered when selecting an appropriate FPGA solution in this complex world of digital design.

For additional information, please reference the URL:

<http://sps.motorola.com/fpga>

4



Using JTAG Boundary Scan with the Motorola MPA1000 Family of FPGAs

Prepared by
Marten L. Smith
Motorola Programmable Logic

4



Using JTAG Boundary Scan with the Motorola MPA1000 Family of FPGAs

INTRODUCTION

JTAG is a standardized boundary scan methodology used for board-level testing to detect faults in package and board connections, as well as internal circuitry. The MPA1000 JTAG architecture is designed to meet the IEEE 1149.1 standard for testability of integrated circuits and to offer our customers a more effective way to test their circuit boards which use the members of the MPA1000 family of FPGAs.

This application note gives the reader information on how JTAG is implemented on Motorola's MPA1000 family of FPGAs. For more information on JTAG and IEEE 1149.1 the reader is encouraged to reference books and articles that directly relate to these topics.

JTAG INTERFACE

The MPA1000 device contains dedicated JTAG IEEE 1149.1 boundary scan circuitry. JTAG can be used on unconfigured devices using the BYPASS or IDCODE instructions. Or JTAG can be used on configured devices using any of the public instructions available.

The first step in setting up any of the MPA1000 family of devices in JTAG mode is to set aside the five TAP signal pins. This is done so that these pins can be utilized as JTAG pins during JTAG mode. These pins are set aside for JTAG when the user first runs the MPA Design System (MPADS) tool. The user should first push the **Select Device** button on the main window of the MPADS tool. This will bring up the **Select Device** window. The user should then select the **JTAG Mode** option along with the desired configuration option. Then the user should push the **OK** button. These pins can still be used as I/O for the user's design configuration, but will revert to the TAP (JTAG) signals when the device is in the JTAG mode as outlined below.

The second step in order to enable JTAG is to raise the MODE[3] pin HIGH after the device is configured and the user

is ready for the device to go into JTAG mode (see Figure 1 and Table 1). When the MODE[3] pin of the MPA1000 device is HIGH, five user I/O pins become the TAP (JTAG) signals and user mode operation of those pins is interrupted. After JTAG testing, these pins can be programmed as normal I/O pins by deasserting the MODE[3] pin. The TAP controller can take control of all device pins, but care must be used to prevent the TAP controller from interfering with device user mode or configuration operation (the configuration pins are the RESETB, CLK, MODE[3:0], and F[4:0] pins). Note: the MPA1000 family of FPGAs can not be configured using the JTAG port.

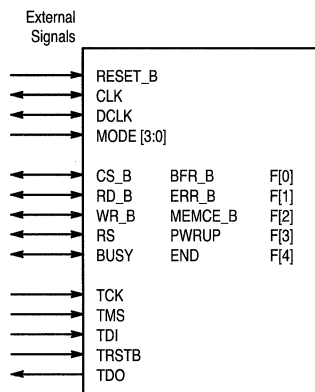


Figure 1. JTAG and Configuration Interface Signals
With the exception of the JTAG signals, the rest of these signals are dedicated and are never available for user I/O.

Table 1. Mode[3:0] Pin Programming

Bits				Description
[3]	[2]	[1]	[0]	
0	X	0	0	Micro Mode – Microprocessor/controller interface circuitry with parallel (byte-wide) data
0	X	0	1	BFR Mode (1) – Boot From ROM, byte-wide data. MPA generates ROM addresses
0	X	1	0	BFR Mode (2) – Boot From ROM, serial data. (Low pin count serial EPROM generates own addresses)
0	X	1	1	BFR Mode (3) – Boot From ROM, byte-wide data. MPA does not generate ROM addresses
0	1	X	X	Use external clock for configuration
1	X	X	X	Enable JTAG circuitry and pins



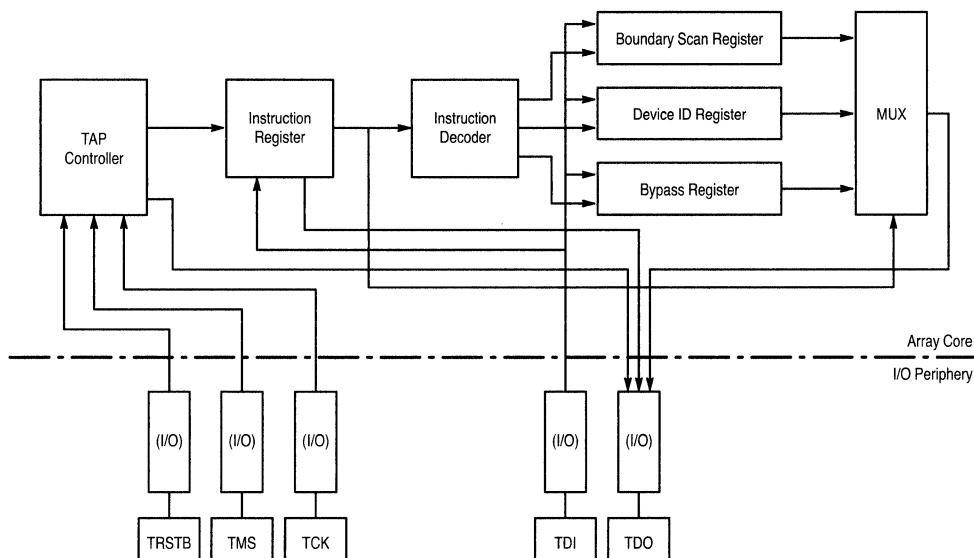


Figure 2. MPA1000 JTAG Architecture

4

JTAG BOUNDARY SCAN

Architecture

Figure 2 shows a block diagram of the JTAG architecture of an MPA1000 device.

TAP and I/O Signals

The TAP (Test Access Port) consists of five externally accessible signals which are used to control and observe boundary scan data. These five pins are: TCK, TMS, TDI, TRSTB, and TDO are multiplexed with normal I/O signal pins. After JTAG testing, these pins can be programmed as normal I/O pins by deasserting the MODE[3] pin. Note: There are no Boundary Scan Cells associated with the TAP signals.

The following is an explanation of the TAP signals

TCK – Test Clock. This clock input is used to synchronize all of the JTAG functions. It is used for functions such as the clock for the TAP controller and the shifting of test data through the boundary scan cells. The maximum frequency that can be used for TCK is 16 MHz.

TMS – Test Mode Select. This input controls the state changes of the TAP controller. TMS determines whether the TAP controller performs an Instruction Register function or a function involving one of the data registers (See Figure 3). TMS is sampled on the rising edge of TCK.

TDI – Test Data Input. Serial test data is received by this input and sent to the Instruction Register or one of the data registers according to the state of the TAP controller. TDI is sampled on the rising edge of TCK.

TDO – Test Data Output. Serial test data is driven from this output, having been sent from a selected data register or the instruction register. Which data register TDO receives data from is determined by the instruction loaded into the Instruction Register. TDO changes on the falling edge of TCK.

TRSTB – Test Reset_Bar. This active low input asynchronously initializes the TAP controller to the Test-Logic-Reset state (see Figure 3). This state initializes all of the test logic of the JTAG circuitry so that normal (i.e. non-JTAG) operation of the MPA1000 device can continue.

JTAG Control and Registers

MPA1000 family JTAG TAP controller and registers are shown in Figure 3. The JTAG architecture of an MPA1000 device consists of a TAP controller, an Instruction Register, Instruction Decoder, a series of data registers, and a Multiplexing function for the data register outputs. The data registers consist of, a Bypass Register, an ID Register, and a Boundary Scan Register. The JTAG circuitry in the MPA1000 family is hard-wired.

The following is an explanation of the controller and registers in the MPA1000 JTAG circuitry:

TAP CONTROLLER – The TAP controller is a synchronous, 16-state, finite state machine which selects the mode of operation for the test circuitry. The state machine diagram is shown in Figure 3. The state transitions of the TAP controller occur based on the value of TMS at the time of a rising edge of TCK. There are two general functions performed by the TAP controller; one is the loading of test instructions into the Instruction Register and the other is the shifting of test data through the data registers.



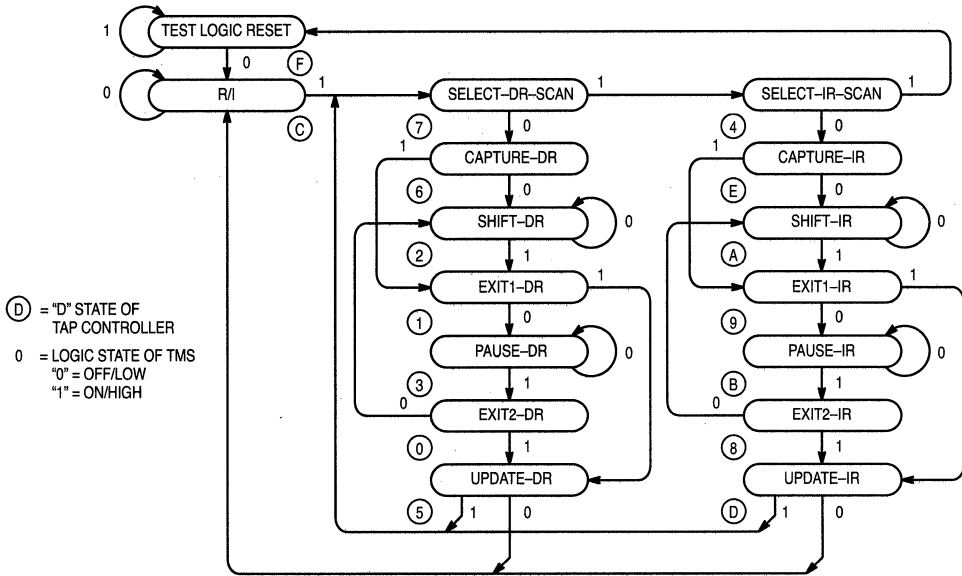


Figure 3. TAP Controller State Diagram

4

INSTRUCTION REGISTER – The Instruction Register is a three-bit register which permits a test instruction (also called a Public Instruction) to be shifted in, which selects the test to be performed. The Instruction Decoder decodes the test instruction and selects which data will be multiplexed into the data registers. Table 2 shows the basic Public Instructions supported by the MPA1000 family.

Table 2. Basic Public Instruction

Three-Bit I ₂ (msb)	I ₁	I ₀ (lsb)	Public Instruction	Register Selected
0	0	0	EXTTEST	Boundary Scan
0	0	1	INTEST	Boundary Scan
0	1	0	SAMPLE	Boundary Scan
1	0	0	IDCODE	Device Register
1	1	1	BYPASS	Bypass Register

BYPASS REGISTER – The Bypass Register is a single-bit register. When selected, the Bypass Register provides the shortest path, a single bit scan path, between TDI and TDO. The Bypass Register makes it possible to reduce the scan path through devices that are not involved in the current board-level test.

DEVICE IDENTIFICATION REGISTER – The Device Identification Register is a 32-bit shift register which holds the Motorola identification code, array identification,

programmable logic products identification, and version number. The bit assignment for the ID code is given in Table 3.

Table 3. Device Register ID Codes

Bit Number	Code Use
0–11	Motorola Identification
12–21	Array Identification
22–27	Programmable Logic Products Identification
28–31	Version Number

The ID codes for Motorola’s MPA1000 family of FPGAs are listed in Table 4.

Table 4. MPA1000 ID Codes

Array	ID Code (Binary)	ID Code (Hex)
MPA1016	0001 001110 0100001110 000000011101	1 390E 01D
MPA1036	0001 001110 0100001110 000000011101	1 391E 01D
MPA1064	0001 001110 0100110100 000000011101	1 3934 01D
MPA1100	0001 001110 0101000000 000000011101	1 3940 01D

BOUNDARY SCAN REGISTER – The Boundary Scan Register is the chain of JTAG boundary scan cells (BSC) that are linked together to form a shift register around the periphery of the array. The test data enters the Boundary Scan Register through the TDI pin. The test data is then shifted around the



array through each boundary scan cell in a counter-clockwise direction and finally exits through the TDO pin (see Figure 4).

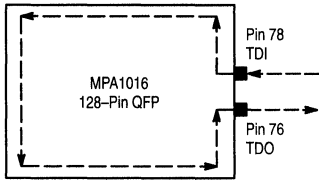


Figure 4. Direction Of Data Shift Around Package
This example is for a MPA1016 device in a 128-pin QFP package, but the concept is the same for all devices and packages.

I/O Pin Boundary Scan Cells – Each I/O pin is designed as a bi-directional pin. Therefore, a boundary scan cell is comprised of a two-bit shift register (see Figure 5). One of the bits of this shift register is the bi-directional data control cell and it is used to drive/receive a data bit to/from the device's I/O pad or monitor a data bit that may be on the device's I/O pad. Going deeper into the data control cell, it is made up of a capture latch and an update latch (See Figure 6. Note: these two latches will be referred to several times in this application note, so it is recommended that the reader be familiar with these latches. See JTAG, Boundary Scan, or IEEE documents for more details). The other bit of this shift register is the I/O control cell. The I/O control cell is used to either control the internal enable of the three-state output buffer or to monitor the status of the three-state output buffer (i.e. during the SAMPLE instruction). The I/O control cell is also used to drive a data bit to the next data control cell in the scan chain. When the bit in the I/O control cell is at a logic 1 then that particular I/O pad's three-state output buffer is enabled. Conversely, when the bit in the I/O control cell is at a logic 0 then that particular I/O pad's three-state output buffer is disabled (Note: Even though the I/O control cell controls whether or not the output buffer is enabled, the MPA is a programmable device so the I/O pads must also be configured as an input or an output in order to pass data). A boundary scan cell resides in every I/O pad with the exception of the TDI, TCK, TMS, TRSTB, and TDO pins.

Control Pin Boundary Scan Cells – Boundary scan cells are also associated with the control pins MODE [3:0], F [4:0], RESETB, and CLK so these cells are included when counting the number of cells in the scan chain, but have limited or no use for parallel loading into or out from the scan chain during JTAG (i.e. as a group, you can only do the SAMPLE instruction because MODE [3:0], RESETB, and CLK are sense only cells – See Figure 7). Also, the user should not attempt to change signal level on one of the control pins. The levels on pins MODE [3:0], RESETB, CLK, and F [4:0] should stay the same in accordance with the requirements for operation found in the MPA data book. Therefore only the SAMPLE test should be used to test them.

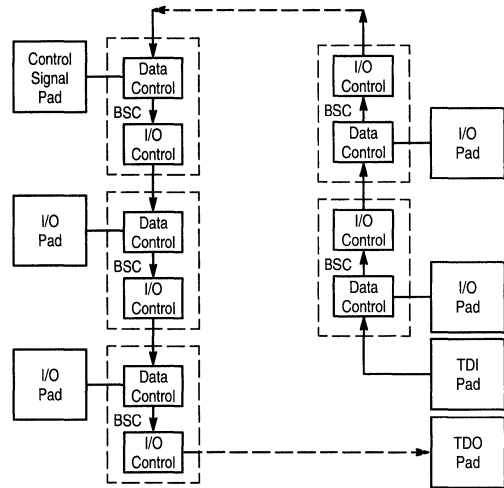


Figure 5. Internal Boundary Scan Cell Chain
The BSC is the Boundary Scan Cell and is comprised of the I/O Control and Data Control Cells. Note the dashed arrows show that there are many more Boundary Scan Cells, in the scan chain, than those pictured here.

4

Shifting Through Both Kinds of Boundary Scan Cells – One important point to remember is that since each Boundary Scan Cell is a two-bit shift register, the number of TCK clock cycles needed to go around the entire scan chain is two times the number of pins that have boundary scan cells associated with them in the MPA device. Or in other words, it takes two TCK clock cycles for the data to shift from one boundary scan cell to another. For example, on an MPA1016, 128-Pin QFP package, there are 80 I/O pads on the integrated circuit die and 80 I/O pins available on the package. Five of these are the TAP signals and are not part of the scan chain therefore, there are 75 I/O pads that have boundary scan cells (BSCs) associated with them. There are also 11 control signals (MODE[3:0], F[4:0], RESETB, and CLK) that have boundary scan cells associated with them. This gives us the total number of boundary scan cells, which is 75+11=86 boundary scan cells on this MPA device and package. Because of this, the data will need to be clocked 86 x 2 or 172 times in order to go around the entire scan chain from TDI to TDO.

Another very important point to remember is that with certain MPA1000 devices there are internal I/O pads on the integrated circuit die that are not brought out to external package pins. These I/O pads are still part of the boundary scan chain and must be taken into account while shifting data around the scan chain (see Data Book, DL201/D, for information on available pin-outs per package). For example, there are 61 total I/O pins (including the TAP signals) on an MPA1016 84-Pin PLCC package, but there are still 80 BSC I/O pads on the MPA1016 integrated circuit die. Therefore the same analysis will need to be performed to count the proper number of boundary scan cells for this MPA device. Therefore,



80 I/O pads – 5 TAP pins = 75 I/O pads with BSCs associated with them. 75 I/O BSCs + 11 control signal BSCs = 86 total BSCs. The data will need to be clocked 86 x 2 or 172 times in order to go around the entire scan chain from TDI to TDO.

JTAG INSTRUCTIONS AND TESTS

There are five JTAG public instructions that have been implemented for the MPA1000 family of FPGAs. As mentioned previously, these instructions are: BYPASS, IDCODE, SAMPLE, INTEST, and EXTEST.

INSTRUCTIONS

BYPASS – The BYPASS instruction selects the Bypass Register to be connected between TDI and TDO. This instruction allows serial data to be transferred through the device from TDI to TDO without affecting the operation of the

device. When selected, the Bypass Register provides the shortest path, a single bit scan path, between TDI and TDO. The Bypass Register makes it possible to reduce the scan path to a single register, thereby bypassing the devices that are not involved in the current board-level test. The decoded Public Instruction for BYPASS in the MPA is i2=1, i1=1, and i0=1 (111 – see Table 2).

IDCODE – The IDCODE instruction selects the Device Identification Register to be connected to the TDO output. The Device Identification Register is a 32-bit shift register which holds the Motorola identity code, array ID, product ID, and version number (see Table 3 and Table 4). By selecting the IDCODE instruction the Device Identification Register contents will be shifted out the TDO output. The decoded Public Instruction for IDCODE in the MPA is i2=1, i1=0, and i0=0 (100 – see Table 2).

4

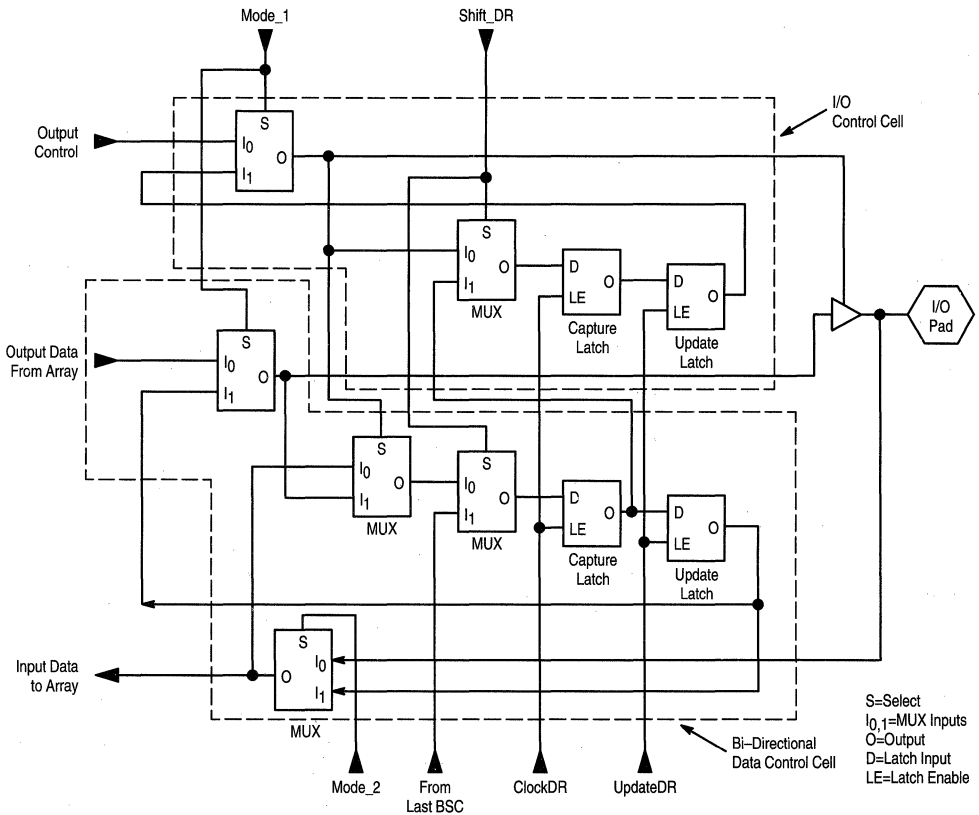


Figure 6. General Structure of Boundary Scan Cell (BSC)
 used for input or 3-state output on general I/O pads (Note: this is a general block diagram and should not be construed to infer design details)



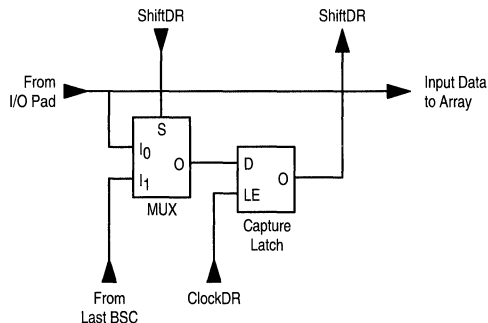


Figure 7. General Structure of Sense-Only Boundary Scan Cell (BSC) used for Pins: CLK, RESETB, MODE [3:0] (Note: this is a general block diagram and should not be construed to infer design details)

(Note: For the SAMPLE, INTEST, and EXTEST commands the MPA1000 device I/O pins are utilized to load data from the inputs to the scan chain and/or to load data from the scan chain to the outputs. In order to do this the I/O pins on the MPA1000 device must first be configured as inputs or outputs using one of the configuration modes: Micro Mode or Boot From ROM (BFR) Modes 1, 2, or 3 (See Table 1). After doing this, the MPA1000 device can then be put into JTAG Mode for test and the I/O pins can be utilized for the loading in to or out of the scan chain).

Note: The user must also remember that there are boundary scan cells which are associated with the pins RESETB, CLK, MODE [3:0], and F [4:0] are included when counting the number of cells in the scan chain for shifting purposes, but can only be tested, as a group, using the SAMPLE instruction.

SAMPLE – The SAMPLE instruction lets you take a sample of the normal operation of the MPA1000 device pins which can be shifted out to TDO. During the SAMPLE function the BSCs take a sample of the functional data that exists on the I/O and control pins. The sample is taken regardless of the pins being configured as inputs, outputs, or bi-directionals. The functional data is entered into the BSC capture latches. Typically the data is then shifted around the scan chain.

The BSCs for the control pins CLK, RESETB, and MODE[3:0] are scan only cells. Because of this the SAMPLE test is the only way to test these pins in JTAG mode (e.g. you can not run an EXTEST test on the control pins that are set as inputs due to a configuration mode, such as MODE [3:0]. See MPA data book (DL201/D), Device Configuration section for more details). Also, the user should not attempt to change a signal level on one of the control pins. The levels should stay the same in accordance with the requirements for operation found in the MPA data book.

The SAMPLE instruction selects the Boundary Scan Register to be connected between TDI and TDO. The

decoded Public Instruction for SAMPLE in the MPA is $i2=0$, $i1=1$, and $i0=0$ (010 – see Table 2).

EXTEST – The EXTEST instruction is generally used to drive test data out of the MPA to another device on a printed circuit board (it can also input data to the scan chain, but SAMPLE is the instruction more often used to do this). The EXTEST instruction places the device in an external boundary test mode and selects the BSCs to be connected between TDI and TDO. The outputs are tested by serially shifting data through the BSC capture latches, loading them into the BSCs' update latches, and driving them out through the I/O pins. In this way the printed circuit board connections can be tested without the need for a physical test point on the printed circuit board. The user must remember that in order to get data loaded out from the scan chain all I/O pins must be configured as outputs before running the EXTEST test. Data can also be loaded into the scan chain during EXTEST by configuring the desired I/O pins as inputs and loading the data on the inputs in the CaptureDR state of the TAP Controller. The decoded Public Instruction for EXTEST in the MPA is $i2=0$, $i1=0$, and $i0=0$ (000 – see Table 2).

INTEST – The instruction is generally used to test the internal logic that was designed and configured by the MPA user. The INTEST instruction places the device in an internal boundary test mode and selects the Boundary Scan Register to be connected between TDI and TDO. Test data is typically loaded from the I/O pads and goes into an input BSC cell. From there the data goes into the user-configured internal logic of the MPA1000. When the data reaches the user-configured internal logic, it is manipulated by that logic, and the manipulated data is latched onto an output BSC cell's capture latch. From the capture latch, the data can be shifted out to TDO. The user must remember that in order to get data loaded into the scan chain all I/O pins must be configured as inputs before running the INTEST test. The decoded Public Instruction for INTEST in the MPA is $i2=0$, $i1=0$, and $i0=1$ (001 – see Table 2).

4

INSTRUCTION EXAMPLES

Below are generic examples for the MPA Public Instructions. These are only simple examples and do not necessarily show all of the functions of each instruction. The MPA used in the examples is an MPA1016 in a 128 QFP package. On this device and package all of the internal I/O pads are brought out to external package pins. It is strongly recommended that the reader refer to the MPA1000 pin assignments page for the MPA1016 in order to follow along. This is found in the MPA Data Book (DL201/D).

Although some semiconductor manufacturers have used the INTEST instruction for internal testing of ICs, there is a lack of interest by board and system manufacturers for this instruction as well as a lack of commercially available software to support it. For this reason, an INTEST example is not shown here.

The examples will refer to the TAP controller state machine diagram, Figure 3. The function of each state occurs when leaving the state unless otherwise noted. The format for the examples is as follows:



C R T T T T T
 L S D D M R C
 K T O I S S K

0 1 H 1 0 1 + / Comment /

Where:

CLK = Configuration Clock for MPA (see Figure 1).
 Used to clock in MPA configuration.
 Remains at logic 0 (after configuration) for
 the purpose of these examples.

RST = Configuration Reset for MPA device (see
 Figure 1). When this input is held to a logic
 0 it clears the MPA's configuration memory.
 When this input is brought to a logic 1 then
 the device will start re-configuring while
 CLK is running. Remains at logic 1 (after
 configuration) for the purpose of these
 examples.

TDO = Test Data Out

TDI = Test Data In

TMS = Test Mode Select

TRS = Test Reset_Bar (Asserts on logic 0)

TCK = Test Clock

0 = Logic Zero (0) for inputs

1 = Logic One (1) for inputs

L = Logic LOW for outputs

H = Logic HIGH for outputs

+ = Clock Transition LO-HI-LO

X = Don't Care

DR = Data Register and IR = Instruction Register

/ Comment = A comment for that particular test
 vector

Note: The TCK transition + occurs after the other signals in
 the vector are setup. For example, when the comment for a
 vector says, "In Run/Idle state. Setup Select-DR-scan state"
 this means that the TAP controller is still in the Run/Idle state
 even though the TMS signal has changed for the next state
 (Select-DR-scan) because TCK is the last signal in the vector
 that transitions.

BYPASS INSTRUCTION EXAMPLE

The following is a generic vector example for a BYPASS
 test:

4

C R T T T T T
 L S D D M R C
 K T O I S S K

0 1 X 0 1 0 0 / Non-JTAG state for tester setup (optional). /
 0 1 X 0 1 1 + / Test-Logic-Reset (TLR) state. /
 0 1 X 0 1 1 + / Test-Logic-Reset (TLR) state (number of /
 / loops in this state is up to the programmer). /
 0 1 X 0 0 1 + / In TLR state. Setup for Run/Idle (R/I) state. /
 / (Number of loops in the Run/Idle state is up to /
 / the programmer) /
 0 1 X 0 1 1 + / In R/I state. Setup for Select-DR-scan state /
 0 1 X 0 1 1 + / In Select-DR-scan state. Setup for Select-IR- /
 / scan state. /
 0 1 X 0 0 1 + / In Select-IR-scan. Setup for Capture-IR /
 / state /
 0 1 X 0 0 1 + / In Capture-IR-state. Setup for Shift-IR state. /
 / Shifts following instruction in. /
 0 1 X 1 0 1 + / In Shift-IR state. TDI = i0 (LSB) of /
 / instruction. /
 0 1 X 1 0 1 + / In Shift-IR state. TDI = i1 of instruction. /
 0 1 X 1 1 1 + / In Shift-IR state. Setup for Exit-IR state. TDI /
 / = i2 (MSB) of instruction. /
 0 1 X 0 1 1 + / In Exit-IR state. Setup for Update-IR /
 / state. /
 0 1 X 0 0 1 + / In Update-IR state. Loads BYPASS /
 / Instruction (111) into IR. Setup for /
 / Run/Idle state. /
 0 1 X 0 0 1 + / In Run/Idle state. /
 0 1 X 0 1 1 + / In R/I state. Setup for Select-DR-scan state. /
 0 1 X 0 0 1 + / In Select-DR-scan state. Setup for Capture- /
 / DR state. /



C L K	R S T	T D O	T D I	T M S	T R S	T C K	
0	1	X	0	0	1	+	/ In Capture-DR state. Setup for Shift-DR / / state. /
0	1	X	1	0	1	+	/ In Shift-DR state. TDI set up for next / / vector. /
0	1	H	0	0	1	+	/ In Shift-DR state. TDI logic 1 from / / previous vector now at TDO output. /
0	1	L	1	0	1	+	/ In Shift-DR state. TDI logic 0 from / / previous vector now at TDO output. /
0	1	H	0	0	1	+	/ In Shift-DR state. TDI logic 1 from / / previous vector now at TDO output. /
0	1	L	1	0	1	+	/ In Shift-DR state. TDI logic 0 from / / previous vector now at TDO output. /
0	1	H	0	1	1	+	/ In Shift-DR state. TDI logic 1 from / / previous vector now at TDO output. / / Setup for Exit1-DR state. /
0	1	X	0	1	1	+	/ In Exit1-DR state. Setup for Update- / DR state. /
0	1	X	0	0	1	+	/ In Update-DR state. Setup for Run/Idle / / state. /
0	1	X	0	0	1	+	/ In Run/Idle state. / /End of BYPASS instruction example. /

4

IDCODE INSTRUCTION EXAMPLE

The following is a generic vector example for an IDCODE test (note: an MPA1016 is being used as an example here so

the ID code that is output on TDO will be for a MPA1016—see Table 3):

C L K	R S T	T D O	T D I	T M S	T R S	T C K	
0	1	X	0	1	0	0	/ Non-JTAG state for tester setup (optional). /
0	1	X	0	1	1	+	/ Test-Logic-Reset (TLR) state. /
0	1	X	0	1	1	+	/ Test-Logic-Reset (TLR) state (number of / / loops in this state is up to the programmer). /
0	1	X	0	0	1	+	/ In TLR state. Setup for Run/Idle (R/I) state. / / (Number of loops in the Run/Idle state is up to / / the programmer) /
0	1	X	0	1	1	+	/ In R/I state. Setup for Select-DR-scan state /
0	1	X	0	1	1	+	/ In Select-DR-scan state. Setup for Select-IR- / / scan state. /
0	1	X	0	0	1	+	/ In Select-IR-scan. Setup for Capture-IR / / state /
0	1	X	0	0	1	+	/ In Capture-IR-state. Setup for Shift-IR state. / / Shifts following instruction in. /
0	1	X	0	0	1	+	/ In Shift-IR state. TDI = i0 (LSB) of / / instruction. /
0	1	X	0	0	1	+	/ In Shift-IR state. TDI = i1 of instruction. /
0	1	X	1	1	1	+	/ In Shift-IR state. Setup for Exit-IR state. TDI / / = i2 (MSB) of instruction. /



C L K	R S T	T D O	T D I	T M S	T R S	T C K	
0	1	X	0	1	1	+	/ In Exit-IR state. Setup for Update-IR / / state. /
0	1	X		0	1	+	/ In Update-IR state. Loads IDCODE / / Instruction (100) into IR. Setup for / / Run/Idle state. /
0	1	X	0	0	1	+	/ In Run/Idle state. /
0	1		0	1	1	+	/ IN R/I state. Setup Select-DR-scan state /
0	1	X	0	0	1	+	/ In Select-DR state. Setup Capture-DR state /
0	1	X	0	0	1	+	/ In Capture-DR state. Setup Shift-DR state /
0	1	H	0	0	1	+	/ Shift-DR state. Bit 0 (LSB) of ID code on TDO /
0	1	L	0	0	1	+	/ Shift-DR state. Bit 1 of ID code on TDO /
0		H	0	0	1	+	/ Shift-DR state. Bit 2 of ID code on TDO /
0	1	H	0	0	1	+	/ Shift-DR state. Bit 3 of ID code on TDO /

4

At this point the rest of the ID code, Bits 4–31, will be output in the same way (the ID code for the MPA1016 is 32 bits long). Per Table 3, the ID code for an MPA1016 is 0001 001110 0100001110 00000001 1101 in binary or 139 0E 01D in hex. As shown above, the four least significant digits (HHLH or

1101) were output first starting with Bit 0, then Bit 1, then Bit 2, and finally Bit 3. The next part of this example will show the end of the example which is the output of the four most significant bits of the ID code (Bits 28–31, which are 0001) and the return of the TAP Controller to the Run/Idle state.

C L K	R S T	T D O	T D I	T M S	T R S	T C K	
0	1	H	0	0	1	+	/ Shift-DR state. Bit 28 of ID code on TDO /
0	1	L	0	0	1	+	/ Shift-DR state. Bit 29 of ID code on TDO /
0	1	L	0	0	1	+	/ Shift-DR state. Bit 30 of ID code on TDO /
0	1	L	0	1	1	+	/ In Shift-DR state. Setup Exit-DR state. Bit 31 / / (MSB) of ID code on TDO /
0	1	X	0	1	1	+	/ In Exit1-DR state. Setup Update-DR / / state /
0	1	X	0	0	1	+	/ In Update-DR state. Setup Run/Idle / / state /
0	1	X	0	0	1	+	/ In Run/Idle state / / End of IDCODE instruction example /



EXTEST INSTRUCTION EXAMPLE

The following is a generic vector example for a simple EXTEST test. Note: an MPA1016 128-pin QFP is being used as an example here in order to have access to all of the

internal I/O pads from the external I/O pins. Also a reminder that in order to get data loaded out from the scan chain the desired output I/O pins must be configured as outputs before running the EXTEST test:

C L K	R S T	T D O	T D I	T M S	T R S	T C K	
0	1	X	0	1	0	0	/ Non-JTAG state for tester setup (optional). /
0	1	X	0	1	1	+	/ Test-Logic-Reset (TLR) state. /
0	1	X	0	1	1	+	/ Test-Logic-Reset (TLR) state (number of / / loops in this state is up to the programmer)./
0	1	X	0	0	1	+	/ In TLR state. Setup for Run/Idle (R/I) state. / / (Number of loops in the Run/Idle state is up to / / the programmer) /
0	1	X	0	1	1	+	/ In R/I state. Setup for Select-DR-scan state /
0	1	X	0	1	1	+	/ In Select-DR-scan state. Setup for Select-IR- / / scan state. /
0	1	X	0	0	1	+	/ In Select-IR-scan. Setup for Capture-IR / / state /
0	1	X	0	0	1	+	/ In Capture-IR-state. Setup for Shift-IR state. / / Shifts following instruction in. /
0	1	X	0	0	1	+	/ In Shift-IR state. TDI = i0 (LSB) of / / instruction. /
0	1	X	0	0	1	+	/ In Shift-IR state. TDI = i1 of instruction. /
0	1	X	0	1	1	+	/ In Shift-IR state. Setup for Exit-IR state. TDI / / = i2 (MSB) of instruction. /
0	1	X	0	1	1	+	/ In Exit-IR state. Setup for Update-IR / / state. /
0	1	X	0	0	1	+	/ In Update-IR state. Loads EXTEST / / Instruction (000) into IR. Setup for / / Run/Idle state. /
0	1	X	0	0	1	+	/ In Run/Idle state. /
0	1	X	0	1	1	+	/ IN R/I state. Setup Select-DR-scan state /
0	1	X	0	0	1	+	/ In Select-DR state. Setup Capture-DR state /
0	1	X	0	0	1	+	/ In Capture-DR state. Setup Shift-DR state /

4

At this time all 1s will be shifted into the scan chain and then parallel loaded out the configured output I/O pins and output control signals

0	1	X	1	0	1	+	/ Shift-DR state. 1 shifted into TDI. TCK 1. /
0	1	X	1	0	1	+	/ Shift-DR state. 1 shifted into TDI. TCK 2. /
0	1	X	1	0	1	+	/ Shift-DR state. 1 shifted into TDI. TCK 3. /
0	1	X	1	0	1	+	/ Shift-DR state. 1 shifted into TDI. TCK 4. /



To determine the number of TCK cycles we take the total number of boundary scan cells in the MPA1016 which can be equated to 80 I/O signals – 5 TAP signals + 11 control signals × 2 cells per BSR = 172 total BSRs (See Boundary Scan

Register section). In order to clock all Ones (1) through the scan chain TCK will need to be clocked 172 times in order to see all 1s coming out of TDO. So in the next vector we will assume that we have clocked TCK 164 times so far.

C	R	T	T	T	T	T	
L	S	D	D	M	R	C	
K	T	O		S	S	K	
0	1	X	1	0	1	+	/ Shift-DR state. Have already shifted TCK 164 / / times. TCK 168. /
0	1	X	1	0	1	+	/ Shift-DR state. 1 shifted into TDI. TCK 169. /
0	1	X	1	0	1	+	/ Shift-DR state. 1 shifted into TDI. TCK 170. /
0	1	X	1	0	1	+	/ Shift-DR state. 1 shifted into TDI. TCK 171. /
0	1	H	1	0	1	+	/ Shift-DR state. H shifted out TDO. TCK 172. /
0	1	H	1	0	1	+	/ Shift-DR state. H shifted out TDO. TCK 173. /
0	1	H	1	0	1	+	/ Shift-DR state. H shifted out TDO. TCK 174. /
0	1	H	1	0	1	+	/ Shift-DR state. H shifted out TDO. TCK 175. /

4

Now that the scan chain is filled with 1s the user can output the 1s from the scan chain out to the I/O pins and the control signal pins that are configured as outputs (also, remember

that the I/O control cell portion of each BSR must have a 1 latched into it in order to enable the output buffer from the BSR to the I/O pin).

C	R	T	T	T	T	T	
L	S	D	D	M	R	C	
K	T	O	I	S	S	K	
0	1	H	1	1	1	+	/ Shift-DR state. Setup Exit1-DR. /
0	1	X	1	1	1	+	/ Exit1-DR state. Setup Update-DR. /
0	1	X	1	0	1	+	/ Update-DR state. Setup R/I state. /

Upon entering the Update-DR state all of the 1s that were shifted into the scan chain, through TDI, are output on the configured I/O pins as Highs (H) as well as on the control pins

that are currently set as outputs (depending on the configuration mode: e.g. F1, F2, and F4 if in BFR).

0	1	X	1	0	1	+	/ In R/I state. End of EXTEST example /
---	---	---	---	---	---	---	---

SAMPLE INSTRUCTION EXAMPLE

The following is a generic vector example for a simple SAMPLE test. Note: an MPA1016 128-pin QFP is being used as an example here in order to have access to all of the internal I/O pads from the external I/O pins. Again, it is strongly recommended that the reader refer to the MPA1000 pin assignments page for the MPA1016 in order to follow along. This is found in the MPA Data Book (DL201/D).

The BSCs for the control pins CLK, RESETB, MODE[3:0], and F[4:0] are scan only cells. They should never be changed by the user. Because of this the SAMPLE test is the only way to test these pins, as a group, in JTAG mode (e.g. you can not

run the output portion of EXTEST on the control pins that are set as inputs due to a configuration mode. This is the case for F[0] and F[3] in BFR mode. See MPA data book (DL201/D), Device Configuration section for more details).

Also a reminder that all user I/O pins that are to be sampled must be configured as inputs, outputs, or bi-directionals before running the SAMPLE test. During the SAMPLE test the I/O control cell portion of the BSC no longer controls, but monitors the configuration of the pin to show whether the configuration is that of an input or an output. The pins that are configured as inputs will read a Low (L), on TDO, from the BSC's I/O control cell and the pins that are configured as



outputs will read a High (H), on TDO, from the BSC's I/O control cell.

In this example, the right side of the MPA1016 128-pin QFP is configured with inputs (roughly I/O and control pins in the

range of pins 67–75) and the left side of the MPA1016 128-pin QFP is configured with outputs (roughly I/O and control pins in the range of pins 4–31).

C L K	R S T	T D O	T D I	T M S	T R S	T C K	
0	1	X	0	1	0	0	/ Non-JTAG state for tester setup (optional). /
0	1	X	0	1	1	+	/ Test-Logic-Reset (TLR) state. /
0	1	X	0	1	1	+	/ Test-Logic-Reset (TLR) state (number of / / loops in this state is up to the programmer)/
0	1	X	0	0	1	+	/ In TLR state. Setup for Run/Idle (R/I) state. / / (Number of loops in the Run/Idle state is up to / / the programmer) /
0	1	X	0	1	1	+	/ In R/I state. Setup for Select-DR-scan state /
0	1	X	0	1	1	+	/ In Select-DR-scan state. Setup for Select-IR- / / scan state. /
0	1	X	0	0	1	+	/ In Select-IR-scan. Setup for Capture-IR / / state /
0	1	X	0	0	1	+	/ In Capture-IR-state. Setup for Shift-IR state. / / Shifts following instruction in. /
0	1	X	0	0	1	+	/ In Shift-IR state. TDI = i0 (LSB) of / / instruction. /
0	1	X	1	0	1	+	/ In Shift-IR state. TDI = i1 of instruction. /
0	1	X	0	1	1	+	/ In Shift-IR state. Setup for Exit-IR state. TDI / / = i2 (MSB) of instruction. /
0	1	X	0	1	1	+	/ In Exit-IR state. Setup for Update-IR / / state. /
0	1	X	0	0	1	+	/ In Update-IR state. Loads / SAMPLE Instruction (010) into IR. / Setup for Run/Idle state./
0	1	X	0	0	1	+	/ In Run/Idle state. /

All Zeros (0) will be shifted into the scan chain.

C L K	R S T	T D O	T D I	T M S	T R S	T C K	
0	1	X	0	1	1	+	/ In R/I state. Setup Select-DR-scan state /
0	1	X	0	0	1	+	/ In Select-DR state. Setup Capture-DR state /
0	1	X	0	0	1	+	/ In Capture-DR state. Setup Shift-DR state /
0	1	X	0	0	1	+	/ Shift-DR state. TCK 1. /
0	1	X	0	0	1	+	/ Shift-DR state. TCK 2. /
0	1	X	0	0	1	+	/ Shift-DR state. TCK 3. /
0	1	X	0	0	1	+	/ Shift-DR state. TCK 4. /

4



AN1618

To determine the number of TCK cycles we take the total number of boundary scan cells in the MPA1016 which can be equated to 80 I/O signals – 5 TAP signals + 11 control signals × 2 cells per BSR = 172 total BSRs (See Boundary Scan Register section). In order to clock all the Zeros (or lows)

through the scan chain TCK will need to be clocked 172 times in order to see all Lows (L) coming out of TDO. So in the next vector we will assume that we have clocked TCK 164 times so far.

C	R	T	T	T	T	T	
L	S	D	D	M	R	C	
K	T	O	I	S	S	K	
0	1	X	0	0	1	+	/ Shift-DR state. Have already shifted TCK 164 /
							/ times. TCK 168. /
0	1	X	0	0	1	+	/ Shift-DR state. TCK 169. /
0	1	X	0	0	1	+	/ Shift-DR state. TCK 170. /
0	1	X	0	0	1	+	/ Shift-DR state. TCK 171. /
0	1	L	0	0	1	+	/ Shift-DR state. L shifted out TDO. TCK 172. /
0	1	L	0	0	1	+	/ Shift-DR state. L shifted out TDO. TCK 173. /
0	1	L	0	0	1	+	/ Shift-DR state. L shifted out TDO. TCK 174. /
0	1	L	0	0	1	+	/ Shift-DR state. L shifted out TDO. TCK 175. /
0	1	L	1	1	1	+	/ Shift-DR state. Setup Exit1-DR. /
0	1	X	1	1	1	+	/ Exit1-DR state. Setup Update-DR. /
0	1	X	1	0	1	+	/ Update-DR state. Setup R/I state. /
0	1	X	1	0	1	+	/ In R/I state. /

4

Now that the scan chain is filled with the Lows (L), the user can now go back through the data path of the TAP controller and sample the I/O and control pins. Also, a reminder is that when you SAMPLE you will sample whatever state is on the MODE [3:0], F [4:0], RESETB, and CLK pins and those states will be sampled into the scan chain (i.e. they will not

necessarily be the Highs that will be sampled in the following part of this example).

By this point all Ones (1) should be put on the input pins and Highs (H) should be put on the output pins in order to SAMPLE them. They will be sampled in the Capture-DR state.

C	R	T	T	T	T	T	
L	S	D	D	M	R	C	
K	T	O	I	S	S	K	
0	1	X	0	1	1	+	/ In R/I state. Setup Select-DR-scan state /
0	1	X	0	0	1	+	/ In Select-DR state. Setup Capture-DR state /
0	1	X	0	0	1	+	/ In Capture-DR state. Setup Shift-DR state /



At this time all the Ones (1) on the input pins (right side of device) will be sampled into the scan chain. They will be read out as Highs (H) on TDO. Note: remember that the pins that

are configured as inputs will read a Low (L), on TDO, from the BSC's I/O control cell.

C	R	T	T	T	T	T	T
L	S	D	D	M	R	C	
K	T	O	I	S	S	S	K
0	1	X	0	1	1	+	/ In Shift-DR state. Then clock for shift out/
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (input) for the BSC / associated with pin 75 (first out to TDO)./ / (See Fig. 5)/
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 75. / / (See Figure 5) /
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (input) for the BSC / associated with pin 74 (See Fig. 5)/
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 74. / (See Figure 5) /

4

The same pattern continues through all of the I/O inputs on the right side of the MPA1016 128-pin QFP. Remember that the CLK and Mode pins 3 and 2 are on the right side of this

package and will also show how they were configured from their respective BSC I/O control cells as well as what logic state was on the pin when it was sampled.

C	R	T	T	T	T	T	T
L	S	D	D	M	R	C	
K	T	O	I	S	S	S	K
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (input) for the BSC / associated with pin 69 (CLK)./ / (See Fig. 5)/
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the Zero (0) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 69. / / (CLK is being held Low throughout these/ / examples) /
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (input) for the BSC / associated with pin 68 (MODE [3])/
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 68. / (MODE [3] is held High to keep the / / device in JTAG mode throughout these / / examples. See Table 1.) /



C	R	T	T	T	T	T	
L	S	D	D	M	R	T	
K	T	O	I	S	S	K	
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (input) for the BSC / associated with pin 67 (MODE [2])/
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the Zero (0) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 67. / / (MODE [2] is held Low because no / / external clock was used in BFR 2 config / / mode (Boot From ROM, serial data) when / / the device was configured. See Table 1.) /

Now that we have gone through the right side (I/O configured as inputs) of the device, we will skip the bottom side of the device (which we did not configure for this example) and go to the left side (I/O configured as outputs) of the device. Starting with F[3] pin 28, remember that the control pins are

not configurable by the user. They are configured depending on the configuration mode that the device is set in. Currently the device is in BFR 2 mode so F[3] is an input and F[4] is an output (See MPA Data Book, DL201/D).

4

C	R	T	T	T	T	T	
L	S	D	D	M	R	T	
K	T	O	I	S	S	K	
0	1	L	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (input) for the BSC / associated with pin 28, F[3]./ / (See Fig. 5)/
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 28. / / F[3] is being held High throughout these/ / examples) /
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (output) for the BSC / associated with pin 26, F[4] /
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 26. / F[4] is held High throughout these / examples. /



Finally, back to the I/O pins which the user configured as outputs.

C	R	T	T	T	T	T	
L	S	D	D	M	R	C	
K	T	O	I	S	S	K	
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (output) for the BSC / associated with pin 25, user I/O pin
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 25. /
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (output) for the BSC / associated with pin 24, user I/O pin
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 24. /

And through all of the user I/O pins until we reach the last of the user I/O pins on the left side of the device.

4

C	R	T	T	T	T	T	
L	S	D	D	M	R	C	
K	T	O	I	S	S	K	
0	1	H	0	0	1	+	/ In Shift-DR state. TDO is the state of the / / I/O control cell (output) for the BSC / associated with pin 4, user I/O pin
0	1	H	0	1	1	+	/ In Shift-DR state. TDO is the One (1) / / state that was loaded in to the data control/ / cell of the BSC associated with pin 4. /
							/ Setup Exit1-DR state. /
0	1	X	1	1	1	+	/ Exit1-DR state. Setup Update-DR. /
0	1	X	1	0	1	+	/ Update-DR state. Setup R/I state. /
0	1	X	1	0	1	+	/ In R/I state. End of SAMPLE example /



MPA1000 Primer for Schematic Designers

Prepared by
Doug M. Shade
Motorola Programmable Logic

4



MPA1000 Primer for Schematic Designers

Introduction

This application note is designed to assist novice Motorola Programmable Array (MPA) users who employ schematics as their preferred design entry method. While the details of the MPA architecture can largely be ignored for most of your designs, achieving the most efficient packing of logic and highest performance marks are facilitated by a better understanding of the MPA internals presented here. You'll find the MPA architecture accommodating to the most popular design topologies, but biased to favor those designs that use fewer non-gated clocks.

The reader is assumed to be familiar with the various controls and options of the MPA Design System.

Internal Array Features

Cells in Tiles in Zones in Quadrants

At the lowest level, the MPA is built of very simple logic building blocks called cells. Each of the four types of cells contain a NAND function and secondary function. The four cell types grouped together form a tile. The cell type numbers (1-4) signify which secondary function is contained and the relative location of the cell within the tile.

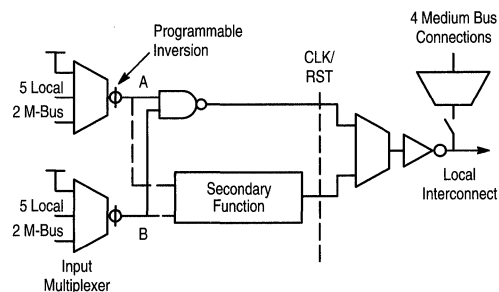


Figure 1. Core Cell Structure

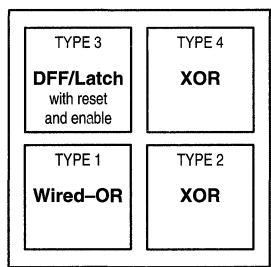


Figure 2. Core Cell Secondary Functions

The tiles are assembled in a 5 x 5 array to form the interior of a zone. The zones are bounded on the top by vertical port cells (routes entering and leaving through these ports run

vertical). The zones are bounded on the right by horizontal port cells. Lastly, the upper right corner of the zone contains the Clock/Reset selection cell (more on this special cell later).

One Hot State Machine Design is Preferred

Designing your state machines as one hot is usually the most efficient method for the register rich MPA.

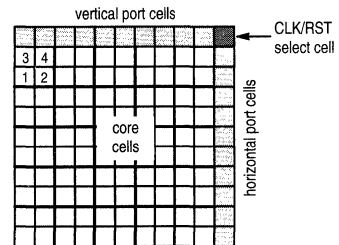


Figure 3. Zone Structure

The zones are arrayed into quadrants. The number of zones varies by MPA family member as shown in Figure 4.

4

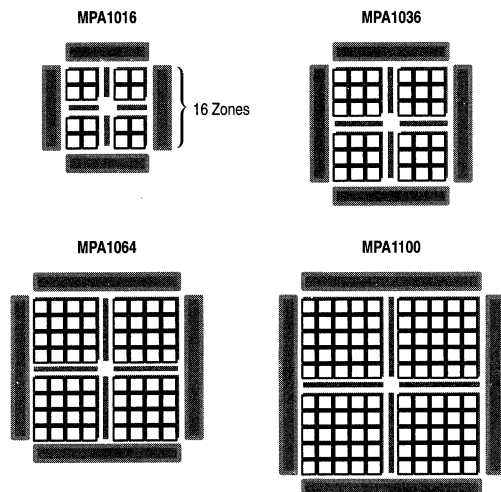


Figure 4. The MPA 1016, MPA1036, MPA1064 and MPA1100

Routing at Three Levels

The autolayout software has 3 levels of routing to choose from to complete connections between cells: Local, Medium and Global. Local routing connects the output of a cell to its 8 nearest orthogonal neighbors. Local routing is the fastest connection available between two points.



Medium routing is used to complete inter-zone routes that can not be made using local routing. Medium routes can also connect points in different zones, intra-zone connections.

Global routing spans entire quadrants and with the use of inter-quadrant switches can span the entire core array area. Global routing is the slowest of the 3 routing levels available to the autolayout software.

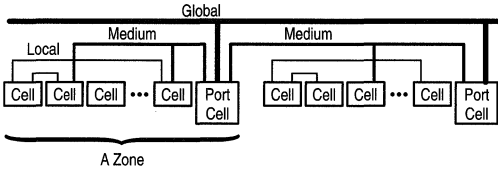


Figure 5. Three Levels of Routing Resources

A common question at this point is, “How do I control which of my nets goes on what level or routing?”. The simple answer is you can’t directly control this. The timing driven autolayout software is designed to make these difficult decisions for you. In the end, all you really need to specify is how fast you want your design to go and let the autolayout software take it from there. There are however things you can do to influence the software. Attributes such as DPLD_CLUSTER_SEED and DPLD_PLACE_PRIORITY can be used to bias the placement and subsequent routing for specific portions of you design. Please reference the on-line help of the MPA Design System software for more detail on these attributes.

4

Autolayout is a Pseudo-Random Process

With all the options available to the autolayout tool to complete the place and route of your design, you’d be correct in guessing that you can get significantly different results with relatively minor changes in design or in the pseudo-random seed. This can be a disquieting characteristic if you discover it on your own, however it can really be quite advantageous. If your autolayout result just missed the desired target, or you’d like a little more margin, simply re-seeding the pseudo-random number generator (from the “Tools → Options” pull down menu) and re-running autolayout can give you a significantly different result.

The routing within the MPA is fully buffered. There is never a need to concern yourself with the loading effects of fan out. This of course is a departure from ASIC and PCB level design where fan out must be considered.

BUFF from the MICROLIB

There are quite a few trivial optimizations that get made to your design during import, one of the most common is getting rid of superfluous ‘buffers’. Examples of which include INV (inverters), AN2 gates with both inputs tied together, AN2 gates with one input tied high etc. (A more complete description of this re-mapping process can be found in the on-line help of the MPA Design System under: “Help on Design → Logic

Optimisation → Summary of Optimisations”) Don’t panic at the above statement that inverters are “gotten rid of”. They are simply mapped to the correct sense on the core cell’s programmable input multiplexers.

No delay penalty is incurred for inserting an INV. BUFF is the only buffer available to the designer that will not get mapped out on import. So with all this effort to get rid of buffers, and no need to worry about fanout loading effects, what is the BUFF library element used for?

- Connecting a primary clock or reset signal directly to an output pad is not allowed. Tapping the clock network with a BUFF and then connecting the output of BUFF to an output macro however works fine.
- BUFF can be used to build a delay line. It should be noted however that asynchronous design is strongly discouraged. Minimum delays are not specified. Also, hard macros are not accommodated by MPADS and as such the delay through a string of BUFFs can vary considerably from autolayout run to run.
- BUFF can be used to break a single large net into two (or more) smaller ones. This has been employed to isolate a speed critical portion of a net, by inserting a buff to feed the non-speed critical portion of the net and assigning that portion of the net the attribute DPLD_IGNORE_TIMING.
- Another example where splitting a net may be advantageous is using BUFF to break a secondary clock net into two tertiary clock nets. (More on clock networks latter.)

Tri-State drivers are not available internally

Because the MPA’s routing resources are fully buffered (actively driven) there are no internal tri-state buffers available. Designers accustomed to using such elements to allow multiple drivers access to a single data line, should instead consider using multiplexers.

Wired-Or a.k.a. Open Drain

In some instances, it may be preferable to use a collection of open drain drivers to drive a single data line. The MPA library elements that accommodate this type of connection include: WINV, WOR2, WND2, and WBUF. It is important to remember that open drain drivers can only actively pull a signal low, a passive pull up resistor is required to pull the net high; that’s the job of the WPUP library element. By default, instantiating a WPUP element results in a single pull resistor being attached to the net. Assigning the attribute DPLD_PUP with a value of BOTH results in two pull up resistors being added in parallel to the net. The low to high transition time is thus improved, but at the expense of more static current drain when any of the attached drivers is holding the net low.



Besides lower speed, another draw back of using open drain drivers in the MPA is the restriction that all the open drain drivers within a zone must reside on the same Wired-OR Bus, and that drivers in other zones must also be placed in the same relative horizontal position. The autolayout tool handles all of this automatically, but it does tend to reduce the number of valid solutions available to the autolayout tool for the remainder of your design.

Wide Inputs Add Delays

Although ASICs are sometimes referred to as “sea-of-gates” devices, they are in reality sea-of-transistors devices. If a designer specified an 3 input AND gate, an ASIC compiler will simply build one using the available transistors. The resulting gate delay for an n-input ASIC AND gate is very close to that of a two input ASIC AND gate. Remember that the MPA architecture is largely constructed of two input logic functions. Multiple input gates are available to you in the MACROLIB, but keep in mind that they are constructed from cascading multiple two input gates. The higher the number of inputs, the more levels of logic your signal will have to pass through slowing it down. It may be worth while to remember that not all logic paths accumulate the same delay through a wide input. Figure 6 shows the descended hierarchy of the construction of an AN3 gate. It would be best to use the “C” input for the speed critical path as it has roughly half the delay of the “A” and “B” inputs.

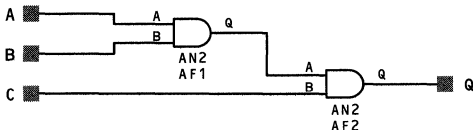


Figure 6. An AN3 Library Element Schematic

Delays in Routing

Both PCB and older ASIC designers share the mind set that delay through a multi-level logic path is principally a function of “gate delay”. In the ASIC world, routing paths are as short as possible and do not pass through multiple levels of pass gates, muxes, and buffers. Similarly, a PCB trace is a simple and hopefully short run of metal, with most of the “gate” delay happening as a function of package input and output delays. A “logical” net in an FPGA however may be a series of several different electrical nodes, each being separated by a mux or switch of some type. The consequence of this is that “routing delays” not gate delays are the first order factor determining the resultant circuit’s speed.

Empirical analysis of several hundred sample designs suggests that a multiplication factor of 2.4 can be applied to the sum of a path’s gate delays to come up with a very rough estimate of what the post autolayout total path delay might be. There are many factors that influence that actual number, so please consider this only as a very crude estimate.

S-R Flops, Avoid the Temptation

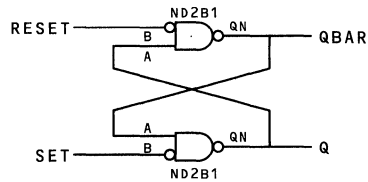


Figure 7. A Classic S-R Flop; An Accident Waiting to Happen

The above construction of an asynchronous S-R flip flop is familiar to all, but should be so for its unfavorable characteristics. Remember that routing delay in an FPGA is the highest order term in delay equation. In the above construction, the (active high) SET pulse width must be greater than the ND2B1 propagation delay plus Q to A routing plus another ND2B1 delay plus QBAR to A routing delay. Without a detailed analysis of the post autolayout path delays, the pulse width specification can not be known. The same holds for the RESET pulse width. A new autolayout run on the same design may alter these path lengths considerably. Additionally this sort of asynchronous feedback loop will generally cause back annotation, simulation and timing analysis tools trouble.

4

Again, avoid asynchronous design.

Delay Lines, Avoid the Temptation

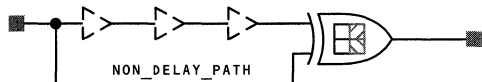


Figure 8. A Delay Line for Turning Edges into Pulses, a Dangerous Proposal

Remember that in an FPGA routing is not just a piece of wire. Routing is comprised of wire, muxes and pass gates. In the above example, the intent is to turn a rising or falling input edge into an output pulse. The assumption is that the “NON_DELAY_PATH” will have a shorter delay than the “delay line” formed by the series of BUFF elements. Again, the MPA design software does not guarantee minimum delays and so it is possible that the autolayout run might result in the



NON_DELAY_PATH to have a delay significantly close the delay line path. The circuit may not work.

Avoid any design habit that makes assumptions about minimum delays, even for just plain routes.

High Fan Out

As mentioned previously, the routing resources of the MPA are fully buffered. There is no reason for the designer to concern himself with loading effects of high fan out net. However, high fan out nets can have an undesirable impact on routing resource consumption. Using only local routing, a single driver could under the most ideal conditions drive only 8 local neighbors. In real world designs however, each of the destinations of a high fan out net has its own downstream circuitry associated with it; there is a vanishingly low probability that they will be placed in the 8 local adjacent locations. For fan outs greater than 8, exclusive local routing is impossible, and both medium and global routes will be used to complete the net. If the fan out is large enough, and the circuitry placed sufficiently far apart in the array, routing resource consumption may become problematic.

The primary clock and reset distribution network may be used to route high fan out signals. Driving the high fan out net internally with an ACKL or ARST buffer, or externally with an IPCLK or IPRST buffer will put the signal on one of the 8 global Clk/Rst distribution lines. The routing congestion can thus be solved, but at the expense of reducing the clock and reset routing solution space. Do not route nets to I/O (other than Clk/Rst) on the primary clock network. There is no mechanism for completing such a route on the MPA devices.

For software versions 2.4 and later, ACKL and ARST insertions for high fanout nets will be automatic.

You Must Use All Macro Inputs, ONE and ZERO

The autolayout tool insists that all MACROLIB and MICROLIB inputs be used. If you don't need a particular input for your design, you are still required to tie it to logic or a ONE or a ZERO (from the MICROLIB). There is no routing consumed when specifying a ONE or a ZERO, the tie off is made at the cell's input selection mux. There is no fan out restriction for a ONE or a ZERO.

Clock Architecture

The MPA architecture is biased toward efficient accommodation of synchronous design topologies, though it has flexibility enough to handle most design styles. One of the most specialized architectural features of the MPA is its low skew clock and reset distribution network. The Primary Clock(/Reset) Distribution Network accepts inputs from two specialized I/O locations on each edge of the die, for a total of 8 inputs. Access to these special inputs is accomplished with IPCLK and IPRST buffers, when bringing an external signal in. If you want to place an internally generated signal on the Primary Clock Distribution Network, you may insert an ACKL or ARST buffer.

ACKL & ARST Consume Clk/Rst I/O Sites

Each ACKL and ARST buffer used resides in one of the 8 clock pad locations. Using an ACKL or ARST consumes this pad location such that it is no longer available to use as an I/O site. The designer is allowed a total of 8 ACKL, ARST, IPCLK, IPRST cells in their design.

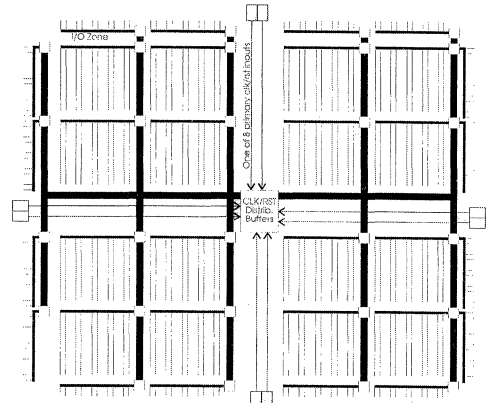


Figure 9. Overview of Primary Clock/Reset Distribution Network

Figure 9 depicts an overview of the primary Clock Distribution Network of an MPA1016. Two specialized Clk/Rst input pads are shown on each edge of the die feeding the centralized Clk/Rst buffers. From there, the heaviest lines represent the 8 lines of the primary Clock Distribution Network. The medium lines represent 2 Clk/Rst pairs. The lightest lines each represent a single Clk or Rst signal. Skew across the entire die is held to less than 1ns on the primary Clock Distribution Network.



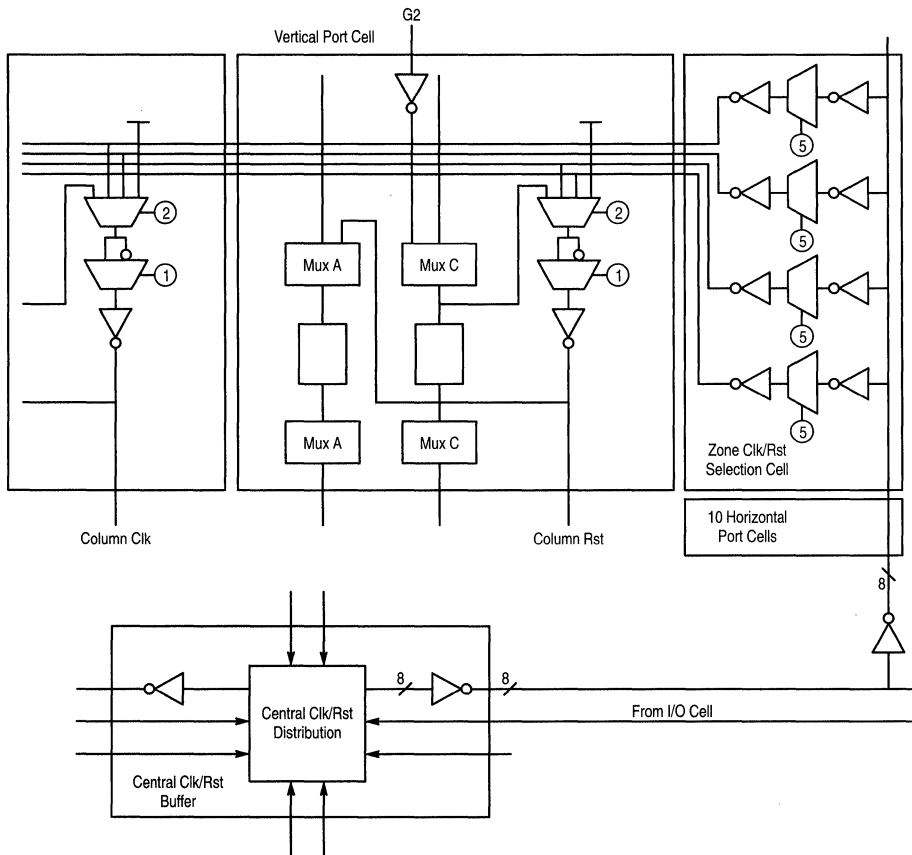


Figure 10. Upper Right-Hand Corner of a Zone, Detail of Primary Clock/Reset Selection Cell

Figure 10 shows the upper right-hand corner of a zone, with more detail visible on how clocks are routed into a zone. From this figure you should note that no more than two primary clocks and two primary resets may be routed into a single zone. There are however provisions for non-primary clock and reset routes into a zone.

Note— Most of the following discussion refers to *clock* and *flops*. It should be noted that this is short hand where flops really refers to flops and latches, and clocks really refers to clocks, resets and latch enables.

There are 10 columns in a zone, half of which each contain 5 Type 3 cells for a total of 25 flops. Each column's flops share common clock and reset lines sourced from the pair of vertical port cells immediately above and just to the right of the column.

Clocking at Three Levels

Primary Clocks

While the 8 lines of the primary Clock Distribution Network, and the fact that no more than two primary Clk/Rst pairs may be routed into a zone at a time may seem a bit limiting at first, there are other ways to get clocks and resets into zone logic.

⚡ Avoiding Layout Spread

The autolayout tool examines the imported netlist, examines all primary clock and reset nets and proceeds with a design partition that assumes clocks and resets are for the most part paired. Logic associated with each Clk/Rst pair is placed into a zone fed by that pair. One primary clock net feeding a set of flops with multiple primary resets (or vice-versa) will result in the flops getting spread



out over multiple zones when they might otherwise have fit into a single zone.

I/O Cells Can Only Be Clocked From the Primary Clock Distribution Network

Clocking I/O macros via secondary or tertiary clocks is prohibited. Reset is however permitted to be sourced from the array or Peripheral Bus (P-Bus).

Again referring to Figure 10, you'll note that the sense of the clock fed to the column of flops is programmed in the column's vertical port cell. In a two phase design, it is not necessary to route both senses of a clock on the primary Clock Distribution Network; instead simply place an INV in front of clock input(s) you wish to run off the inverted version of the clock. The autolayout tool will do the rest of the job for you.

Clock Sense Selection is Made in the Vertical Port Cell

All flops within a column will have the same clock and reset (or will lie unused).

4

Do Not Use the Primary Clk/Rst Distribution Network to Route Clock Enable Signals

Referring to Figure 10, note that a clock is paired with a reset and brought down to all 5 of the Type 3 cells within a column. If the associated clock enable (if used) is also on the primary clock network, there would be no efficient route available to get it down to the target flops. Do not use the Primary Clock Distribution Network to route clock enables. (Do use it for "Latch Enable" signals.)

Secondary Clocks

The autolayout software recognizes any network not on the primary Clock Distribution Network with 5 or more Clk/Rst loads on it as a "secondary clock (reset)" net. In order to continue to guarantee low clock skew for even these secondary networks, the autolayout software will construct a secondary clock tree. Skew is held below $2nS$ with a secondary clock tree. Construction of a secondary clock tree is accomplished automatically with the use of horizontal global bus switching to vertical global buses (G2 lines) as necessary to connect to the clocked logic. Figure 10 shows the G2 line as this secondary vertical route into the Clk/Rst resources of the flops within a zone column.

Secondary Clock Networks Consume Routing Resources

The MPA architecture easily handles a fair number of secondary clock networks, but networks with large numbers of Clk/Rst loads are more efficiently accommodated by moving onto the primary Clock Distribution Network using ACLK or ARST buffers mentioned previously. Version 2.4 will do this automatically as an import option.

Avoid Assigning Normal I/O to Clk/Rst Pads

Using an ACLK or ARST consumes the internal circuitry of one of the 8 special Clk/Rst pads. If these pads have been previously consumed by (designer assigned) I/O signals, then they will not be available for ACLK or ARST buffers.

From Left to Right Only

The software currently restricts input buffers to secondary clock nets to be on the left or right edge of the die. This may change in latter revisions of the autolayout software.

Tertiary Clocks

The autolayout software recognizes any network not on the primary Clock Distribution Network with 1 to 4 Clk/Rst loads on it as a "tertiary" clock (reset) net. Skew is not controlled nor is it guaranteed on such networks. The unlabeled input into "Mux C" of Figure 10 actually represents several of the normal routing resources in the MPA including medium and global buses that could be used for tertiary clock routing.

Remember BUFF

BUFF may be inserted into the middle of a secondary clock net in order to break it up into two or more tertiary clock nets. Under rare circumstances, heavy usage of secondary clocks can cause routing congestion problems. It may be beneficial to break a few of these up into tertiaries.

Head Scratcher...

Suppose you have a design with three primary clocks (C1, C2 & C3) and two primary resets (R1 & R2). Suppose further that your design uses a single flop for each of the following combinations of clock and reset: C1R1, C1R2, C2R1, C2R2 and C3R1. Because each of the clock and reset inputs of the flops within a zone's columns are hardwired together, this design would require at least 5 columns worth of clock resources. Because all the clocks and resets were specified as 'primary', and since only two primary clocks and resets are allowed per zone, the design would be fit into not less than two zones. The fit of the design could be improved by moving C3 off the primary clock network. The now tertiary C3 could route into the fifth available column containing Type 3 cells within the single zone. Since the design only used a single flop per Clk/Rst pair, there will be 4 unused flops per column.

While the design is an obviously extreme and contrived example, it does demonstrate how some design styles may not fit efficiently into the MPA.

Synchronous Design

The MPA architecture is biased to accommodate synchronous designs. The autolayout neither guarantees nor reports "minimum" delays through any network so asynchronous design techniques are strongly discouraged. The "minimum" delays are not reported for two reasons, the



first is that the autolayout software algorithms are designed solely to beat maximum delay requirements, so the length of a route may vary considerably from autolayout run to run (while never exceeding the maximum number specified). The second reason to not report minimum delays is that with continuously improving process technologies, the trend is to make the MPA silicon ever faster. Any design that is constructed assuming some minimum delay can not guaranteed to work in the future.

☞ Skipping Clocks, Strongly Discouraged

In some cases you might be tempted to construct a “synchronous” circuit that employs the questionable practice of “skipping clocks”. In this (discouraged) design practice, the designer may have a deep combinatorial circuit that takes much longer than a single clock period to settle. The practice of skipping clocks forces the assumption that the combinatorial circuit takes more than one clock period and less than two clock periods to resolve. In this instance, the very first result is ignored, but from then on the results are assumed valid two clocks after the input changes.

While the design might be shown to work on a given unit, Motorola does not guarantee minimum delays so a subsequent process lot might leave the designer with silicon that goes “too fast” and a broken design. Also, subsequent autolayout runs might result in a significant reduction in a combinatorial path’s delay time also rendering the original assumption of “more than one clock period” invalid.

☞ Using Both Clock Edges

The autolayout software and architecture does allow the designer to use both edges of a clock, but as mentioned above the timing driven portion of the software only knows about what clock is being used, not which edge of the clock it has to meet timing for. If you’re going to use both edges of a 50% duty cycle clock when going from one flop to the next, you’ll have to halve your desired “Target Delay” in the Tools → Options → Autolayout panel. If your clock is not 50% duty cycle, you’ll have to set Target Delay to a value equal to the shorter of the two states of the clock.

☞ Too Many Clocks

The MPA is best suited for designs with a few primary clocks, but multiple clocks are supported. The problem with tertiary and especially secondary clock networks is that they consume a fair amount of routing resources. An otherwise easy to fit design may not be routable once multiple secondary clocks are accommodated by the software.

Further, because of the granularity of the clock architecture (1 Clk/Rst pair per zone column of 5 flops, 2 primary and 2 secondary clocks per zone),

multiple clocks in a design tends to limit how “small” or how tightly packed the resultant layout can be. Remember that of the three levels of routing resources available, local routing is the fastest. A design that gets “spread out” because it uses multiple Clk/Rst pairs can not be routed with the faster local routing resources and so may suffer some performance loss.

☞ Gated Clocks, Avoid When Possible

Inserting anything but an INV in a clock path will result in the clock being pulled off the primary clock network and placed either secondary or tertiary routing (depending on the number of clock loads downstream of the inserted gate). As mentioned above, this tends to spread the resulting layout out a bit more and consequently can slow things down some. If a gated clock is desired, try instead using register elements with clock enables.

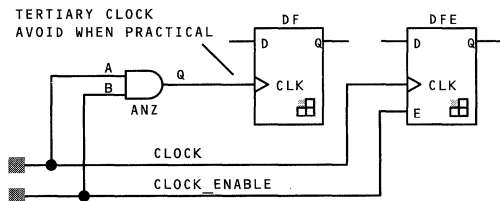


Figure 11. A Standard Register with a Gated Clock and the Preferred Clock Enabled Register

I/O and P-Bus Features

The MPA I/O cell is sometime referred to as “Complex I/O” and for good reason. The flexibility of the I/O Cell in terms of functionality and routing is enormous, but it comes at the cost of being a difficult structure to fully understand. A good place to begin is the on-line help “Help On Device → Functional Description → I/O Cell”. Bookmark it; you’ll be going there again. The Complex I/O is made even more flexible via its association with the Peripheral Bus (P-Bus). The P-Bus is an 8 bit wide bus that runs around the entire chip, broken at each of the chip’s four corners by programmable switches. The corner switch structure also contains pull-up resistors for each of the 32 P-Bus segments to support the optional open-drain driving of the P-Bus.

The P-Bus is intended to make routing of common signals from the interior of the MPA array to the I/O Cells more efficient. Such signals include tri-state enables, clock enables, resets, even data. Instead of asking the autolayout software to route a tri-state enable from the array to every T (enable) pin of bank of OPTBUFs, the user should employ the P-Bus by inserting an APBUF (from the MICROLIB) after the logic sourcing the enable signal. The autolayout software will place the enable signal from the array interior onto the P-Bus and again tap off as required by the I/Os; only a single route from array interior to I/O area will be required. Figure 12 shows a similar situation; in this case the open-drain APWBUF driver has been selected (for no good reason other than as an



example of how to use it). The P-Bus signal “ENABLE” can be driven by either of the APWBUFs to its low state. A P-Bus wired-or pull up (PWPUP) is required in order to ensure ENABLE can go to a valid high state if not being otherwise pulled low. The autolayout software handles all the issues of

assigning drive strength to the PWPUP and closing corner switches of the P-Bus signal to ensure that all of the OPTBUFs get the required ENABLE signal, no matter which chip edge they end up on after autolayout.

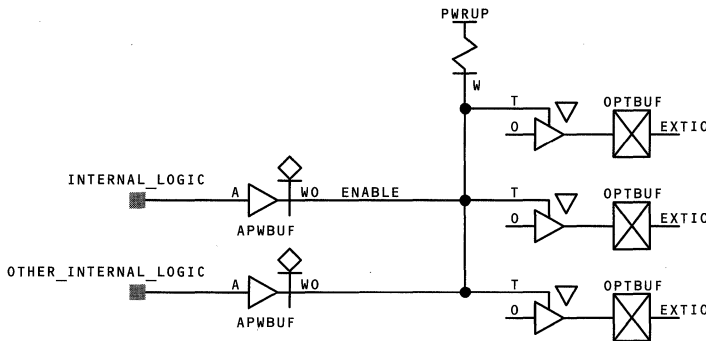


Figure 12. Using the P-Bus with Open-Drain Drivers

4

A more common example of using the P-Bus is to use a single APBUF to drive all the enables of a bank data bus transceivers.

Use the P-Bus to route enable signals

Whenever an enable signal goes to more than one I/O cell, it is recommended that the designer employ the P-Bus (by inserting an APBUF). Version 2.4 will do this automatically as an import option.

Start Off Easy, Begin with IPBUF, OPBUF, IPCLK, IPRST

The Complex I/O can be a space and a time saver for your more critical designs, but you may want to consider starting off slow and use the simpler I/O structures.

Enable and Reset Pins on Complex I/Os do not have to be tied

There are too many permutations possible in the I/O cell to make each available as a macrocell in the IOLIB. Consequently a short cut has been made available to the designer using Complex I/O, namely it is not necessary to tie reset or enable inputs high or low when using elements of the IOLIB. (N.B. This is not true for elements from the MICROLIB or MACROLIB. Each of these inputs must be used or otherwise tied off.) The autolayout software will make the obvious assumptions about how the unused input should be tied and make the tie off for you.

Twinning I/Os, Not Recommended but...

The MPA does provide open-drain output drivers that may be safely shorted together externally, but

only for the purposes of forming a wired-and, not for increasing total current sinking capacity. Shorting or ‘twinning’ outputs together is not a recommended method of increasing current source and sink capacity for a particular signal. The twinned outputs may not change state simultaneously and may as a result be in contention, putting undue stress on the device. If you still feel compelled to twin outputs there are some things you can do to mitigate the hazards.

Use clocked outputs and be sure they both use the same sense of the same clock. This will reduce the possible contention period down to less than 1nS (assuming the PCB trace length is negligible between the two outputs). If possible, place the two clocked outputs in the same I/O Zone. Clock skew within a single I/O Zone is down in the pico-seconds, further reducing the possible time in contention. (The Rev 1 version of the databook does not show I/O Zone information, this is being amended in subsequent revisions.)

Simultaneous Switching

The MPA was designed to accommodate no more than 16 simultaneously switching outputs each driving 6mA. Increasing the number of simultaneously switching outputs beyond this number may result in ground bounce troubles (though this is largely an effect of PCB design and power supply capacity and bypassing conditions as well).

One method to avoid such problems is to avoid putting wide output buses on clocked I/O. Avoiding clocked I/O will tend to spread out the range of clock-to-output arrival times, thus reducing the instantaneous current demand.



Another method available is to assign the attribute `DPLD_OPSLEW` with a value of `LOW` to the output port. The dt term of Ldi/dt gets larger, thus lessening the tendency for voltage rise above ground.

Bad Attribute Combinations

There are just a few combinations of I/O attributes that are not recommended:

Using `DPLD_OPSLEW=LOW` and `DPDL_OPDRIVE=4mA` on the same output. When an I/O cell is programmed for 4mA drive, only the first of the pair of output drivers is connected. Only the second of the pair (added in parallel when 6mA is required) has programmable slew rate control. Only 6mA outputs have the ability to modify slew rate.

`PULLUP=1` and `PULLDOWN=1` would have the (permitted, but discouraged) effect of enabling both the pull-up and pull-down resistors for a particular I/O. This would have the undesirable effect of causing static current drain for no good reason.

On a bi-directional I/O, assigning the attribute combination of `DPLD_IPLLEVEL=CMOS` and `DPLD_OPLEVEL=3V` can leave the input at a level close to threshold, possibly resulting in unwanted current.

Default I/O attributes are CMOS out but TTL in

The default attributes for the MPA I/Os are `DPLD_OPLEVEL=5V` and `DPLD_IPLLEVEL=TTL`; this ensures that without re-assigning attributes to

the MPA's I/Os, that the resulting design will be compatible with most logic signaling schemes, however it may leave you with a lower than optimal noise margin because of the lower threshold TTL inputs.

Don't fix your I/O locations unnecessarily

Fixing your I/O locations using `DPLD_PAD_PLACE` attribute may place an undue burden on the autolayout tool. Most designs will route to a higher performance level if the autolayout tool is given as much freedom as possible with regards to I/O pin placement.

Pull-Ups Not Intended for Open-Drain Buses

The pull-ups and pull-downs available on the I/O cells are best classified as "very weak". Their intended use to pull an otherwise floating input to a valid logic high or low. They are not intended to drive any external loads such as an open drain bus.

Summary

A little up front research into the MPA architecture specifics and the appropriate adjustments to your current design style can go a long way towards making your design fit fast and efficiently in our MPAs. We've presented here all the most popular design tips for working with the MPA. After reading this application note we're confident you'll find designing with the MPA to be a rewarding experience.

Don't forget help is never far away. Start with your MPA representative or our web site at:

<http://sps.motorola.com/fpga>

Thank you for selecting MPA.

4



HDL Techniques for Faster Synthesized Counters

Prepared by
Marc Greenberg
Motorola Programmable Technology Centre

4

This application note contains information on how HDL coding style can affect the size and speed of a design. By describing the synthesised structure in the HDL code, results can be achieved that rival schematic capture in a lot less time.



HDL Techniques for Faster Synthesized Counters

Introduction

Synthesis tools and Hardware Description Languages (HDLs) provide designers with a great number of benefits compared with schematic capture tools. Designers using HDLs are often more productive than they are with schematic capture. HDL designs are much more portable, easier to document, easier to partition, easier to modify and easier to prototype than their schematic capture counterparts. But schematic capture designs have two important advantages over HDL designs: size and speed. This report shows how to use HDL and still achieve circuit size and speed on a par with schematic capture.

Counters, timers and frequency dividers are often on the critical path in a design. By improving the speed of these elements the performance of the entire design can be improved, and this could make the difference between using a faster speed grade of device or not. While the discussion centres around counters, the principles involved can be applied to many other circuit elements.

This application note discusses methods of building 16-bit counters, both in schematic capture and through the synthesis of Hardware Description Languages (HDLs). The benefits of different counter architectures and HDL coding styles are explored.

VHDL has been used in this study, however Verilog users should be able to apply the same techniques to their code. The Synario synthesis tool was used in this study. The findings may differ slightly for users of other synthesis tools but the principles involved are the same.

The VHDL written for this application note is intended for Motorola MPA FPGAs, but it is not specific to MPA FPGAs.

No MPA components are instantiated, so it should be possible to re-synthesise the VHDL and retarget it to any FPGA or ASIC without modification. The schematics have been designed with Synario Generic library elements, so the schematics can also be re-used in any Synario device kit that supports the Synario Generic library. This demonstrates that it is possible to retain portability in an HDL design without sacrificing too much circuit performance.

All the test cases have been placed and routed using the fully automatic place-and-route in the MPA Design System (MPA DS). Speed information is derived from static timing information produced by MPA DS.

The Basics

All the counters in the study are synchronous, 16 bits wide with clock enable and asynchronous reset. Counters with and without data load inputs are explored. Three main counter architectures are used: the very simple ripple counter, the look ahead counter, and the prescaled counter. Counters are described using schematic capture, behavioral VHDL or structural VHDL.

4

Schematic Capture

The simple ripple counter is probably the most familiar. A section of ripple counter is reproduced in Figure 1. The simple ripple counter has the advantage that it is the simplest possible synchronous counter and so it is smaller than other counters. It is also one of the slowest architectures, as the critical path 'ripples' through the counter, resulting in one gate delay per bit width of the counter.

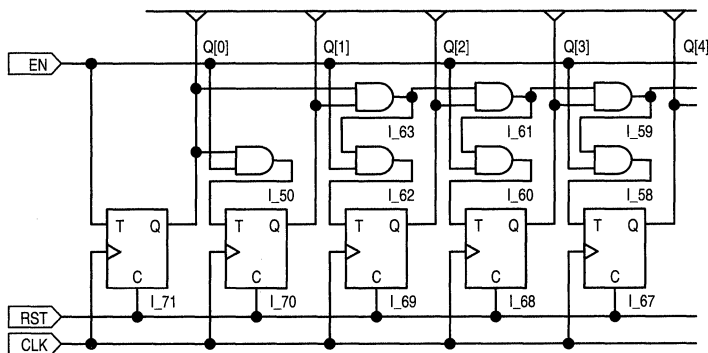


Figure 1. The First Few Stages of the Simple Ripple Counter



The look ahead counter is a variation of the ripple counter. The look ahead counter breaks up the ripple carry path by precalculating what the ripple carry value should be at certain points in the counter and injecting this value into the ripple carry path at these points. For 16 bit counters in MPA

devices this works out to be at the 8th, 11th and 14th stages in the counter. In the schematic designs this resulted in a 40% speed improvement over the simple ripple counter. Figure 2 shows the ripple carry counter.

4

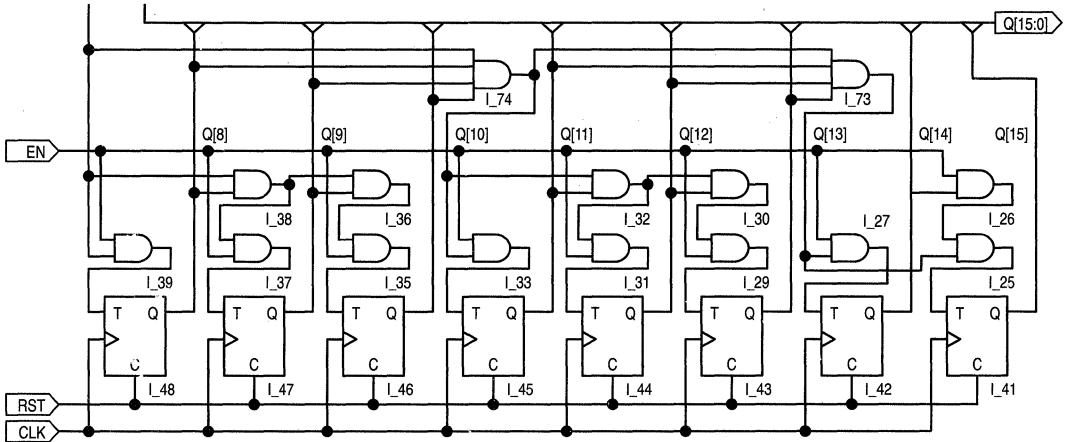


Figure 2. The Look Ahead Carry Counter, Showing the 11th and 14th Stage Look Aheads

The prescaled counter is another variation on the ripple carry counter. The two least significant bits of the counter form their own counter which is capable of operating at high speed. Once every 4 clock ticks, the clock enable for the 14 most significant bits in the counter is enabled, thus the ripple carry path has four clock ticks to complete the ripple carry calculation. Note that the static timing analysis in the MPA Design System does not take circuit functionality into

account, so it reports a speed figure that is lower than the true figure possible. Circuit analysis using the timing path data provided by MPA DS shows a speed improvement of over 50% compared with the ripple carry counter. Prescaled counters are not suitable for implementation in counters with data load. Figure 3 shows the first few stages of the prescaled counter.

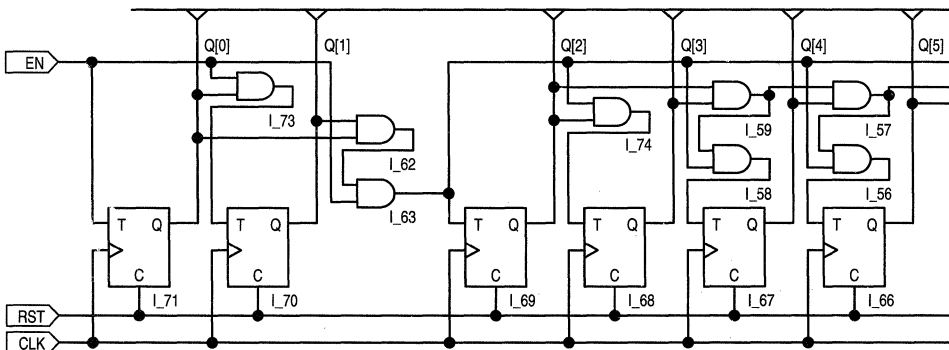


Figure 3. The first few stages of the prescaled counter.



Synthesis

The synthesis code that will be familiar to HDL users is the very basic `COUNT <= COUNT + 1`; as shown in Figure 4. This is the correct form to use when prototyping some HDL code or making a test bench, but it doesn't synthesise to a

very efficient structure. In fact, this produces a structure that is more than 30% slower and almost twice as large as the slowest schematic capture counter. So what went wrong?

```
architecture behv of vhd16e is
    signal COUNT : std_logic_vector ( 0 to 15 );
begin
count16: process (RST, CLK)
    begin
        if ( RST = '1' ) then
            COUNT <= (others=>'0');
        elsif ( CLK'event and CLK = '1' ) then
            if ( EN = '1' ) then
                if ( COUNT = B"1111111111111111" ) then
                    COUNT <= (others=>'0');
                else
                    COUNT <= COUNT + 1;
                end if;
            end if;
        end if;
    end process count16;
Q <= COUNT;
end behv;
```

Figure 4. The Behavioral VHDL for a Ripple Carry Counter

The problem with the behavioral model of the counter is that it gives the synthesis tool many degrees of freedom. Most synthesis tools would produce a ripple carry counter from the VHDL in Figure 4; it is unlikely that a look ahead or a prescaled counter would be generated.

We can improve upon the behavioral model for the counter by specifying exactly what is wanted. For example, examine the VHDL of Figure 5.

```
architecture struct of vhd16er is
    signal RIPPLE : std_logic_vector ( 0 to 15 );
    signal ENABLE : std_logic_vector ( 0 to 15 );
    signal COUNT : std_logic_vector ( 0 to 15 );
    signal i : integer;
    signal j : integer;
    component tflop port (
        RST, CLK, EN : in std_logic;
        Q : out std_logic);
    end component;
begin
G1: for i in 0 to 15 generate
    count_stage: tflop port map (
        RST=>RST, CLK=>CLK,
        Q=>COUNT(i),
        EN=>ENABLE(i));
    end generate;
RIPPLE(0) <= '1';
ENABLE(0) <= EN;
G2: for j in 1 to 15 generate
    RIPPLE(j) <= (COUNT(j-1) and RIPPLE(j-1));
    ENABLE(j) <= (RIPPLE(j) and EN);
    end generate;
Q <= COUNT;
end struct;
```

Figure 5. Structural VHDL Description of a Ripple Counter

4



The VHDL of Figure 5 uses the VHDL 'generate' statement to instantiate 16 T-type flip flops. The T-type flip flops used are architecture independent; the VHDL that describes these flip flops is shown in Figure 6. Another generate statement is used to describe the ripple carry path and the logic for the enable input to the T-type flip flops. The VHDL of Figure 5 is

```
architecture rtl of tflop is
    signal Q_INT : std_logic;
begin
flop: process (RST, CLK)
    begin
        if ( RST = '1' ) then
            Q_INT <= '0';
        elsif ( CLK'event and CLK = '1' ) then
            if ( EN = '1' ) then
                Q_INT <= not Q_INT;
            end if;
        end if;
    end process flop;
Q <= Q_INT;
end rtl;
```

Figure 6. The VHDL for the tflop Component Used in Figure 5

not that much more complicated than the basic counter, and yet it is 23% faster and 46% smaller than the circuit produced by the VHDL of Figure 4. The circuit produced by the VHDL of Figure 5 is the same size as the schematic capture version.

4

A slightly different arrangement of the VHDL used in Figure 5 is shown in Figure 7. The VHDL is very similar, except that the statement that ANDs the ripple and enable

signals together has been moved into the tflopa component. This results in a ripple carry circuit that is 22% larger than the schematic capture version, but just 1% slower.

```
architecture struct of vhd16era is
    signal RIPPLE : std_logic_vector ( 0 to 15 );
    signal COUNT : std_logic_vector ( 0 to 15 );
    signal i : integer;
    signal j : integer;
    component tflopa port ( RST, CLK, EN, RIN : in std_logic;
        Q : out std_logic);
    end component;
begin
    G1: for i in 0 to 15 generate
        count_stage: tflopa port map (
            RST=>RST, CLK=>CLK,
            Q=>COUNT(i),
            EN=>EN,
            RIN=>RIPPLE(i));
    end generate;
    RIPPLE(0) <= '1';
    G2: for j in 1 to 15 generate
        RIPPLE(j) <= (COUNT(j-1) and RIPPLE(j-1));
    end generate;
    Q <= COUNT;
end struct;
```

Figure 7. Structural Ripple Carry VHDL that is Just 1% Slower than the Schematic Capture Version



Synthesis of the Advanced Counters

The behavioral version of the look-ahead counter is shown in Figure 8. This complicated piece of VHDL is almost 40% faster and 13% larger than the behavioral VHDL version

of the ripple carry counter. However, it is twice as large as the schematic capture ripple counter and it is slower.

```

architecture behv of vhd161a is
    signal COUNT : std_logic_vector ( 0 to 15 );
    signal LA8 : std_logic;
    signal LA11 : std_logic;
    signal LA14 : std_logic;

begin
    LA8 <= (COUNT(0) and COUNT(1) and COUNT(2) and COUNT(3) and COUNT(4) and COUNT(5) and COUNT(6)
    and COUNT(7));
    LA11 <= (COUNT(8) and COUNT(9) and COUNT(10) and LA8);
    LA14 <= (COUNT(11) and COUNT(12) and COUNT(13) and LA11);
    count16: process (RST, CLK)
    begin
        if ( RST = '1' ) then
            COUNT <= (others=>'0');
        elsif ( CLK'event and CLK = '1' ) then
            if ( EN = '1' ) then
                if ( COUNT(0 to 7) = B"11111111" ) then
                    COUNT(0 to 7) <= (others=>'0');
                else
                    COUNT(0 to 7) <= COUNT(0 to 7) + 1;
                end if;
                if ( LA8 = '1' ) then
                    if ( COUNT(8 to 10) = B"111" ) then
                        COUNT(8 to 10) <= (others=>'0');
                    else
                        COUNT(8 to 10) <= COUNT(8 to 10) + 1;
                    end if;
                end if;
                if( LA11 = '1' ) then
                    if ( COUNT(11 to 13) = B"111" ) then
                        COUNT(11 to 13) <= (others=>'0');
                    else
                        COUNT(11 to 13) <= COUNT(11 to 13) + 1;
                    end if;
                end if;
                if ( LA14 = '1' ) then
                    if ( COUNT(14 to 15) = B"11" ) then
                        COUNT(14 to 15) <= (others=>'0');
                    else
                        COUNT(14 to 15) <= COUNT(14 to 15) + 1;
                    end if;
                end if;
            end if;
        end if;
    end process count16;
    Q <= COUNT;
end behv;

```

4

Figure 8. The Behavioral VHDL for a Look-Ahead Counter – Faster than the Behavioral VHDL Ripple Carry Counter, But Slower than (and twice the size of) the Schematic Capture Ripple Carry Counter



The structural VHDL for the look ahead counter is a lot more efficient. Figure 9 shows structural VHDL that the same size as the schematic capture version of the look ahead counter and is less than 1% slower.

```

architecture struct of vhd161aa is
    signal COUNT : std_logic_vector ( 0 to 15 );
    signal RIPPLE : std_logic_vector ( 0 to 15 );
    signal LA8 : std_logic;
    signal LA11 : std_logic;
    signal LA14 : std_logic;
component tflopa port ( RST, CLK, EN, RIN: in std_logic;
                       Q : out std_logic);
    end component;

begin
LA8 <= (COUNT(0) and COUNT(1) and COUNT(2) and COUNT(3) and COUNT(4)
       and COUNT(5) and COUNT(6) and COUNT(7));
LA11 <= (COUNT(8) and COUNT(9) and COUNT(10) and LA8);
LA14 <= (COUNT(11) and COUNT(12) and COUNT(13) and LA11);
    G1: for i in 0 to 15 generate
        count_stage: tflopa port map (      RST=>RST, CLK=>CLK,
                                           Q=>COUNT(i),
                                           EN=>EN,
                                           RIN=>RIPPLE(i));
        -- Note use of conditional generate statement
    G2 : if ((i /= 0) and (i /= 1) and (i /= 8) and (i /= 11)
           and (i /= 14)) generate
        RIPPLE(i) <= (RIPPLE(i-1) and COUNT(i-1));
    end generate;
    end generate;
    -- look ahead carries
RIPPLE(0) <= '1';
RIPPLE(1) <= COUNT(0);
RIPPLE(8) <= LA8;
RIPPLE(11) <= LA11;
RIPPLE(14) <= LA14;
Q <= COUNT;
end struct;

```

4

Figure 9. Structural VHDL for a Look-Ahead Counter – the Same Size and Speed as a Schematic Capture Version, and Less Complicated than the Behavioral Version

The results are similar for the prescaled counter. The schematic capture version is the smallest and fastest as expected. The behavioral VHDL prescaled counter of Figure 10 is 6% smaller and 47% faster than the behavioral

VHDL look ahead counter – and the VHDL is much simpler. The structural VHDL version of the prescaled counter the same size as, and slightly slower than, the structural version of the look ahead counter.

```

architecture behv of vhd16psb is
    signal COUNT : std_logic_vector ( 0 to 15 );
    signal EN_INT : std_logic;
begin
prescale: process (RST, CLK)
    begin
        if ( RST = '1' ) then
            COUNT(0 to 1) <= (others=>'0');
        elsif ( CLK'event and CLK = '1' ) then
            if ( EN = '1' ) then

                if ( COUNT(0 to 1) = B"11" ) then
                    COUNT(0 to 1) <= (others=>'0');
                else
                    COUNT(0 to 1) <= COUNT(0 to 1) + 1;
                end if;
            end if;
        end if;
    end process;
end behv;

```



Table 1. Results for the Counters without the Data Load Function

Name Of Example	Entry Method	# of Core Cells	Speed (MHz)	Effort to create	Notes
App_note	Schematic	45	33.9	45	Utilisation target set to 80%
Appnote6	VHDL Behavioral	84	22.8	18	COUNT <= COUNT + 1;
Appnote8	VHDL Structural	45	28.1	22	Utilisation target set to 80%
Appnote9	VHDL Structural	58	33.7	20	Slightly different arrangement to Appnote8
Appnote4	Schematic	55	47.4	55	Look Ahead Carry
Appnot13	VHDL Behavioral	95	31.6	45	COUNT <= COUNT + 1; with Look Ahead Carry
Appnot14	VHDL Structural	55	47.2	28	Look Ahead Carry
Appnot16	VHDL Structural	63	41.5	35	Look Ahead Carry, Different arrangement to Appnot14, Utilisation target set to 80%
Appnote5	Schematic	44	51.3 (Note 1.)	44	2-stage Prescaled Counter, Utilisation target set to 80%
Appnot17	VHDL Behavioral	79	46.5 (Note 2.)	34	COUNT <= COUNT + 1; with 2-stage Prescaled Counter
Appnot10	VHDL Structural	55	44.4 (Note 3.)	29	2-stage Prescaled Counter, Utilisation target set to 80%

1. Based on detailed timing information. MPADS static timing analysis reports 38.9MHz.
2. Based on detailed timing information. MPADS static timing analysis reports 19.5MHz.
3. Based on detailed timing information. MPADS static timing analysis reports 34.6MHz.

4

Table 2. Results for the Counters with Data Load

Name of Example	Entry Method	# of Core Cells	Speed (MHz)	Effort to create	Notes
Appnote7	VHDL Behavioral	160	23.9	20	COUNT <= COUNT + 1;
Appnote3	Schematic	119	35.5	119	Look Ahead Carry
Appnot11	VHDL Behavioral	150	30.8	47	COUNT <= COUNT + 1; with Look Ahead Carry
Appnot12	VHDL Structural	122	30.6	30	Look Ahead Carry
Appnot15	VHDL Structural	127	39.2	37	Look Ahead Carry, Different arrangement to Appnot12 (Same as Appnot16)

Interpretation of the Results

Some of the results of the study are not surprising; the largest and slowest counters were produced with behavioral VHDL, and the smallest and fastest counters were produced with schematic capture.

What is more surprising is the range of results found. Looking at the results for the counters without the data load function, it can be seen that changing the architecture and the coding style of a VHDL counter can result in a speed improvement of over 100%. Similarly, the largest behavioral VHDL counter is more than 100% larger than the smallest structural VHDL version.

The size of the counter implementation is not necessarily

related to the speed of the counter produced. Referring to the graph in Figure 11, it can be seen that there is no strong correlation between size and speed. Indeed, the smallest counter is also the fastest and the second largest counter is also the slowest.

What can also be seen from Figure 11 is that the three behavioral VHDL counters are the largest by a significant margin. It can also be seen that structural VHDL is almost capable of matching schematic capture on the basis of size, speed, or both. For the very best in size and speed, it is still necessary to use schematic capture.



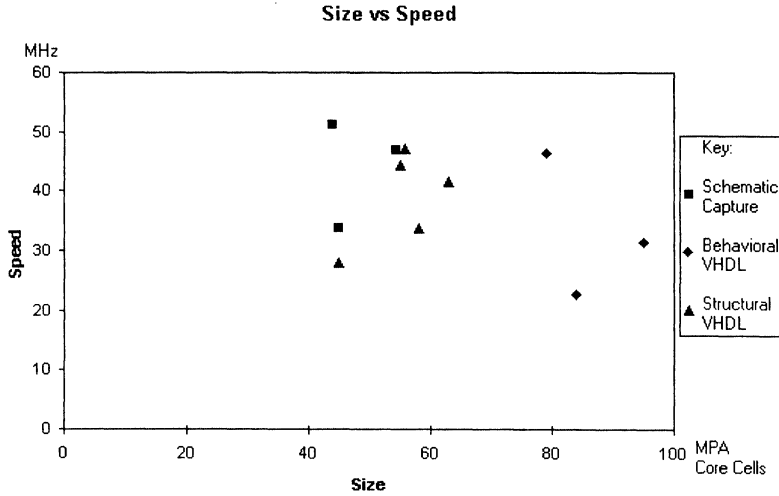


Figure 11. Size Plotted Against Speed for the Counters in the Study Without the Data Load Feature.
Bigger is not necessarily faster.

The results are a little different if the effort to produce the required solution is taken into account. The graph of Figure 12 introduces two new terms: quality and effort. For the purposes of this comparison the 'quality' of a counter is defined as the speed of a counter divided by the number of core cells required to implement that counter. The 'effort' required to produce the counter is defined as either the number of lines of VHDL required to produce the counter or the number of instances in a schematic captured counter.

'quality' of the solution produced. As we showed in Figure 11, schematic capture is capable of producing the highest 'quality' counters, but it also requires significant effort to produce them. On the other hand, the very simple behavioral VHDL counter requires little effort to produce, but results in a poor 'quality' counter implementation. What is interesting to note is that in most cases, structural VHDL is capable of producing results of comparable 'quality' to schematic capture, but with much less effort.

4

Figure 12 shows some correlation between effort and

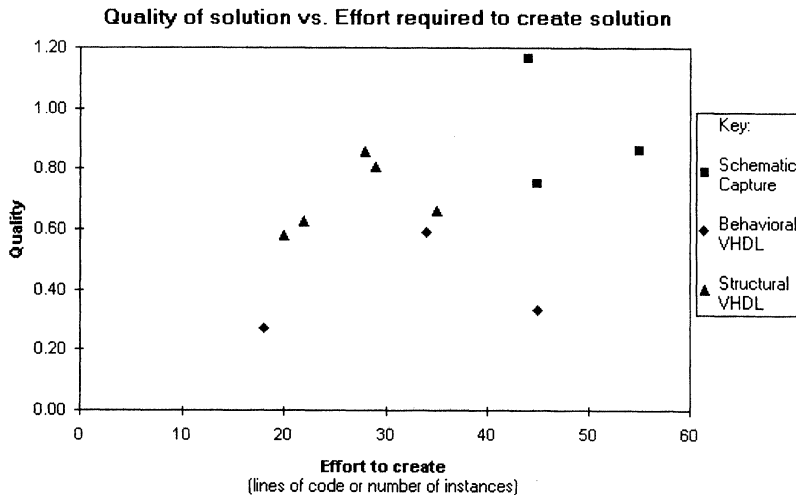


Figure 12. 'Quality' of Counter Implementation Plotted Against the Effort Required to Produce It.
More effort does not always guarantee a better solution.



Conclusions

Two different VHDL code styles have been compared and contrasted with schematic capture, and it has been shown that using VHDL does not necessarily mean having to accept a reduction in circuit performance. Three different counter architectures have been explored, and it has been shown that it is possible to gain significant performance benefits by changing counter architecture, without a dramatic increase in counter size or the effort required to generate the counter.

This study has shown that VHDL coding style and selection of different circuit architectures can have a sizeable effect on the quality of the solution produced. If you think about the structural implications of your VHDL, you can

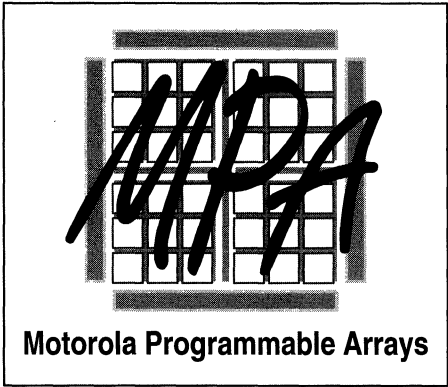
produce designs that rival the performance of schematic capture in a lot less time.

The Synario – MPA Device Kit will be released with version 2.4 of the MPA Design System. The kit supports MPA-specific and Synario Generic schematic capture libraries, VHDL and ABEL-HDL design entry, VHDL and Verilog simulation. You can download a demonstration version of the kit and a free version of the MPA Design System from the Motorola MPA web site,

<http://sps.motorola.com/fpga>

4





How to Reach Us **5**

Three Ways To Receive Motorola Semiconductor Technical Information

Literature Distribution Centers

Printed literature can be obtained from the Literature Distribution Centers upon request. For those items that incur a cost, the U.S. Literature Distribution Center will accept Master Card and Visa.

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado 80217.
Phone: 303-675-2140 or 1-800-441-2447

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 4-32-1,
Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan. **Phone: 81-3-5487-8488**

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park, 51 Ting Kok Road,
Tai Po, N.T., Hong Kong. **Phone: 852-26629298**

Mfax™ - Touch-Tone Fax

Mfax offers access to over 30,000 Motorola documents for faxing to customers worldwide. With menus and voice instruction, customers can request the documents needed, using their own touch-tone telephones from any location, 7 days a week and 24 hours a day. A number of features are offered within the Mfax system, including product data sheets, application notes, engineering bulletins, article reprints, selector guides, Literature Order Forms, Technical Training Information, and HOT DOCS (4-digit code identifiers for currently referenced promotional or advertising material).

A fax of complete, easy-to-use instructions can be obtained with a first-time phone call into the system, entering your FAX number and then, pressing 1.

How to reach us:

Mfax: RMFAX0@email.sps.mot.com -TOUCH-TONE (602) 244-6609
USA & Canada ONLY 1-800-774-1848

or via the <http://motorola.com/sps> home page, select the Mfax icon.

Motorola SPS World Marketing Internet Server

Motorola SPS's Electronic Data Delivery organization has set up a World Wide Web Server to deliver Motorola SPS's technical data to the global Internet community. Technical data such as the complete Master Selection Guide along with the OEM North American price book are available on the Internet server with full search capabilities. Other data on the server include abstracts of data books, application notes, selector guides, and textbooks. All have easy text search capability. Ordering literature from the Literature Distribution Center is available on line. Other features of Motorola SPS's Internet server include the availability of a searchable press release database, technical training information, with on-line registration capabilities, complete on-line access to the Mfax system for ordering faxes, an on-line technical support form to send technical questions and receive answers through email, information on product groups, full search capabilities of device models, a listing of the Domestic and International sales offices, and links directly to other Motorola world wide web servers. *For more information on Motorola SPS's Internet server you can request BR1307/D from Mfax or LDC.*

How to reach us:

After accessing the Internet, use the following URL:

<http://motorola.com/sps>

Motorola's Programmable Array Group

How to reach us:

After accessing the Internet, use the following URL:

<http://sps.motorola.com/fpga>

Our toll free Support Access Line at:

1-800-449-3742



MOTOROLA AUTHORIZED DISTRIBUTOR & WORLDWIDE SALES OFFICES

NORTH AMERICAN DISTRIBUTORS

UNITED STATES

ALABAMA

Huntsville
 Arrow/Schweber Electronics . . . (205)837-6955
 FAI . . . (205)837-9209
 Future Electronics . . . (205)830-2322
 Hamilton/Hallmark . . . (205)837-8700
 Newark . . . (205)837-9091
 Time Electronics . . . 1-800-789-TIME
 Wyle Electronics . . . (205)830-1119

ARIZONA

Phoenix
 FAI . . . (602)731-4661
 Future Electronics . . . (602)968-7140
 Hamilton/Hallmark . . . (602)414-3000
 Wyle Electronics . . . (602)804-7000

Tempe

Arrow/Schweber Electronics . . . (602)431-0030
 Newark . . . (602)966-6340
 PENSTOCK . . . (602)967-1620
 Time Electronics . . . 1-800-789-TIME

CALIFORNIA

Agoura Hills
 Future Electronics . . . (818)865-0040
 Time Electronics Corporate . . . 1-800-789-TIME

Belmont

Richardson Electronics . . . (415)592-9225

Calabassas

Arrow/Schweber Electronics . . . (818)880-9686
 Wyle Electronics . . . (818)880-9000

Chatsworth

Time Electronics . . . 1-800-789-TIME

Costa Mesa

Hamilton/Hallmark . . . (714)789-4100

Culver City

Hamilton/Hallmark . . . (310)558-2000

Garden Grove

Newark . . . (714)-893-4909

Irvine

Arrow/Schweber Electronics . . . (714)587-0404
 FAI . . . (714)753-4778
 Future Electronics . . . (714)453-1515
 Wyle Laboratories Corporate . . . (714)753-9953
 Wyle Electronics . . . (714)863-9953

Los Angeles

FAI . . . (818)879-1234
 Wyle Electronics . . . (818)880-9000

Manhattan Beach

PENSTOCK . . . (310)546-8953

Mountain View

Richardson Electronics . . . (415)960-6900

Newberry Park

PENSTOCK . . . (805)375-6680

Palo Alto

Newark . . . (415)812-6300

Riverside

Newark . . . (909)784-1101

Rocklin

Hamilton/Hallmark . . . (916)632-4500

Sacramento

FAI . . . (916)782-7882
 Newark . . . (916)565-1760
 Wyle Electronics . . . (916)638-5282

San Diego

Arrow/Schweber Electronics . . . (619)565-4800
 FAI . . . (619)623-2888
 Future Electronics . . . (619)625-2800
 Hamilton/Hallmark . . . (619)571-7540
 Newark . . . (619)453-8211
 PENSTOCK . . . (619)623-9100
 Wyle Electronics . . . (619)565-9171

San Jose

Arrow/Schweber Electronics . . . (408)441-9700
 Arrow/Schweber Electronics . . . (408)428-6400

FAI . . . (408)434-0369
 Future Electronics . . . (408)434-1122

Santa Clara

Wyle Electronics . . . (408)727-2500

Sierra Madre

PENSTOCK . . . (818)355-6775

Sunnyvale

Hamilton/Hallmark . . . (408)435-3500
 PENSTOCK . . . (408)730-0300
 Time Electronics . . . 1-800-789-TIME

Thousand Oaks

Newark . . . (805)449-1480

Torrance

Time Electronics . . . 1-800-789-TIME

Tustin

Time Electronics . . . 1-800-789-TIME

Woodland Hills

Hamilton/Hallmark . . . (818)594-0404
 Richardson Electronics . . . (615)594-5600

COLORADO

Lakewood

FAI . . . (303)237-1400
 Future Electronics . . . (303)232-2008

Denver

Newark . . . (303)373-4540

Englewood

Arrow/Schweber Electronics . . . (303)799-0258
 Hamilton/Hallmark . . . (303)790-1662
 PENSTOCK . . . (303)799-7845
 Time Electronics . . . 1-800-789-TIME

Thornton

Wyle Electronics . . . (303)457-9953

CONNECTICUT

Bloomfield

Newark . . . (203)243-1731

Cheshire

FAI . . . (203)250-1319
 Future Electronics . . . (203)250-0083
 Hamilton/Hallmark . . . (203)271-2844

Southbury

Time Electronics . . . 1-800-789-TIME

Wallingford

Arrow/Schweber Electronics . . . (203)265-7741

FLORIDA

Altamonte Springs

Future Electronics . . . (407)865-7900

Clearwater

FAI . . . (813)530-1665
 Future Electronics . . . (813)530-1222

Deerfield Beach

Arrow/Schweber Electronics . . . (305)429-8200
 Wyle Electronics . . . (305)420-0500

Ft. Lauderdale

FAI . . . (305)428-9494
 Future Electronics . . . (305)436-4043
 Hamilton/Hallmark . . . (305)484-5482
 Newark . . . (305)486-1151
 Time Electronics . . . 1-800-789-TIME

Lake Mary

Arrow/Schweber Electronics . . . (407)333-9300

Largo/Tampa/St. Petersburg

Hamilton/Hallmark . . . (813)547-5000
 Newark . . . (813)287-1578
 Wyle Electronics . . . (813)576-3004
 Time Electronics . . . 1-800-789-TIME

Orlando

FAI . . . (407)865-9555

Tallahassee

FAI . . . (904)668-7772

Tampa

PENSTOCK . . . (813)247-7556

Winter Park

Hamilton/Hallmark . . . (407)657-3300
 PENSTOCK . . . (407)672-1114
 Richardson Electronics . . . (407)644-1453

GEORGIA

Atlanta

FAI . . . (404)447-4767
 Time Electronics . . . 1-800-789-TIME
 Wyle Electronics . . . (404)441-9045

Duluth

Arrow/Schweber Electronics . . . (404)497-1300
 Hamilton/Hallmark . . . (404)623-4400

Norcross

Future Electronics . . . (770)441-7676
 Newark . . . (770)448-1300
 PENSTOCK . . . (770)734-8990
 Wyle Electronics . . . (770)441-9045

IDAHO

Boise

FAI . . . (208)376-8080

ILLINOIS

Addison

Wyle Laboratories . . . (708)620-0969

Bensenville

Hamilton/Hallmark . . . (708)797-7322

Chicago

FAI . . . (708)843-0034
 Newark Electronics Corp. . . . (312)784-5100

Hoffman Estates

Future Electronics . . . (708)882-1255

Itasca

Arrow/Schweber Electronics . . . (708)250-0500

LaFox

Richardson Electronics . . . (708)208-2401

Palatine

PENSTOCK . . . (708)934-3700

Schaumburg

Newark . . . (708)310-8980
 Time Electronics . . . 1-800-789-TIME

INDIANA

Indianapolis

Arrow/Schweber Electronics . . . (317)299-2071
 Hamilton/Hallmark . . . (317)575-3500
 FAI . . . (317)469-0441
 Future Electronics . . . (317)469-0447
 Newark . . . (317)259-0085
 Time Electronics . . . 1-800-789-TIME

Ft. Wayne

Newark . . . (219)484-0766
 PENSTOCK . . . (219)432-1277

IOWA

Cedar Rapids

Newark . . . (319)393-3800
 Time Electronics . . . 1-800-789-TIME

KANSAS

Kansas City

FAI . . . (913)381-6800

Lenexa

Arrow/Schweber Electronics . . . (913)541-9542
 Hamilton/Hallmark . . . (913)663-7900

Olathe

PENSTOCK . . . (913)829-9330

Overland Park

Future Electronics . . . (913)649-1531
 Newark . . . (913)677-0727
 Time Electronics . . . 1-800-789-TIME

MARYLAND

Baltimore

FAI . . . (410)312-0833

Columbia

Arrow/Schweber Electronics . . . (301)596-7800
 Future Electronics . . . (410)290-0600
 Hamilton/Hallmark . . . (410)720-3400
 Time Electronics . . . 1-800-789-TIME
 PENSTOCK . . . (410)290-3746
 Wyle Electronics . . . (410)312-4844

Hanover

Newark . . . (410)712-6922

5



AUTHORIZED DISTRIBUTORS – continued

UNITED STATES – continued

MASSACHUSETTS

Boston
Arrow/Schweber Electronics (508)658-0900
FAI (508)779-3111

Bolton
Future Corporate (508)779-3000

Burlington
PENSTOCK (617)229-9100
Wyle Electronics (617)271-9953

Norwell
Richardson Electronics (617)871-5162

Peabody
Time Electronics 1-800-789-TIME
Hamilton/Hallmark (508)532-9893

Woburn
Newark (617)935-8350

MICHIGAN

Detroit
FAI (313)513-0015
Future Electronics (616)698-6800

Grand Rapids
Newark (616)954-6700

Livonia
Arrow/Schweber Electronics (810)455-0850
Future Electronics (313)261-5270
Hamilton/Hallmark (313)416-5800
Time Electronics 1-800-789-TIME

Troy
Newark (810)583-2899

MINNESOTA

Bloomington
Wyle Electronics (612)853-2280

Burnsville
PENSTOCK (612)882-7630

Eden Prairie
Arrow/Schweber Electronics (612)941-5280
FAI (612)947-0909
Future Electronics (612)944-2200
Hamilton/Hallmark (612)881-2600
Time Electronics 1-800-789-TIME

Minneapolis
Newark (612)331-6350

Earth City
Hamilton/Hallmark (314)291-5350

MISSOURI

St. Louis
Arrow/Schweber Electronics (314)567-6888
Future Electronics (314)469-6805
FAI (314)542-9922
Newark (314)453-9400
Time Electronics 1-800-789-TIME

NEW JERSEY

Bridgewater
PENSTOCK (908)575-9490

Cherry Hill
Hamilton/Hallmark (609)424-0110

East Brunswick
Newark (908)937-6600

Fairfield
FAI (201)331-1133

Long Island
FAI (516)348-3700

Marlton
Arrow/Schweber Electronics (609)596-8000
FAI (609)988-1500
Future Electronics (609)596-4080

Pinebrook
Arrow/Schweber Electronics (201)227-7880
Wyle Electronics (201)882-8358

Parsippany
Future Electronics (201)299-0400
Hamilton/Hallmark (201)515-1641

Wayne
Time Electronics 1-800-789-TIME

NEW MEXICO

Albuquerque
Alliance Electronics (505)292-3360
Hamilton/Hallmark (505)828-1058
Newark (505)828-1878

NEW YORK

Bohemia
Newark (516)567-4200

Hauppauge
Arrow/Schweber Electronics (516)231-1000
Future Electronics (516)234-4000
Hamilton/Hallmark (516)434-7400
PENSTOCK (516)724-9580

Konkoma
Hamilton/Hallmark (516)737-0600

Melville
Wyle Laboratories (516)293-8446

Pittsford
Newark (716)381-4244

Rochester
Arrow/Schweber Electronics (716)427-0300
Future Electronics (716)387-9550
FAI (716)387-9600
Hamilton/Hallmark (716)272-2740
Richardson Electronics (716)264-1100
Time Electronics 1-800-789-TIME

Rockville Centre
Richardson Electronics (516)872-4400

Syracuse
FAI (315)451-4405
Future Electronics (315)451-2371
Newark (315)457-4873
Time Electronics 1-800-789-TIME

NORTH CAROLINA

Charlotte
FAI (704)548-9503
Future Electronics (704)547-1107
Richardson Electronics (704)548-9042

Raleigh
Arrow/Schweber Electronics (919)876-3132
FAI (919)876-0088
Future Electronics (919)790-7111
Hamilton/Hallmark (919)872-0712
Newark (919)781-7677
Time Electronics 1-800-789-TIME

OHIO

Centerville
Arrow/Schweber Electronics (513)435-5563

Cleveland
FAI (216)446-0061
Newark (216)391-9330
Time Electronics 1-800-789-TIME

Columbus
Newark (614)326-0352
Time Electronics 1-800-789-TIME

Dayton
FAI (513)427-6090
Future Electronics (513)426-0090
Hamilton/Hallmark (513)439-6735
Newark (513)294-8980
Time Electronics 1-800-789-TIME

Mayfield Heights
Future Electronics (216)449-6996

Solon
Arrow/Schweber Electronics (216)248-3990
Hamilton/Hallmark (216)498-1100

Worthington
Hamilton/Hallmark (614)888-3313

OKLAHOMA

Tulsa
FAI (918)492-1500
Hamilton/Hallmark (918)459-6000
Newark (918)252-5070

OREGON

Beaverton
Arrow/Almac Electronics Corp. ... (503)629-8090
Future Electronics (503)645-9454

Hamilton/Hallmark (503)526-6200
Wyle Electronics (503)643-7900

Portland

FAI (503)297-5020
Newark (503)297-1984
PENSTOCK (503)646-1670
Time Electronics 1-800-789-TIME

PENNSYLVANIA

Coatesville
PENSTOCK (610)383-9536

Ft. Washington
Newark (215)654-1434

Mt. Laurel
Wyle Electronics (609)439-9110

Montgomeryville
Richardson Electronics (215)628-0805

Philadelphia
Time Electronics 1-800-789-TIME
Wyle Electronics (609)439-9110

Pittsburgh
Arrow/Schweber Electronics (412)963-6807
Newark (412)788-4790
Time Electronics 1-800-789-TIME

TENNESSEE

Franklin
Richardson Electronics (615)791-4900

Knoxville
Newark (615)588-6493

TEXAS

Austin
Arrow/Schweber Electronics (512)835-4180
Future Electronics (512)502-0991
FAI (512)346-6426
Hamilton/Hallmark (512)219-3700
Newark (512)338-0287
PENSTOCK (512)346-9762
Time Electronics 1-800-789-TIME
Wyle Electronics (512)833-9953

Benbrook
PENSTOCK (817)249-0442

Carrollton
Arrow/Schweber Electronics (214)380-6464

Dallas
FAI (214)231-7195
Future Electronics (214)437-2437
Hamilton/Hallmark (214)553-4300
Newark (214)458-2528
Richardson Electronics (214)239-3680
Time Electronics 1-800-789-TIME
Wyle Electronics (214)235-9953

El Paso
FAI (915)577-9531

Ft. Worth
Allied Electronics (817)336-5401

Houston
Arrow/Schweber Electronics (713)647-6868
FAI (713)952-7088
Future Electronics (713)785-1155
Hamilton/Hallmark (713)781-6100
Newark (713)894-9334
Time Electronics 1-800-789-TIME
Wyle Electronics (713)879-9953

Richardson
PENSTOCK (214)479-9215

San Antonio
FAI (210)738-3330

UTAH

Salt Lake City
Arrow/Schweber Electronics (801)973-6913
FAI (801)467-9696
Future Electronics (801)467-4448
Hamilton/Hallmark (801)266-2022
Newark (801)261-5660
Wyle Electronics (801)974-9953

West Valley City
Time Electronics 1-800-789-TIME
Wyle Electronics (801)974-9953

5



AUTHORIZED DISTRIBUTORS – continued

UNITED STATES – continued

WASHINGTON

Bellevue

Almac Electronics Corp. (206)643-9992
 Newark (206)641-9800
 PENSTOCK (206)454-2371
 Richardson Electronics (206)646-7224

Bothell

Future Electronics (206)489-3400

Redmond

Hamilton/Hallmark (206)882-7000
 Time Electronics 1-800-789-TIME
 Wyle Electronics (206)881-1150

Seattle

FAI (206)485-6616
 Wyle Electronics (206)881-1150

WISCONSIN

Brookfield

Arrow/Schweber Electronics (414)792-0150
 Future Electronics (414)879-0244
 Wyle Electronics (414)521-9333

Milwaukee

FAI (414)792-9778
 Time Electronics 1-800-789-TIME

New Berlin

Hamilton/Hallmark (414)780-7200

Wauwatosa

Newark (414)453-9100

CANADA

ALBERTA

Calgary

Electro Sonic Inc. (403)255-9550
 FAI (403)291-5333

BRITISH COLUMBIA

Future Electronics (403)250-5550
 Hamilton/Hallmark (800)663-5500

Edmonton

FAI (403)438-5888
 Future Electronics (403)438-2858
 Hamilton/Hallmark (800)663-5500

Saskatchewan

Hamilton/Hallmark (800)663-5500

Vancouver

Arrow Electronics (604)421-2333
 Electro Sonic Inc. (604)273-2911
 FAI (604)654-1050
 Future Electronics (604)294-1166
 Hamilton/Hallmark (604)420-4101

MANITOBA

Winnipeg

Electro Sonic Inc. (204)783-3105
 FAI (204)786-3075
 Future Electronics (204)944-1446
 Hamilton/Hallmark (800)663-5500

ONTARIO

Kanata

PENSTOCK (613)592-6088

Mississauga

PENSTOCK (905)403-0724

Ottawa

Arrow Electronics (613)226-6903
 Electro Sonic Inc. (613)728-8333
 FAI (613)820-8244
 Future Electronics (613)820-8313
 Hamilton/Hallmark (613)226-1700

Toronto

Arrow Electronics (905)670-7769
 Electro Sonic Inc. (416)494-1666
 FAI (905)612-9888
 Future Electronics (905)612-9200
 Hamilton/Hallmark (905)564-6060
 Newark (905)670-2888
 Richardson Electronics (905)795-6300

QUEBEC

Montreal

Arrow Electronics (514)421-7411
 FAI (514)694-8157
 Future Electronics (514)694-7710
 Hamilton/Hallmark (514)335-1000
 Richardson Electronics (514)748-1770

Quebec City

Arrow Electronics (418)687-4231
 FAI (418)682-5775
 Future Electronics (418)877-6666

INTERNATIONAL DISTRIBUTORS

AUSTRALIA

AVNET VSI Electronics (Australia) (61) 2 878-1299
 Veltek Australia Pty Ltd (61) 3 9574-9300

AUSTRIA

EBV Austria (43) 1 8941774
 Elbatex GmbH (43) 1 866420
 Spoerle Austria (43) 1 31872700

BELGIUM

Diode Spoerle (32) 2 725 4660
 EBV Belgium (32) 2 716 0010

CHINA

Advanced Electronics Ltd. (852)2 305-3633
 AVNET WKK Components Ltd. . (852)2 357-8888
 China El. App. Corp. Xiamen Co
 (86)592 513-2489
 Nanco Electronics Supply Ltd. (852) 2 333-5121
 Qing Cheng Enterprises Ltd. . (852) 2 493-4202

DENMARK

Arrow Exatec (45) 44 927000
 Avnet Nortec A/S (45) 44 880800
 EBV Denmark (45) 39690511

ESTONIA

Arrow Field Eesti (372) 6503288
 Avnet Baltronic (372) 6397000

FINLAND

Arrow Field OY (35) 807 775 71
 Avnet Nortec OY (35) 806 13181

FRANCE

Arrow Electronique (33) 1 49 78 49 78
 Avnet Components (33) 1 49 65 25 00
 EBV France (33) 1 64 68 86 00
 Future Electronics (33) 1 69821111
 Newark (33) 1-30954060
 SEI/Scab (33) 1 69 19 89 00

GERMANY

Avnet E2000 (49) 89 4511001
 EBV Elektronik GmbH (49) 89 99114-0
 Future Electronics GmbH (49) 89-957 270
 Jermyn GmbH (49) 6431-5080
 Newark (49)2154-70011
 Sasco HED (49) 89-46110
 Spoerle Electronic (49) 6103-304-0

HOLLAND

EBV Holland (31) 3465 623 53
 Diode Spoerle BV (31) 4054 5430

HONG KONG

AVNET WKK Components Ltd. . (852)2 357-8888
 Nanshing Clr. & Chem. Co. Ltd . (852)2 333-5121

INDIA

Canyon Products Ltd (91) 80 558-7758

INDONESIA

P.T. Ometraco (62) 21 619-6166

ITALY

Avnet Adelsy SpA (39) 2 38103100
 EBV Italy (39) 2 660961
 Silverstar SpA (39) 2 66 12 51

JAPAN

AMSC Co., Ltd. 81-422-54-6800
 Fuji Electronics Co., Ltd. 81-3-3814-1411
 Marubun Corporation 81-3-3639-8951
 Nippon Motorola Micro Elec. . 81-3-3280-7300
 OMRON Corporation 81-3-3779-9053
 Tokyo Electron Ltd. 81-3-5561-7254

KOREA

Jung Kwang Sa (82)2278-5333
 Lite-On Korea Ltd. (82)2858-3853
 Nasco Co. Ltd. (82)23772-6800

NEW ZEALAND

AVNET VSI (NZ) Ltd (64)9 636-7801

NORWAY

Arrow Tahonic A/S (47)2237 8440
 Avnet Nortec A/S Norway (47) 66 846210

PHILIPPINES

Alexan Commercial (63) 2241-9493

SINGAPORE

Future Electronics (65) 479-1300
 Strong Pte. Ltd (65) 276-3996
 Uraco Technologies Pte Ltd. . (65) 545-7811

SPAIN

Amitron Arrow (34) 1 304 30 40
 EBV Spain (34) 1 804 32 56
 Selco S.A. (34) 1 637 10 11

SWEDEN

Arrow-Th:s (48) 8 362970
 Avnet Nortec AB (48) 8 629 14 00

SWITZERLAND

EBV Switzerland (41) 1 7456161
 Elbatex AG (41) 56 43751111
 Spoerle (41) 1 8746262

S. AFRICA

Advanced (27) 11 44423333
 Rheutec Components (27) 11 8233357

THAILAND

Shapiphat Ltd. (66)2221-0432 or 2221-5384

TAIWAN

Avnet-Mercuries Co., Ltd (886)2 516-7303
 Solomon Technology Corp. (886)2 788-8989
 Strong Electronics Co. Ltd. (886)2 917-9917

UNITED KINGDOM

Arrow Electronics (UK) Ltd . (44) 1 234 270027
 Avnet/Access (44) 1 462 488500
 Future Electronics Ltd. (44) 1 753 763000
 Macro Marketing Ltd. (44) 1 628 60600
 Newark (44) 1 420 543333

5



MOTOROLA WORLDWIDE SALES OFFICES

UNITED STATES

ALABAMA	
Huntsville	(205)464-6800
ALASKA	
Nome	(800)635-8291
ARIZONA	
Tempe	(602)302-8056
CALIFORNIA	
Calabasas	(818)878-6800
Irvine	(714)753-7360
Los Angeles	(818)878-6800
San Diego	(619)541-2163
Sunnyvale	(408)749-0510
COLORADO	
Denver	(303)337-3434
CONNECTICUT	
Wallingford	(203)949-4100
FLORIDA	
Clearwater	(813)524-4177
Maitland	(407)628-2636
Pompano Beach/Ft. Lauderdale	(305)351-6040
GEORGIA	
Atlanta	(770)729-7100
IDAHO	
Boise	(208)323-9413
ILLINOIS	
Chicago/Schaumburg	(847)413-2500
INDIANA	
Indianapolis	(317)571-0400
Kokomo	(317)455-5100
IOWA	
Cedar Rapids	(319)378-0383
KANSAS	
Kansas City/Mission	(913)451-8555
MARYLAND	
Columbia	(410)381-1570
MASSACHUSETTS	
Marlborough	(508)481-8100
Woburn	(617)932-9700
MICHIGAN	
Detroit	(810)347-6800
Literature	(800)392-2016
MINNESOTA	
Minnetonka	(612)932-1500
MISSOURI	
St. Louis	(314)275-7380
NEW JERSEY	
Fairfield	(201)808-2400
NEW YORK	
Fairport	(716)425-4000
Fishkill	(914)896-0511
Hauppauge	(516)361-7000
NORTH CAROLINA	
Raleigh	(919)870-4355
OHIO	
Cleveland	(216)349-3100
Columbus/Worthington	(614)431-8492
Dayton	(513)438-6800
OKLAHOMA	
Tulsa	(918)459-4565
OREGON	
Portland	(503)641-3681
PENNSYLVANIA	

Colmar	(215)997-1020
Philadelphia/Horsham	(215)957-4100
TENNESSEE	
Knoxville	(423)584-4841
TEXAS	
Austin	(512)502-2100
Houston	(713)251-0006
Plano	(214)516-5100
VIRGINIA	
Richmond	(804)285-2100
UTAH	
CSI Inc.	(801)572-4010
WASHINGTON	
Bellevue	(206)454-4160
Seattle Access	(206)622-9960
WISCONSIN	
Milwaukee/Brookfield	(414)792-0122

Field Applications Engineering Available
Through All Sales Offices

CANADA

BRITISH COLUMBIA	
Vancouver	(604)293-7650
ONTARIO	
Ottawa	(613)226-3491
Toronto	(416)497-8181
QUEBEC	
Montreal	(514)333-3300

INTERNATIONAL

AUSTRALIA	
Melbourne	(61-3)98870711
Sydney	(61-2)29661071
BRAZIL	
Sao Paulo	55(11)815-4200
CHINA	
Beijing	86-10-8437222
Guangzhou	86-20-7537888
Shanghai	86-21-3747668
Tianjin	86-22-5325072
DENMARK	
Denmark	(45) 43488393
FINLAND	
Helsinki	358-0-351 61191
car phone	358(49)211501
FRANCE	
Paris	33134 635900
GERMANY	
Langenhagen/Hanover	49(511)786880
Munich	49 89 92103-0
Nuremberg	49 911 96-3190
Sindelfingen	49 7031 79 710
Wiesbaden	49 611 973050
HONG KONG	
Kwai Fong	852-2-610-6888
Tai Po	852-2-666-8333
INDIA	
Bangalore	91-80-5598615
ISRAEL	

Herzlia

Herzlia	972-9-590222
ITALY	
Milan	39(2)82201
JAPAN	
Kyusyu	81-92-725-7583
Gotanda	81-3-5487-8311
Nagoya	81-52-232-3500
Osaka	81-6-305-1801
Sendai	81-22-268-4333
Takamatsu	81-878-37-9972
Tokyo	81-3-3440-3311
KOREA	
Pusan	82(51)4635-035
Seoul	82(2)554-5118
MALAYSIA	
Penang	60(4)228-2514
MEXICO	
Mexico City	52(5)282-0230
Guadalajara	52(36)21-8977
Marketing	52(36)21-2023
Customer Service	52(36)669-9160
NETHERLANDS	
Best	(31)4998 612 11
PHILIPPINES	
Manila	(63)2 822-0625
PUERTO RICO	
San Juan	(809)282-2300
SINGAPORE	
.....	(65)4818188
SPAIN	
Madrid	34(1)457-8204
or	34(1)457-8254
SWEDEN	
Solna	46(8)734-8800
SWITZERLAND	
Geneva	41(22)799 11 11
Zurich	41(1)730-4074
TAIWAN	
Taipei	886(2)717-7089
THAILAND	
Bangkok	66(2)254-4910
UNITED KINGDOM	
Aylesbury	44 1 (296)395252

FULL LINE REPRESENTATIVES

CALIFORNIA, Loomis	
Galena Technology Group	(916)652-0268
NEVADA, Reno	
Galena Tech. Group	(702)746-0642
NEW MEXICO, Albuquerque	
S&S Technologies, Inc.	(602)414-1100
UTAH, Salt Lake City	
Utah Comp. Sales, Inc.	(801)561-5099
WASHINGTON, Spokane	
Doug Kenley	(509)924-2322

HYBRID/MCM COMPONENT SUPPLIERS

Chip Supply	(407)298-7100
Elmo Semiconductor	(818)768-7400
Minco Technology Labs Inc.	(512)834-2022
Semi Dice Inc.	(310)594-4631

5





MOTOROLA

How to reach us:

USA/EUROPE/Locations Not Listed: Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217. 303-675-2140 or 1-800-441-2447

Mfax™: RMFAX0@email.sps.mot.com - TOUCHTONE 602-244-6609
- US & Canada ONLY 1-800-774-1848

INTERNET: <http://motorola.com/sps>

JAPAN: Nippon Motorola Ltd.: SPD, Strategic Planning Office, 4-32-1,
Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan. 81-3-5487-8488

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd.; 8B Tai Ping Industrial Park,
51 Ting Kok Road, Tai Po, N.T., Hong Kong. 852-26629298

DL201/D