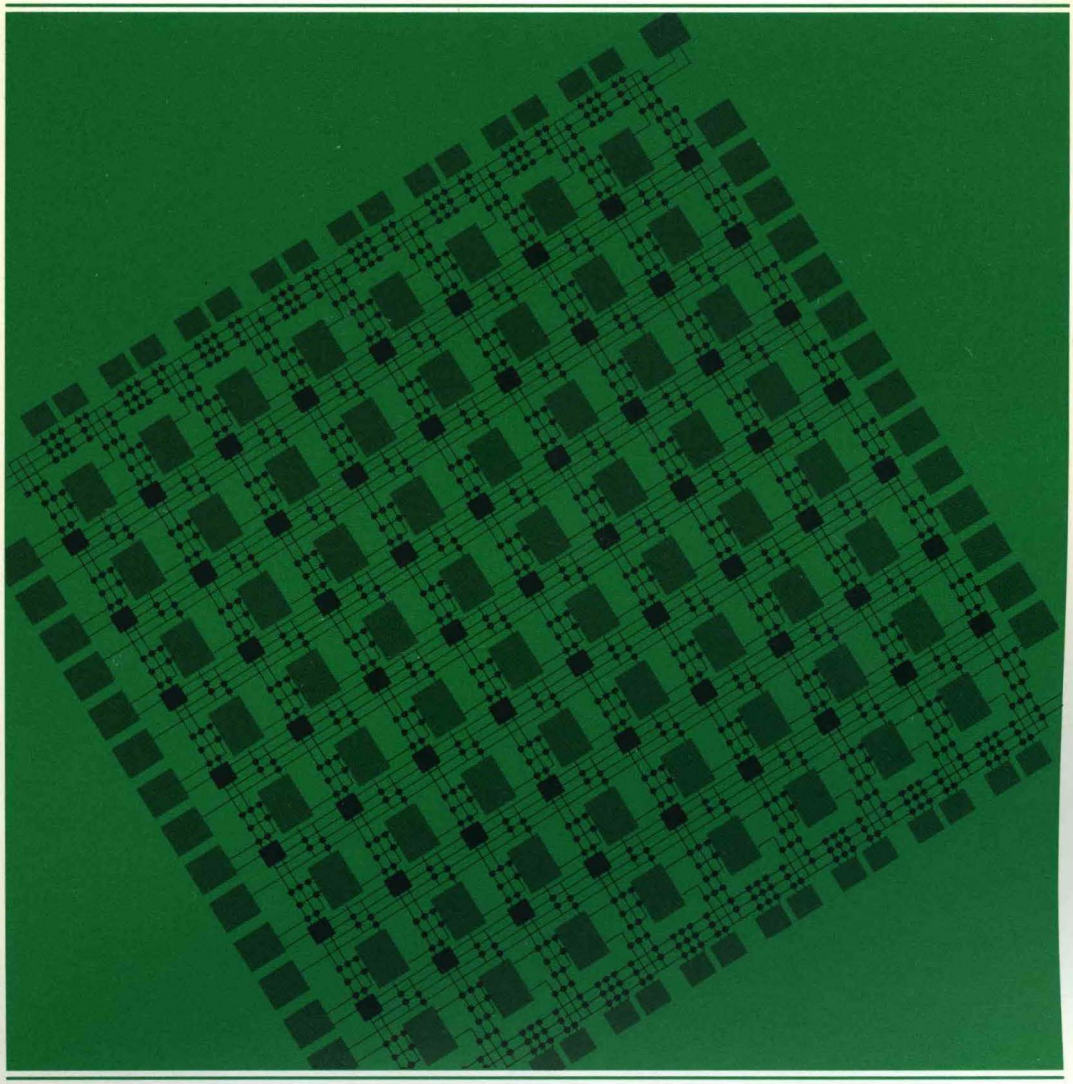
1988
Data Book

PGA Data Book

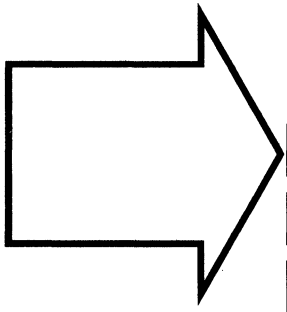
AMD

PGA
Data Book

Advanced
Micro
Devices



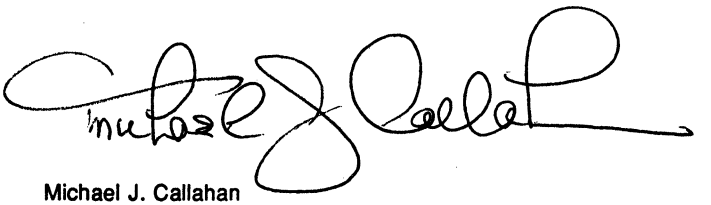
Advanced
Micro
Devices



PGA Databook

Since the original printing of this material, Monolithic Memories has merged with Advanced Micro Devices. References in this handbook to either company now pertain to the new combined entity, which markets all products under the AMD name.

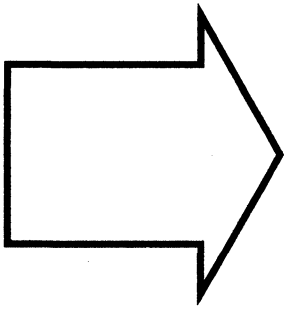
This book contains information about AMD's Programmable Gate Arrays, an exciting extension of our commitment to the field of programmable logic. The leader in programmable logic products, AMD continues to provide you with the quality, reliability and innovation you demand. As with every product we sell, AMD's Programmable Gate Arrays are backed by an extensive force of knowledgeable sales personnel and fully-trained field applications engineers. After reviewing the information in this book, you will see how PGAs can fit into your applications. Please contact your local AMD sales office, authorized representative or franchised distributor so that we can together, solve your technical problems with AMD's Programmable Gate Arrays.



Michael J. Callahan
Senior Vice President
Programmable Products Group

Table of Contents

Advanced
Micro
Devices



THE 2000 SERIES LOGIC CELL ARRAY

DESIGN HANDBOOK

BY

AMD, ADVANCED MICRO DEVICES

2

901 Thompson Place
P.O. Box 3453
Sunnyvale, CA 94088

TEL: 408-732-2400
TWX: 910339-9280
TELEX: 34-6306
TOLL FREE: 800-538-8450

APPLICATIONS HOTLINE: 800-222-9323

2000 SERIES LCA DESIGN HANDBOOK

TABLE OF CONTENTS

PREFACE

This handbook introduces the Logic Cell™ Array (LCA™) device by

- comparing it with other digital logic devices
- describing its three major components
- providing testing and data reliability specifications.

This handbook places each title and major topic at the top of a new page. Each chapter introduction lists the major topics therein. Each major topic introduction identifies the second-level topics to watch for, and so on.

Note: Abbreviations used in this handbook that are not explicitly defined are those deemed standard by the IEEE.

AUDIENCE

The reader of this handbook should have a working knowledge of the design, testing, and reliability of digital logic devices.

ACKNOWLEDGEMENTS

For information provided in this handbook, I would like to thank the following people from Xilinx, in alphabetical order.

Chuck Erickson
Steve Elischu
Rick Farabaugh
Dave Galli
Steve Knapp
Steve Landry
Dave Lautzenheiser
Richard Ravel
Rob Stransky
Craig Wooster
Pardner Wynn

For their great help during the preparation of this handbook, I would like to thank the following people from Advanced Micro Devices, in alphabetical order.

Audrey Dickey
Chris Jay
Vivian Kong
Jana McNulty
Joseph Parenteau
Arlo Radcliffe
Dieter Rathjens
Gail Tiberi
Joe Walcek

CHAPTER 1

INTRODUCTION TO THE LCA DEVICE

1 INTRODUCTION TO THE LCA DEVICE

This chapter introduces the AMD-supplied LCA family and covers the following topics.

- The overview, 1.1, introduces the Logic Cell™ Array, or LCA™ device.
- The discussion on architectural comparisons, 1.2, explains how LCA devices compare to other semiconductor devices.
- The discussion on LCA benefits, 1.3, compares the benefits of using an LCA device with those of other, more traditional semicustom devices.
- The discussion on LCA development software, 1.4, describes the available LCA design and verification software.

1.1 OVERVIEW

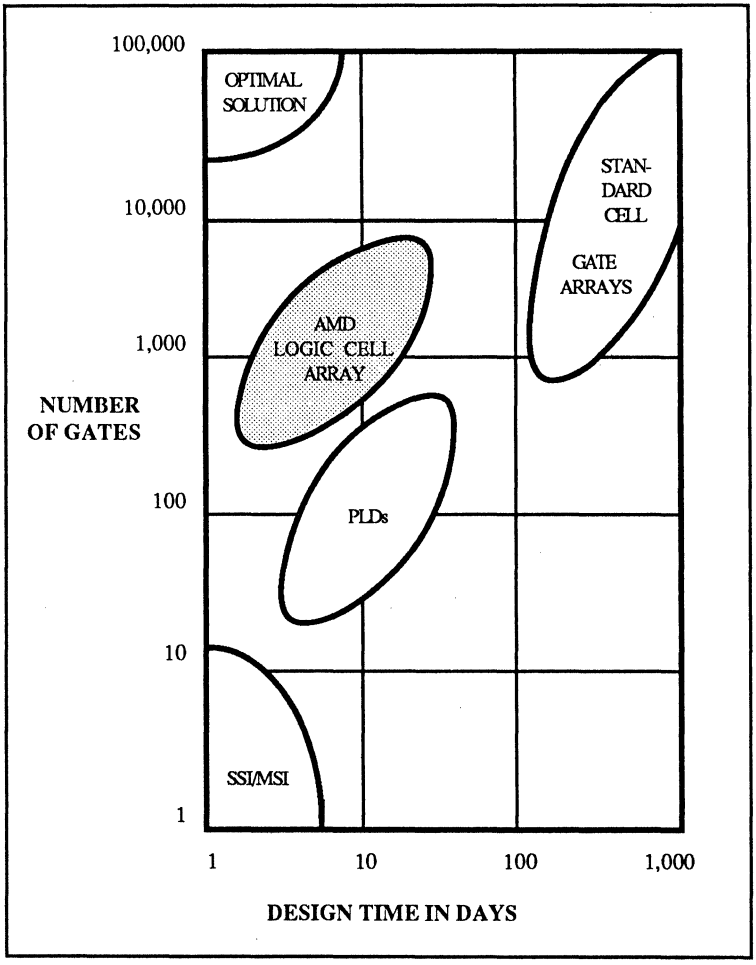
The following product features and improvements are motivating manufacturers of electronic systems to use high-density VLSI circuits.

- Lower cost
- Higher performance
- Reduced power consumption
- Smaller size
- Increased reliability

Microprocessors and memory devices are standard product ICs that have best exploited the advances in VLSI technology. Density improvements in these product types outpaced those in other digital integrated circuits and widened the technology gap between them and other logic devices. To achieve comparable densities for their proprietary functions, designers of digital equipment must consider using factory-programmed custom and semicustom application-specific integrated circuits (ASICs).

The advent of user-programmable gate arrays combines the production cost-effectiveness of VLSI with all the benefits of a standard product. The following figure illustrates the tradeoffs of density and development time for several digital logic device types.

The **optimal solution**, in the upper-left corner of the following diagram, represents the best tradeoff of density and complexity with design time. As illustrated, a new digital logic technology called the Logic Cell Array, or LCA, offers improvements over the traditional tradeoffs in both design complexity and design time.



Logic Technology Tradeoffs

Several common device classes are introduced below.

- 1.1.1, SSI/MSI Devices
- 1.1.2, Programmable Logic Devices (PLDs)
- 1.1.3, ASIC Devices
- 1.1.4, LCA Devices

1.1.1 SSI/MSI DEVICES

Standard **SSI/MSI devices**, which are well understood by most logic designers, provide much design flexibility and are readily available. However, they offer less density and consume more power than other device types. Also, they usually are manufactured in maturing technologies with limited opportunity for further cost reductions.

1.1.2 PROGRAMMABLE LOGIC DEVICES

Programmable logic devices (PLDs) include a number of competing alternatives, all based on a programmable AND/OR plane architecture. This architecture is most efficient for applications requiring up to a few hundred usable gates.

Typically, each programmable logic device replaces five to ten SSI/MSI devices. Because PLDs are user-programmable, designers achieve this gain in density with only a small increase in design time and little schedule risk. The design and device pattern for a specific application can be developed within days.

Bipolar PLDs are programmed by opening fuse links. CMOS PLDs are either one-time programmable, UV erasable (EPLD), or electrically erasable (EEPLD).

PLDs are best suited for state machines and decoders. For functions that are readily expressed as a sum-of-products, the PLD architecture provides efficient multi-variable decoding and high performance. Architectural restrictions limit the application of PLDs to general logic replacement, consolidation of miscellaneous glue logic and control functions, or complex processing tasks.

1.1.3 ASICs

Factory-programmable ASIC devices include gate arrays, standard cells, and compiled silicon. ASICs provide logic densities of up to 100,000 equivalent logic gates, and are sufficiently flexible for most digital logic functions. Fabricating factory programmed ASICs typically requires two to four months after a designer completes and verifies a design prototype. Manufacturing the first production quantities requires a similar period of time. Because of their high design costs and limited production flexibility, factory-programmed ASICs are most economical in very high production volumes. The logistics of verifying a workable design, testing ASIC devices, and coordinating production demand require substantial resources on the part of the manufacturer.

1.1.4 LCA DEVICES

The **LCA device** is a user-programmable gate array that provides the usable density of gate arrays and the short development times and low risk of standard logic circuits. The LCA device combines the design and production benefits of a standard logic device with the system benefits of increased reliability, power savings, space savings, and lower production costs than usually associated with ASIC device types.

The next discussion compares LCA devices to other semiconductor devices.

1.2 ARCHITECTURE COMPARISONS

Your benefits of the LCA device result from its flexible array architecture. This architecture is based on a number of technical breakthroughs that have resulted in patent disclosures. Discussions below compare familiar device architectures with that of the new LCA device.

- 1.2.1, Programmable Logic Devices
- 1.2.2, Gate Arrays
- 1.2.3, Logic Cell Arrays

1.2.1 PROGRAMMABLE LOGIC DEVICES

In PLD architectures, dedicated device input pins and some user-selectable input/output (I/O) pins or feedback paths directly drive the inputs to the AND/OR planes. Outputs are driven directly from the sum-of-products logic outputs, or from device flip-flops.

The primary limitations of the PLD architecture are the rigidity of the AND/OR plane logic and its dedicated interconnections. Flip-flops are typically driven by a common clock and are closely associated with specific output pins. As a result, gate use rarely exceeds 15%. Consequently, the practical upper limit of usable gates appears to be a few hundred, and the extension of this basic architecture to higher densities is limited. Also, PLD performance is fixed for each level of logic; each path through the AND/OR plane exhibits the same delay, typically 25 to 45 ns.

1.2.2 GATE ARRAYS

Array architectures provide more flexible resources than PLD architectures, both for I/O functions and for logic structures. A gate array typically implements user logic by interconnecting two-input NAND gates into more complex functions using mask-programmed metal segments. Factory processing customizes each gate array by creating the metal interconnections on standard, partially processed arrays.

Larger arrays can be generated through straightforward extensions of the I/O blocks, logic building blocks, and interconnection resources, much like extending the capacity of a memory device. Gate arrays offer usable densities of 25,000 gates or more. Usability of 80% to 90% is possible because of the architecture's flexibility and regularity.

Gate array performance depends on the placement and interconnection of the elements that make up each logic function. In a gate array characterized by 2 ns gate delays, frequently-used functions can have a total delay of 15 ns or more, due to the number of interconnections and gating levels needed to implement them.

1.2.3 LCA DEVICES

The LCA architecture resembles a gate array with an interior matrix of logic blocks and a surrounding ring of I/O interface blocks. LCA devices also share the gate array architecture's flexibility and ease of extension to higher densities. However, they do not share the gate array's need for factory programming. Instead, a configuration bit stream stored in on-chip memory defines and controls the function of the LCA device's configurable logic blocks (CLBs), I/O blocks (IOBs), and user-programmable interconnections. Distributed memory cells are adjacent to the logic, I/O, or interconnection elements they control. The interconnections are located in the channels between the rows and columns of configurable logic blocks, and between the configurable logic blocks and I/O blocks.

Straightforward extensions of the LCA architecture can increase a 1200-gate array to one with more than 1800 gates. Further extensions of the LCA architecture have increased the number of usable gates to 9000.

Like other standard IC components, LCA devices provide a selection of low- and high-speed parts. You

can choose the most cost-effective speed grade for your application.

LCA performance depends on the fixed delays of the logic and storage elements plus the interconnection delays. During design, the LCA delay calculation software can quickly display worst-case timing. Typically, LCA performance is specified by the maximum toggle rate for a logic block storage element when it is configured as a toggle flip-flop. For typical configurations, a 70 MHz toggle rate translates to a system clock rate of up to 35 MHz.

Unlike conventional gate arrays, the LCA device requires no custom factory fabrication. Each device is identical until it is loaded with its application-specific configuration bit stream. During normal operation, the configuration bit stream is loaded automatically from an EPROM or a processor, either when the LCA device is powered up or on command while the system is operating. You can copyright your LCA configuration bit stream to protect your designs from unauthorized copying under the same legal precedents that are used effectively to protect microprocessor-based systems.

The next discussion lists and explains the main benefits of using LCA devices as logic devices.

1.3 LCA BENEFITS

LCA devices have three important logic design benefits discussed below.

- 1.3.1, Advanced process
- 1.3.2, High quality
- 1.3.3, Proven reliability

1.3.1 PROCESS

Over the last five years, the most pronounced trend in the semiconductor manufacturing process is the shift to CMOS technology. This shift is especially pronounced for ASIC devices. The advantages of advanced CMOS processes include both high speed and low power consumption.

LCA devices are fabricated using AMD's 1.5µ advanced process. Two metal layers are essential for an efficient array architecture; the array must propagate logic signals in horizontal and vertical directions, with minimum delays. The LCA manufacturing process is very similar to that used for high-speed memories; it exploits the photolithography and wafer diameter advances achieved in memory process technology. These advances result in ever-higher device density and performance at ever-decreasing cost.

1.3.2 QUALITY

As quality consciousness has grown among semiconductor users, awareness of the importance of testing has also increased among manufacturers.

Testability is an important consideration in the design of microprocessors, memories, and other standard products. These devices are tested exhaustively by AMD with carefully developed programs.

The testing of most application-specific ICs is less comprehensive, due to limitations of design and test program development. However, the LCA device is

100% testable; each device is comprehensively tested during the manufacturing process. Testing is accomplished by AMD without involving you in the definition of test programs or the generation of test vectors.

1.3.3 RELIABILITY

The LCA manufacturing process used is based on a process developed for high-performance CMOS static memories. Extensive process-development work ensures the most reliable memory devices and provides the same benefits to the LCA device. Data collected over millions of operating hours confirm the reliability of the LCA design and the CMOS process.

Compared with other logic devices, the LCA device exhibits extremely low power dissipation. This translates to lower operating temperatures and higher reliability. Also, packaging materials for the LCA device are selected to match closely the thermal coefficient for the expansion of the silicon. This match minimizes thermal stresses and further improves reliability.

The memory cell used to store the LCA configuration bit stream is particularly robust. Memory is written only during device configuration and its static output controls the logic elements in the array. Because the two circularly linked inverters that make up the static latch are adjacent, transients cause only minor differences in voltages. Each inverter is a true complementary transistor pair, so that a low impedance path to the supply rail always exists regardless of the state. Furthermore, tests involving bombardment with high levels of alpha radiation verify that the storage cell is not disturbed by alpha particles.

1.4 LCA DEVELOPMENT SOFTWARE

The development system for LCA devices is similar in capability and usage to those for microprocessors.

Development support for the LCA device includes complete software-based design entry, analysis, and verification. The LCA development system offers a complete basic configuration and several powerful options to enhance designer productivity. LCA development system features include the following.

A. A consistent, user-friendly, menu-driven environment for all LCA development software

B. Schematic entry

C. Macro library support for standard AMD-supplied and user-defined functions

D. Simulation interface support that includes netlist extraction

E. Automatic placement and routing

F. Interactive timing calculation and design optimization

G. In-circuit emulation for one or multiple LCA devices

The LCA development system for the 2000 series requires the following hardware.

- IBM® PC-XT™, PC-AT™, or 100% compatible computer
- 640 kBytes of internal RAM
- A serial mouse

A system that must interface with printers and other output devices also requires

- a single parallel port
- two serial ports.

A variety of schematic editors and design workstation platforms are also available.

Chapter 2 discusses the LCA design cycle in detail.

CHAPTER 2

THE LCA DESIGN CYCLE

2 THE LCA DESIGN CYCLE

This chapter provides an overview of the LCA design cycle and explains each step of the cycle in detail.

- The overview of the design cycle, 2.1, introduces the LCA design process and software.
- The discussion on design entry, 2.2, explains entering an LCA design as a schematic and converting the schematic design to netlist data.
- The discussion on logic simulation, 2.3, describes pre-route simulation.
- The discussion on automatic partitioning, and placement and routing, 2.4, explains translating the netlist into the LCA file format as well as automatic placement and routing.
- The discussion on design optimization, 2.5, explains routing optimization and delay calculation.
- The discussion on timing verification, 2.6, describes post-route simulation.
- The discussion on in-circuit design verification, 2.7, explains three optional ways to verify an LCA design.

2.1 OVERVIEW

Before you begin your LCA design, it is important for you to understand the LCA design cycle and, in particular, how the the design cycle relates to the LCA environment and the available software.

- Discussion 2.1.1 explains the LCA design system.
- Discussion 2.1.2 shows the relationship between the tasks in the LCA design cycle and the software tools you use.

2.1.1 LCA DESIGN SYSTEM

The **LCA design system**, shown in the following figure, consists of two independent design environments and a file format that can be used as a bridge among the formats of these environments.

- External design environment
- Internal design environment
- External netlist format

In the **external design environment**, all software packages deal with **silicon-independent** tasks such as schematic capture and simulation. These software packages frequently use incompatible file formats. Also, they may be supported on hardware platforms different from that of the internal design environment.

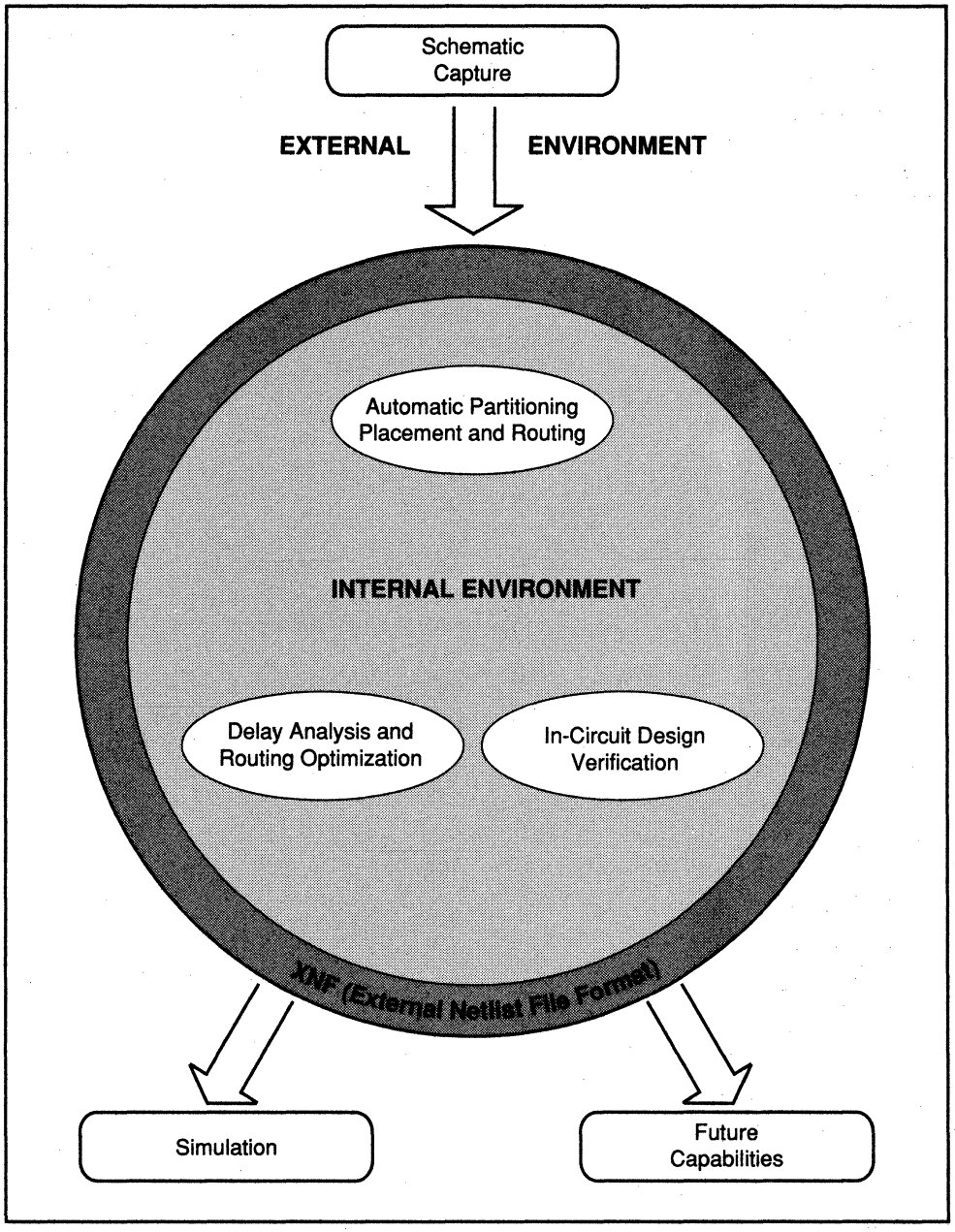
The **internal design environment**, in contrast, supports software packages dealing with **silicon-dependent** tasks such as logic partitioning, automatic placement and routing, and those tasks performed under the LCA software development system. These software packages communicate with each other through a common file format, the LCA format, which is proprietary. The LCA format contains silicon-dependent information such as delay, routing, and programming data.

The **external netlist format,** or **XNF**, provides the bridge among different file formats. This bridge permits files produced in one environment to be converted to a format that permits designers to use them in the other environment; it also permits conversion of files among different formats in the external environment.

AMD integrates the LCA design system with its interface software packages. These packages translate the different file formats into the XNF format, or the XNF format into the required file format. For example, the schematic to XNF interface translates the schematic file format into the XNF format; whereas the XNF to simulation interface translates the XNF format into typical simulation files such as the simulator netlist and input stimulus files.

After external files are translated into the XNF format, another software package offered by AMD can be used to convert the XNF format into the LCA format.

AMD will provide additional translation capabilities as new software packages in the external design environment require access to the LCA design system.


Schematic Capture
EXTERNAL
ENVIRONMENT
Automatic Partitioning Placement and Routing
INTERNAL ENVIRONMENT
Delay Analysis and Routing Optimization
In-Circuit Design Verification
XNF (External Netlist File Format)
Simulation
Future Capabilities

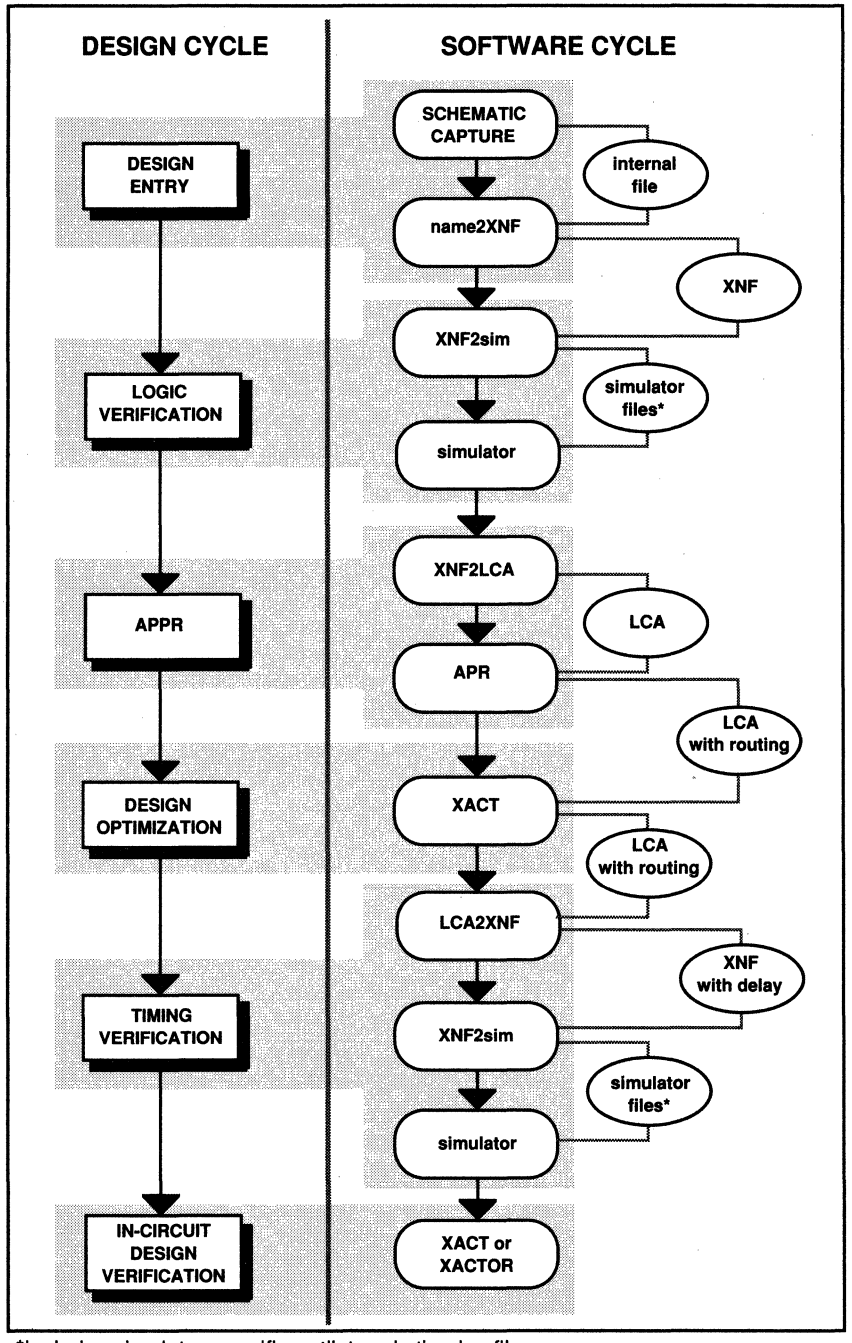
AMD supplies software that translates a design file from a unique schematic-capture format into the XNF format: *name*2XNF. AMD also supplies software to translate the XNF format to a simulation format, XNF2*sim*, and software that translates files between the XNF and LCA formats. These interfaces provide an environment that lets you perform all the tasks required to produce a consistent, integrated LCA design.

2.1.2 LCA SOFTWARE AND DESIGN CYCLE

The diagram on the next page is divided into two vertical segments that show the relationship between each task in the LCA design cycle and the actual software you use to perform each task.

In the figure, **square boxes** on the left show major tasks in the **design cycle**. Although the tasks are depicted serially, the design cycle is actually an iterative process in which you repeat a task or sequence of tasks.

Rounded boxes in the LCA software cycle represent the product used for the task connected by the shading. **Ovals** show the format of the input and/or output file for each step.



*includes simulator-specific netlist and stimulus files

To produce an LCA design, you complete six major design tasks.

I. **Design entry** using a supported schematic-capture platform includes **design conversion** to produce an external netlist format (XNF) file.

II. **Logic verification** includes translating the XNF file to specific simulation file formats. You then run a unit-delay simulation.

III. **Automatic partitioning, and placement and routing** (APPR) reduces the design, minimizes or compresses the logic, and partitions the design into required CLBs and IOBs. You can then place and route automatically.

IV. **Design optimization** includes delay analysis and routing optimization. Any logic changes you make here dictate that you return to task I and repeat subsequent tasks.

V. **Timing verification** includes translation of the completely routed design from its LCA format into the XNF format, translation of the XNF format into a simulation netlist, and simulation. You compare the timing simulation results with those of the previous logic simulation to ensure that there are no logic changes.[1]

VI. **In-circuit design verification** can include any of the following: using the download cable, programming a PROM, and using the in-circuit design verifier.

1 Unless you complete timing verification on a completely placed and routed design, you cannot be sure the design will function under worst-case process, voltage, and temperature conditions.

2.2 DESIGN ENTRY AND CONVERSION

When using a supported schematic-capture platform, the first two steps toward completing an LCA design are as follows.

1. Produce a schematic using the AMD-supplied LCA design library with a corresponding, supported schematic-capture platform.

2. Create an external netlist of the schematic-based design using the *name*2XNF interface software.

Schematic entry shortens your product development time by letting you enter complex LCA designs efficiently.

You enter the LCA design as a schematic using any AMD-supported schematic-capture platform. Rather than basing your design schematic on TTL or other standard parts, you base the schematic on the available **LCA library** parts.

The LCA library for each supported schematic-capture platform includes common logic functions and standard parts, such as gates, latches, and 7400-series parts.

While entering the schematic, you can constrain nets to direct design placement and routing with APR software later in the design cycle. You can also include constraints in a text file that APR can read. APR recognizes the following constraints.

- Critical nets that should be routed with as little interconnection delay as possible
- Nets that should be placed on long lines
- Specific placement of logic blocks
- Logic block grouping

- Locking nets and blocks

Once the schematic is produced, you must convert it to an external netlist file (XNF) format. AMD supplies a software interface product, *name*2XNF, for each supported schematic-capture platform. The interface software converts the schematic-based data into a netlist file that's needed to produce a simulation file or LCA design file.

Note: If the a schematic contains CLB or IOB macro symbols, which directly specify LCA elements, the XNF file must be translated into an LCA file and then back into an XNF file prior to logic verification. AMD supplies this interface in the XNF/LCA software product.

2.3 LOGIC VERIFICATION

The next step toward completing your design is to verify, during unit-delay simulation, that the **logic** is correct. This is done with a **gate-level** XNF file: one that does not have the physical routing paths specified in the LCA design. At this point it is not possible to make a realistic estimate of the timing because the design is not yet implemented as an LCA device.

The following steps outline the procedure for verifying the logic of an unrouted design with AMD-supplied software. For details, refer to specific topics in the LCA Development System manual before simulating.

1. Translate the XNF file into the simulation netlist file with XNF2*sim*.

   The netlist includes logic parameters and setup and hold times based on the selected LCA speed grade, operating under worst-case conditions.

2. Edit the input stimulus file created by XNF2*sim*, using a text editor.

   You specify simulation stimuli with a set of clock statements or with an input pattern for either pad inputs or internal nodes.

3. Simulate the design.

   Simulation results are available in tabular, plotted, and graphic formats. This flexibility makes it easy to correct the function and timing of the circuit.

4. Repeat design entry and logic verification until your design simulates correctly.

After schematic entry and before automatic partitioning, and placement and routing, you use a logic simulator to debug the design. This pre-route simulation saves design time because you can detect and correct logic

errors before final placement and routing. Later, you compare pre-route simulation with post-route simulation results to detect whether or not logic changes were introduced during optimization.

There are several software products available to assist you with simulation at various times in the design cycle. For each simulator there is one software interface product, XNF2*sim*, that translates the XNF file into the file format your particular simulator uses.

2.4 AUTOMATIC PARTITIONING, AND PLACE-MENT AND ROUTING

Once the design file is in the netlist format, you use the LCA2XNF translator to convert the schematic elements into LCA elements. You then partition, place, and route the LCA design. The following steps give an outline of this procedure.

1. Partition the design into CLBs and IOBs with the XNF2LCA software.

   This process efficiently groups as much logic as possible into each CLB, and translates the design into the LCA file format required by the LCA development system software.

2. Place and route the design.

   You can use automatic placement and routing (APR) software, or you can proceed directly to the LCA design editor, EDITLCA, and place and route the design interactively.

The XNF2LCA software completes the following tasks.

- Reduces and minimizes the logic
- Partitions the design into CLBs and IOBs
- Translates the design to the LCA format
- Performs design checking

The partitioning that occurs during translation may not result in the optimal placement of CLBs and IOBs. You can improve the layout of the design by including additional constraints to APR.

Both APR and XACT use the LCA file format. Thus, you can optimize the placement and routing of individual CLBs and IOBs in your design interactively with the interconnection feature of the LCA design editor,

EDITLCA. Refer to discussion 2.5 for more information on design optimization.

You can also take advantage of APR when you develop LCA designs incrementally. You can lock in place a partial LCA layout while APR places and routes additions to the design.

Note: If you do not lock in place a partially placed-and-routed design, the design is rearranged to yield a new placement when APR places and routes an **addition** to the design.

The APR software is extremely flexible. Through directives, you optimize the placement for a particular design. You can also specify routing resources to minimize clock skews and signal delays for critical paths. This results in faster product development.

Refer to Chapter 5 of this manual for a discussion of manual placement and routing. Refer to the LCA Development System manual, Volume II, Section I, for specific instructions on using APR.

2.5 DESIGN OPTIMIZATION

After initial placement and routing, you can optimize your design as follows.

1. Use the LCA design editor, EDITLCA, to optimize placement and routing.

2. Use the LCA design editor's delay calculator to check point-to-point timing after optimization.

2.5.1 ROUTING OPTIMIZATION

Whether you enter a design using schematic-entry software or lay out a design manually, you may have to use manual placement-and-routing and delay analysis for design optimization. You can modify the placement of your design by moving CLBs and re-routing the affected interconnection.

You also can lay out a **complete** design manually by using EDITLCA to configure design elements such as CLBs, IOBs, or system macros.

In either case, individual elements are configured directly with EDITLCA, either through Boolean equations or Karnaugh maps. A macro can be selected to automatically configure a block or group of blocks for a specified function.

The AMD LCA device supports a variety of routing resources, including long lines, global clock buffers, and direct connection.

Refer to Chapter 5 of this manual for more information on how to manually place and route LCA devices, and how to optimize the placement and routing of your LCA design to improve its performance. Two good ways to monitor the performance of your LCA design are with the LCA design editor's delay calculator, discussed next, and through timing verification, which is discussed under 2.6.

2.5.2 DELAY CALCULATION

You can perform timing analysis on a partially or completely routed LCA design to check its performance. Typically, you monitor the timing on critical paths as you complete the design. You can perform these timing checks quickly and efficiently using the LCA design editor's delay calculator.

The delay calculator is an interactive design tool that calculates and displays the worst-case delays associated with CLBs, IOBs, and interconnections. It is particularly useful for evaluating various placement-and-routing options during design optimization. The calculated delay represents the worst-case delay from the source block for that signal to the destination selected.

The delays are calculated based on the selected speed grade for the design; you can select an alternate speed grade to examine its impact on critical-path timing. Also, the worst-case delay calculations are from clock-edge to clock-edge for clocked systems.

The LCA design editor's delay calculator also flags any paths over which the signal is significantly degraded. In addition to displaying timing for individual networks, this delay calculator can produce a listing showing timing for all logic networks in a design.

Chapter 5 also contains a detailed discussion of timing analysis with the LCA design editor's delay calculator. See the DELAY command description in the LCA Development System manual, Volume I, Chapter 3.

2.6 TIMING VERIFICATION

At this point in the design cycle you should have an optimized, placed, and routed layout. You are ready to verify the timing and the logic.

Simulating a design's timing can be done either before or after the design has been verified in-circuit. You should simulate the critical paths to ensure that the design will function under worst-case process, voltage, and temperature conditions. The results obtained here should be compared with the logic simulation results to ensure that any layout editing did not change your design's functionality.

Timing verification lets you verify critical timing over worst-case power supply, temperature, and process conditions. It also helps you select the correct LCA speed grade for your application. To do this,

1. Use the LCA2XNF command to translate the routed CLB-based LCA design file into a CLB-based netlist that includes routing delays.

2. Use XNF2*sim* to translate the XNF netlist to an appropriate format for your simulator.

3. Simulate, review the timing information, and ensure no logic changes have occurred.

4. Use the LCA design editor, EDITLCA, to edit and optimize the design's timing until it meets your design specifications. Then repeat steps 1 through 4 above.

Once you are satisfied with the layout, you can proceed to in-circuit design verification. However, to make design changes, always return to the original schematic, make the design changes at that level, and then repeat all succeeding steps.

2.7 IN-CIRCUIT DESIGN VERIFICATION

Once the design meets your specifications, you can verify the design in-circuit. **In-circuit design verification** is the final stage of the LCA design cycle. You ensure that your design meets the specifications in the **target system**, the one for which it was designed. In-circuit design verification includes the following.

1. Use the XACT Bit-Stream Generator to automatically create the LCA design's **configuration bit stream**.

2. Verify your design's logic and timing in any of the following three ways.

   - Use the download cable to transfer the configuration bit stream to the LCA device in the target system.

   - Program a PROM and use it to configure the LCA device(s) in the target system.

   - Use XACTOR, the in-circuit emulator, to emulate the LCA device(s) in the target system.

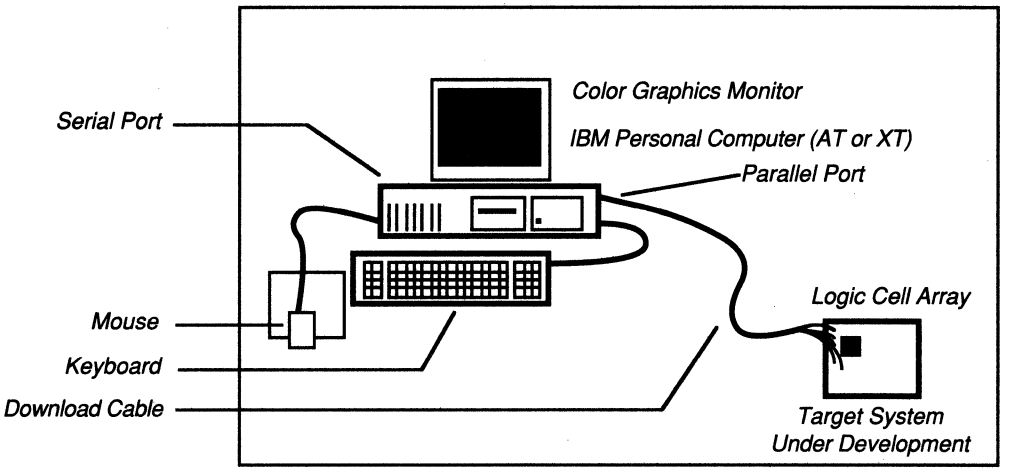
These three methods are discussed below. Refer to Chapter 6 in this manual for a detailed discussion of the configuration process, which includes generating a configuration bit-stream file from a design file in the LCA format.

2.7.1 DOWNLOAD CABLE

After you generate the configuration bit stream, you can use the bit-stream generator and download cable to program the LCA device for in-circuit verification, as described below.

The download cable connects the LCA device in the system under development to the parallel port on the LCA workstation, as shown in the next figure. The bit-

stream generator controls the operation of the download cable.



Download Cable Setup

To program the LCA device, you use the bit-stream generator to transfer the configuration bit stream across the download cable to the LCA device in the target system.

Note: During development and debugging, you can use this capability to save time because you do not need to reprogram a PROM each time you modify the configuration bit stream.

For more information on the XACT bit-stream generator, MAKEBITS, and the download cable, refer to the LCA Development System manual, Volume I, Chapter 7.

2.7.2 PROM PROGRAMMING

After creating the configuration bit stream for your LCA design, you can use it to program a PROM and let the PROM configure the target system LCA device(s), as explained below.

To load a PROM with the configuration bit stream for one or more LCA device(s), first convert the configuration bit stream into a **PROM-format file**. The LCA PROM formatter can create files automatically in a variety of standard PROM formats, for PROMs from 2 kB to 8 kB and larger. Each PROM file can represent one or more LCA designs, so one PROM can configure one LCA device or several daisy-chained LCA devices.

The LCA Development System manual, Volume I, Chapter 6, provides a complete description of the LCA PROM formatter, including the formatter commands.

2.7.3 XACTOR

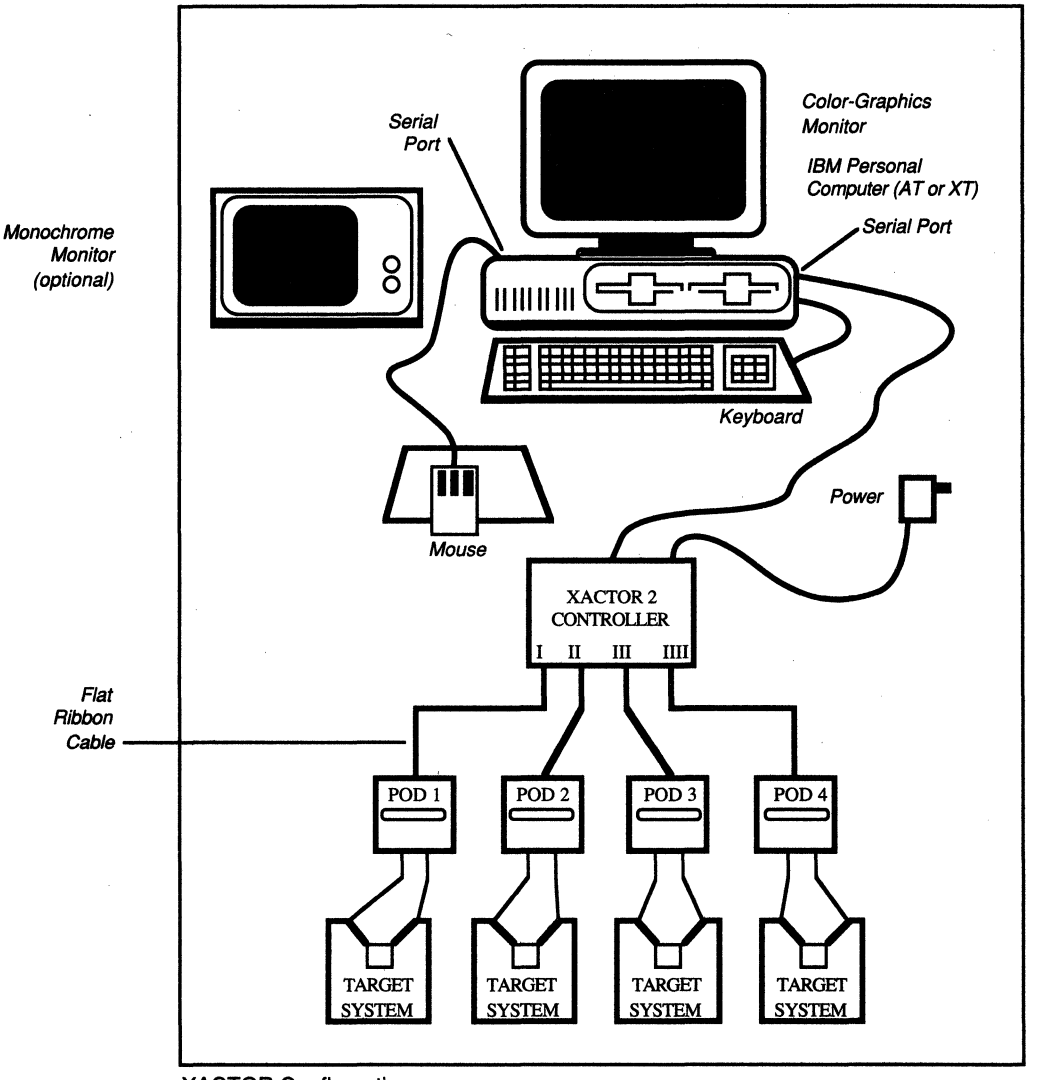
The XACTOR in-circuit emulator provides real-time, interactive target-system emulation of up to four LCA devices from the host PC.

> **Note:** In-circuit emulation enhances design simulation by letting you verify your LCA design's functionality in the target system in real time, while working with all other circuits and system software.

The XACTOR configuration, shown next, consists of a microcomputer-based controller and from one to four universal emulation pods, each with an emulation header. One pod and header is included with the basic LCA system. The XACTOR controller, connected to the PC workstation through a serial port, provides the following.

- Local storage of configuration bit streams
- Control of individual device configurations
- Control of the isolation of the pod device(s) from the target system

You set the state and isolation for each control signal to provide debugging of the target system hardware.



XACTOR Configuration

To emulate your design, the XACTOR controller programs the emulation LCA device in each of up to four emulation pods. After programming, each emulation pod provides the functionality of one or more LCA devices plugged directly into the target system. You can program the LCA devices in the target system individually or in a daisy chain. XACTOR supports daisy chains of up to seven LCA devices from any of the four emulation pods. You control each device's isolation and configuration with menu or keyboard commands. These can be supplemented by user-defined setup files for easy system debugging.

Using XACTOR, you can read back the device configuration to verify the configuration process and to interrogate the internal states. After you perform a read-back operation, XACTOR displays the state of all internal storage elements, isolation switches, and control signals. XACTOR also automatically reports asynchronous status changes in the target system. Refer to the discussion on reading back the configuration bit stream in Chapter 6 of this manual for more specific information.

Due to the speed with which you can modify a design in-circuit with XACTOR, you may find it useful to implement temporary debugging circuitry in your design during development. For example, you could temporarily connect unused I/O blocks to internal nodes for viewing with a logic analyzer or an oscilloscope.

For complete information on the XACTOR in-circuit emulator, refer to the LCA Development System manual, Volume I, Chapter 8.

CHAPTER 3

CONFIGURABLE LOGIC BLOCKS

3 CONFIGURABLE LOGIC BLOCKS

This chapter introduces configurable logic blocks, the basic design element of the Logic Cell Array (LCA).

- The overview, 3.1, provides a brief introduction to the design of LCA devices and to the LCA development system's part in this process.
- The discussion of the LCA device structure, 3.2, explains configurable logic blocks (CLBs) and describes their basic structure.
- The discussion on CLBs, 3.3, explains how to generate logic designs with CLBs.
- The discussion on timing, 3.4, explains the timing characteristics of CLBs.
- The discussion on logic design, 3.5, discusses LCA design using macrocells from the LCA macrocell library.

The macrocell library mentioned in this chapter is the LCA development system macrocell library. The AMD-supplied LCA logic libraries that support the schematic design-entry method are discussed in Chapter 2.

3.1 OVERVIEW

Each new technology available for digital design offers the designer a new set of characteristics. These characteristics include speed, power, integration level, reliability and selection of logic functions. A good designer makes the best use of a technology's design characteristics by matching the design's methodology and logic architecture to these characteristics.

The digital design and architecture for LCA devices is similar to that of conventional TTL SSI/MSI or gate arrays. However, the designer of LCA devices has additional design flexibility because of the lack of typical design limitations, which could include logic in four-bit or eight-bit increments, a specific set of inputs and outputs, or a combination of logic functions.

The core of the CMOS LCA integrated circuit is an array of user-programmable logic elements called configurable logic blocks, or CLBs. User-programmable interconnections of the CLBs create the required logic networks. Individually programmable input/output blocks, or IOBs, provide the interfaces for the LCA device's input/output. With these resources, you are free to tailor the LCA logic; you are not confined to standard product devices or gate array library elements.

AMD's LCA device gives you a higher level of integration than other standard products. The benefits of this higher integration level include

- increased performance and reliability,
- reduced printed circuit board space,
- lower power requirements,
- shorter design time, and
- smaller component inventories.

To create the logic capacity of one LCA device usually requires 40 to 100 SSI/MSI packages. Also, the LCA user-programmability gives you a single, fully-tested inventory item you can use in multiple products.

Using the conventional gate array definition of one **gate** as a 2-input NAND function, the LCA 2000 family provides a logic capacity up to 2000 gates. Using a single LCA device to construct part of a system design can reduce the package pin count of the design from hundreds of SSI/MSI pins to 48, 64, or 84 LCA pins. These three available LCA packages provide up to 84 pins that you can program as logic input, output, or both.

The next discussion introduces two main components of the LCA device, the CLB and the IOB.

3.2 LCA STRUCTURE

This discussion details the structure of a CLB. It also briefly discusses the IOB, which is described in more detail in Chapter 4.

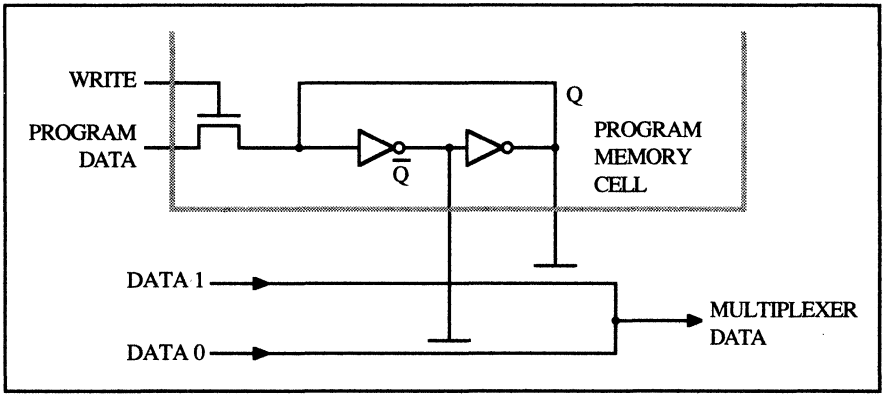
3.2.1 CLBs

The CLB is the basic logic building block in an LCA design. Each CLB has four logic inputs and two logic outputs. It includes a **combinational-function** portion and a **storage-element** portion. You can configure the combinational function to perform any function of four variables. You can configure the storage element as a transparent latch or an edge-triggered flip-flop.

You enter and verify an LCA design using the LCA development system, then generate a configuration bit stream that defines the appropriate functions within the CLB.

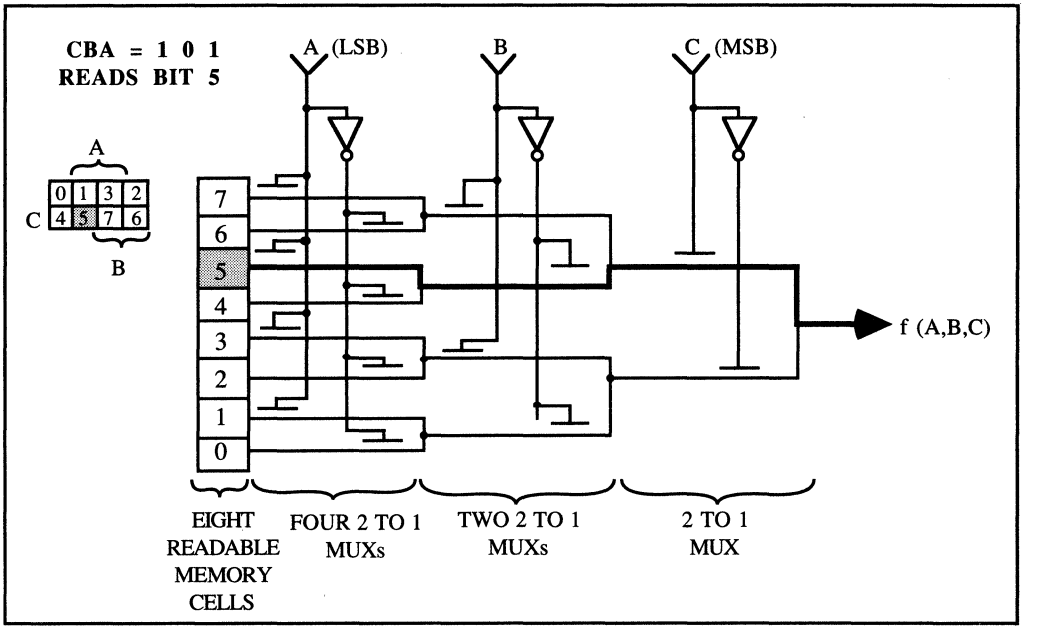
The interconnection of CLBs consists of a two-layer grid of metal segments. These metal segments are joined at each intersection by a switching matrix of controlled pass transistors that creates the interconnection paths of the CLBs and IOBs. Additional pass transistors connect these metal interconnections to the IOBs.

The following figure shows a single memory cell that controls a simple two-to-one multiplexer made of two pass transistors.



Memory Cell Multiplexer Control

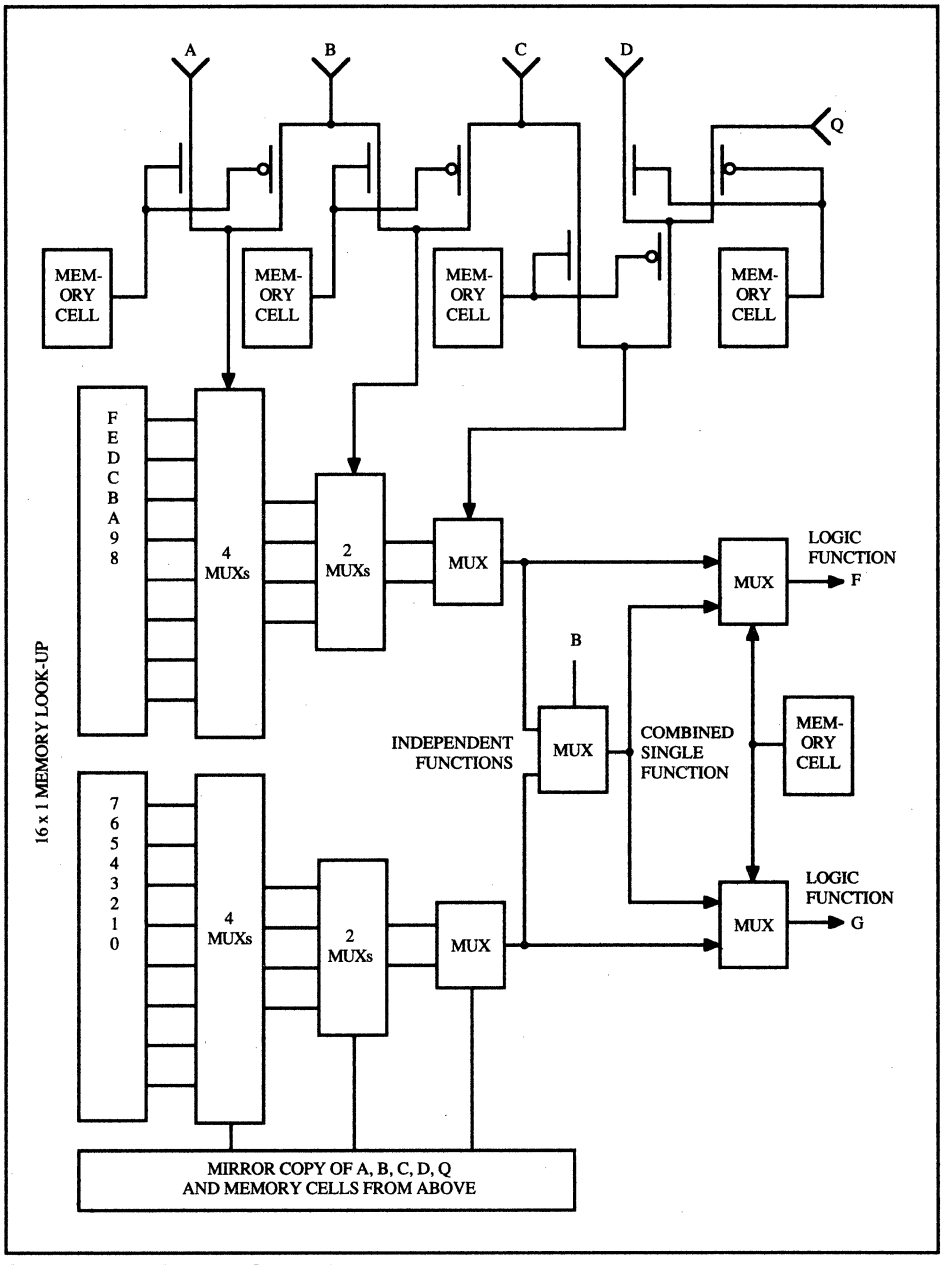
Combining eight of these readable memory cells to control an eight-to-one multiplexer tree, as shown below, creates a circuit capable of generating any logic function of the three-input variables A, B, and C.



Look-Up Function Generator

As illustrated above, the C, B, A input code 101 reads the contents of memory cell five. The data pattern of the readable memory cells defines the logic function. Doubling the look-up table and multiplexer creates a circuit that can generate any function of four variables, which is the basis of the CLB's combinational portion. The CLB includes programmable multiplexers for input variables A, B, C, D, and Q, shown in the next figure, and a selection of outputs, to create either a single function of four variables or two functions of three variables each.

The following figure shows the CLB's combinational function generator. As you can see, some paths are shorter than others.



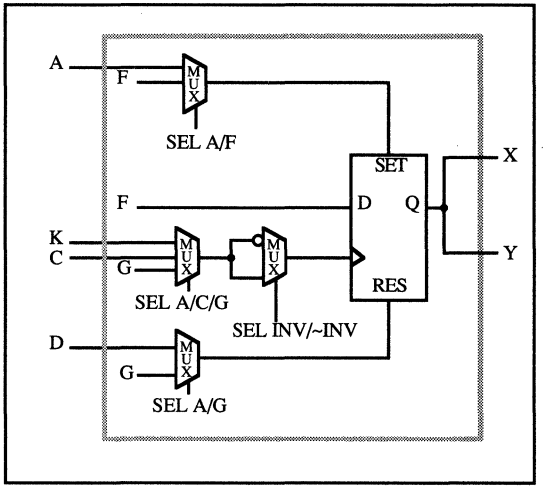
Combinational Function Generation

When a CLB generates a four-variable function, both halves of the look-up table select either the input variable D, or the Q of the storage element; the single result produces both F and G outputs.

When a CLB generates two functions of three variables, the D vs. Q selection is independent for the functions F and G. Each function can then use any three of the five available variables as input: A, B, C, D, or Q.

A CLB can generate a third type of function by using the input variable B to select between the two three-variable combinational functions. This configuration results in a compound function that can involve some combinations of all five variables.

The programmable features of the CLB storage element are shown below.



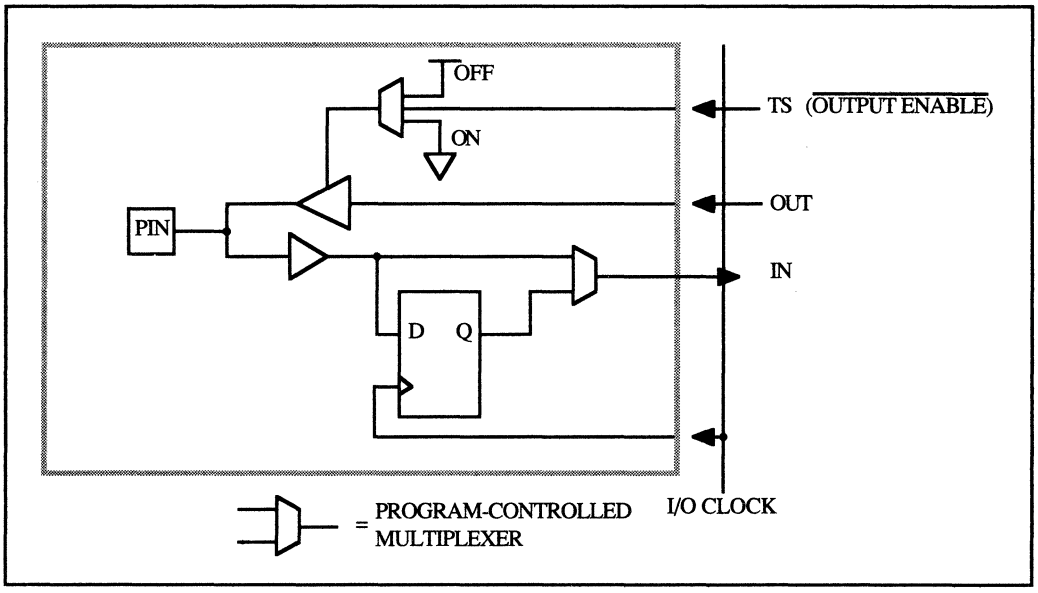
CLB Storage Element

You can leave the storage element part of a CLB unused, or program it as a level-transparent latch or an edge-triggered flip-flop. The combinational function F supplies its data input. You can also select the

invertible flip-flop clock or latch enable from any of three sources, on a CLB-by-CLB basis. Each CLB storage element has available an active-HIGH asynchronous SET, and a RESET. RESET is dominant over SET; the active-LOW chip input, ~RESET, clears all storage elements.

3.2.2 THE INPUT/OUTPUT BLOCK

The IOBs provide access between the CLBs and the world external to the LCA device. As illustrated below, IOBs can provide a direct or registered input to the chip.



I/O Block

The positive-edge clock for the register function is common along each die edge. The chip configuration process, as well as the active-LOW chip reset, ~RESET, clears the storage elements.

Each IOB includes an input/output buffer you can enable continuously to create an output pin, disable continuously to create an input or unused pin, or enable by logic signals to create an I/O or bus pin.

For more information on IOBs, refer to Chapter 4 of this manual.

3.3 LOGIC DESIGN WITH CLBs

This discussion explains how to use CLBs to design basic logic elements. It explains how to use CLBs, partial CLBs, and multiple CLBs to generate common functions.

3.3.1 CREATE BASIC LOGIC

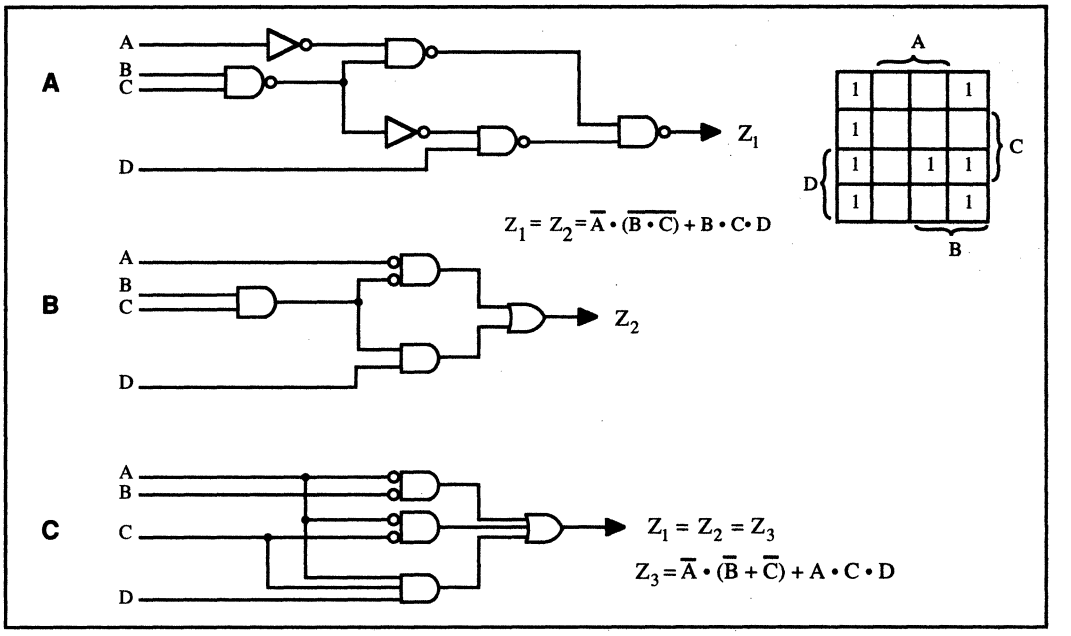
To create **basic logic**, you can choose from several equivalent ways of representing a CLB function, both schematically and mathematically. The LCA design editor, EditLCA, supports design entry through Karnaugh maps, truth tables, and Boolean equations. The CLB's ability to accommodate either sense of input variables, and to generate either sense of an output, lets you eliminate extraneous inverters. In most cases, however, it is practical to route only active-HIGH signals, thus avoiding the duplicate routing of both true and complement signals. The following figure shows

A. A typical four-variable combinational function as a logic diagram, a Boolean equation, and a Karnaugh map.

B. Equivalent forms of the function.

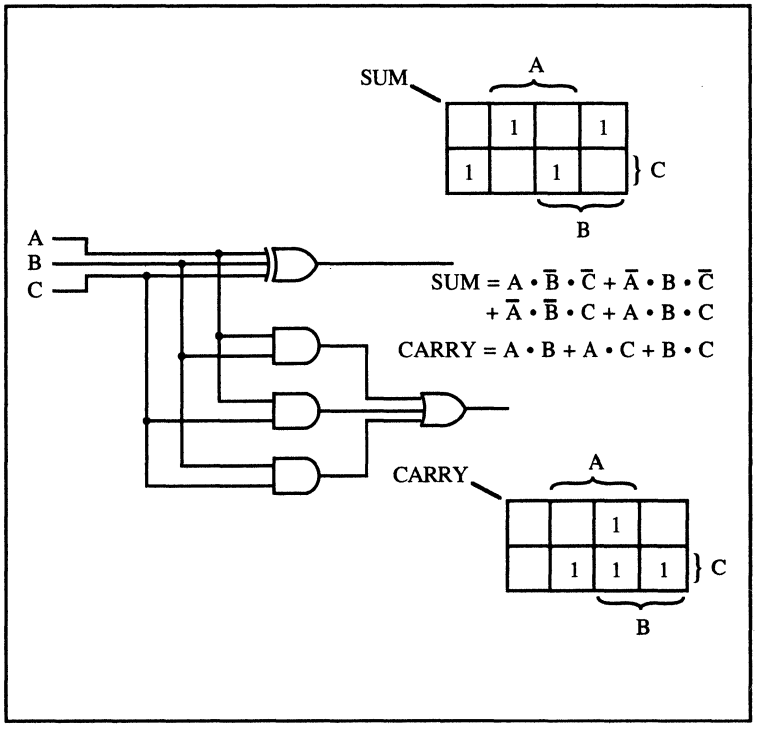
C. Equivalent forms of the function.

The active-LOW inputs replace the inverters of the conventional representation and the output symbol is an OR. AMD-supported schematic-capture interface software converts the logic in an LCA schematic design into an equivalent representation and groups the combinational gates while translating your design into LCA design files.



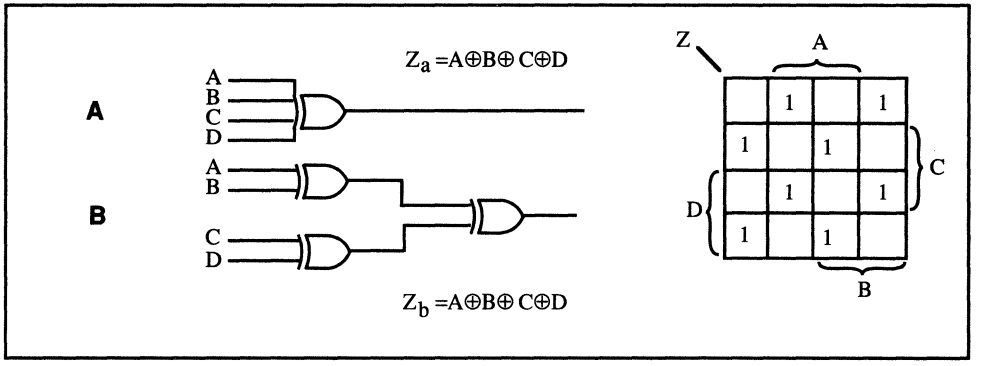
Alternate Representations of the Same Function in an LCA Device

The ADDER shown below illustrates how to use two combinational functions of three variables. The SUM and CARRY functions are usually grouped in the same CLB because of their common input variables.



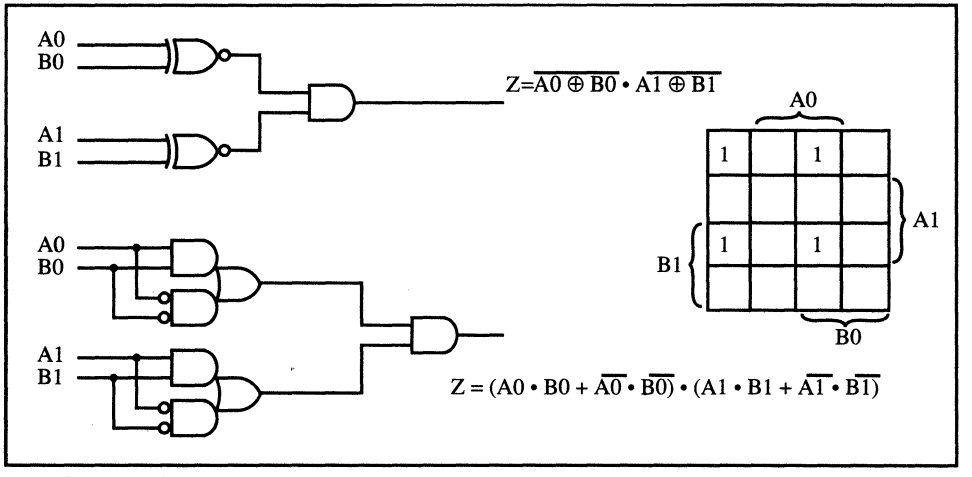
One-Bit Adder with Carry In

The four-input exclusive-OR gates in the following figure are an example of a common logic function that is not obviously four-variable. It is a modulo-2 add without a carry.



Four-Input Exclusive OR

COMPARE, which is similar, is usually a two-input function. The figure below shows a CLB-generated **dual compare** function, which compares two bits from each of two sources.

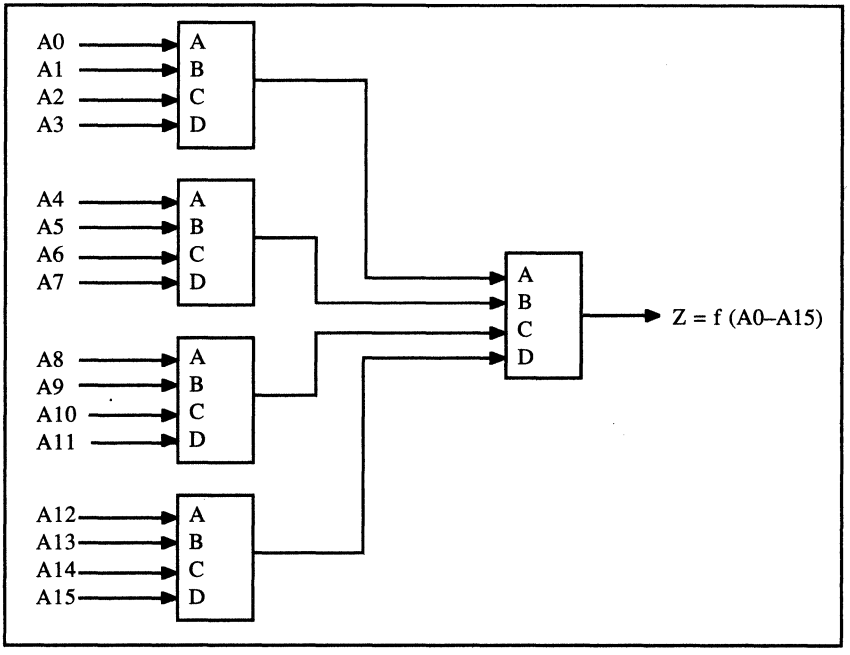


Dual Compare CLB

3.3.2 COMBINE OR SHARE CLBs

You can combine several CLBs to create an expanded function, or two different CLB functions can share a single CLB.

By using multiple CLB levels, you can expand the basic CLB design element of four input variables. For example, one CLB driven by four others can produce the sixteen-variable function shown below. If you select the decodes to use common terms in several functions, you can share those CLBs.

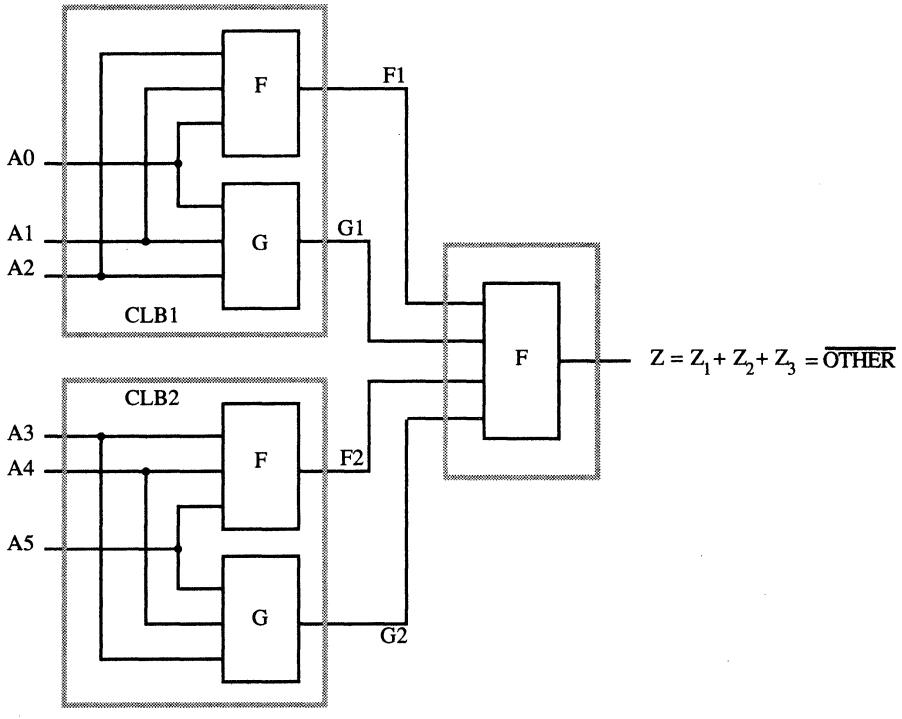


A Function of 16 Variables

You can use a related technique to encode the results of a pair of three-input, two-output CLBs. Use one of three output codes to indicate which of three selected input conditions exist. Possible combinations of the two CLB outputs can represent four conditions, namely

One, Two, Three, or Other. Each CLB encodes a three-input subset of the variables. When two of these first-level codes are input to another CLB, its result can be a complex function of six inputs.

The next figure shows **two encoded results, each a function of three inputs**. Each CLB responds with the selected code when its inputs match its part of the desired minterm. A HIGH output indicates that both codes match the same selected value, yielding a sum of three six-variable products.

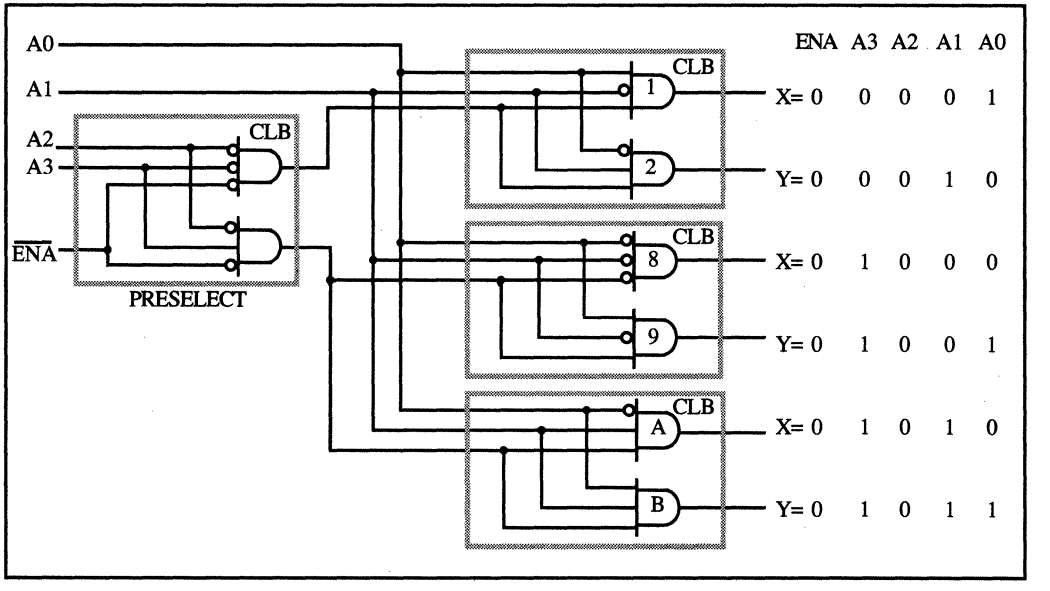


	CLB2		CLB1			
	INPUT	OUTPUT CODE	INPUT	OUTPUT CODE	CODE	RESULT
	$A_5 A_4 A_3$	$F_2 G_2$	$A_2 A_1 A_0$	$F_1 G_1$	$F_2 G_2 F_1 G_1$	Z
Z1	0 0 1	0 1	0 1 0	0 1	0 1 0 1	1
Z2	1 0 0	1 0	0 1 1	1 0	1 0 1 0	1
Z3	1 0 1	1 1	1 1 0	1 1	1 1 1 1	1
$\overline{Z}$	OTHER	0 0	OTHER	0 0	$F_2 G_2 \neq F_1 G_1$	0

Encoding Partial Results of Six Variables

If a design does not require some part of a conventional logic function, you need not create that portion of the function. For example, the above design of the encoder does not use all of the decodes of a set of input variables, so you can omit the unused decodes.

In an output-intensive function, using each CLB to generate two functions of three shared variables can be more efficient than generating one function per CLB. The next figure illustrates this technique. The **PRESELECT enabling gates** created in this figure are an example of a common term of a wider input function. To improve system speed, you can use the input variables that become stable first at the first level of logic. Those variables can propagate, while the design's more timing-critical inputs drive the shorter propagation path.



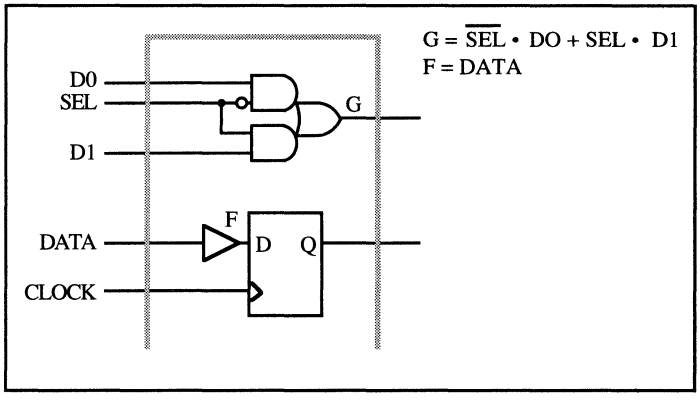
Decoder with Common Term and Only Required Outputs

You can build **wide multiplexer functions** from a tree of 2-to-1 multiplexers. This kind of structure,

illustrated below, leaves the storage element and one input variable of each CLB available for use as an independent register function. In other cases, the multiplexer

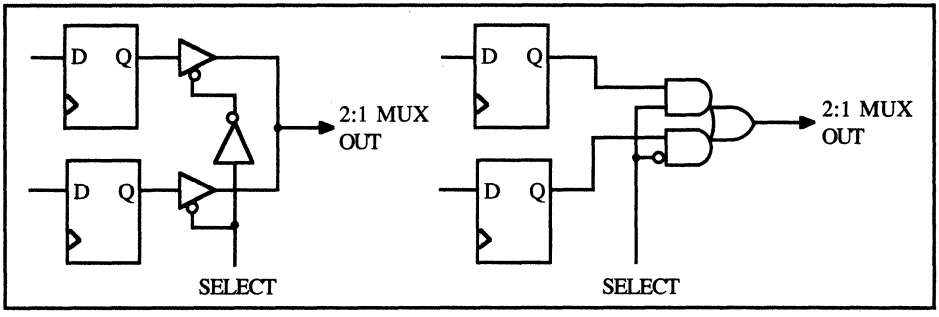
- can be the data input to the storage element,
- can share input variables, or
- can use the output of the storage element.

These examples provide a natural grouping of shared functions in a CLB.



CLB Sharing MUX and Register Element

A common element in digital systems is a **group of registers with sets of enabled output buffers bused together**. The structure shown below is not always recognized as a multiplexer; however, the multiple sources provide the inputs, and the enables represent the select lines. All inputs driven by the bus are driven by the multiplexer output.



Three-State Function Creates a MUX

3.4 CLB TIMING

This discussion explains the general timing factors in CLB design. It then discusses the specific timing for various CLB functions, including latches, flip-flops, registers, and counters. It also compares the timing of synchronous versus asynchronous design, and timing considerations for asynchronous inputs and clock skew.

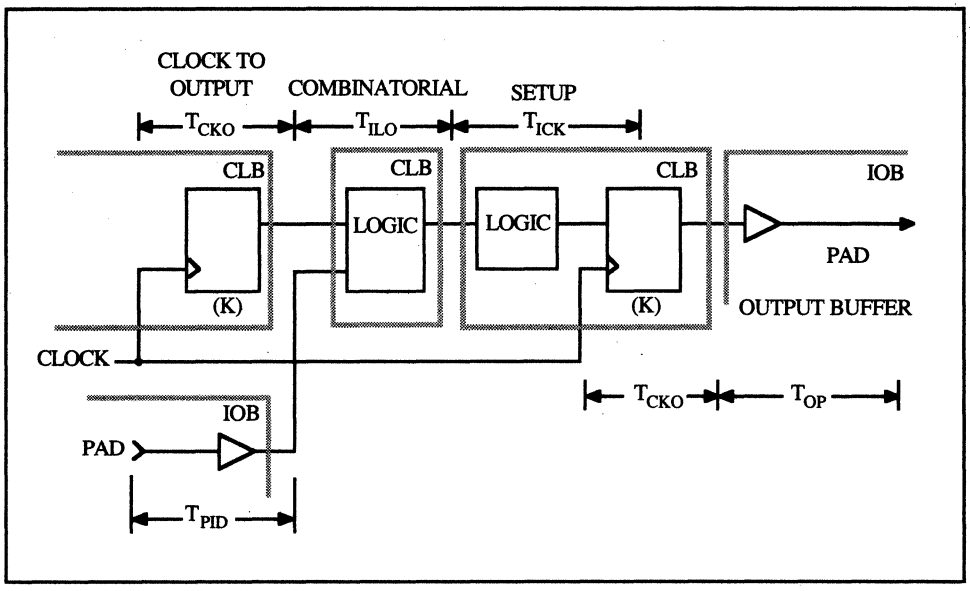
3.4.1 TIMING FACTORS

Any Boolean function generated by a CLB has the same timing delay as any other CLB-generated function. The concept of levels of logic or gate delay loses its significance with LCA devices, in which higher-level primitives perform logic.

The primary timing factors involved in designing an LCA device are listed below.

- The propagation time of a CLB
- The clock-to-block output via Q
- The input setup time for the CLB flip-flop input variables
- The input and output pad buffer delays
- The interconnection timing

Although other switching characteristics are specified in the LCA data sheet, the timing factors listed above are the most important in determining LCA performance. Some of these factors are illustrated and discussed below.



LCA Timing Factors

MSI devices typically have matched internal delay paths and low-impedance outputs that are independent of loading. Logic delays are more sensitive to output loading in programmable CMOS array architectures than in bipolar devices. As with CMOS gate arrays, variations in internal signal delays are significant in the LCA device. Synchronous design techniques can minimize the complexities of signal timing caused by delay accumulations in CMOS designs. An additional advantage of synchronous design is better control of output timing.

The clock distribution resources of the LCA device simplify synchronous design. When you can program any function of the input variables, it is simple to include such control signals as RESET, CLOCK ENABLE, and PARALLEL ENABLE in the logic function for the data input of flip-flops. All flip-flops can then use a common clock. With the flexibility of the LCA device you can generate and use individual CLB clocks, as well as

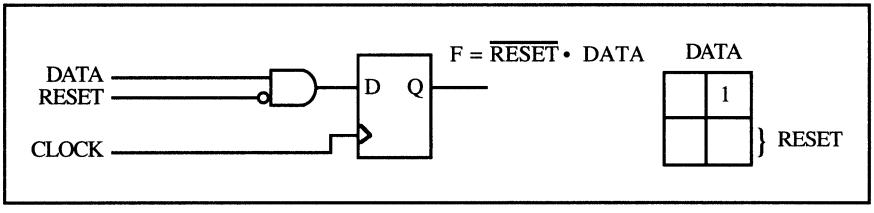
asynchronous SET and RESET, if required by the application.

3.4.2 LATCHES, FLIP-FLOPS, AND REGISTERS

LCA configurability lets you tailor the storage elements of the CLBs to fit your applications. Together with complex combinational data functions, this configurability lets you construct a wider variety of latches and flip-flops in LCA devices than is found in standard parts or gate array cell libraries.

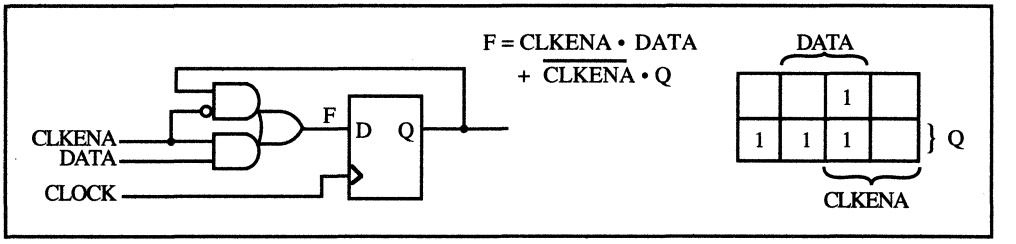
The level-transparent form of the storage element is the **D latch**. The edge-clocked form is the **D flip-flop**. In both cases, the function F supplies the data input, and the K or C pin or the function G supplies the clock (LOAD ENABLE). You select the data input, the clock, and the active sense of the signal on a block-by-block basis.

Including a RESET variable in the combinational input of a flip-flop produces a synchronous RESET, as shown below.



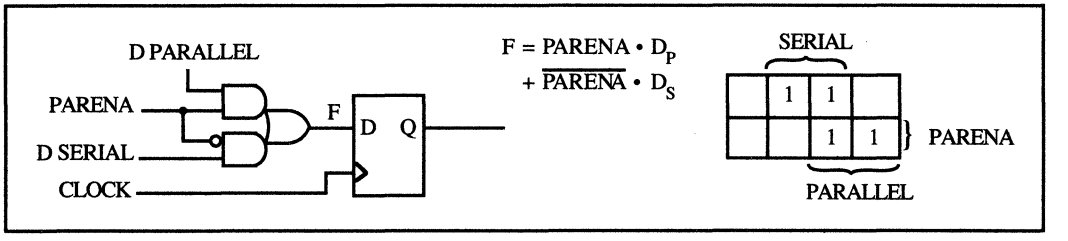
Synchronous Reset

You can use a combinational function of Q with input variables to generate a CLOCK ENABLE, as illustrated in the next figure.



Synchronous Clock Enable

As shown below, you can also create a multiplexer as the input of a flip-flop to provide a PARALLEL ENABLE.



Synchronous Parallel Enable

A flip-flop can have parallel data or reset inputs that do, or do not, depend on the clock enable. As with the J-K flip-flop, an interesting derivative of the set-reset flip-flop is one that does **not** change state in the case of simultaneous set and reset conditions. The other types of flip-flops are set-dominant or reset-dominant. The availability of this variety of synchronous set-reset flip-flops provides you with alternatives for logic creation that can help you minimize next-state control conditions.

A group of related flip-flops with similar functions can form a register. You can group registers into two categories, namely data registers and shift registers. A data register is a set of flip-flops with independent parallel input paths and common control. A shift register is a set of flip-flops with a serial data relationship. Both

data and shift registers consist of combinational variations of signals supplying the data input of the basic, edge-triggered D flip-flop.

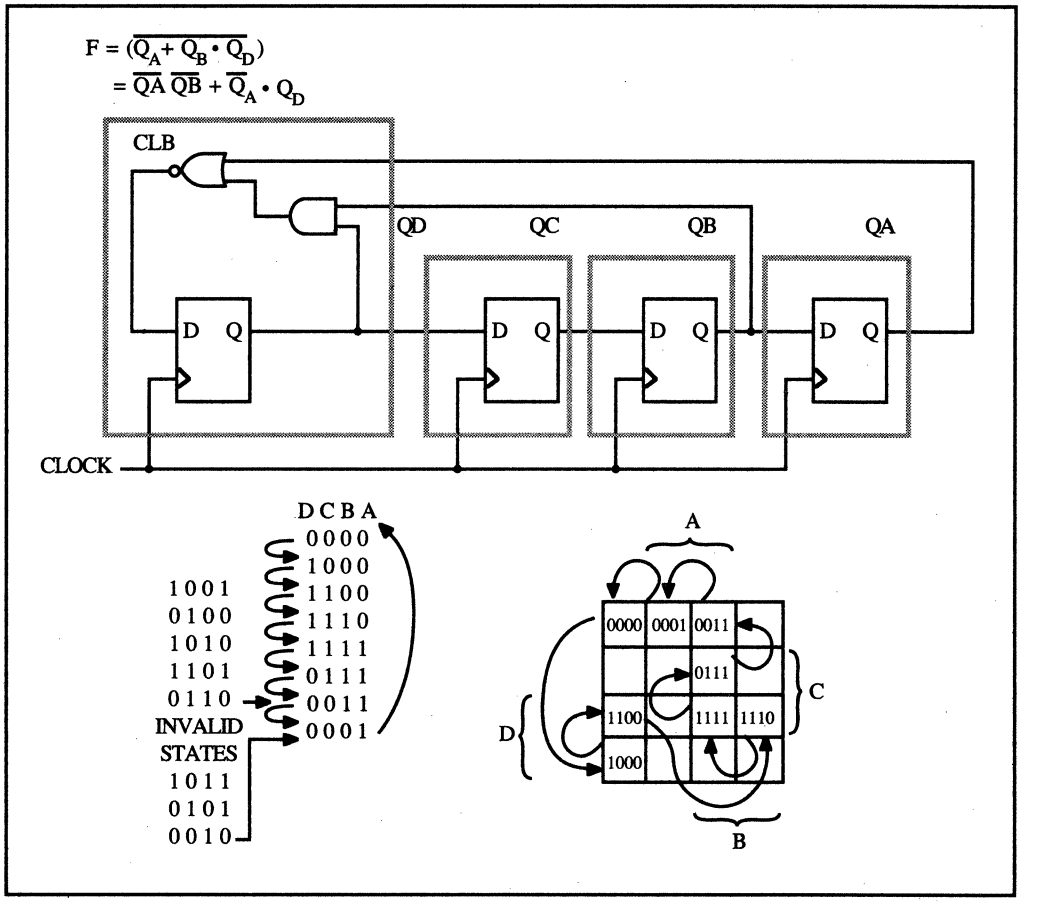
3.4.3 COUNTERS

Counters are a simple example of a state machine with a regular sequence. The most familiar counters are the Johnson or Mobius counter discussed under 3.4.3.1, the binary weighted sequence discussed under 3.4.3.2, and the Linear Feedback Shift Register discussed under 4.2.6.7.

3.4.3.1 Johnson Counters

Johnson counters often offer advantages for counter designs with a modulo of less than 10 to 12. Also, they are simple to place and route, and the basic combinational functions shown below are compatible with maximum clock frequency.

In a Johnson counter, decodes of single or consecutive states are simple and glitch-free. Initializing the LCA device clears all storage elements. However, due to the presence of unused states, the Johnson counter could enter an alternate state sequence if there are any asynchronous control inputs. As shown in the next figure, additional input variables from QB and QD in the feedback function can return the count to the proper sequence.

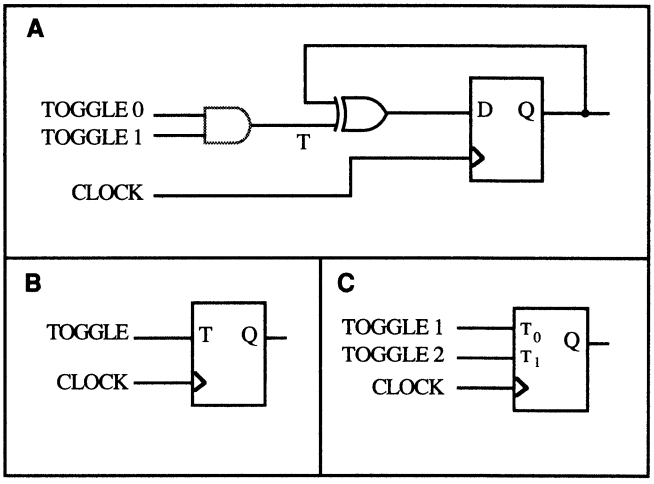


Divide-by-8 Johnson Counter

3.4.3.2 Binary-Weighted Sequence Counters

When creating a large-modulo CMOS binary-weighted counter, you are presented with a number of trade-offs. In terms of physical resources, the most effective counter of this type is a simple ripple counter; however, the accumulation of multiple clock-to-Q flip-flop delays can be prohibitive because the outputs ripple for varying times.

A synchronous toggle flip-flop is shown below.



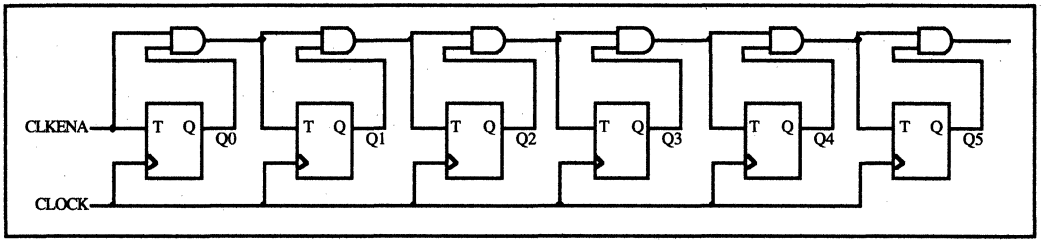
Toggle Flip-Flop

A. Changes state synchronously if T is HIGH.

B. Shows the simplified symbol for the flip-flop.

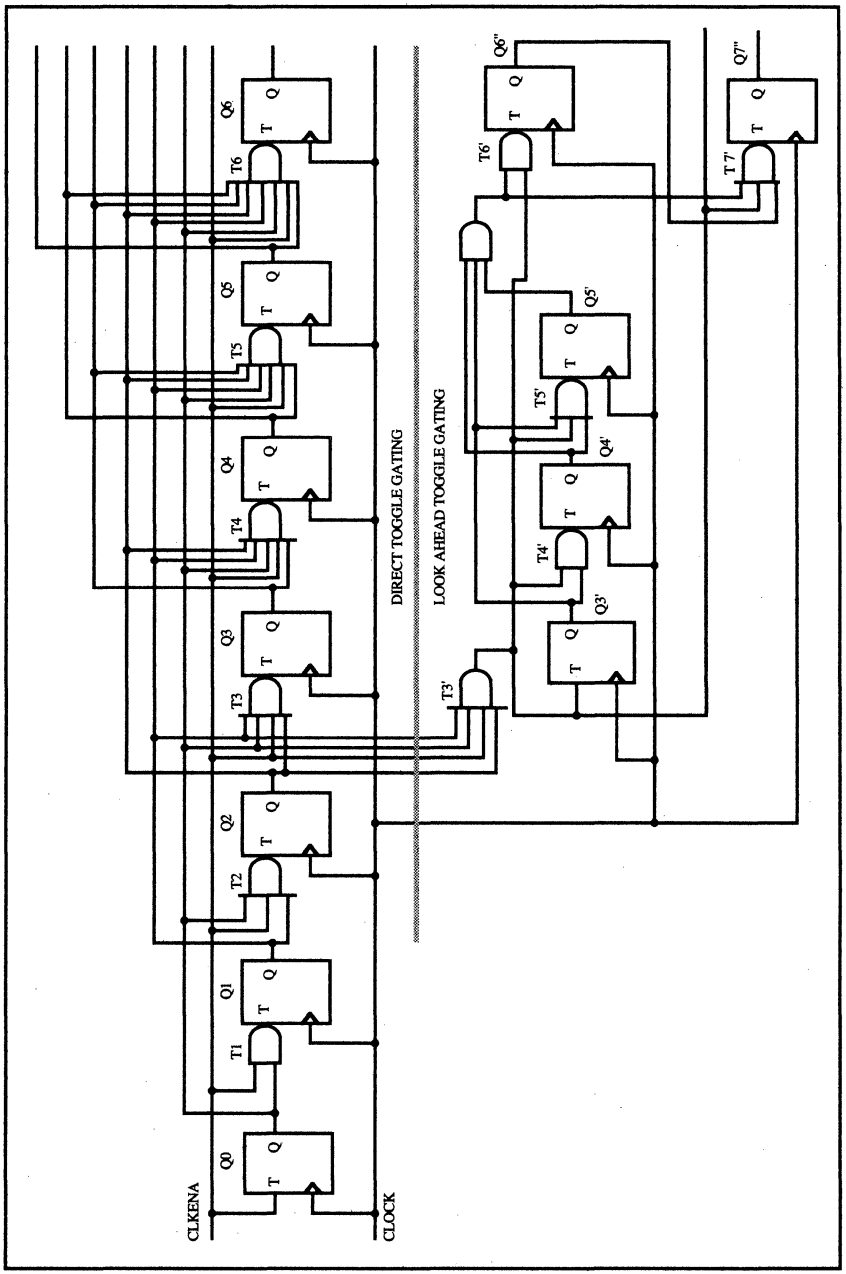
C. Illustrates the AND of two inputs, which produces the T.

The following figure shows a fully synchronous counter solution composed of T flip-flops. This design generates the counter's toggle ripple carry for each bit by adding a carry gate, which tests the previous toggle carry and the state of its flip-flop in a daisy-chain fashion. The counter's maximum clock rate is determined by the total propagation time for the carry path from CLKENA to data setup of the last bit.



Synchronous Binary Counter with Ripple Carry

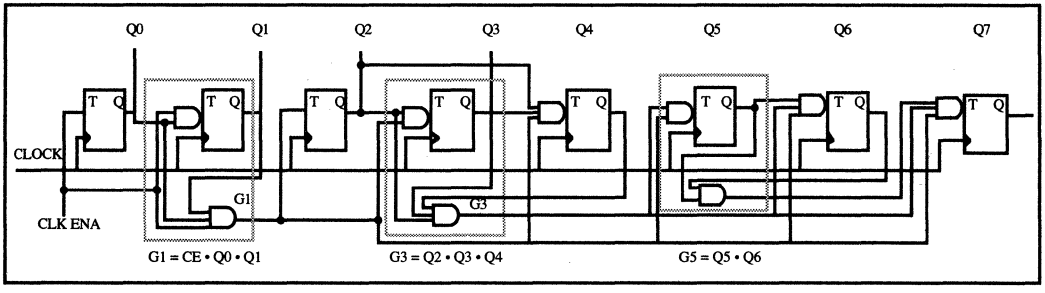
The fully parallel counter shown next generates each toggle function directly. This counter design requires an n-wide gate for toggle control of each bit of the counter. You can extend this design to 12 bits. The total delay would then consist of one combinational propagation delay between register CLBs, one clock-to-Q delay, one set-up time delay, and the interconnection delay.



Synchronous Binary Counter

When designing counters, you can use block-level carries. The 3-bit segment size shown in the lower half of the above figure accommodates the three stages and a carry-in within the single-CLB 4-input limit. The only combinational delays are those of T3' and T6'. Without a clock enable input, the first section could be 4 bits, followed by 3-bit sections. When designing LCA devices, you must watch for design trade-offs and not try to fit a standard solution into all applications.

The figure below illustrates another synchronous 8-bit counter with a single level of combinational propagation delay. The figure shows the merging of the sequential and combinational elements of the CLB. This counter uses periodic look-ahead carry terms to make efficient use of variables within the block.



Eight-Bit Synchronous Counter Generated in CLBs

3.4.4 SYNCHRONOUS VERSUS ASYNCHRONOUS DESIGN

Efficient LCA-based designs can differ from MSI designs. MSI elements are general purpose building blocks that exploit the strengths of a different technology. Most MSI parts are designed to fit a set of standard package sizes and the pin functions are chosen to provide a useful standard product. The LCA design goals and techniques are very different.

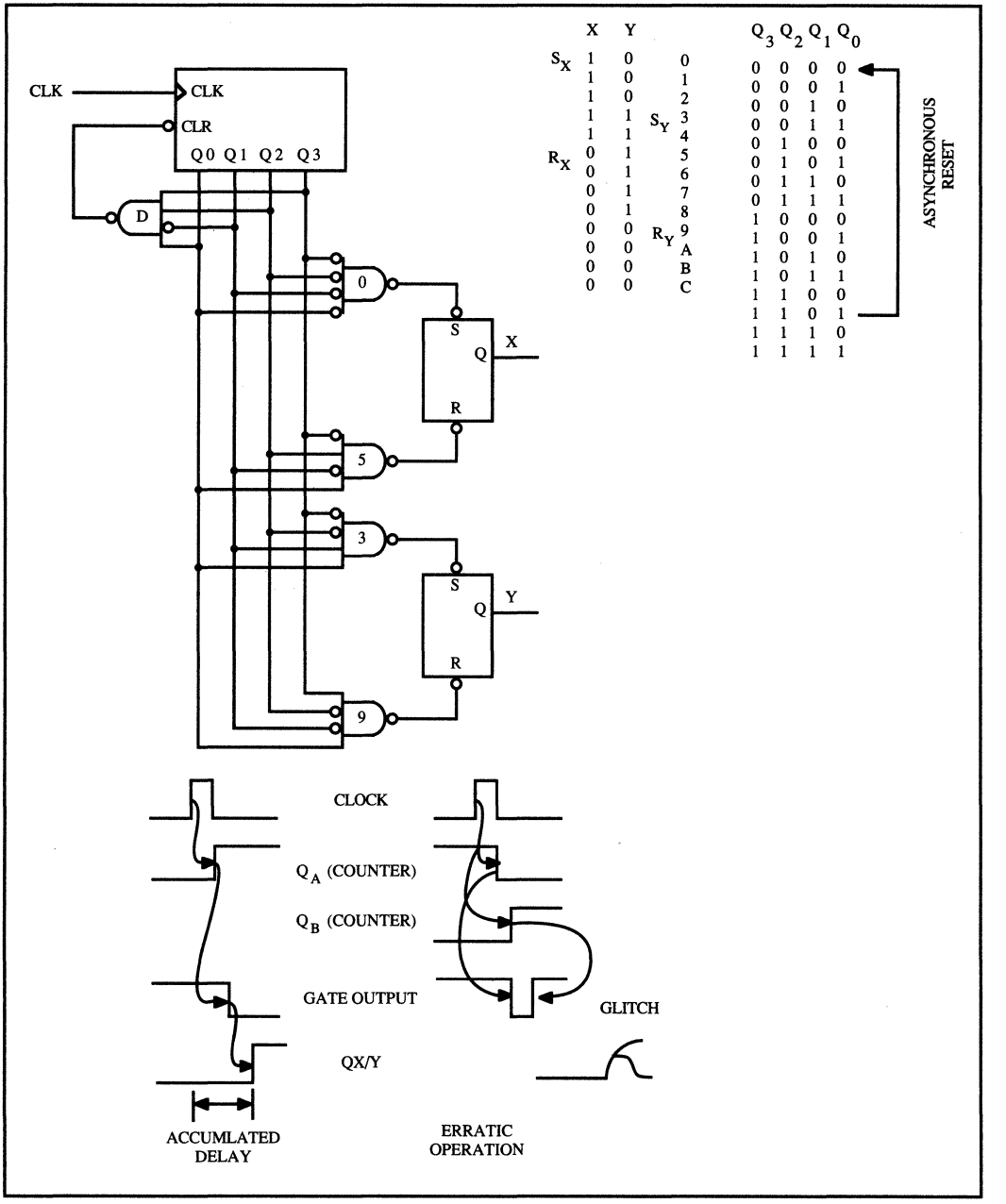
When designing LCA devices, your goal is to minimize the routing and number of blocks. In fact, you can often adapt your design's

logic to minimize the constraints of the available logic and routing resources and to optimize the LCA device's logic capacity and performance.

The three- and four-variable capability of the CLB is a good balance in LCA devices and gate arrays. You can use a conventional logic diagram of your design and group the combinational functions to give an approximate CLB count. In a register-intensive design, the number of flip-flops required by the design determines the logic capacity and related combinational functions merge with the sequential portions.

3.4.4.1 Asynchronous Ripple Counters

The ripple counter shown next incorporates a counter that sets and resets output control bits at specific times in the sequence. NAND gates that decode the desired states drive the asynchronous set and reset inputs of the flip-flops. When the counter increments to state D, it should asynchronously reset to 0.



Simple Asynchronous Ripple State Machine

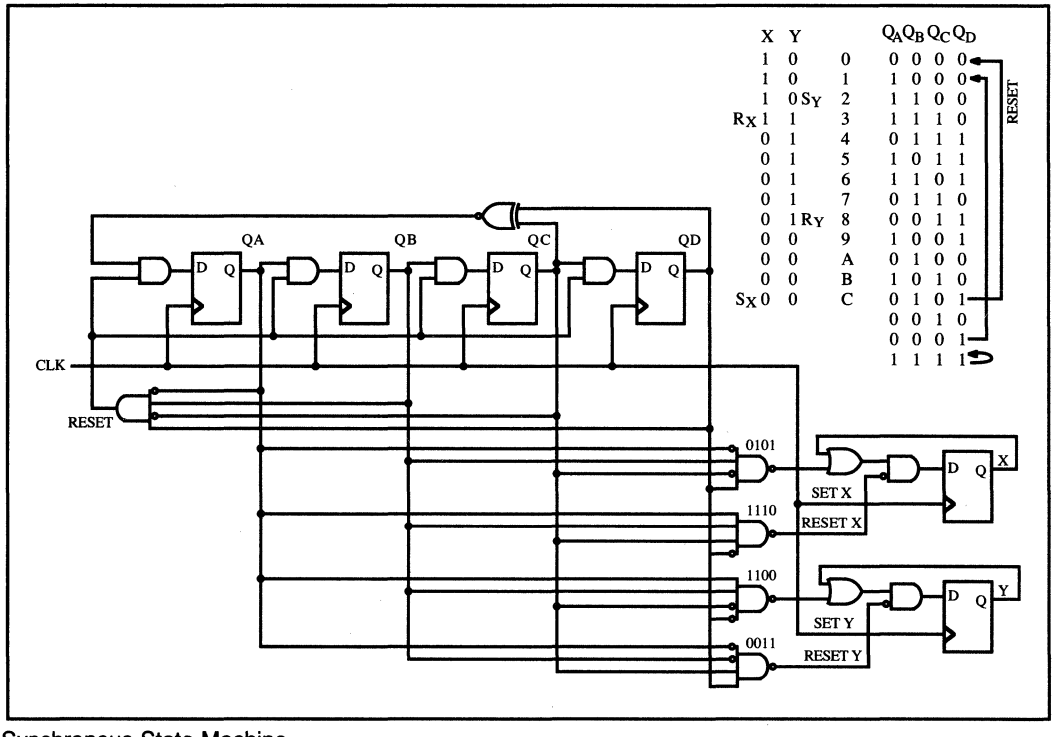
The above counter might operate properly if it were an MSI device because the counter bits in such a device are matched. To use this counter in a gate array or LCA device, however, the decodes of the counter states involve mismatched loading and layout of various counter bits. As a result, the decode gates are likely to produce output spikes, causing erratic operation of the output control flip-flops that use these signals as asynchronous inputs. Although the decode spikes can be so narrow that you do not notice them during design verification, they might produce erratic output control changes during operation. The decode of the terminal count also has the potential for spurious outputs. Even with a valid terminal count decode, a mismatch in counter bit speeds could result in **some** bits resetting and terminating the reset state decode signal before **all** bits of the counter are reset. This timing problem could leave the counter in an undefined or incorrect state. A reasonable alternative to the asynchronous ripple counter is the synchronous linear feedback shift register, which is discussed next.

3.4.4.2 Synchronous Linear Feedback Shift Registers

The following figure shows a Linear Feedback Shift Register, which is a fully synchronous alternative to the asynchronous-reset binary counter. This class of counters follows a less familiar sequence, but its decodes of specific counts are predictable. Use of OR/AND feedback for inputs on the output flip-flops results in a synchronous SET/RESET function for the output control bits, making them immune to decoding spikes.

Notice that the resulting X and Y sequences are identical, although the counter sequences differ and the control decodes of the synchronous version represent the state before X or Y transition. This synchronous design revision also lets the clock control the output timing; in the ripple counter of the previous figure, the output timing is controlled by the

accumulation of delay from the clock to the counter output, then to the state decode, and finally through the flip-flop to the output. The use of a fully synchronous counter reset provides more reliable counter operation, and you may be able to increase the maximum clock rate due to the elimination of the terminal count delay path. Generating the flip-flop synchronous reset requires no more resources than that of the asynchronous reset described above for the ripple counter.

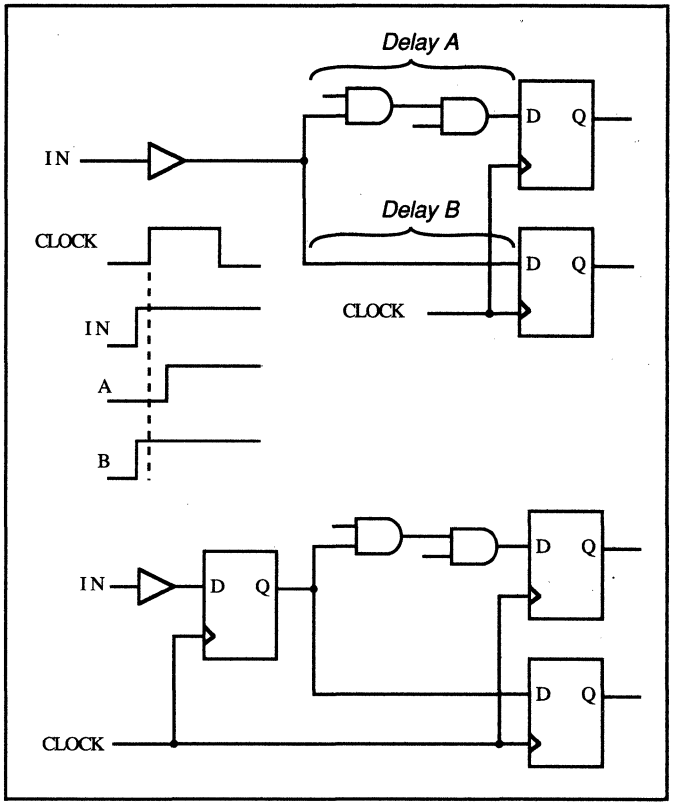


Synchronous State Machine

3.4.5 ASYNCHRONOUS INPUTS

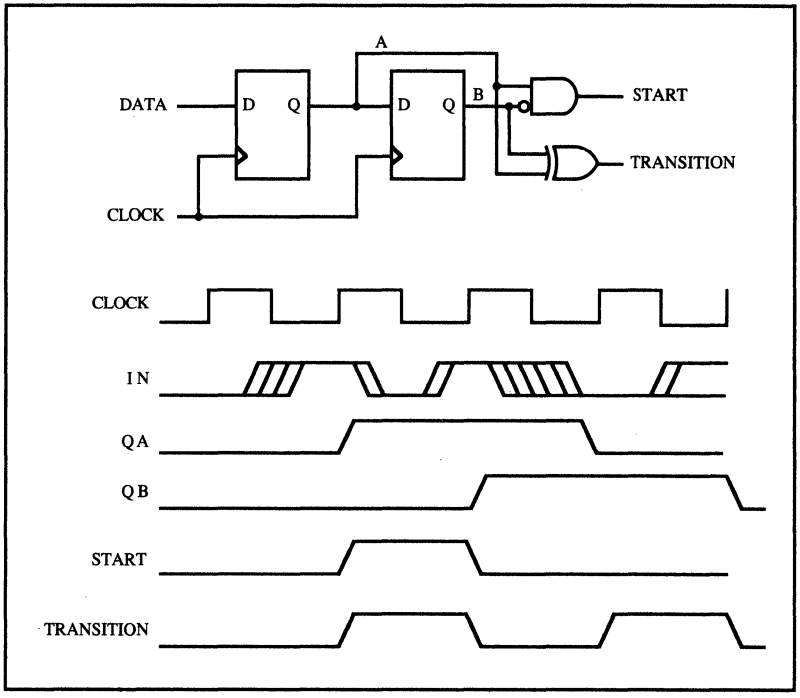
Another common source of design problems is the timing of asynchronous input signals that affect more than a single flip-flop. An example is a counter in which an asynchronous parallel load is removed near a clock edge. Various bits of the counter may change in response to the clock, while others retain the previous state. The result is an invalid value in the counter.

Another common design problem is that of an asynchronous system reset. If the reset signal is removed near a clock-edge, different parts of the logic may respond differently, resulting in invalid states as the logic tries to begin operation. Clearly, you should always synchronize asynchronous signals at their input, as shown in the following figure.



Synchronization of an Asynchronous Input

You can accommodate asynchronous inputs that require a response to their transition by using a resynchronizer, as illustrated below. The additional input delay this solution imposes may be undesirable in some applications but it results in a more reliable design when input latency is not a limiting factor. The resynchronizer also acts as a simple noise filter.



Data Resynchronizer and Filter

3.4.6 CLOCK SKEW

The problem of clock skew accounts for numerous gate array design iterations. Clock skew problems are caused by mismatched delay paths. For example, you could clock one flip-flop, and its new output level, lightly loaded, could propagate to another flip-flop input, arriving as much as one set-up time before the original clock reaches a second flip-flop. The difference in clock timing can occur because of clock gating or unequal routing delays. To minimize clock skew in an LCA device, the LCA clock buffers drive a dedicated metal clock distribution network.

3.5 LOGIC DESIGN WITH XACT MACROCELLS

Macrocells, also called macros, are predefined CLB or IOB configurations that use common logic functions. The LCA development system includes a library of macros that you can use to create your LCA designs. You can also create your own macros. This discussion provides detailed information about macros and gives you an overview of how to create one. It also shows some sample macros.

3.5.1 MACRO OVERVIEW

Using macros when you optimize your LCA design is a quick way to specify a function. Each macro is actually a file that contains all of the executable **EditLCA** commands required to define the macro's function in the LCA design. When you invoke a macro, you provide the set of parameters needed to execute the file in the required order.

The parameters you supply customize each occurrence, or **instance**, of the macro in your design. The parameters include such information as an instance name, the names of networks providing inputs, and the block locations for each CLB or IOB in the macro. The instance name is used during macro execution to compose unique block and net names that distinguish each occurrence of a particular macro in your design.

Macros in the AMD macrocell library are stored in the \MACROS directory. Each macro file has an assigned name with a .MAC file extension that identifies the logic function. You can find a list of the macros in the AMD library and the order of their required parameters in the Quick Reference Card and the LCA Macrocell Library Manual. The macro documentation indicates the required parameter order in the syntax statement for each macro.

In addition to using the available macros, you can create customized macros as follows.

3.5.2 MACRO CREATION

EDITLCA commands and macro executions let you create an LCA design as well as modify macros. After editing, you may want to incorporate part of that new design or modified macro into a new macro. For example, you might create a new macro that is a one-bit slice of useful logic and which may include several CLBs. Then you can place several instances of that macro to create a more complex logic unit, such as a data path. Another useful user-defined macro could describe a section of a special counter that generates a unit of control logic.

To create a macro, you first use the keyboard or mouse to specify the individual blocks that must be included. The LCA development system assigns a parameter for each network that the user has selected as a macro input, as well as for each CLB and IOB that this macro uses. All block names and net names sourced by the macro blocks are included in the new macro's .MAC file, which is created in the current directory.

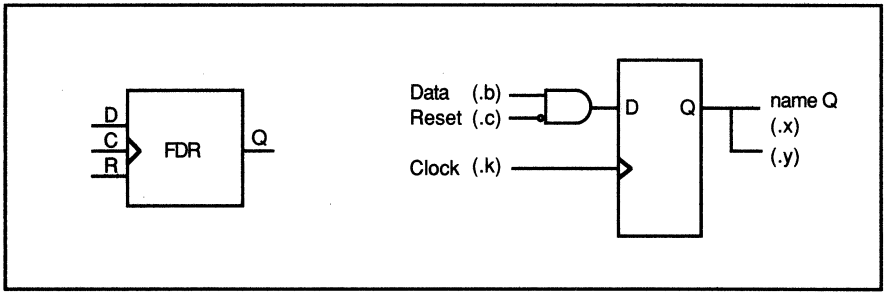
When you invoke the macro, you are prompted for parameters in the order they are needed. The first required parameter is always the **instance name**. This name differentiates one instance of a macro from any other instance of the same macro in the design. The instance name is added as a prefix to the macro's original net names for all nets driven by blocks included in the macro. The instance name is also added as a suffix to the original block name of all blocks in the macro to allow the first characters of the block names to show in the editor display.

Note: All block and net names must be unique. Reserved names include A, B, C, D, K, I, O, X, and Y, which are already used for block pin names. Also reserved are AA through last row/column of the LCA design, and P1 through highest pin number of the LCA design, as block names. Some additional reserved names are assigned to configuration and power pins.

Refer to Chapter 4 in the LCA Development System manual Volume I for specific information on how to create macros.

3.5.3 SAMPLE MACROS

Several sample macros are illustrated below. The first figure shows the logic diagrams for FDR, a simple D flip-flop with synchronous reset.

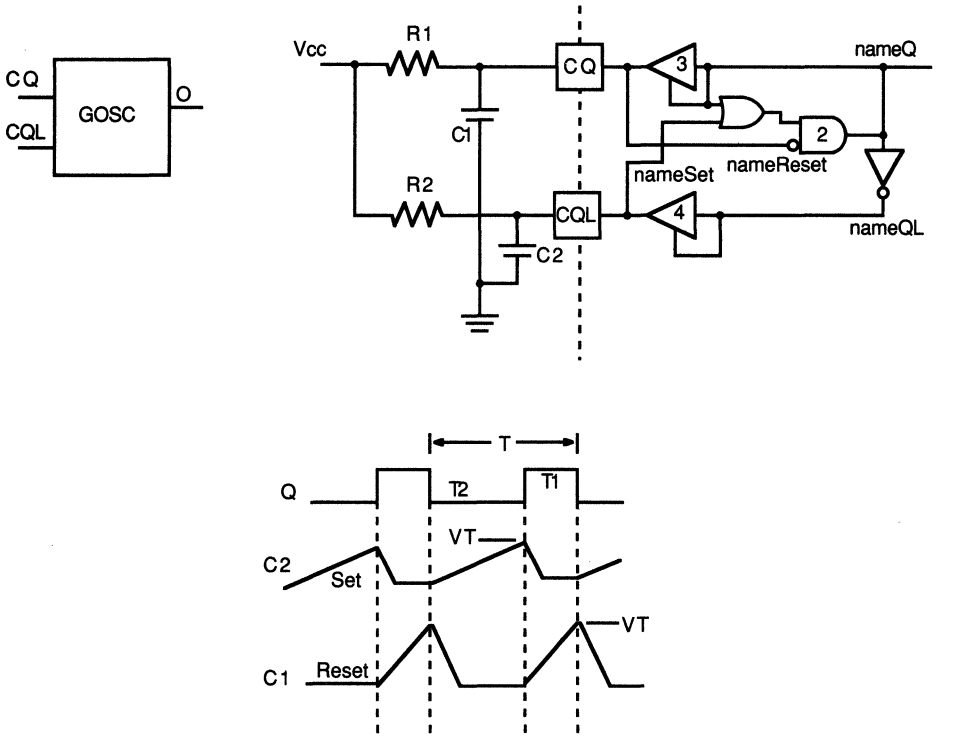


The macro for this figure is shown below. The first two lines in the macro are comment lines indicating the syntax and parameter order for macro execution. The comments in the macro are self-explanatory.

Note: %1 through %5 represent the parameter values you supply.

```
;MACRO          FDR    Name    Clock    Data    Reset    Location        nameblock

;                      %1      %2       %3      %4       %5              NAME

Parameter NAME  ?  Enter instance name:                    | Parameter statements specify
Parameter NET Clock Select Clock net:                     | parameter type, the default
Parameter NET Data Select Data net:                       | names for nets, followed
                                                            Parameter
Parameter NET Reset select Reset net:                     | by the Select prompts
                                                          | for the editor screen.
Parameter CLB  ?  Select  %1  block:

Nameblk  %5  %1                                           | Editor commands to name the
Editblk  %5                                               | block  %5  (fifth parameter)
Base 3var                                                 | with the instance name  (%1)
Config X:Q  Y:Q  F:B:C  G:  Q:FF  SET: RES:  CLK:K        | Edit the block (%5) and define
Equate F = B*~C                                           | its configuration and equation.
Endblk

Addpin  %2  %5.K                          | Addpin commands define the nets.  The first
Addpin  %3  %5.B                          | parameter variable is the name (or default)
Addpin  %4  %5.C                          | supplied by that parameter in the installation
Addpin  %1Q  %5.X                         | statement.  The  %1Q is a Q concatenation on
                                          | the instance name  %1.
```

The GOSC macro in the following figure is a simple oscillator that uses two external R-C networks, two IOBs and one CLB that functions as a set-reset latch.



T = T1 = T2 = N ((R1 C1) + (R2 C2))
where N = approx. 0.35 for TTL threshold
= approx. 0.75 for CMOS threshold
when each capacitor is allowed to be discharged by the LCA during opposite timing phase. Capacitor might partially charge due to a delayed 3-state routing.

The macro for this GOSC is shown below. Refer to Chapter 4 Volume I of the LCA Development System manual for specific information about how to use and create macros. Also, refer to the discussion in Chapter 2 of this manual for details about the AMD-supported logic libraries used in schematic entry of LCA designs.

```
:macro    GOSC          Name          LocQ          LocCQ          LocCQL

;                       %1            %2            %3             %4

Parameter NAME  ? Enter instance name:                    | Parameter statements
Parameter CLB   ? Select  %1  CLB block:                  | defining parameter type
Parameter IOB   ? Select  CQ%1  I/O block:                | and screen prompt.
Parameter IOB   ? Select  CQL%1  I/O block:

Nameblk  %2  %1                                           | Assigns the first
Editblk  %2                                               | parameter (%1) as
Base 3var                                                 | block name to block
Config X:F  Y:G  G:A:C:B  G:A  Q:  SET:  RES:  CLK:       | specified by (%2)
Equate F = ~B* (C+A)                                      | and configures it.
Equate G = ~A
Endblk

Nameblk  %3  CQ%1                                         | Assigns the CQ prefix
Editblk  %3                                               | to instance name for
Base IO                                                   | the block selected as
Config  I:PAD  BUF:TRI                                    | the third parameter
Endblk                                                    | and configures it.

Nameblk  %4  CQL%1                                        | Assigns the CQL prefix
Editblk  %4                                               | to block name for the
Base IO                                                   | fourth parameter and
Config  I:PAD  BUF:TRI                                    | configures it.
Endblk

Addpin  %1Q  %2.X  %2.A  %3.0  %3.T                       | Creates nets of names
Addpin  %1Reset  %3.I  %2.C                               | with concatenation to
Addpin  %1Set  %4.I  %2.C                                 | the pins .x, .a, .o, .t
Addpin  %1QL  %2.Y  %4.O  %4.T                            | etc. of the blocks
                                                          | identified by the %2, %3,
                                                          | %4 parameters.
```

CHAPTER 4

INPUT/OUTPUT BLOCKS

4 INPUT/OUTPUT BLOCKS

This chapter discusses IOBs, which comprise the LCA I/O structures. The chapter has the following structure.

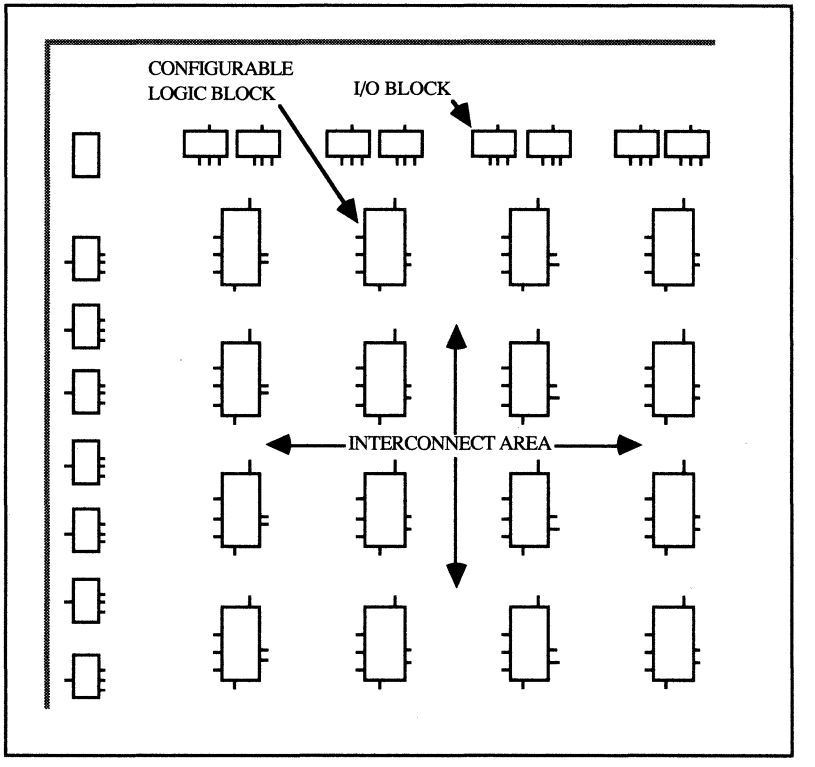
- The I/O block overview, 4.1, introduces IOBs and discusses some specific IOB operating characteristics.
- The discussion on LCA I/O Structures, 4.2, illustrates the wide variety of I/O structures available for use in LCA designs.

4.1 I/O BLOCK OVERVIEW

This discussion explains LCA Input/Output Blocks, registered inputs, and metastability.

4.1.1 IOB INTRODUCTION

The IOBs in an LCA design surround the array of CLBs, as illustrated below. Where the CLBs are the logic building blocks in an LCA design, IOBs are the building blocks for LCA input, output, and bidirectional I/O structures.



Logic Cell Array Structure

The 2064 and 2018 IOBs are identical. Each IOB can

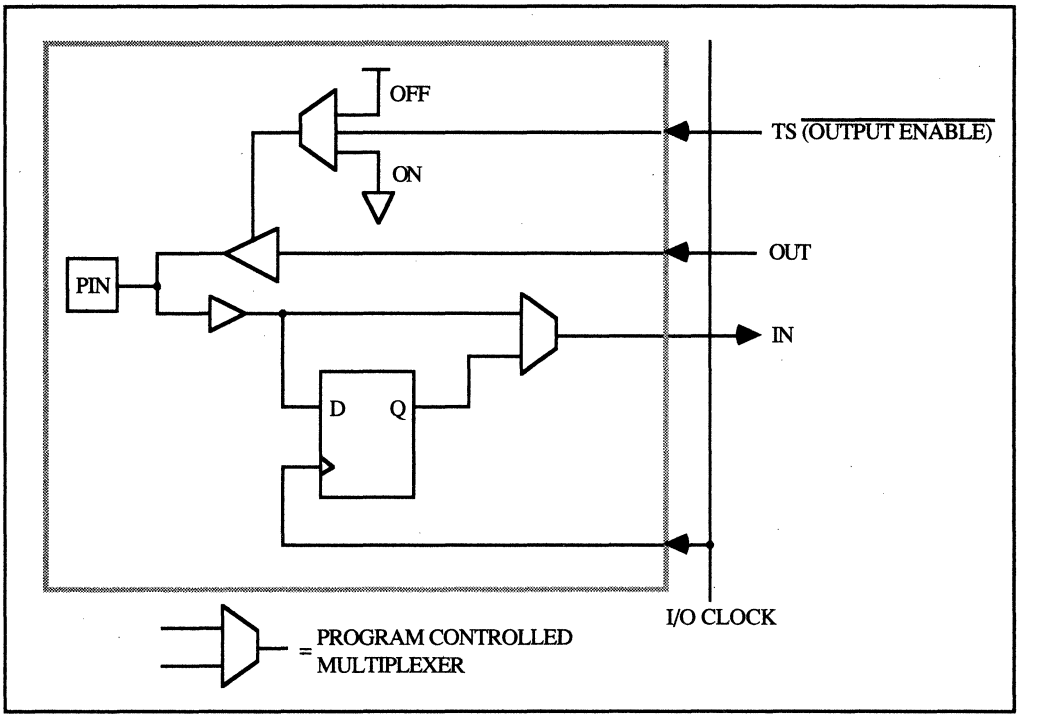
- drive an output,
- receive an input,
- clock the input into a flip-flop, or
- be both an input and an output under three-state control.

You can configure an IOB to perform a variety of logic functions.

The architecture of the LCA device provides great design flexibility in using inputs and outputs. The IOBs in an LCA device are not dedicated to any fixed logic. Therefore, you can use IOBs for logic structures beyond simple inputs or outputs. Often, designs do not use all IOBs available within the LCA device. You then can use the extra IOBs to build such logic structures as shift registers or Johnson counters.

The following figure shows the schematic of an IOB. The trapezoidal structures in the figure are data-path selectors or multiplexers. How you program these data-path selectors determines what function the IOB performs. For example, you can configure the IOB to perform as

- a direct or registered input,
- a direct or three-state output, or
- a bidirectional data line.



Input/Output Block (IOB)

4.1.1.1 Input Signals

Along each edge of the LCA die, the IOBs share a common I/O clock signal that drives each input register. All internal registers are reset to a zero state after configuration, or after the ~RESET pin is asserted LOW. Data is clocked into the input register on the positive edge of the I/O clock signal.

A logic signal external to the LCA device comes in through an I/O pad and non-inverting buffer, as shown above. The logic signal is then either directly propagated or fed into the input register, depending on the configuration of the data-path selector.

4.1.1.2 Output Signals

Similarly, output data is driven by the non-inverting buffer shown in the previous figure. The output buffer is forced into a high-impedance state whenever the three-state (TS) control line is HIGH (TS = 1). Conversely, the output buffer propagates the output signal when the three-state control line is LOW (TS = 0). All outputs can source and sink 4 mA under specified worst-case conditions.

4.1.1.3 Voltage Levels

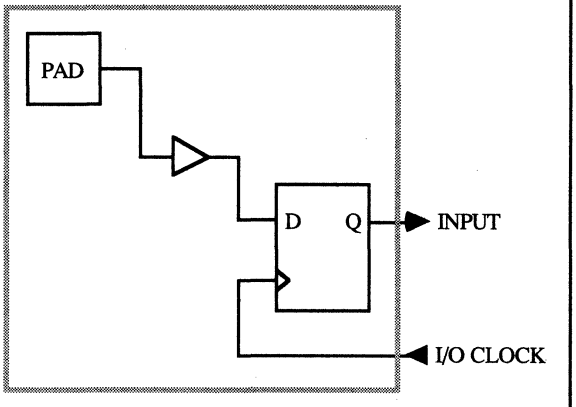
You can configure all IOBs to recognize either TTL-level (V_{TH} = 1.4 V) or CMOS-level (V_{TH} = 2.2 V) input thresholds. The selected voltage level affects overall device power consumption; power consumption is lower when you select CMOS input levels.

4.1.2 REGISTERED INPUTS AND METASTABILITY

The following schematic shows a registered input within the LCA device. LCA devices are manufactured with a high-speed CMOS process that allows these IOB input registers to achieve flip-flop **loop delays** of three to five nanoseconds. These short loop delays provide very good performance under asynchronous clock and data transitions. Short loop delays also minimize the probability of a metastable condition that can result when the input to the flip-flop is still in transition while the clock is asserted.

The IOB's short loop-delay characteristics make them effective in synchronizing external signals. After the IOB synchronizes the external signals, you can use the signals internally without further consideration of their relative timing, except as it applies to internal logic and routing-path delays. Chapter 7 of this manual provides further information regarding the metastable behavior of flip-flops and registers in an LCA device.

I/O TYPE: Pad Input with Storage (registered input)
MACRO NAME: PINQ
SCHEMATIC:



CONFIGURATION:
I:Q
BUF:

Input Pad with Storage Register

LCA I/O Structures are discussed next.

4.2 LCA I/O STRUCTURES

This discussion explains some I/O and other logic functions available by configuring IOBs in various ways.

- Discussion 4.2.1 explains standard I/O structures.
- Discussion 4.2.2 explains open-collector structures.
- Discussion 4.2.3 explains Schmitt-trigger structures.
- Discussion 4.2.4 explains general purpose oscillator structures.
- Discussion 4.2.5 explains on-chip crystal oscillator structures.
- Discussion 4.2.6 explains registers and counters.
- Discussion 4.2.7 explains increased drive current structures.

The structures described below use the following conventions for input paths and output buffers. Discussions here show each structure in schematic form and describe its IOB configuration.

You can configure the input path as any of these functions.

- I:PAD - Direct input from the device pad
- I:Q - Registered input
- I: - No input

You can configure the output buffer as follows.

- BUF:ON - Direct output
- BUF:TRI - Three-state output
- BUF: - No output

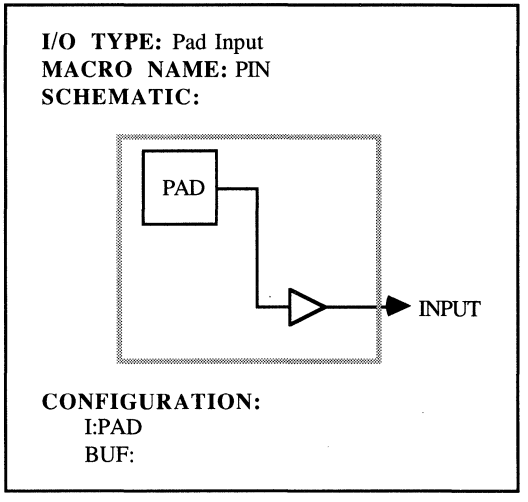
If an LCA library macro exists for any structures described below, the macro is listed in the schematic as *macro name*.

4.2.1 STANDARD I/O STRUCTURES

The six standard LCA I/O structures are listed and pictured below. These **standard structures** are the basic input, output, and bidirectional I/O configurations for an LCA device.

- Input pad
- Input pad with storage register
- Output pad
- Output pad with three-state control
- Bidirectional pad
- Bidirectional pad with input storage

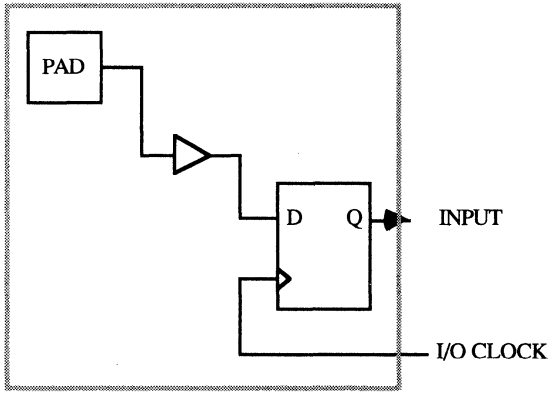
The input pad is shown first.



Input Pad

The next two figures show the input pad with a storage register and the output pad, respectively.

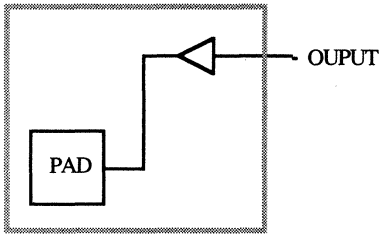
I/O TYPE: Pad Input with Storage (registered input)
MACRO NAME: PINQ
SCHEMATIC:



CONFIGURATION:
I:Q
BUF:

Input Pad with a Storage Register

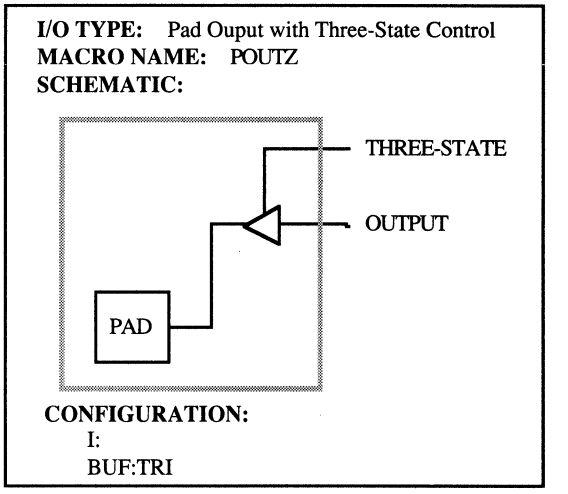
I/O TYPE: Pad Ouput
MACRO NAME: POUT
SCHEMATIC:



CONFIGURATION:
I:
BUF:ON

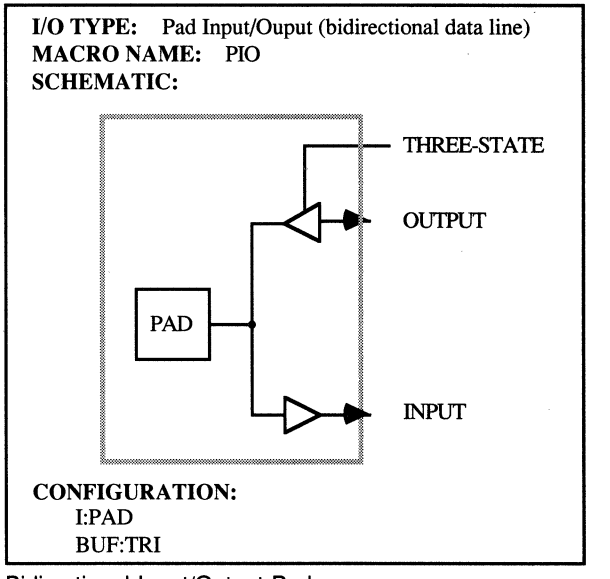
Output Pad

The figure below shows the output pad with three-state control.



Output Pad with Three-State Control

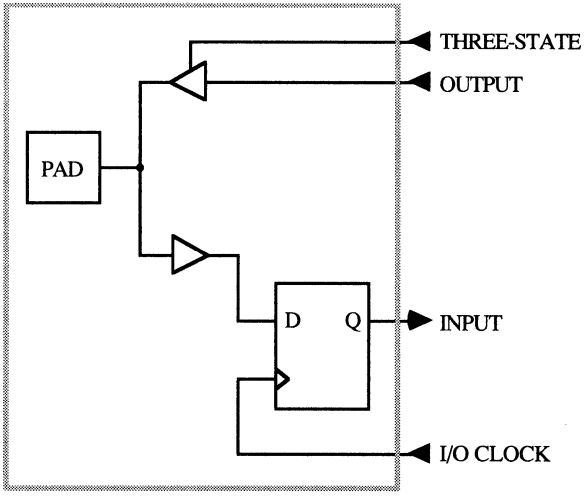
The next figure shows a bidirectional input/output pad.



Bidirectional Input/Output Pad

Next, you'll see a bidirectional pad with input storage.

I/O TYPE: Pad Input/Output with Input Storage
MACRO NAME: PIOQ
SCHEMATIC:



CONFIGURATION:
I:Q
BUF: TRI

Bidirectional Pad with Input Storage

4.2.2 OPEN-COLLECTOR STRUCTURES

The LCA macro library contains a variety of output functions, including wired-AND and wired-OR structures, based on available LCA **open-collector** structures. The MOS transistor has no collector; therefore, **open-drain outputs** is a more accurate term for MOS devices like the LCA device.

To build an open-drain-output structure in an LCA device, you tie together both the output and the three-state control lines. For an active-HIGH signal, the three-state control engages (high impedance), and the output signal is disabled through the output buffer.

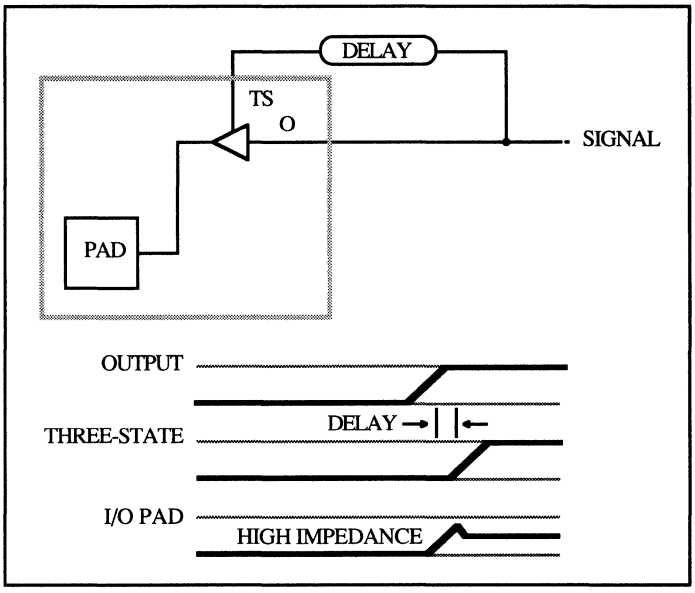
The signal at the output pad is also high impedance, which allows that particular signal line to float. Connecting this signal line to VCC through a resistor pulls this line up for an active-HIGH output. However, for active-LOW signals the three-state control line is driven LOW to turn on the output buffer and allow the LOW signal to propagate directly to the I/O pad.

4.2.2.1 Open-Drain Structures and Routing

When designing with open-drain structures, you should be aware of an LCA-specific phenomenon caused by the different routing delays between the signal source and the output and three-state control loads.

Because a routed signal may take longer to reach an IOB's three-state control line than its output line, the pad can be driven for a short period of time during a LOW to HIGH transition, as shown in the next figure. This could occur if the output line (O) starts to go HIGH before the three-state control line does. Depending on how much routing delay there is between the output and three-state lines, the PAD output could start to go HIGH and then be driven into a high-impedance state. Excessive routing delay differences between the output and the three-state control line may cause a brief output glitch, as shown below. Careful design prevents this.

The above situation is not a problem in most designs. You can check the actual routing delay difference between the TS and O terminals of an IOB using the delay calculator in the LCA development system. See Chapter 2 of this manual for information about the delay calculator.



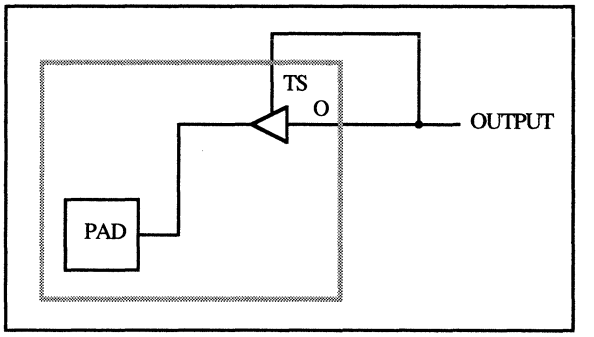
Brief Output Glitch Caused by Three-State Routing Delay

The next three figures show the available LCA macro library open-drain structures listed below.

- Open-Collector Output Pad
- Open-Collector Bidirectional Input/Output Pad
- Open-Collector Output Pad with Storage

First, the open-collector output is shown, followed by the open-collector bidirectional I/O pad.

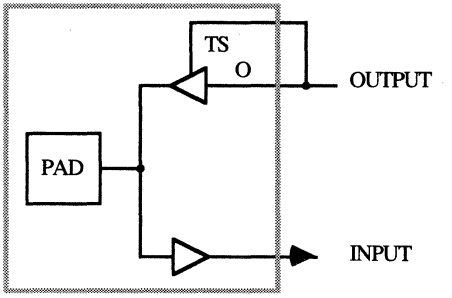
I/O TYPE: Pad Output with "Open-Collector"
MACRO NAME: POUTC
SCHEMATIC:



CONFIGURATION
I:
BUF: TRI

Open-Collector Output Pad

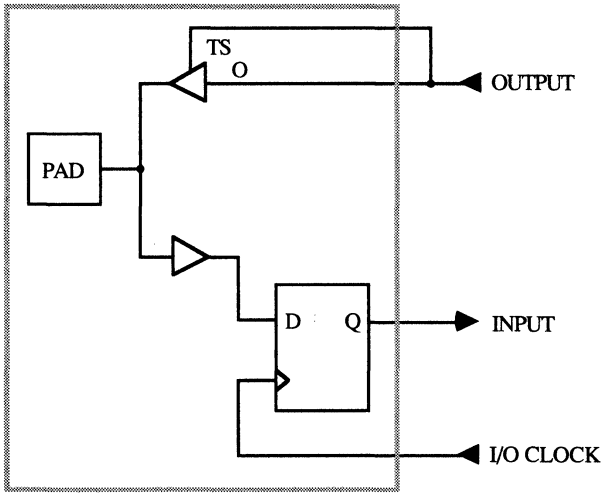
I/O TYPE: Pad Input/Output with "Open Collector"
MACRO NAME: PIOC
SCHEMATIC:



CONFIGURATION:
I: PAD
BUF: TRI

Open-Collector Bidirectional Input/Output Pad

I/O TYPE: Pad Input/Output with Storage, "Open Collector"
MACRO NAME: PIOQC
SCHEMATIC:



CONFIGURATION:
I:Q
BUF: TRI

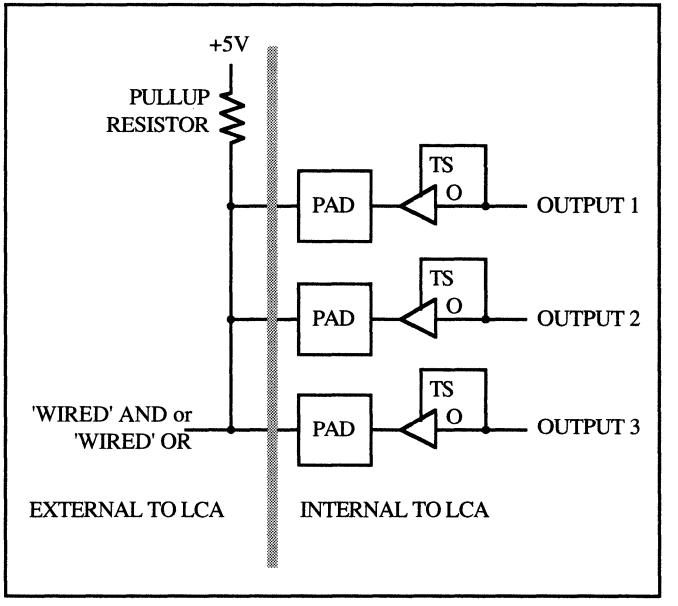
Open-Collector Output Pad with Storage

4.2.2.2 Wired-AND and Wired-OR Structures

The open-drain capability of an IOB allows wired-AND and wired-OR structures to become part of the LCA macro library. The AND and OR implementations are essentially the same; the only difference between their forms is the type of logic each uses. You design with wired-AND structures in positive-logic implementations and wired-OR structures in negative-logic implementations.

The next figure shows a typical wired-AND or wired-OR structure. All output pads from the IOBs are externally wired together as a common signal. In a positive-logic system, when all of the logic outputs to the IOBs are TRUE, the three-state control is enabled and the IOB

output PADs are forced to high-impedance. However, since all of the IOBs are tied to Vcc through a pull-up resistor, the line is pulled up to Vcc. If the logic signal to any of the IOBs is FALSE, the corresponding output buffer is turned on and that LOW signal propagates to the common line, pulling the entire line LOW. The entire structure then acts as an AND function; when all outputs are HIGH, the common line is HIGH. If any output is LOW, then the common line is also LOW.



Wired-AND or Wired-OR Function

The following equation describes the wired-AND logic.

$$IOB1 \cdot IOB2 \cdot IOB3 \cdot ... \cdot IOBn = TRUE$$

A wired-OR structure is similar to the wired-AND, except that it is implemented in negative logic. It ORs together a number of active-LOW signals to generate a logic function. The logic equation for a wired-OR, shown below, is merely a DeMorgan-equivalent inversion of the previous equation.

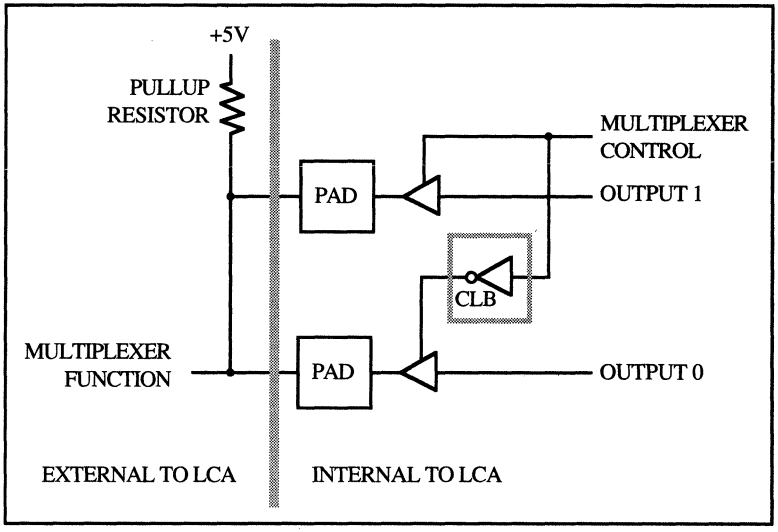
$$IOB1 + IOB2 + IOB3 + ... + IOBn = FALSE$$

A typical application of a wired-OR structure is an active-LOW common-interrupt line. An interrupt request from any peripheral pulls the common interrupt line LOW, which informs the processor of the request. You can build a wired-AND or wired-OR function from any number of open-collector outputs.

4.2.2.3 Multiplexers from Open Collector I/O Structures

The LCA macro library also contains another structure built using open-drain IOBs: an n-bit multiplexer, as shown below. All pad outputs are tied together outside of the package, on a common line that becomes the multiplexer output. Each IOB in the multiplexer is configured as an output with three-state control using the LCA library macro POUTZ. The output line (O) of each IOB becomes an input for the multiplexer. Driving the corresponding three-state control line LOW, T = 0, selects a signal; the selected signal propagates to the common output line. The three-state control lines can be driven with a CLB.

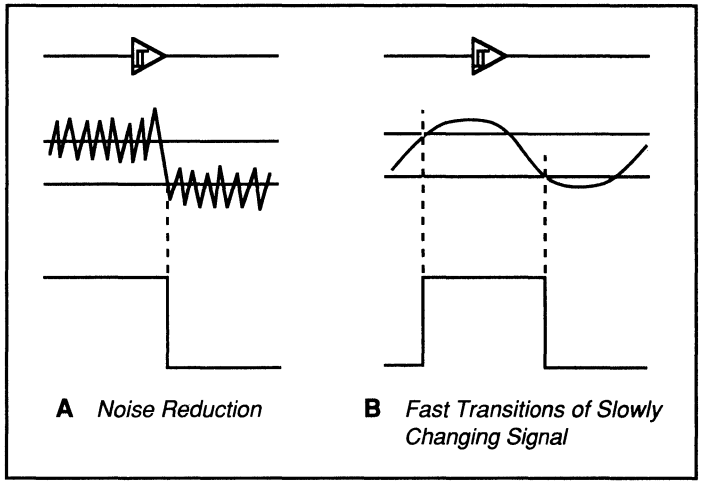
Caution: You must avoid contentions on the common output line.



Open-Collector Multiplexer Function

4.2.3 SCHMITT-TRIGGER STRUCTURES

The Schmitt trigger has numerous applications in digital designs. Two of the most common applications are shown below.



A *Noise Reduction*

B *Fast Transitions of Slowly Changing Signal*

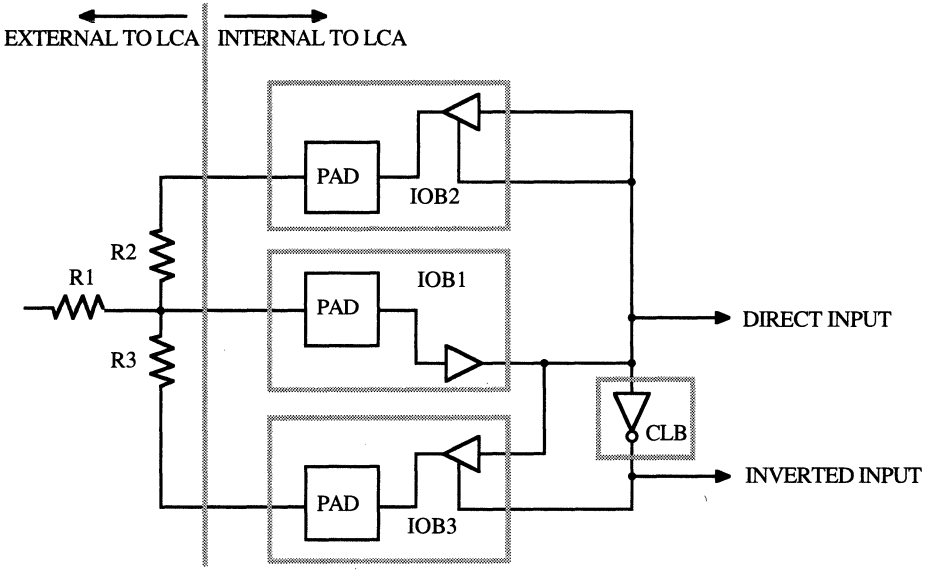
A. Schmitt-triggered inputs filter signal noise because of the hysteresis inherent in the switching characteristics of a Schmitt trigger.

B. A Schmitt trigger generates a fast transition for a slowly changing input function when that function reaches a predetermined level. Again, this capability is available due to the hysteresis of the Schmitt trigger.

You can build a variety of Schmitt-triggered input structures in an LCA device. For example, using three IOBs, a CLB, and three resistors, you can create a Schmitt trigger with selectable voltage hysteresis, as shown below.

Note: If the amount of hysteresis is not critical, then only two resistors and two IOBs are required.

I/O TYPE: Schmitt-Triggered Input With Selectable Hysteresis
MACRO NAME: None
SCHEMATIC:



CONFIGURATION:
IOB 1 - Input
I:PAD
BUF:
IOB 2 - Output
I:
BUF:TRI
IOB 3 - Output (inverted through CLB)
I:
BUF:TRI

Note: V_{TH} = input threshold voltage for CMOS inputs
V_{TH} = 2.2 V for TTL inputs
V_{TH} = 1.4 V ± supply tolerance

COMMENTS:
Resistors pairs R1:R2 and R1:R3 form two voltage dividers that set the HIGH-going and LOW-going input hysteresis.
Resistors R1 and R2 set the HIGH-going hysteresis (V_H) according to this equation:

$$V_H = V_{TH}[(R1 + R2)/R2] - V_{OL}$$

Resistors R1 and R3 set the LOW-going hysteresis (V_L) according to this

$$V_L = V_{TH}[(R1 + R3)/R3] - V_{OH}$$

Schmitt-Triggered Input with Selectable Hysteresis

Three resistors in the above macro select the threshold voltage and the amount of hysteresis for the Schmitt trigger. The three resistors are separated into two-

resistor network pairs, R1:R2 and R1:R3. Each pair forms a voltage divider to set the input-voltage level. One voltage divider sets the HIGH-going transition level (V_H), the other sets the LOW-going transition level (V_L). The value at the input to IOB1 is inverted through a CLB, and then routed to the three-state control line IOB3. The CLB logic adds a small amount of time hysteresis to the signal because the CLB logic and the routing cause delay. The logic delay can be balanced by buffering the input before sending it to the three-state control of IOB2.

An inverting Schmitt trigger is similar to the non-inverting one shown above, except that the sense of the logic is inverted inside the LCA device.

Assume that the input voltage is near ground. The output voltage of IOB2 is at V_{OL}, which pulls resistor R2 toward ground. There is then no potential difference across R2. The output buffer of IOB3 is high-impedance because its three-state control pin is HIGH. Resistor R3 is effectively removed from the circuit and the input voltage is divided by the resistor network formed by resistors R1 and R2. As the input voltage continues to increase, the IOB1 pad voltage eventually reaches its switching threshold.

As soon as the threshold is crossed, IOB1 goes HIGH. This drives the output of IOB2 into high-impedance (IOB2 TS = 1) and enables the output buffer of IOB3 (IOB3 TS = 0). At V_{OH} then, IOB3 pulls the input of IOB1 HIGH through resistor R3. In this state, resistor R2 is effectively removed from the circuit because IOB2 is high-impedance.

The Schmitt-trigger structure remains in this state even if the input voltage fluctuates, unless it fluctuates to the opposite hysteresis limit. Then, the Schmitt trigger goes to the opposite state. In other words, the Schmitt

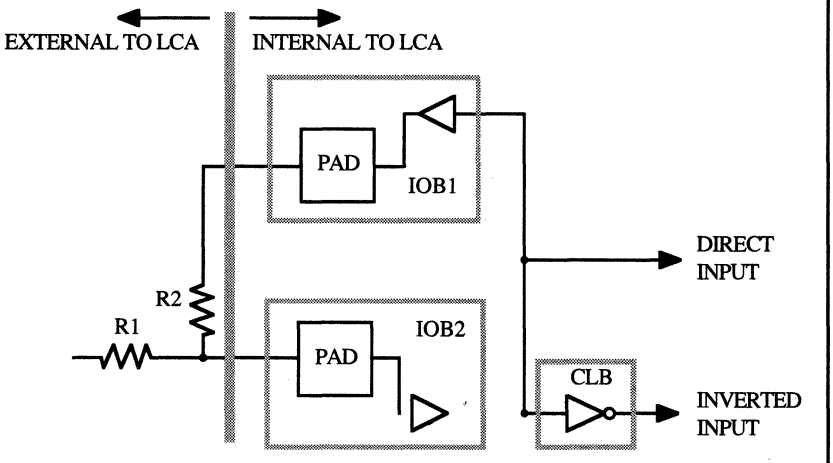
trigger stays HIGH until the input to IOB1 drops below the LOW-going hysteresis limit, and vice versa.

If the hysteresis values are not critical, the Schmitt-trigger structure requires only two IOBs and two resistors, as shown in the following figure. However, the range of V_H and V_L is very limited. This IOB configured as an output pulls the input HIGH or LOW, depending on the transition direction.

I/O TYPE: Schmitt-Triggered Input With Limited Hysteresis

MACRO NAME: None

SCHEMATIC:



CONFIGURATION:

IOB 1 - Output
I:
BUF:ON

IOB 2 - Input
I:PAD
BUF:

COMMENTS:

Hysteresis values are limited:

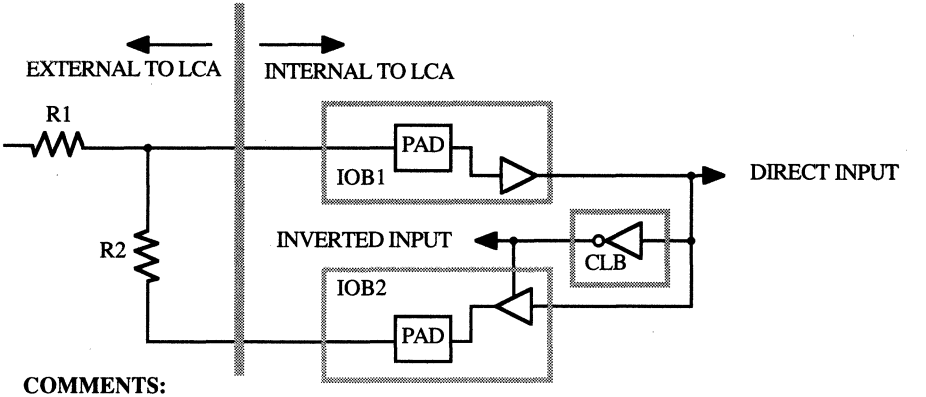
$V_L = [(R1 + R2)/R2]\ V_{TH} - V_{OH}(R1/R2)$
$V_H = [(R1 + R2)/R2]\ V_{TH} - V_{OL}(R1/R2)$

Schmitt-Triggered Input with Limited Hysteresis

If you need a selectable Schmitt trigger only for a single transition direction (HIGH going LOW, or LOW going HIGH), then you can use one of the Schmitt triggers shown in the following figure. These circuits are simpler versions of the one above.

Note: A single CLB is required to invert the sense of the input signal, which then enables or disables the output buffer for IOB2 (the one configured as a three-state output.

I/O TYPE: Unidirectional Schmitt-Triggered Input
MACRO NAME: None
SCHEMATIC:

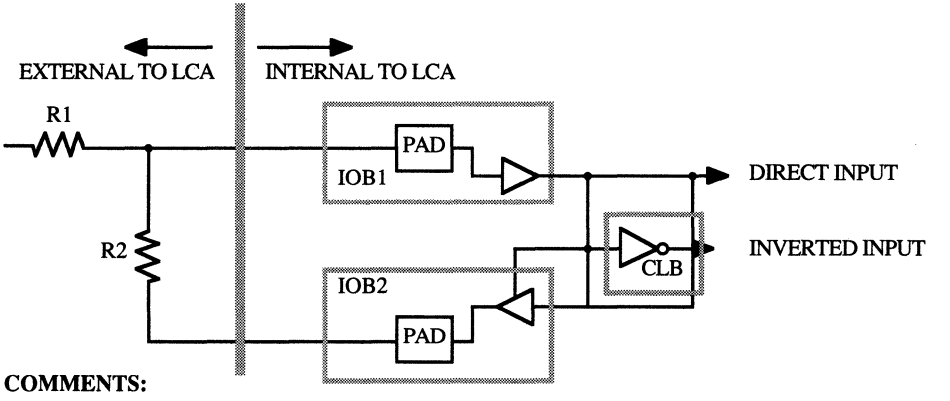


COMMENTS:

$V_H = V_{TH}$
$V_L = V_{TH}[(R1 + R2)/R2] - V_{OH}(R1/R2)$

Unidirectional Schmitt-Triggered Input HIGH Going LOW

SCHEMATIC:



COMMENTS:

$V_H = V_{TH}[(R1 + R2)/R2] - V_{OL}(R1/R2)$
$V_L = V_{TH}$

Unidirectional Schmitt-Triggered Input LOW Going HIGH

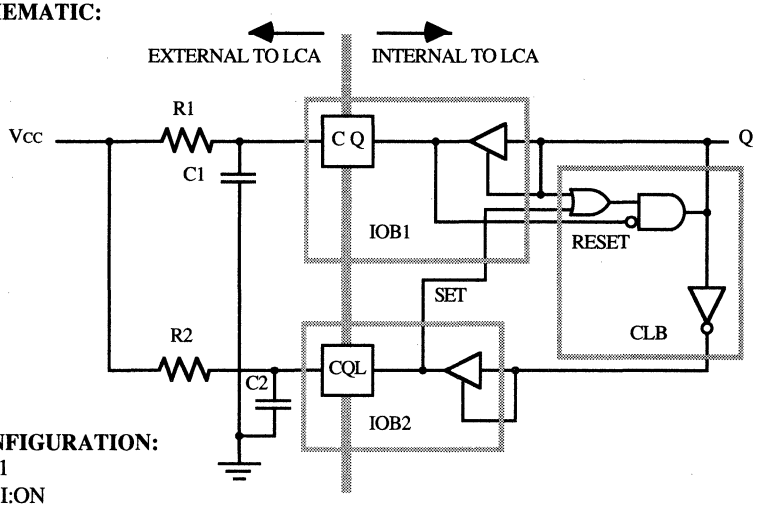
4.2.4 GENERAL PURPOSE OSCILLATOR STRUCTURES

The LCA macro library includes general purpose oscillators built using two IOBs and a CLB. The general theory of operation is similar to that described for Schmitt triggers.

In the oscillator shown and described below, the charging and discharging of two capacitors generates the oscillating signal. Capacitor C2 charges to a voltage threshold, on SET, to set a latch. As soon as the voltage across C2 exceeds the threshold, the SET line causes the Q line to go HIGH and discharges C2 by driving the IOB called CQL. After crossing the threshold, the RESET line, which has been held LOW, is allowed to rise as capacitor C1 charges. When capacitor C1 charges to its threshold, the Q output is reset and forced LOW. Capacitor C1 is then discharged by the IOB named CQ and capacitor C2 begins charging again. This process is repeated, creating a low-frequency resistor-capacitor oscillator.

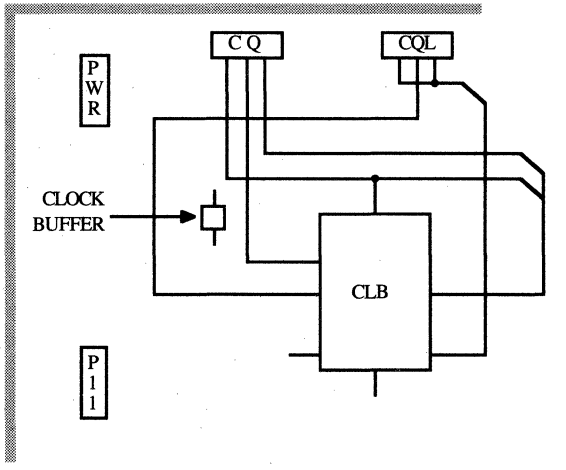
Consider the routing delay of the three-state control lines within the IOBs, named CQ and CQL in the figure. The time period of the oscillator depends on each capacitor being completely discharged during the opposite timing phase. Also, timing depends on both capacitors beginning their charge near ground. A routing-delay difference between the output (O) of an IOB and the three-state control can prevent the capacitors from completely discharging.

I/O TYPE: Low-Frequency Resistor-Capacitor Oscillator
MACRO NAME: GOSC
SCHEMATIC:



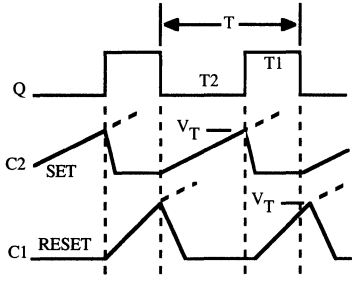
CONFIGURATION:
1OB1
I:ON
BUF:TRI
1OB2
I:ON
BUF:TRI

SAMPLE ARRANGEMENT



Low-Frequency Resistor-Capacitor Oscillator

The following figure illustrates the oscillator's low-frequency timing characteristics.



T (time period) = T1+T2 = N ((R1 X C1) = (R2 X C2)) [11]
where N = approximately 0.35 for TTL threshold
= approximately 0.75 for CMOS threshold

ASSUMPTIONS:

1) Each capacitor is discharged during the opposite timing phase.
2) Capacitors begin charging from GROUND.
3) Effect of three-state routing delay is assumed minimal.

Low-Frequency Resistor-Capacitor Oscillator Timing Diagram

You can use any number of these low-frequency oscillators in a design. Most designs, however, require only one or two.

Note: If the oscillator output is used throughout the design to clock the registers in the CLBs, then you should place the oscillator near one of the clock buffers and use the clock buffer to route the signal.

The sample array element in the low-frequency oscillator figure above shows the oscillator built near the main clock buffer in the upper-left corner of the die. A similar low-frequency oscillator could drive the auxiliary clock buffer located in the lower-right corner of the die.

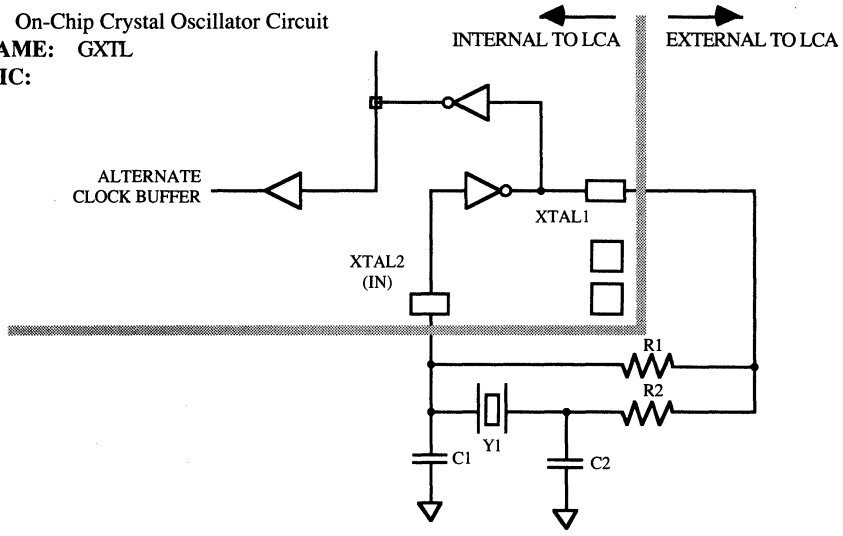
You should be aware that the low-frequency oscillator circuit causes an error when you use the timing calculator to examine the oscillator. The timing calculator in the LCA development system detects combinational loop conditions and flags them as errors. Because the oscillator circuit depends on a combinational loop for operation, it causes an error message. You can safely ignore such error messages if you detect them only in the oscillator circuit.

4.2.5 ON-CHIP CRYSTAL OSCILLATOR STRUCTURES

You can configure two special I/O Blocks to interface to the oscillator on-chip, located in the lower-right corner of the die. This oscillator is associated with the auxiliary clock buffer located near it. When you select the interconnection to drive the auxiliary clock buffer, two special pins interface to the internal high-speed inverting amplifier to form the oscillator. Externally, you should attach these pins to the crystal oscillator components, as shown below. The best way to configure the crystal oscillator is by using the GXTL macro in the LCA library.

Even before you finish configuring the LCA device, the on-chip oscillator begins operation so that its circuitry can stabilize. However, the actual internal connection of the oscillator to other circuitry on the chip is delayed until the device configuration completes.

I/O TYPE: On-Chip Crystal Oscillator Circuit
MACRO NAME: GXTL
SCHEMATIC:



SUGGESTED COMPONENT VALUES:

R1: 1-4 MΩ
R2: 0-1 KΩ
(may be required for low frequency, phase shift and/or compensation level for crystal Q)
C1, C2: 5-20 pF
Y1: 1-10 MHz AT cut

	XTAL1	XTAL2
48-PIN DIP	33	30
68-PIN PLCC	46	43
68-PIN PGA	J10	L10
84-PIN PLCC	56	53
84-PIN PGA	K11	L11

On-Chip Oscillator Circuit

The feedback resistor R1, from output to input, biases the amplifier at threshold and should be as large a value as practical, up to 4 MΩ. The inversion and delay of the amplifier, together with the R-C networks and crystal, produce a 360-degree phase shift, forming a Pierce oscillator.

Note: You can include the series resistor R2 to add to the amplifier output impedance, when needed. This may be needed for phase-shift control, crystal-resistance matching, or for limiting the amplifier input swing to control clipping at large amplitudes.

The ratio of capacitor C2 to C1 adjusts the excess feedback voltage. The amplifier operates in the range of 1 MHz, up to one-half the specified CLB toggle frequency. Using the oscillator at frequencies below 1 MHz requires individual characterization with respect to a series resistance. Operating at frequencies above 20 MHz is also more complex because it generally requires that the crystal operates in a third overtone mode in which the R-C networks must suppress the fundamental frequency.

4.2.6 REGISTERS AND COUNTERS

The previous examples in this chapter describe how to use IOBs in conventional I/O applications, using the IOB for input, output, or both. For any IOB that is **not** required for input or output, you can use the storage element within the IOB to create registers and various types of counters. The following designs use the output buffer (BUF:ON) fed back into the input register (I:Q). This configuration is shown below. These IOB's pads usually are not connected to anything externally, although you may do this if necessary.

4.2.6.1 IOB-Based Register Delays

If you want to construct registers using IOBs, you must understand the delays in IO-based registers. The delays incurred through an IOB-based register depend on the sum of two parameters: the delay through the output buffer, and the delay back through the input buffer to the register. While these values are defined in the data sheet for an output load of 50 pF, they change only slightly for no output capacitance. The delay into an IOB-based register is shown below.

T(IOB-reg)	=	Top	=	Tpl

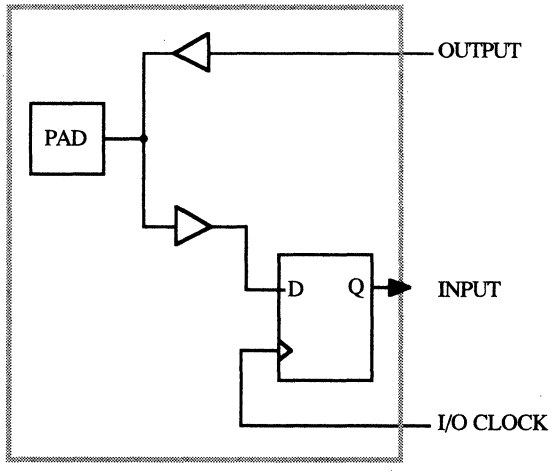
Where:

Top	=	Output to Pad output
Tpl	=	Pad input set up to I/O clock (minimum)

4.2.6.2 Wide Storage Registers

The first type of IOB-based registers is a wide storage register. The basic I/O structure illustrated below creates a wide storage register.

I/O TYPE: Pad With Input Storage (IOB-based)
MACRO NAME: PREG
SCHEMATIC:

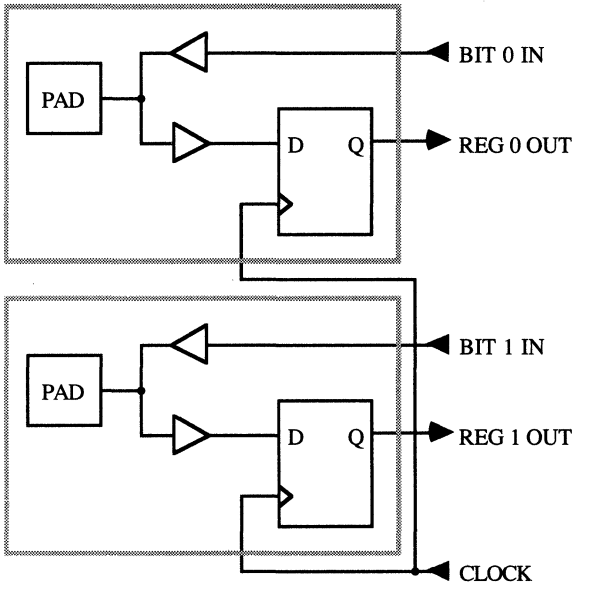


CONFIGURATION:
I:Q
BUF: ON

Pad with Input Storage (IOB-Based Register)

The following figure shows the construction of an n-bit-wide storage register built from these IOBs. Wide storage registers are ideal for IOBs because the I/O clock feeding an IOB is common to all IOBs along each edge of the die.

I/O TYPE: N-bit Storage Register (IOB-based register)
MACRO NAME: None
SCHEMATIC:



EXPANDABLE TO N BITS

CONFIGURATION:
All IOBs
I:Q
BUF:ON

COMMENTS:
The I/O clock into each IOB is common to all IOBs along each edge of the die. For best resource utilization, group the storage elements along one edge of the die.

N-Bit Storage Register (IOB-Based Register)

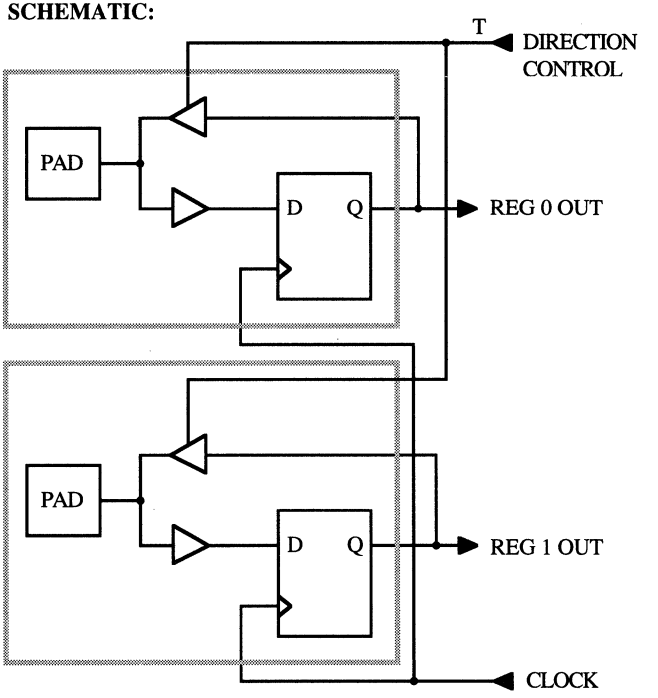
4.2.6.3 Read/Write Registers

Another variation of the basic IOB-based register is a simple read/write register. This structure allows external devices to write data into registers within the LCA device, and also to read the data back.

The following figure shows the structure of a read/write register. In this example, each IOB's input and output is connected. The three-state control line (T) controls the direction of data flow, where T = LOW for a read operation by the external device, and T = HIGH for a write operation to the LCA device. Typically, the read/write control line (three-state control) originates outside the LCA device and comes in through an additional I/O block.

The input register data from the read/write register can be read from within the LCA device but the data cannot be written to the LCA device. Writing the register from inside the LCA device would require that two network sources be active, which is not allowed.

I/O TYPE: N-bit Read/Write Storage register
MACRO NAME: None
SCHEMATIC:



EXPANDABLE TO N BITS

CONFIGURATION:
All IOBs
I:Q
BUF:TRI

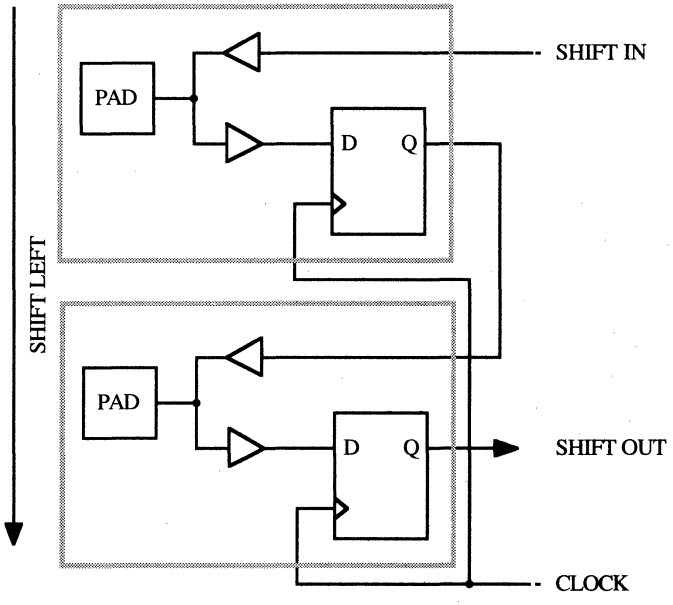
COMMENTS:
The I/O clock into each IOB is common to all IOBs along each edge of the die. For best resource use, group the storage elements along one edge of the die. Also, the three-state control line controls the direction of data flow (T = LOW for read, T = HIGH for write). This control line typically originates off-chip and comes in through an additional IOB.

N-Bit Read/Write Storage Register

4.2.6.4 Shift Registers

You can easily construct shift registers with IOBs by feeding the output of one IOB to the input of the next. The figures below describe two shift registers: one shifts to the left, the other shifts to the right. The shift direction depends on the connections of each IOB's inputs and outputs.

I/O TYPE: Shift Left Register (IOB-based register)
MACRO NAME: None
SCHEMATIC:



CONFIGURATION:
All IOBs
I:Q
BUF:ON

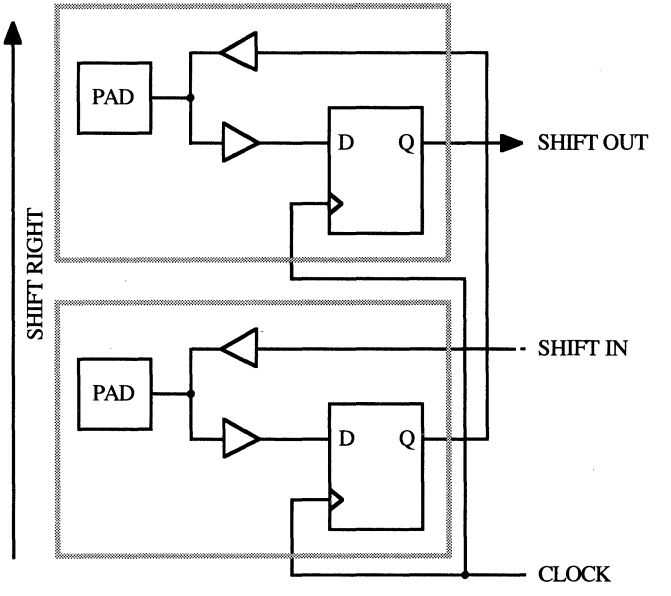
COMMENTS:
Notice that the routing of the output of a given IOB goes to the intput of the IOB on the left (shift left).

Shift-Left Register (IOB-Based Register)

Note: Because the I/O clock line of an IOB is common to all IOBs along each edge of the die, the IOB-based shift registers are placed along one edge of the die.

The following figure shows a shift-right IOB-based register.

I/O TYPE: Shift Right Register (IOB-based register)
MACRO NAME: None
SCHEMATIC:



CONFIGURATION:
All IOBs
I:Q
BUF:ON

COMMENTS:
Notice that the routing of the output of a given IOB goes to the input of the IOB on the right (shift right).

Shift-Right Register (IOB-Based Register)

4.2.6.5 Johnson Counters

An n-bit Johnson counter counts to 2n states as opposed to standard binary counters that count to 2^n possible states. Johnson counters have a variety of uses in digital design, including low-modulo counters and glitch-free decoders.

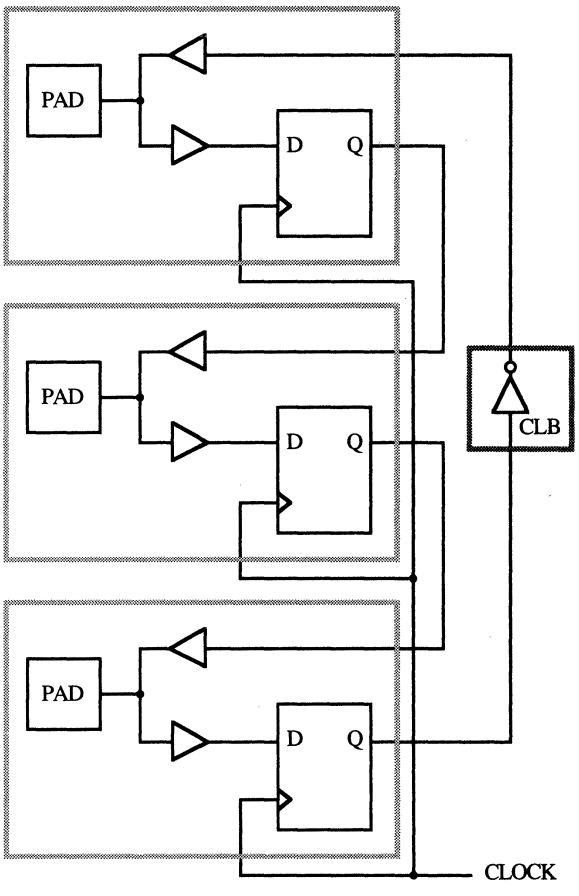
In an IOB-based design you can think of a Johnson counter as special shift register. Only one bit changes during a state transition, as shown in the following table for a three-bit Johnson counter.

Transitions of a Three-Bit Johnson Counter
000
100
110
111
011
001

You can build a Johnson counter of unused IOBs, as shown below; however, it requires at least one CLB to perform an inversion. The Johnson counter is automatically reset to an all-zeroes state upon configuration or on a ~RESET pulse.

Refer to the application note on counters for more information on creating Johnson counters in LCA devices.

I/O TYPE: N-bit Johnson Counter
MACRO NAME: None
SCHEMATIC:



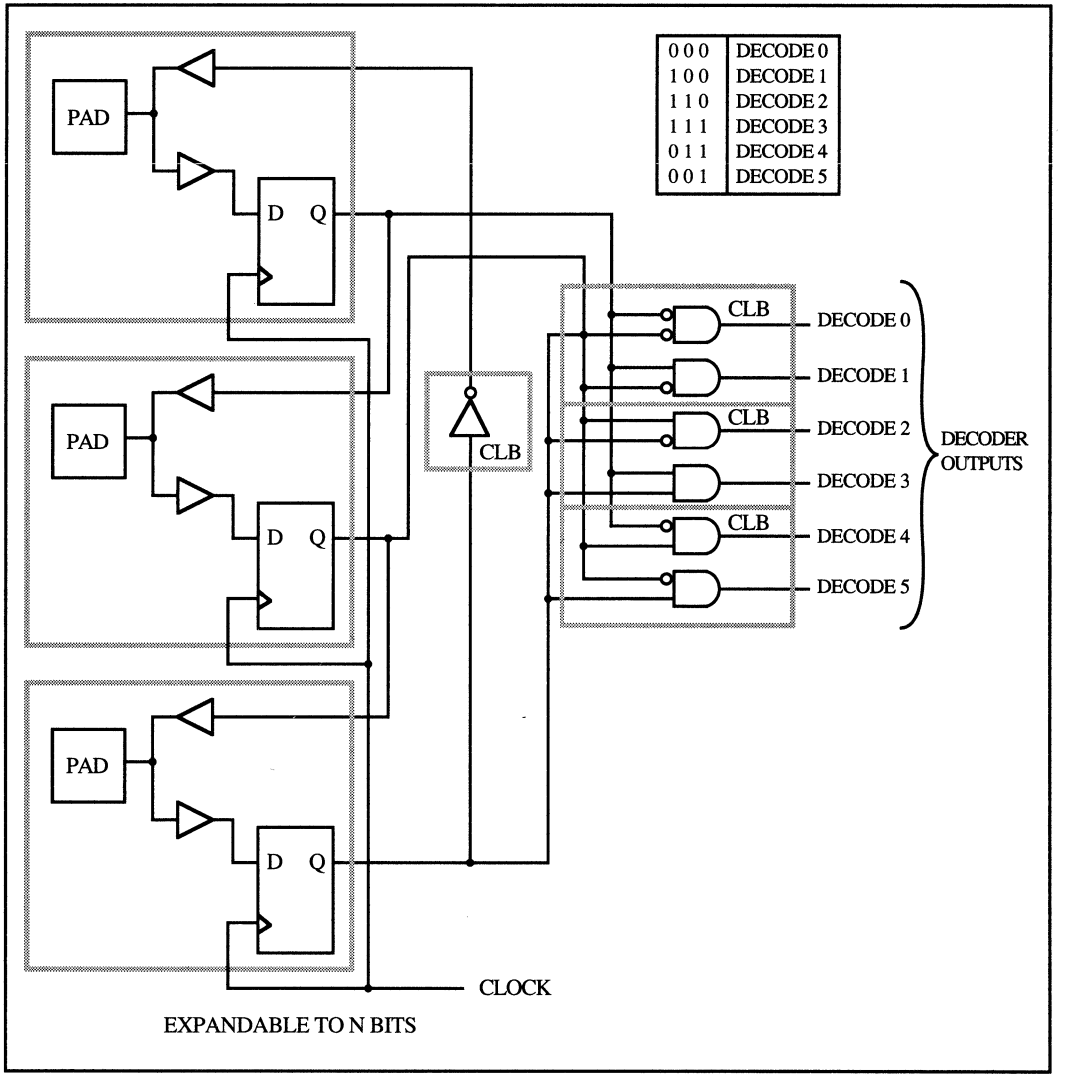
EXPANDABLE TO N BITS

CONFIGURATION:
All IOBs
I:Q
BUF:ON

N-Bit Johnson Counter (IOB-Based)

4.2.6.6 Glitchless Johnson Decoder

You can build a glitch-free decoder using IOBs and CLBs. The decoder is glitch-free because only one bit changes during a state transition. An n-bit Johnson counter/decoder can decode any one of the 2n possible states, or any number of contiguous states, by decoding (ANDing) together just two of the appropriate counter bits. You can also create counters of various modulo and duty-cycle by using different Johnson decoders. For example, the next figure shows the schematic implementation of a Johnson counter/ decoder with various two-input decode states.



Johnson Counter/Decoder

As the above figure illustrates, you can decode any state of a Johnson counter, glitch-free, using only a two-input logic function.

4.2.6.7 Linear Feedback Shift Registers

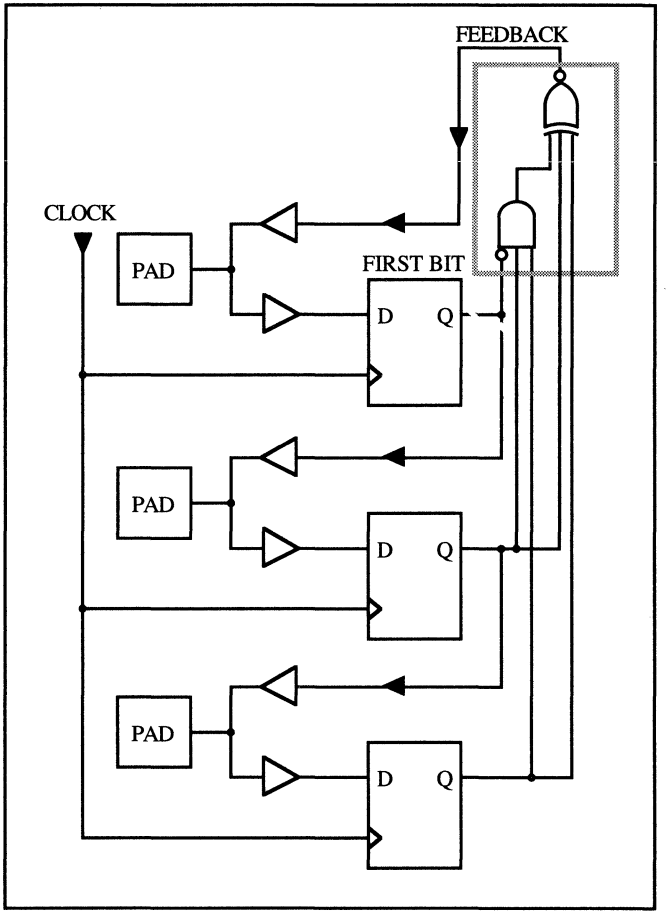
Linear Feedback Shift Registers (LFSRs) are yet another modification of a simple shift register. An LFSR consists of a shift register that feeds back the appropriate bits to the first bit position. An LFSR requires some logic function in the feedback path, usually an exclusive-OR (XOR) function.

LFSRs have numerous applications, such as implementing the encryption and decryption functions in a UART.

The next figure shows the schematic for a three-bit LFSR that implements a modulo 5 (divide-by-five) counter. An n-bit LFSR counter can produce a pseudo-random sequence of up to 2^n-1 unique states. By adding logic to the feedback path, you can force the LFSR counter to **skip** any number of states, from one to 2^n-1. By forcing the counter to skip some number of states, m, an LFSR counter can implement any modulo as described in the following equation.

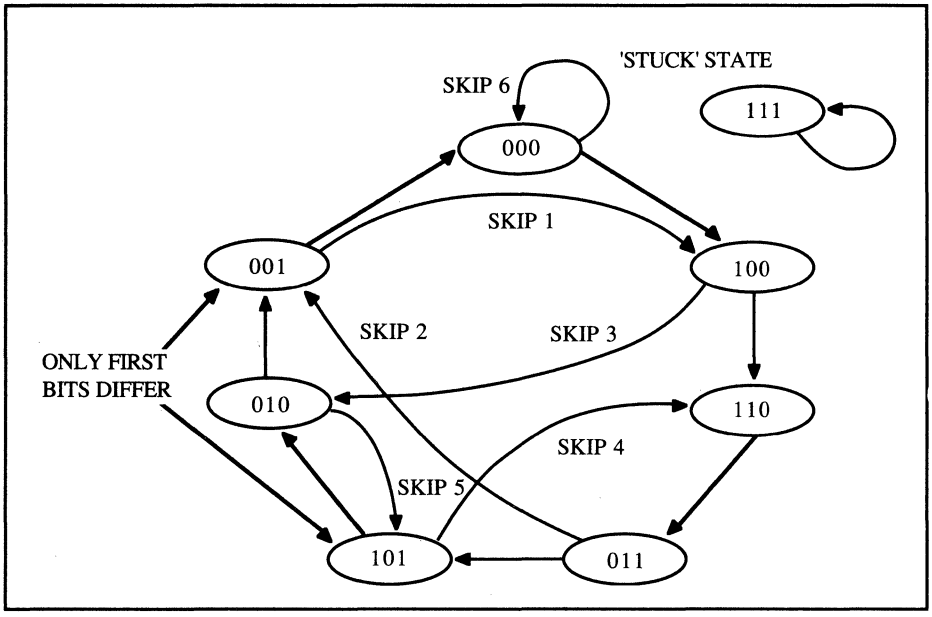
$$\text{MODULO} = (2^n\text{-}1) - m$$

- n = number of shift-register bits
- m = number of skipped states



Schematic for Modulo 5 LFSR Counter

The figure below shows the counting sequence for a three-bit LFSR counter with an exclusive-NOR (XNOR) in the feedback path. This figure also shows all possible **skip paths** and the **stuck** state.



Three-Bit LFSR Counting Sequence Skip Path and Stuck State

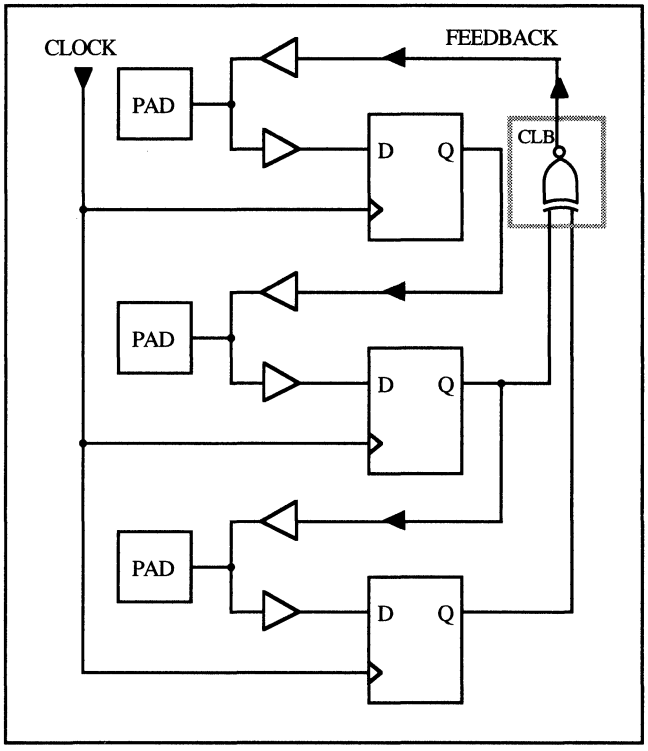
In the counting sequence there are two counter states where only the first bits differ. For example, refer to the states 101 and 001. By forcing the feedback logic to invert the sense of the feedback into the first bit, the counter can be forced to **skip** all of the states between the two indicated values. You can accomplish this skip by decoding (ANDing) the state just previous to the state to be skipped.

Using the modulo 5 counter as an example, locate the initial value that allows the counter to skip two states, 101 for example. By decoding the state 011, which is the state just prior to the initial skip state of 101, you can invert the sense of the feedback into the first bit. The counter then skips from state 101 to state 010, implementing a modulo 5 counter. Using this method and the proper feedback into the register, you can build a counter of any modulo from 1 to 2^n-1.

Either device configuration, or an externally driven active-LOW RESET signal, resets all storage elements used in the LFSR counter to zero.

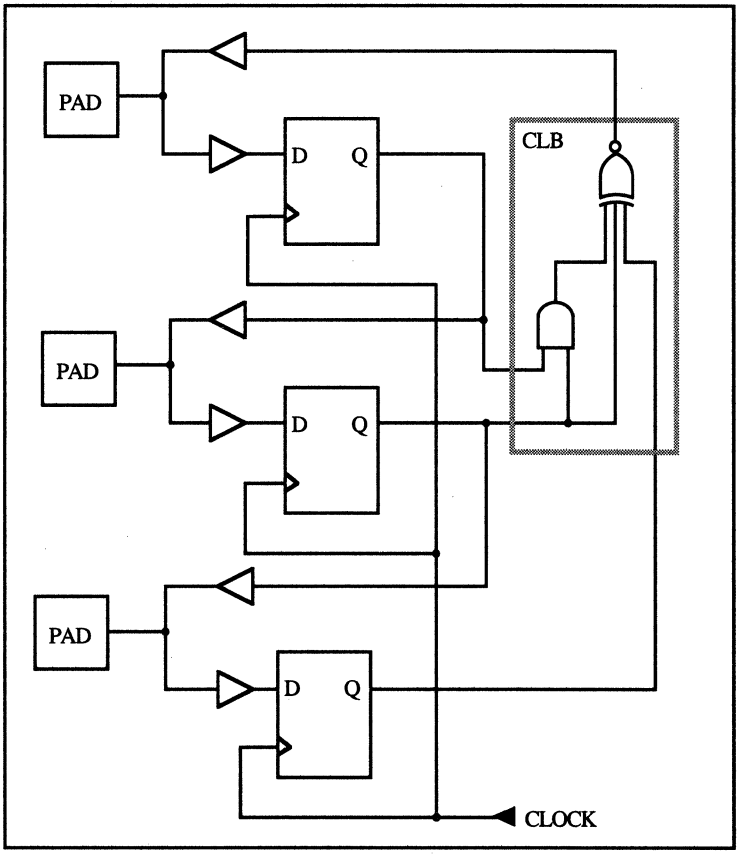
Be careful to avoid the **stuck** state in your designs. This is the missing state in the 2^n-1 counting sequence. If the stuck state is included, the LFSR counter has 2^n possible states. The stuck state occurs when the feedback path forces the counter into an ever-repeating single state.

As a simple example, assume that you build an LFSR counter with a two-input XOR feedback path, as shown next. Upon configuration or an external active-LOW RESET signal, the counter begins operation in the all-zeroes state (000) and becomes stuck in that state due to the type of feedback used.



Simple LFSR with a **STUCK** State

An interesting situation occurs when all bits except the last bit of the stuck state are decoded (ANDed together) and included in the feedback path. Instead of counting over a possible range of 2^n-1 states, the extra decoding causes the LFSR counter to count to all 2^n states, as shown in the following figure.



An LFSR Forced to Count to 2^n Possible States

You can build longer LFSR counters with higher possible modulos and more complex feedback mechanisms, but their discussion is well beyond the scope of this chapter. However, the following table presents some of the possible feedback combinations for LFSR counters of three to ten bits.

I/O Block

2n-1 Modulo	7	15	31	63	127	255	511	1023
Feed-back Options into Bit 1	1,3 2,3	1,4 3,4	2,5 3,5	1,6 5,6	1,7 3,7 4,7 6,7	1,2,7,8	4,9 5,9	3,10 7,10

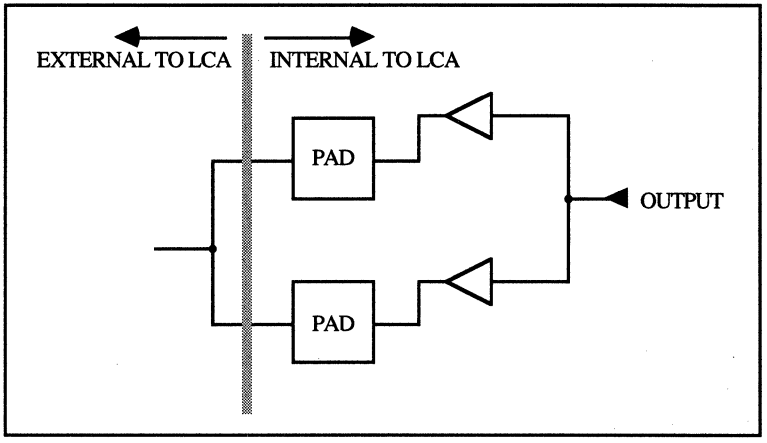
4.2.7 INCREASED DRIVE-CURRENT STRUCTURES

LCA devices are specified to have 4 mA worst-case source and sink capabilities at V_{OL} = 0.32 V and V_{OH} = 3.68 V. However, you obtain increased drive current at the cost of decreased voltage margins. For example, the following table illustrates the effect on V_{OL} and V_{OH} of increasing the drive current through a single IOB.

Output Current and Output Voltage Levels for an IOB

	4 mA	6 mA	8 mA
V_{OH}	3.86 V	3.54 V	3.22 V
V_{OL}	0.32 V	0.48 V	0.64 V

An alternate method of increasing the drive current is to parallel the output drivers of two IOBs. Paralleling two outputs enables the IOBs to source-and-sink double the worst-case current, with no reduction in voltage margins. This method is diagrammed below.



Parallel Outputs Have Increased Drive Capability

Caution: You should minimize the difference in routing delay between the two IOBs connected in parallel. Excessive delays can cause output contentions.

Chapter 5 discusses placement and routing.

CHAPTER 5

PLACEMENT AND ROUTING

5 PLACEMENT AND ROUTING

This chapter discusses placement and routing of LCA designs. Placement and routing play an important part in determining both the performance of your design and how efficiently it uses the available LCA resources.

- The overview, 5.1, introduces LCA placement and routing.
- The discussion on interconnection resources, 5.2, describes different options for interconnecting the CLBs and IOBs in an LCA design.
- The discussion on placement, 5.3, explains how to optimize placement of CLBs and IOBs in a design.
- The discussion on routing, 5.4, explains how to route, and how to edit the routing of, an LCA design.
- The discussion on timing analysis, 5.5, describes how to use the delay calculator to analyze an LCA design's timing.
- The summary, 5.6, presents conclusions and recommendations for placing and routing LCA designs.

5.1 OVERVIEW

As with other high density ASIC devices, the LCA device offers placement and routing alternatives that can affect the use and the performance of your final design. In gate arrays and other factory-programmed solutions, you can explore these layout alternatives only through simulation. With the LCA device, however, you can see and modify the placement and routing at design time using the LCA development system.

The LCA development system includes several powerful capabilities that let you optimize your design for performance and use of resources. This chapter investigates these capabilities and the operations that you can employ to optimize your design.

If you are not familiar with LCA placement and routing, you should read this entire chapter. If you have some level of knowledge about the LCA device and have completed some design work, you may want to study only those discussions that interest you.

Note: If you intend to use the Automatic Placement and Routing software, APR, you should still read this chapter.

APR helps you plan your design so that the LCA resources are used as efficiently as possible. After using APR to place and route your design, you may need to optimize or complete the routing with XACT. All of the following guidelines can help.

5.2 INTERCONNECTION RESOURCES

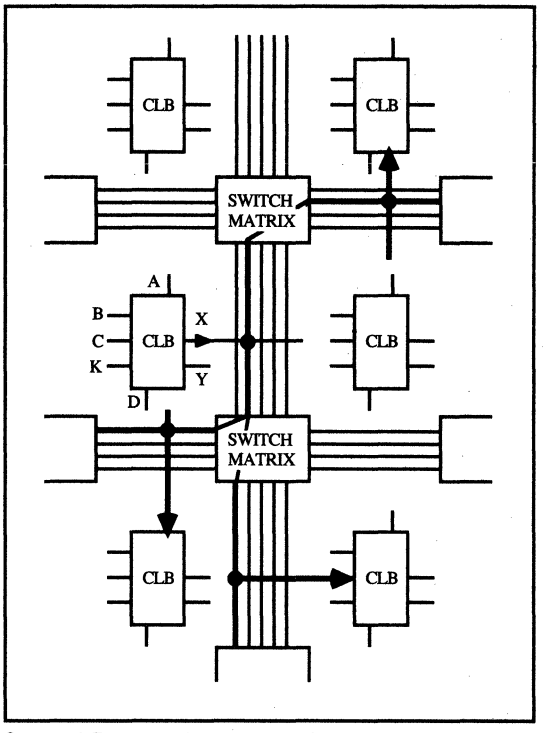
Placement and routing are closely interrelated. You make placement decisions based on efficient use of interconnection resources. Any changes in placement can change the routing and routability of a design and consequently, impact the design's performance. To make the best possible placement and routing decisions, you must understand the capabilities and trade-offs of the various types of routing available in an LCA device.

This discussion explains the following available interconnections.

- 5.2.1, General-Purpose Interconnection
- 5.2.2, Direct Connections
- 5.2.3, Long Lines
- 5.2.4, Clock Buffers

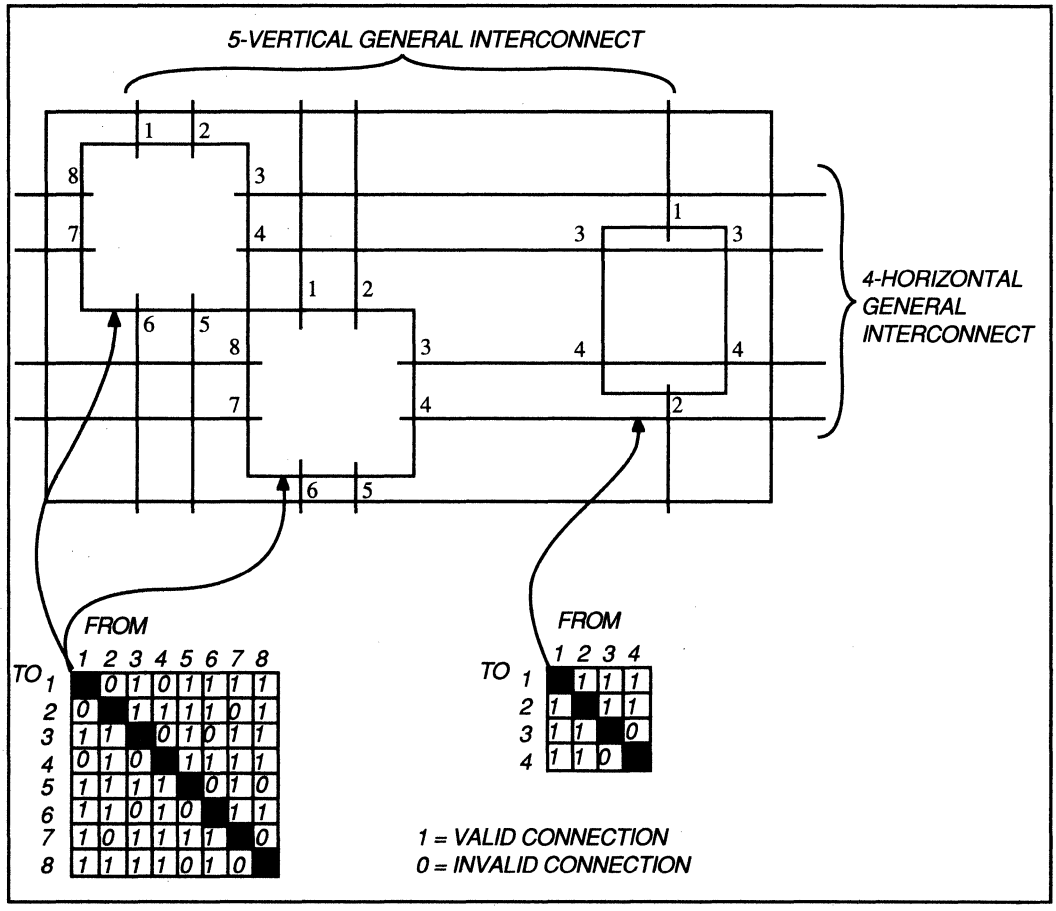
5.2.1 GENERAL-PURPOSE INTERCONNECTION

General-purpose interconnection provides routing for most signals on the LCA device. As shown below, it consists of four horizontal metal segments between adjacent rows and five vertical metal segments between adjacent columns of the CLBs and IOBs. The vertical segments are the same height as a CLB; the horizontal segments are the same width. A switch matrix at each row and column intersection controls how the segments are interconnected.



General-Purpose Interconnection

A switch on each CLB or IOB output can connect the output to the adjacent interconnection segments. Configuration bits in the LCA configuration file set up the switch connections in each matrix and on the block outputs. Configuration bits also program the multiplexers at the inputs of the CLBs and IOBs to select the appropriate input connections from the adjacent interconnection.



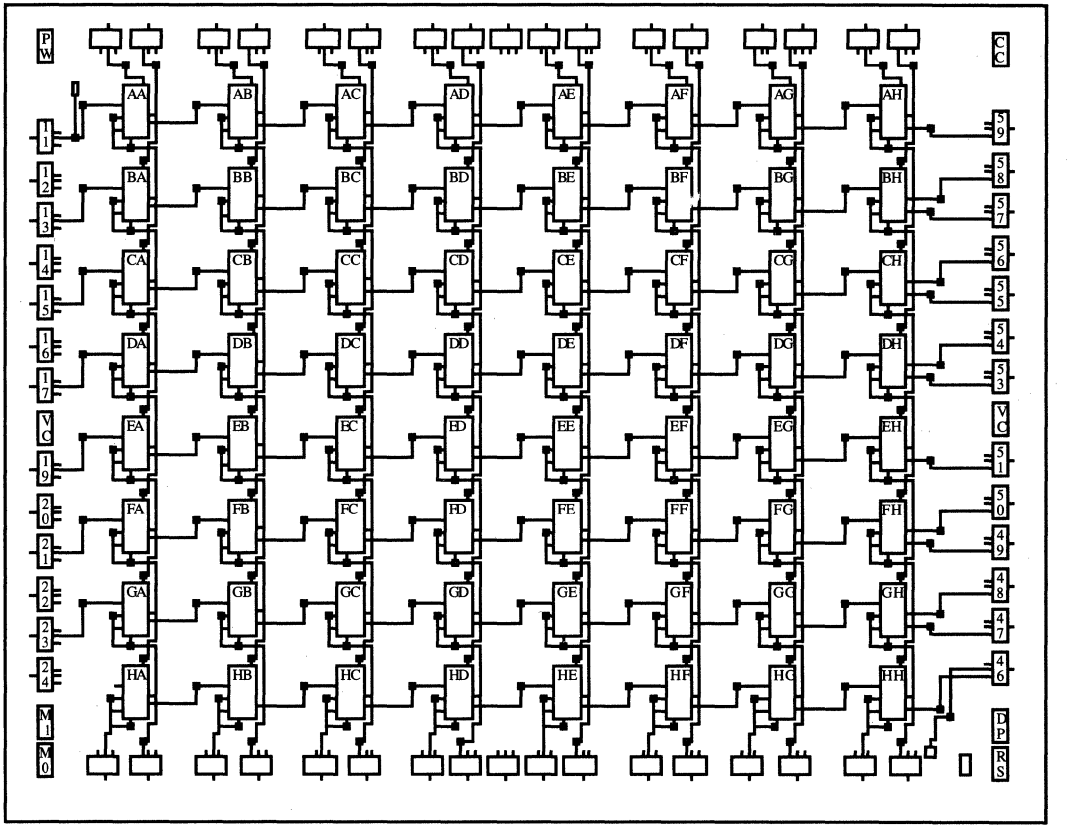
Interconnection Switching Matrix

Special **repowering buffers** in the general-purpose interconnection provide periodic signal isolation and restoration for higher fanout and improved performance. Each LCA device is divided into nine sections, with buffers provided at the section boundaries. These buffers are bidirectional because signals on a general interconnection segment must be able to propagate in both directions.

Note: Repowering buffers are provided only for the general-purpose interconnection; direct connections and long line interconnections do not require them.

5.2.2 DIRECT CONNECTIONS

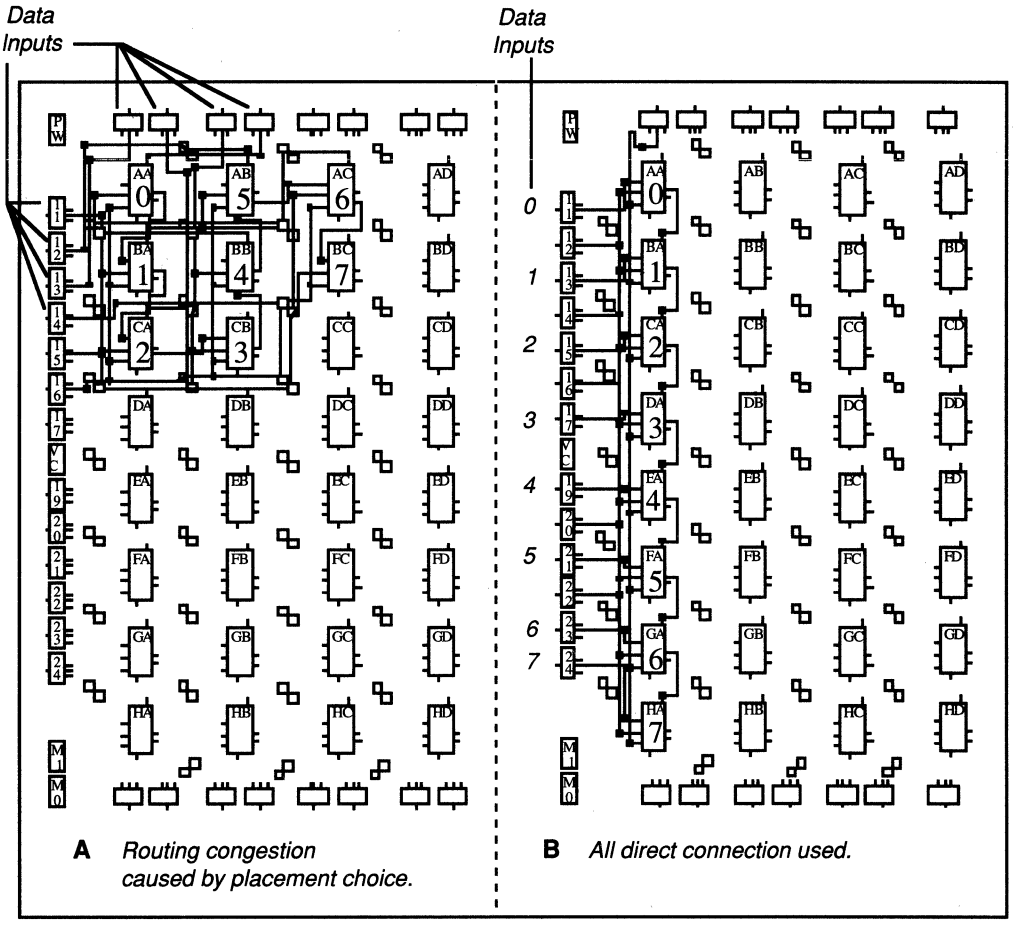
Each CLB and IOB can connect directly to adjacent blocks, as shown below. A direct connection is a **signal path** with virtually **zero delay**, which does not use any of the general interconnection or long line resources.



Direct Connection Resources

For maximum device performance and minimum routing resource impact, use direct connections as much as possible. Direct connections exist primarily in the vertical and left-to-right directions; therefore, you should arrange blocks that represent stages in a process sequentially: either vertically or from left to right. Left-edge I/O blocks naturally become data inputs, while right-edge I/O blocks become data outputs. Top- or bottom-edge I/O blocks can be either direct inputs or outputs, with alternate blocks having direct-out or direct-in paths. As an example, consider the following circuit.

An 8-bit parallel-load shift register loads the data byte into the shift register in parallel, then shifts it out one bit per clock cycle. The following figure shows two alternate implementations of this circuit.



8-Bit Parallel Load Shift Register

A. The eight CLBs used for the stages of the shift register are arranged in a rectangular area in the upper-left corner of the device, with general interconnection providing many of the signal paths.

B. The design uses direct connections exclusively, which provides zero-delay paths from block to block and allows higher performance.

The general interconnection used in design A is still available for other uses. Also, the LOAD signal has been routed on a long line driven from a directly-connected I/O block. Long lines are discussed next.

For the design shown in B, the following table of worst-case delays shows that the maximum load and shift clock rate for the 33 MHz device is 31.3 MHz, and 43.5 MHz for the 50 MHz device.

Partial Delay Report for Direct Connection Placement of an 8-Bit Shift Register with Parallel Load

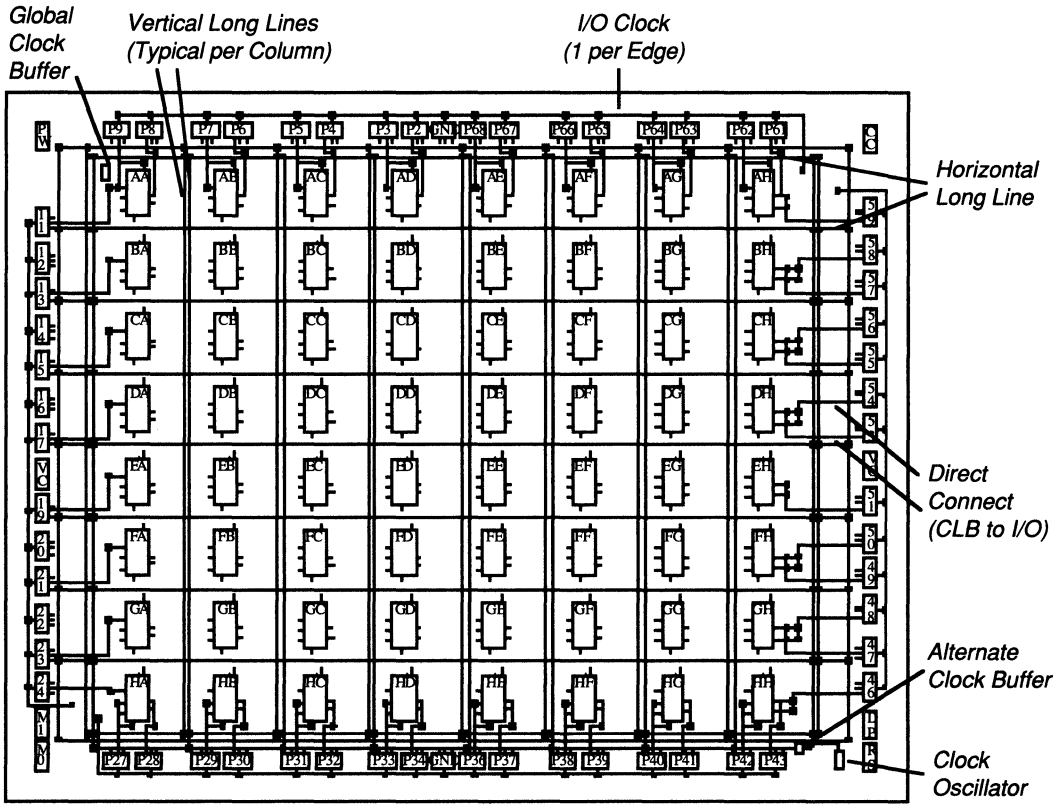
```
                                :
                                :
            For -30  (33 MHz)  speed device
                                :
                                :
                                :
From:       BLK GA              (CLOCK to GA.X)     :    20 ns   ( 20 ns)
Thru:       NET S6              (GA.X to HA.A)  :         0 ns   ( 20 ns)
To:         BLK HA              (HA.A to SETUP)     :    12 ns   ( 32 ns)*
                                :
                                :
            For -50  (50 MHz)  speed device
                                :
                                :
                                :
From:       BLK GA              (CLOCK to GA.X)     :    15 ns   ( 15 ns)
Thru:       NET S6              (GA.X to HA.A)  :         0 ns   ( 15 ns)
To:         BLK HA              (HA.A to SETUP)     :     8 ns   ( 23 ns)*
                                :
                                :
* Total Clock-to-Clock Worst-Case Delay
```

In designs that are not highly synchronous, such as those in glue logic replacement, you usually cannot exploit direct connections to the degree shown in the example. However, if possible, use a direct connection to route a signal from one block to another. Direct

connection considerations should be a primary factor in block placement. For each direct connection, you release a general interconnection resource that can then be used in some other function. Extensive use of direct connections can boost the logic use of the LCA device by up to 30%.

5.2.3 LONG LINES

Long lines are continuous metal segments that span the width or length of the LCA device, providing **minimum-delay and skew for long distance signal paths**. Although the automatic router uses long lines for general signal routing when other types of connections are not available, you should direct the use of long lines for specific signals. Ensure that long lines are efficiently used by considering their capabilities and interconnection potential. The following figure illustrates the locations of the long lines and shows the clock buffers that work with them. Clock buffers are described under 5.2.4.

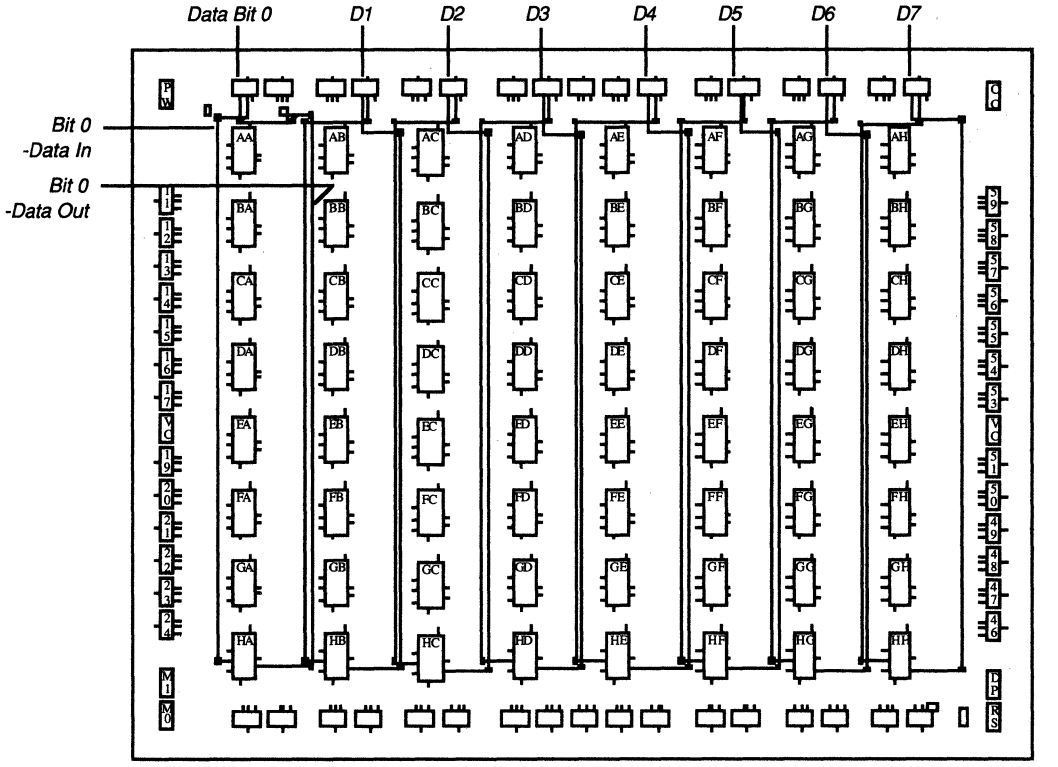


2064 Long Lines and Clock Buffers

Signals that can most effectively use long lines are generally classified as data distribution, or low-skew, control signals. Whether originating at an I/O block or a CLB, data signals typically have several destinations each of which uses the data differently. To follow the natural data flow of the device, you should route these signals on long lines with one bit per row or column.

An important consideration in data routing is the direction of the data flow. In the 2064/2018 series of LCA devices, internal signals must be unidirectional. For systems that require bidirectional data paths, you can use a pair of long lines in each column to carry input

data and output data, respectively. This bidirectional signal routing requires that the data input/output pins be located at the top or bottom of the device. The following figure shows an 8-bit bidirectional data bus with vertical routing on pairs of long lines.



Bidirectional Data Bus Using Long Lines

Control signals, such as clocks, reset/set controls, and count or shift direction controls may have critical timing requirements between their source and their multiple destinations. For these signals, you must control skew to ensure that all of the destination blocks perform the desired function at the same time or on the same clock edge. If possible, you should arrange destination blocks in a single column or row and you should route the control function onto the appropriate long line. The

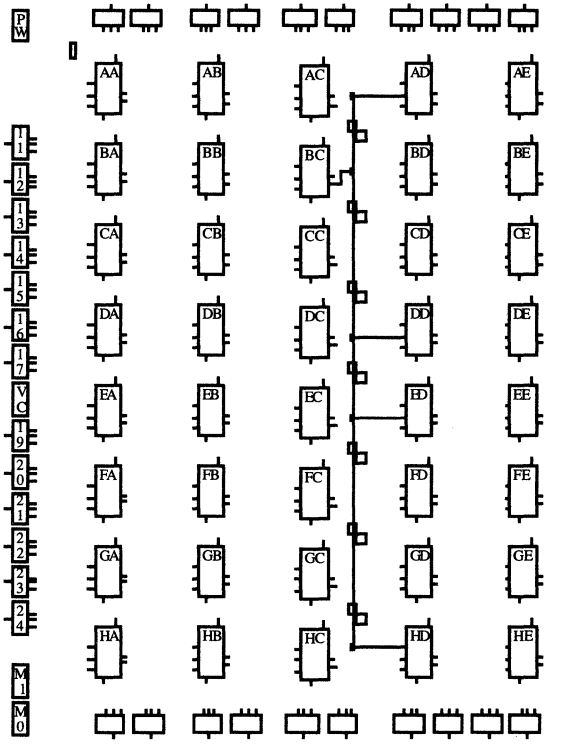
figures below show two different implementations of a reset function generated in a logic block and routed to four destination blocks. You can see the skew reduction associated with the use of the long line from their accompanying tables.

Delay: bidibus8.1ca, XACT 1.3

From:	BLK BC	(BC.X)	:	0ns	(0ns)
Thru:	NET RESET	(BC.X to HD.B)	:	24ns	(24ns)
To:	BLK HD	(HD.B)	:	0ns	(24ns)
From:	BLK BC	(BC.X)	:	0ns	(0ns)
Thru:	NET RESET	(BC.X to ED.C)	:	14ns	(14ns)
To:	BLK ED	(ED.C)	:	0ns	(14ns)
From:	BLK BC	(BC.X)	:	0ns	(0ns)
Thru:	NET RESET	(BC.X to DD.C)	:	12ns	(12ns)
To:	BLK DD	(DD.C)	:	0ns	(12ns)
From:	BLK BC	(BC.X)	:	0ns	(0ns)
Thru:	NET RESET	(BC.X to AD.C)	:	3ns	(3ns)
To:	BLK AD	(AD.C)	:	0ns	(3ns)

21ns SKEW

DELAYS FOR GENERAL INTERCONNECT



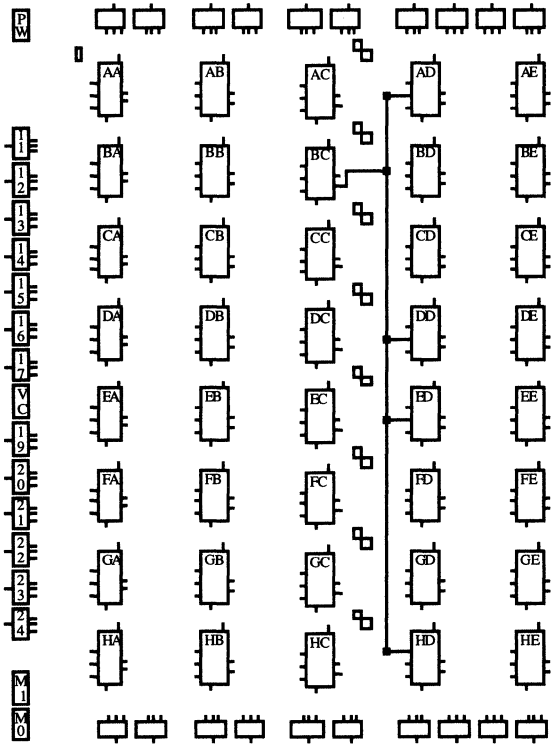
Signal Routed Via General Interconnection

Delay: bidibus8.1ca, XACT 1.3

From:	BLK BC	(BC.X)	:	0ns (	0ns)
Thru:	NET RESET	(BC.X to HD.B)	:	5ns (	5ns)
To:	BLK HD	(HD.)	:	0ns (	5ns)
From:	BLK BC	(BC.X)	:	0ns (	0ns)
Thru:	NET RESET	(BC.X to ED.C)	:	5ns (	5ns)
To:	BLK ED	(ED.C)	:	0ns (	5ns)
From:	BLK BC	(BC.X)	:	0ns (	0ns)
Thru:	NET RESET	(BC.X to DD.C)	:	5ns (	5ns)
To:	BLK DD	(DD.C)	:	0ns (	5ns)
From:	BLK BC	(BC.X)	:	0ns (	0ns)
Thru:	NET RESET	(BC.X to AD.C)	:	5ns (	5ns)
To:	BLK AD	(AD.C)	:	0ns (	5ns)

0ns SKEW

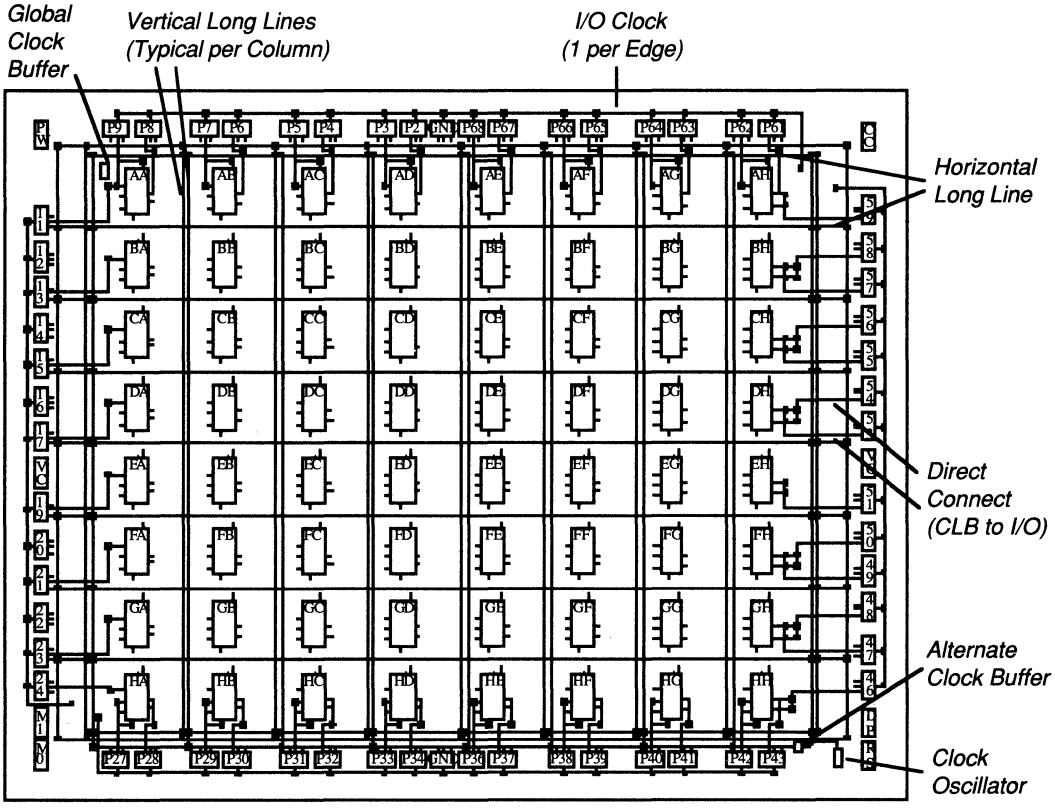
DELAYS FOR ROUTING VIA LONG LINE



Signal Routing Via Long Line

5.2.4 CLOCK BUFFERS

The LCA device has special-purpose on-chip buffers to provide high-fanout, low-skew signal distribution. You normally use these buffers for clock signals; however, you can also use them for any general-purpose signal that requires **high-fanout or low-skew routing to multiple blocks**. Clock buffers work with specific long lines for routing on a column basis. The global clock buffer, located in the upper-left corner of the LCA device, directly drives a long line in each column. The alternate clock buffer, located in the lower-right corner, drives a horizontal line that can be selectively connected to a long line in each column. The following figure shows the buffer locations.



2064 Clock Buffers and Long Lines

In systems with a single common clock for all state elements, you can best distribute that clock using the global buffer. More difficult cases involve systems with multiple clocks and other critical control signals. If a system has two separate clocks, one clock can use the global buffer and the other can use the alternate buffer. This is particularly true when one clock is derived from the other.

Note: When you use the crystal oscillator, its output drives the alternate buffer directly; therefore, it can be the primary clock for the system.

In any case, you should drive the predominantly used clock with the global buffer.

For systems requiring more than two clocks, you should still drive the primary clock with the global buffer. Route other clocks onto vertical long lines and arrange their respective CLBs in columns adjacent to the long line that carries the appropriate clock signal. You can use direct connections to drive these column-oriented clocks either from an adjacent (to the left) CLB, or from an I/O block on the edge at either end of the column. When selecting the long line to be used, note that one of the nondedicated vertical long lines can be connected to the CLB K inputs, while the other cannot. Since most clock signals are best routed into the K input, you should choose the former long line.

You can also use one of the clock buffers to route a control signal to many CLBs or IOBs. By placing the source of the signal near the alternate buffer, you provide a low-delay path from the source to the buffer, and then to all the destination CLBs or IOBs. The following figure shows a shift register that has been placed and routed using both buffers: the global buffer for the overall shift register clock and the alternate buffer for a low-skew shift/hold control signal. If the shift/hold control logic timing is not well controlled, skews in the control signal, as seen by the blocks, could cause a partial shift. In a partial shift, some blocks could get the signal while others may not get it in time to hold relative to the next clock edge.

Note: This timing skew becomes less of a factor in choosing routing for control signals as you relax your timing constraints.

5.3 PLACEMENT

Placement and routing determine how efficiently you can use an LCA device for a particular design. **Placement of logic in an LCA design primarily involves determining the relative locations of the functions that most effectively use CLB, IOB, and routing resources.** Efficient use greatly simplifies signal routing within functions and between groups of functions.

The following discussion explains how to accomplish efficient logic placement in your LCA designs. First, it provides some guidelines for system-level design partitioning to help you determine what parts of your system to implement in an LCA design. Then, it explains how to analyze the data flow for your LCA design. Finally, it describes CLB and IOB placement, and gives some examples.

This discussion has the following organization.

- 5.3.1, Partition the System Design
- 5.3.2, Analyze the Data Flow
- 5.3.3, Place the Logic Blocks
- 5.3.4, Place the I/O Blocks
- 5.3.5, Examples
- 5.3.6, Modification Guidelines

5.3.1 PARTITION THE SYSTEM DESIGN

You must partition your system-level design before you can implement an LCA design. Partitioning is a two-part process. First, you separate your design into external and internal LCA functions; then you group the internal LCA functions into related clusters.

The following are useful guidelines for determining what part of your design to implement in an LCA device.

- Implement **standard LSI** or VLSI functions with appropriate components.

- Use an LCA device to implement **non-standard** VLSI functions if a large portion or all of the LCA device is available for that function.

- Place required portions of **standard** VLSI functions inside an LCA device.

- Place **random** SSI and MSI functions inside an LCA device. You can implement functions or sub-functions that require four or fewer inputs, two or fewer outputs, and a single storage element using a single CLB.

- Group all parts of a **complex function** inside one LCA device, instead of placing them at remote locations in the circuit; this simplifies implementation and debugging.

- As a rule, use MSI components to implement a function with a large fan-out or fan-in and with few logic levels. Decoders and multiplexers fall into this category.

5.3.2 ANALYZE THE DATA FLOW

After determining which part of your system design to place in an LCA design, you must analyze that part of the design to determine optimal data flow. You analyze the data flow of your design as follows.

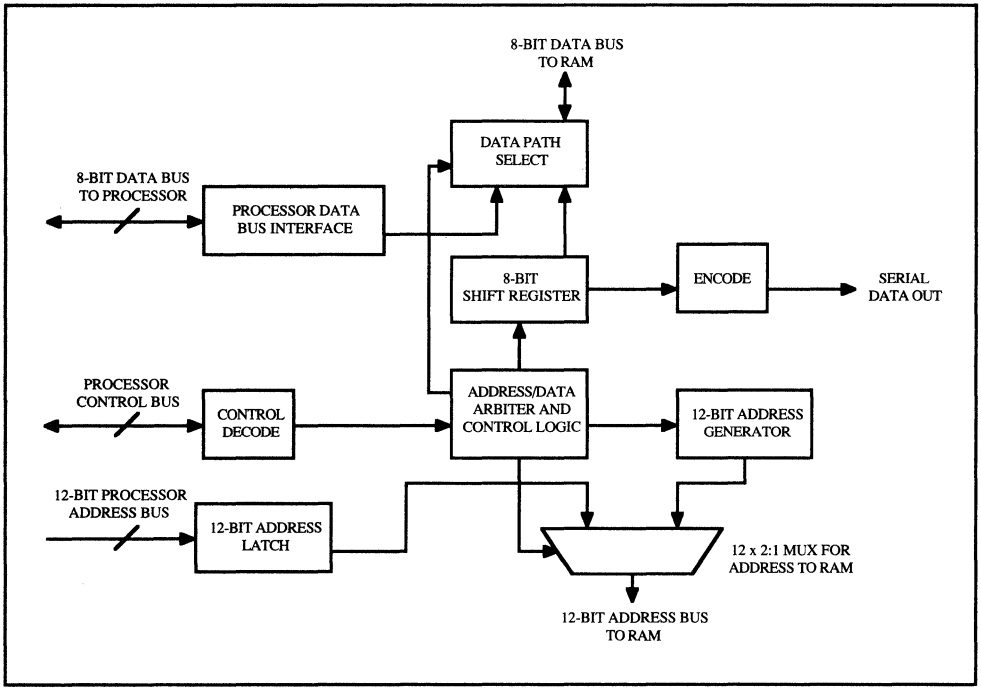
To make placement and routing decisions, you evaluate the sequential nature of the logic to be created in the LCA device. Examine the data flow to determine the best placement of required logic blocks, and to determine the most effective signal routes.

In general, data processing in the LCA device flows most naturally either from left to right or vertically; flow up and flow down are virtually identical. **To minimize interconnection**, arrange structures such as counters in nearly square rectangles. Implement

registers in columns of CLBs to take advantage of the direct interconnection between adjacent CLBs and the common clocking in columns of CLBs.

If you must implement several independent functions, you should position the function that requires the most CLBs so that the data flows through it from left to right. The exceptions are counters and shift registers, which should be arranged in columns to share common clocking.

To illustrate data flow analysis, consider the block diagram below.



Serializer Block Diagram

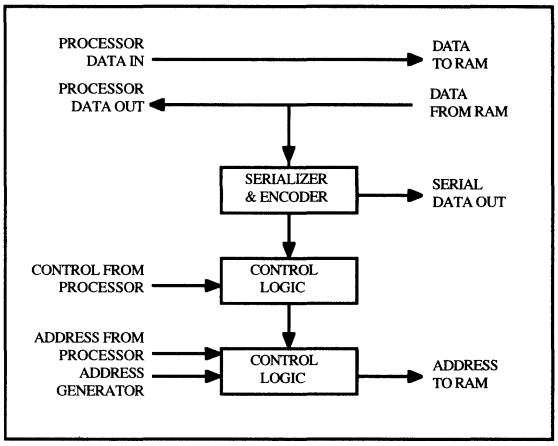
This diagram shows a dual-ported memory interface used as a high-speed serializer, which is a typical application in video pixel processing or serial

communications systems. In the serializer, both the serial-data output device and the microprocessor must have access to the memory. In analyzing the data flow, you can see a need for an 8-bit bidirectional path between the memory and the microprocessor, as well as an 8-bit path from the memory to the serializer. The serializer requires an 8-bit parallel-to-serial data flow. The interface generates the memory addresses internally for the serialization process; external memory addresses are supplied by the microprocessor. The memory addresses are always outputs to the external memory.

The data flow paths are summarized below.

- 8 bits of data from microprocessor to memory
- 8 bits of data from memory to microprocessor
- 8 bits of data from memory to serializer
- 12 bits of address from address generator to memory
- 12 bits of address from microprocessor to memory

Clearly, the design requires a bidirectional data path between the memory and the microprocessor. This same path must supply data to the serializer. You can view the serializer as a process **perpendicular** to the data flow because it serializes parallel data. The memory address path is wider than the data paths but it is unidirectional and has a common connection only at the output point. The following diagram shows a flow analysis of this design example.



Data Flow Analysis of Serializer

Based on this flow analysis, place the 8-bit data path vertically to take advantage of direct connection in the up and down directions. Then place the serializer, which could be connected in a left-to-right or vertical orientation, perpendicular to the vertical data path and use the direct connection left-to-right capabilities. Route the address path between CLBs, which are near the edge of the device, and the adjacent I/O blocks, that drive the address. This can be done because the path is unidirectional. Use direct connection as much as possible.

You should follow the general guidelines below to determine the block placement and the routing alternatives within the LCA device.

5.3.3 LOGIC BLOCK PLACEMENT

One of the most critical elements in achieving an efficient design with an LCA device is the proper placement of CLBs and IOBs. CLB placement is more critical than IOB placement for two reasons.

- it offers more degrees of freedom

- the final CLB placement can dictate most of the IOB placement.

You can improve both the performance and the routing of your design by proper placement. Good placement relieves routing problems and generally results in good initial performance, minimizing the placement and routing iterations.

Note: Maximizing the use of the direct connections between blocks is an important goal.

5.3.3.1 Placement Guidelines

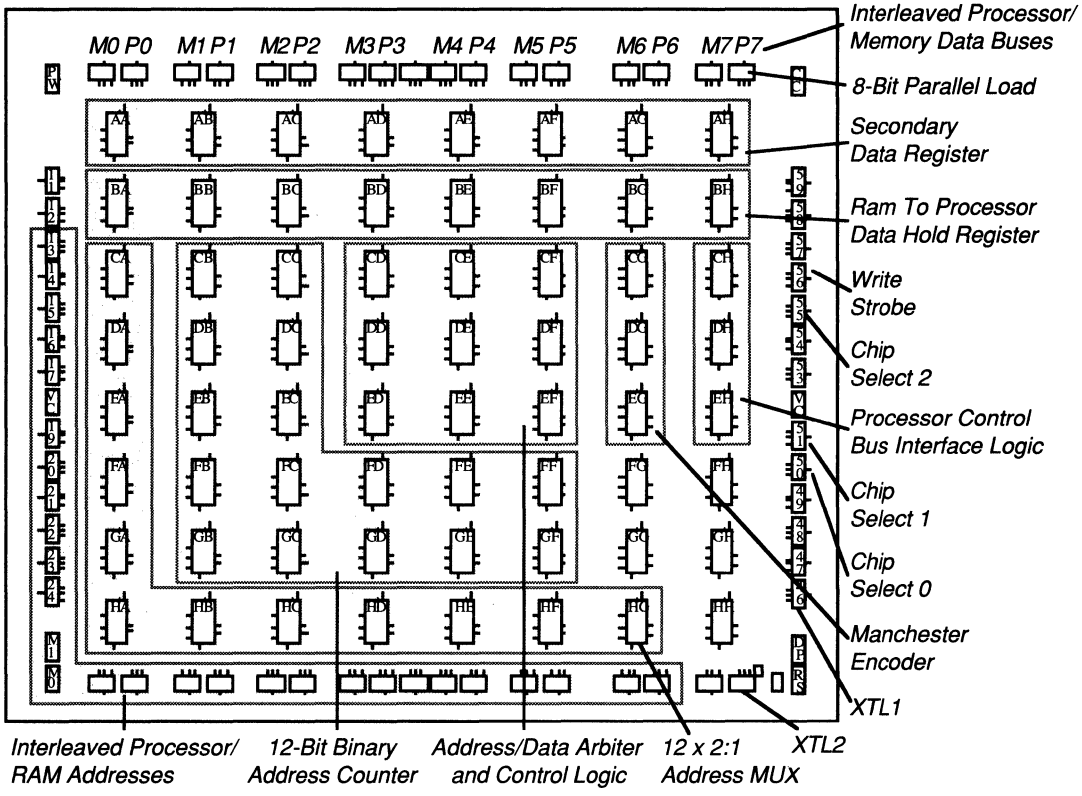
The following guidelines should help you achieve optimum placement efficiently.

1. Consider the various functional elements in the design, the shapes that each can take, and their relative interconnection.

   Try the placement of these functional blocks on a printout of a blank LCA device to see how they might fit together. The layout in the following figure was obtained using this basic analysis technique.

2. Examine the internal and external inputs and outputs for each block of logic.

   Place blocks with a high number of common interconnections near each other.



Serializer Placement Plan

When you consider the relative placements of individual CLBs and IOBs, remember the following.

3. Use direct connections wherever possible.

4. Arrange related groups of logic blocks in rectangular shapes, if possible.

5. Place CLBs and IOBs with the greatest number of interconnections next to other blocks at the perimeter of any rectangular shapes.

6. Arrange logic blocks in a **long, thin** shape only where data flow through them to some other logic that is perpendicular to the shape's long axis.

7. Place blocks of control logic or miscellaneous functions that have minimal external I/O near the center of the device.

8. Minimize the number of different clocks in the design where possible, particularly those clocks generated internally. A completely synchronous design with a single clock that uses the global clock buffer is ideal.

Many of these recommendations are similar to those applied to the layout of printed circuit boards using SSI/MSI devices. The examples following the discussion of I/O Block placement, below, should help illustrate effective placement.

5.3.3.2 Optimization Guidelines

The following guidelines will help you optimize the placement of CLB designs.

1. Use only necessary functions. Many designs use only a portion of a standard-logic part and disable the unused inputs. **LCA-based designs should never include unused inputs of standard-logic devices.**

2. **Use the function, not the equivalent logic gates.** Many standard-logic designs use multiple gate levels. LCA-based designs are not subject to the same logic-gate restrictions; you can create any function of three or four variables, regardless of complexity, with one CLB.

3. **Share CLBs wherever possible.** Two independent functions can share one CLB. For example, a CLB configured as a data latch uses

one input and one output. The remaining three inputs, the remaining output, and the output of the latch can be used to perform another function. A data-latching function and a function that uses the latched data can both be incorporated in one CLB, which effectively expands the CLB's four inputs to five.

4. **Group common intermediate outputs into one function.** If the same sub-function is performed for several inputs, CLBs are wasted. Perform the function to generate the intermediate output and use the single result in each place required. A common technique is to divide the design into functional pieces and look for commonality.

5.3.4 I/O BLOCK PLACEMENT

The placement of I/O blocks usually is dictated by the placement of the CLBs connected to them. However, you must consider IOB placement constraints because they can have a significant impact on overall placement and routing. General guidelines for I/O block placement are listed below.

1. Locate IOBs adjacent to the CLBs that use the most associated signals.

If I/O blocks are being used as buses, note the following considerations.

2a. Locate data buses that are to be latched on a single device edge to allow use of the flip-flops in the I/O block, and to share the single I/O clock on the edge of the device.

2b. Limit address buses to the top of the device if the pins are used during configuration as the external EPROM/ROM address lines.

3. Make unused IOBs data or shift registers.

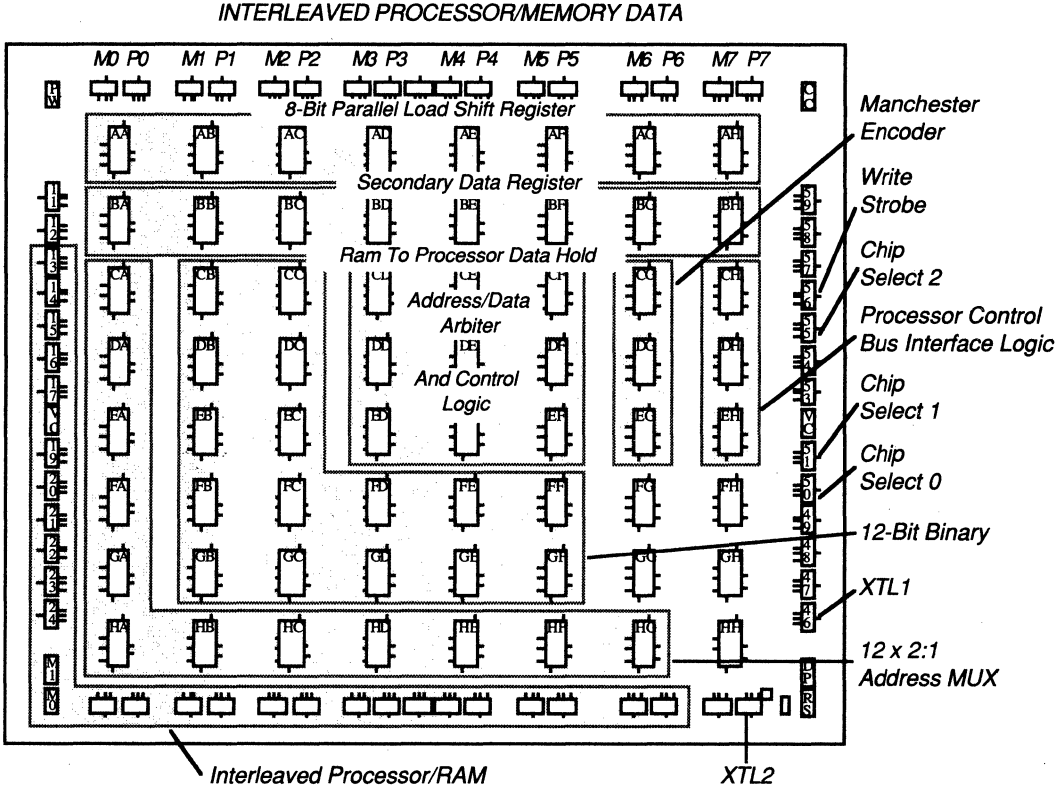
Note: The IOBs must have available the I/O clock on that edge of the device.

When specifying IOB usage, use the following guideline.

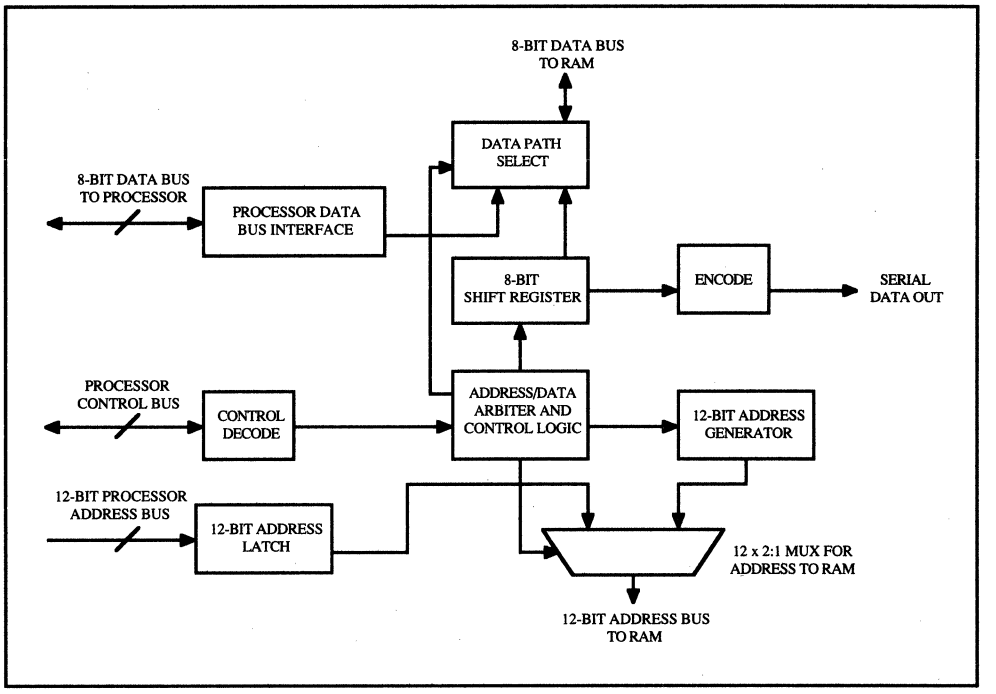
4. Note which pins have special functions during configuration. In general, you should use pins that are inputs during configuration as user inputs during operation, while using pins that are outputs during configuration only as user outputs during operation. Careful specification generally eliminates any possible contention between configuration use and operation use.

5.3.5 EXAMPLES

The following examples illustrate many of the placement and optimization guidelines discussed above. This sample design is the data serializer used previously for data flow analysis. Individual elements of the design are used to illustrate each topic. The first figure below repeats the LCA layout of the data serializer. The figure following it shows the data serializer block diagram.



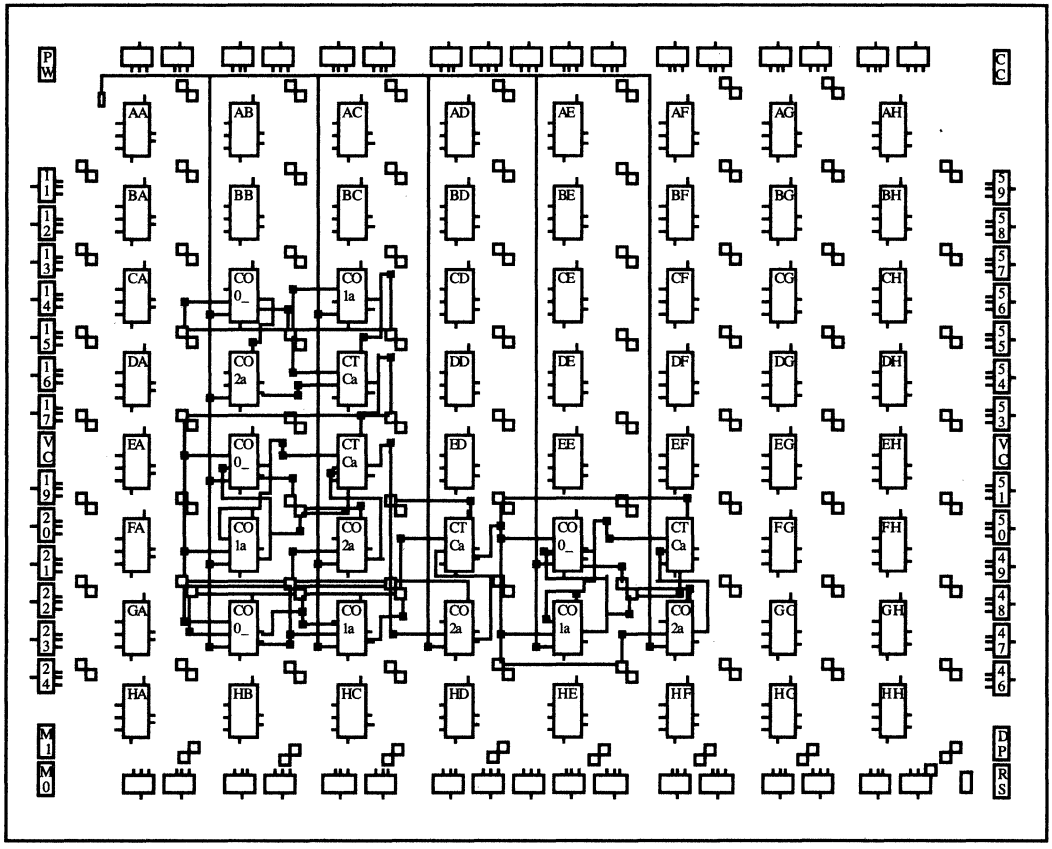
Serializer Placement Plan



Serializer Block Diagram

5.3.5.1 Using Macros, Example 1

Macros are one method of implementing individual functional blocks. The address generator portion of the serializer is shown below. This function is a 12-bit binary counter that addresses the external RAM holding the data to be serialized. In generating the counter, this example uses macros, each of which represents three bits of the counter.



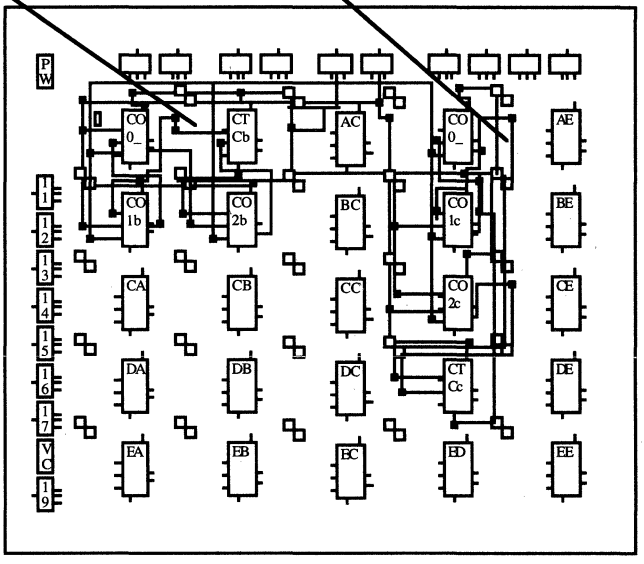
12-Bit Address Counter

The placement of the logic blocks in the macro illustrates the advantages of rectangular placements.

The next figure shows the 3-bit macro C8BCR (Counter, modulo 8, Binary sequence, Clock enable and Reset synchronous controls) placed in two ways. Placement A is linear. Placement B uses the recommended placement from the LCA Macrocell Library manual.

Placement B:
Local Routing Does Not Block Other Routes

Placement A:
Routing Congestion Makes Other Vertical Routes Difficult



3-Bit Counter Macro (8BCR) Placement Alternatives

Notice that in the linear placement, all of the signals must travel the height of the macro to reach the terminal count (CTC name) block. This placement congests the routing in the columns to the left and right of the column containing the macro. The rectangular placement shown in B makes the routing more compact and provides additional space for routing around the module. Also, the square structure allows easier placement in a dense design.

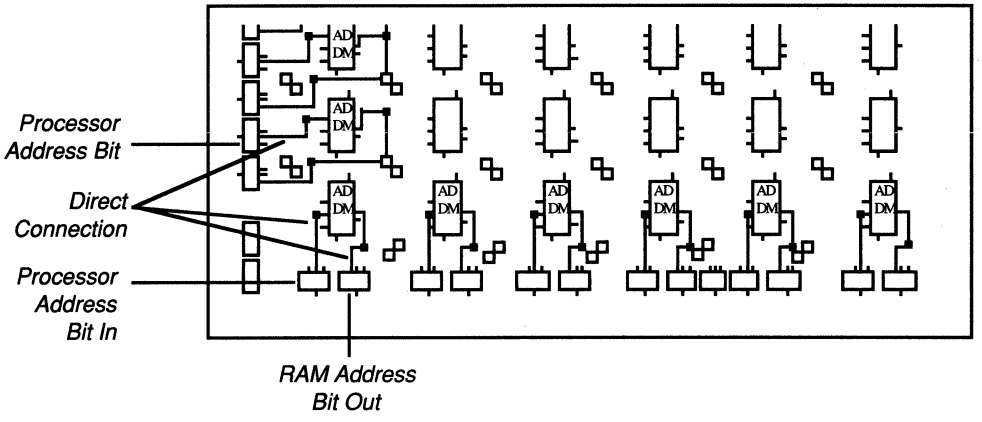
5.3.5.2 The Long and Thin Approach, Example 2

An example of the long and thin approach is the data shift register at the top of the data serializer shown in discussion 5.3.5. The data comes in from the pins at the top and flows into the blocks of the shift register. If these blocks are placed in a traditional rectangular shape, some bits would have to travel long distances to reach the appropriate shift register block. With this long, thin arrangement, the secondary data register in the B-row of CLBs can receive the data as it flows from the pins through the shift register.

The address multiplexer also uses long, thin shapes, with emphasis on the direction of the information flow. The next figure shows how two different bits of the multiplexer use the direct connection paths. Direct connection can reduce the congestion in general interconnection and improve placement. The initial placement has alternate I/O blocks connected either to the memory or processor address bus. After examining the direct connection, use it by modifying the position of the processor bus interface I/O relative to the multiplexer block. This provides an input to each block. Along the bottom, place the processor blocks to the left of the memory block of the same bit, thus taking advantage of direct connection for input and output paths.

5.3.5.3 Trade Off Resources for Performance, Example 3

In some cases, you may need to make placements that trade off resource use for performance. In the following example, the primary performance-limiting element is the speed of the address generator, in particular the 12-bit counter. The initial serializer placement plan shown in discussion 5.3.5 uses macros to build the counter. Macros let you create the function quickly but may not provide optimum resource use or performance.



Use of Direct Connection in Address MUX

Each 3-bit section of the counter can operate at high speed because only a single logic function is required between clock edges. However, additional logic block delays inserted between each 3-bit stage reduce the overall performance. To obtain higher performance, you could generate the toggle condition of each bit in the counter in the minimum number of logic levels. This optimization requires approximately 18 blocks and much more care in placement and routing.

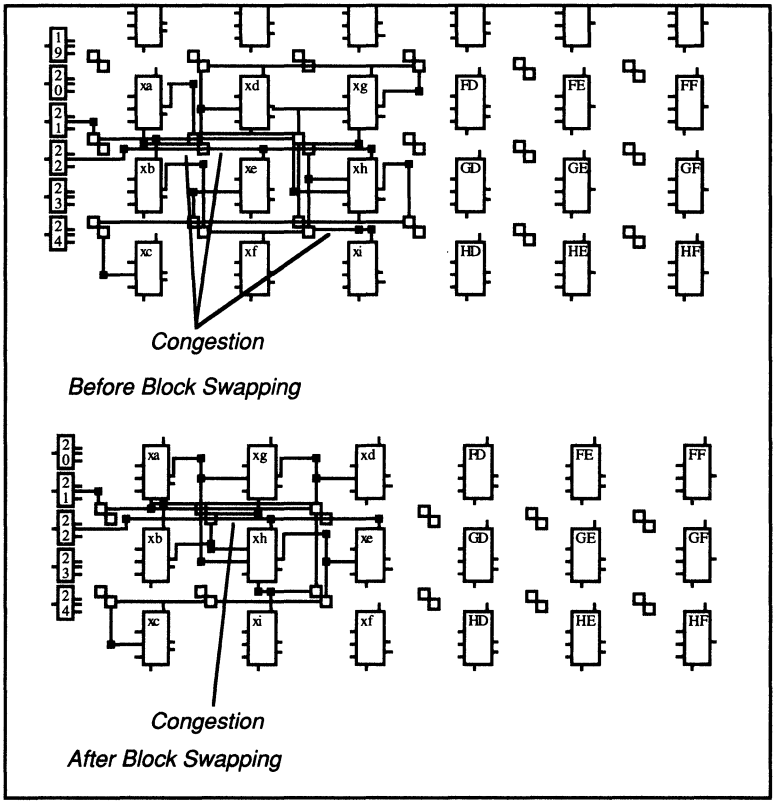
5.3.6 MODIFICATION GUIDELINES

If your initial placement of logic and I/O blocks fails to produce a design you can readily route, you must modify the placement. Some guidelines for doing this are described below.

If congestion exists in the middle of the placement, make the following changes.

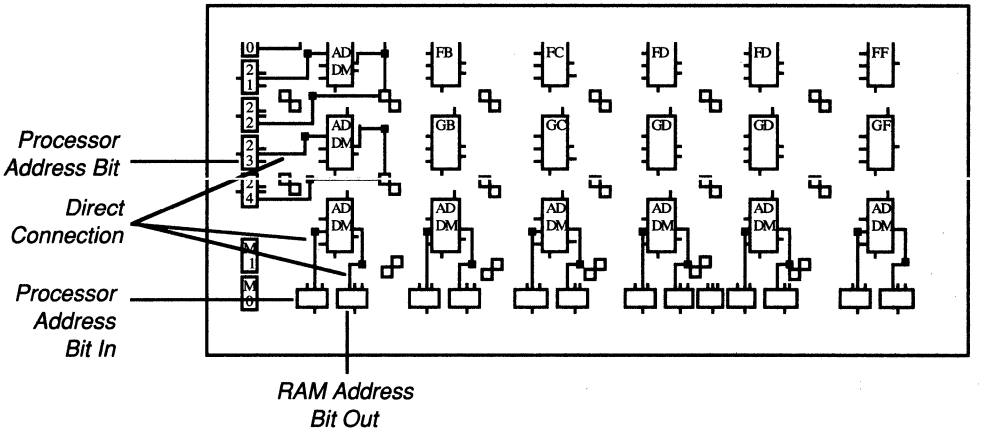
1. Move the blocks inside the congested area to the outside and move the outside blocks inside. This method of **turning it inside out** normally alleviates congestion, except in cases where the original exterior blocks have a large number of

connections to resources outside the area being examined. The next figure shows a block of logic that has interior congestion and an alternative placement that relieves the congestion.



Block Swapping to Relieve Congestion

2. Spread I/O connections out, rather than clustering them together. This often relieves congestion near the edge of the device. Some I/O-intensive applications can benefit from interleaving related I/O blocks, which is done in the address bus area of the next figure.



Use of Direct Connection in Address MUX

3. Orient the majority of signals vertically to improve the placement for groups of logic with horizontal routing congestion. Remember, there are effectively 9 vertical connections in each column (five general-purpose, three long lines, and one direct connection to the block above and below), and only six horizontal connections in each row.

4. Move data register functions out of the logic block areas and have unused I/O blocks perform that function. This approach is particularly effective for function control registers written with an external data bus. You can use pins adjacent to the data bus input pins for direct data input connections.

5.4 ROUTING

This discussion explains manual routing and describes how to edit and pre-route your design's interconnection manually using the LCA development system.

The LCA routing resources consist of general purpose interconnections, long lines and direct connections from a block to adjacent blocks, and special buffers. You must balance the use of routing resources with the partitioning and placement of the logic to generate a complete design.

5.4.1 MANUAL EDITING

With some LCA designs, you must interact with the routing process to perform any of the following tasks.

- Relieve congestion to route a signal.
- Force use of selected resources to meet specific performance or use criteria.
- Modify existing routes to tune delays for a particular requirement.
- Complete the routing on a dense, partially-routed design.

EDITNET is the XACT design editor command used to manually route signals or nets. EDITNET selectively enables or disables any of the **programmable interconnection points** (PIPs) on the device with the following operations.

1. Select the EDITNET command, either with the mouse and cursor or type EDITNET from the keyboard.
2. Specify the net you wish to manipulate manually. The net must have the source and destination connections on block pins defined.

3. Move the cursor over a PIP for the desired path and press the SELECT button on the mouse to toggle the selected connection.

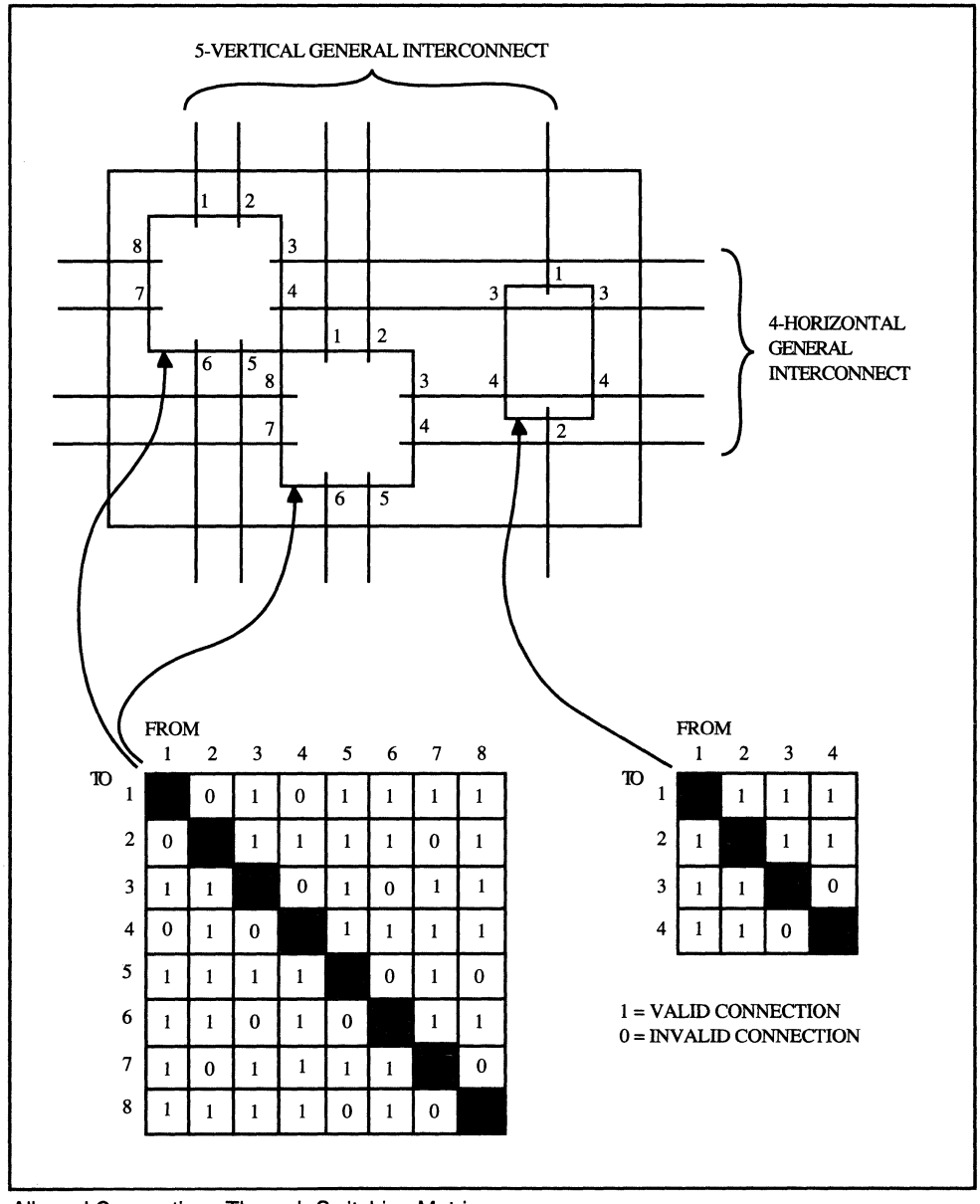
   - If the PIP was previously connected, pressing SELECT disconnects it.

   - If it was not connected, pressing SELECT connects it.

To modify the switching matrices located where the general interconnection segments meet, you must

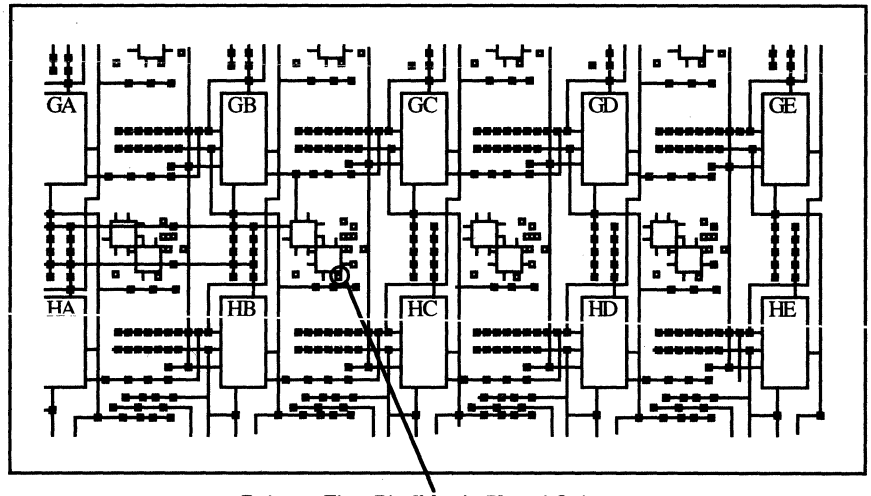
4. Select a pair of magic pins, which are connected to the switch matrix.

The table below shows the allowed connections for the various switching matrices. You make or break connections by selecting the desired pair of pins. Selecting the second pin breaks current connections or connects unconnected pins.

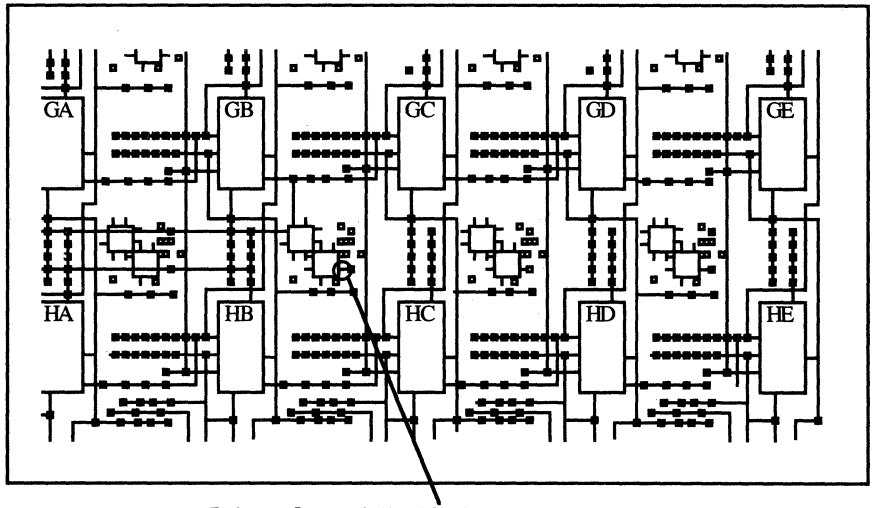


Allowed Connections Through Switching Matrices

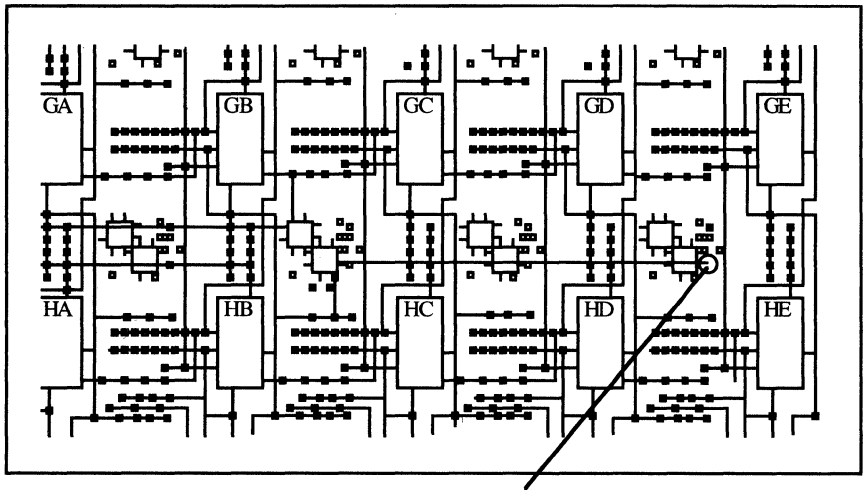
The following figures show the sequence of operations for editing connections in the switch matrices.



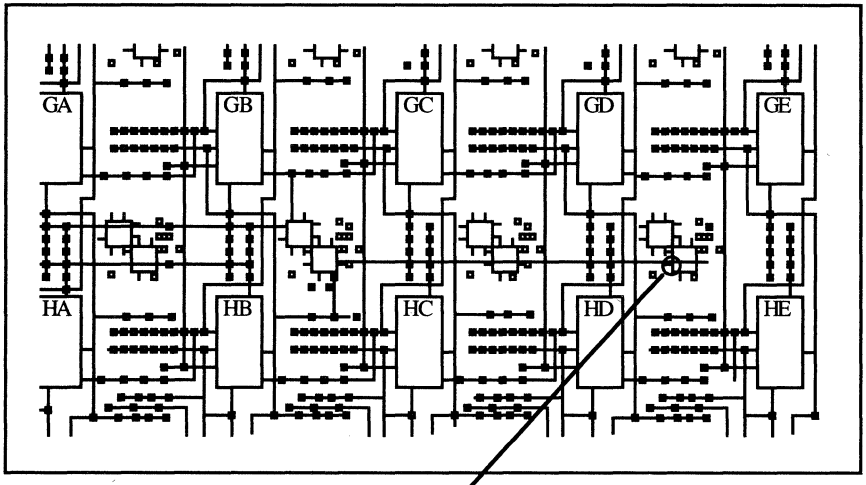
Point to First Pin [Magic 5] and Select



Point to Second Pin [Magic 4] and Select

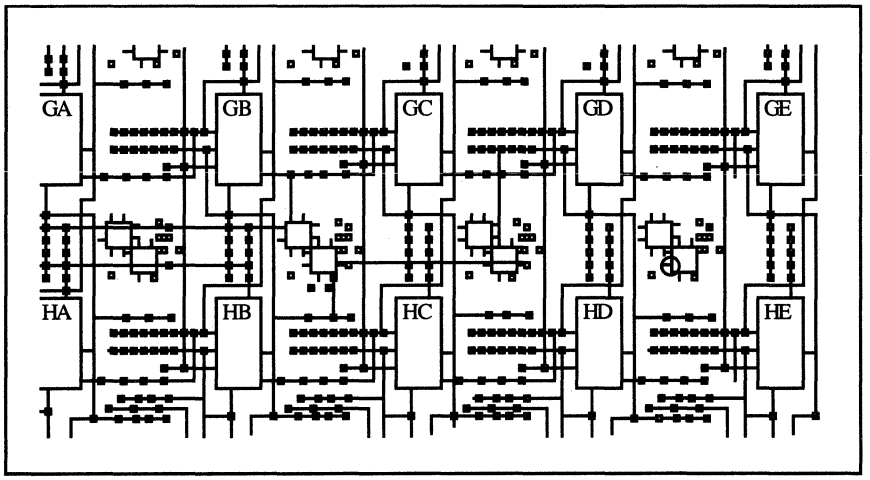


Result - Disconnect by Point to First Pin [Magic 4] and Select



Point to Second Pin [Magic 7] and Select

Result



Sequence of Operation for Connecting Through Switch Matrices

When you have made all the connections,

5. Select the DONE option. The XACT system automatically calculates the delays associated with the interconnections and makes them available for display. The delay from the source of a net to its destination is shown whenever you position the cursor at a destination pin.

When you use EDITNET to make a connection, this error message may appear: *connection shorts pin zz.v* . This indicates that with this connection you have assigned a signal to a block pin that has not been assigned to the net being routed. If you must connect that pin to the net, assign it using the ADDPIN command.

Although you cannot directly connect certain pairs of switch matrix pins, you can use a combination of valid connections to accomplish the desired routing. For example, a connection from pin 1 to pin 4 is not valid; however, you can accomplish it by connecting pin 1 to pin 5, and pin 5 to pin 4. This routing connection

causes an additional switch delay, yet may be essential for routing in a congested area.

Some additional routing guidelines are given below.

- Do not route through inputs and outputs.
- Do **seed** the routing for a net before using auto-routing.
- Do place **pre-route** selected nets onto long lines.
- Do route high fanout items first or last.

Pre-routing, or seeding the routing, is explained below.

5.4.2 MANUAL PRE-ROUTING

An effective technique to improve the resource use of the XACT router is to manually **pre-route** or **seed the routing** of particular nets prior to using the router. This seeding can take two forms, depending on the desired effect.

Even if you want to use a particular long line resource for a signal path, based on delays or general placement, the router typically does not route the signal onto that long line if an alternate path is available. One technique is suggested below.

I. Force a signal onto a long line to pre-route it onto the long line before you route the signal, as described in the stages below.

   1a. If you have already entered the net, use UNROUTE or CLEARPIN to deconfigure the routing for each pin on the net.

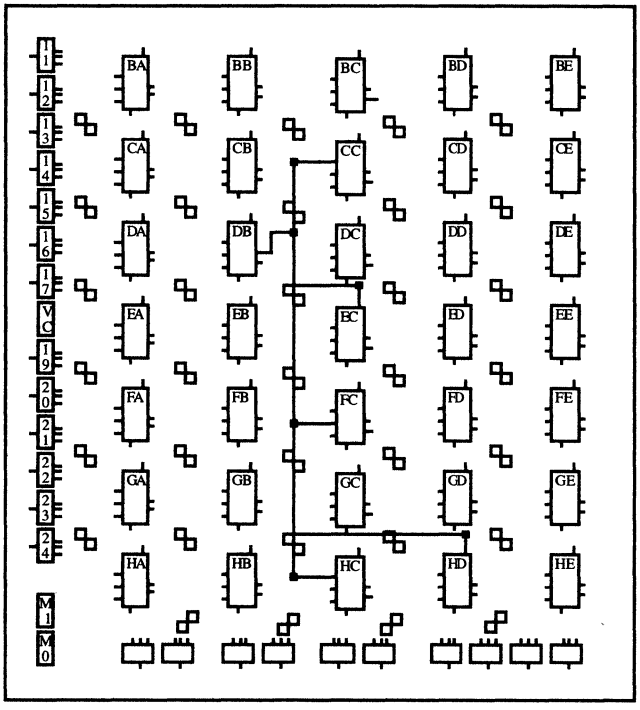
1b. If you have not entered the net, disable the automatic router with the AUTOROUTE OFF command on the PROFILE menu and define the net with the ADDNET command. This avoids the delays caused by routing each pin and the necessity to unroute them after they are entered.

2. Use EDITNET to choose the net to route on the long line and turn the appropriate switches on or off to get the signal from its source block onto the long line.

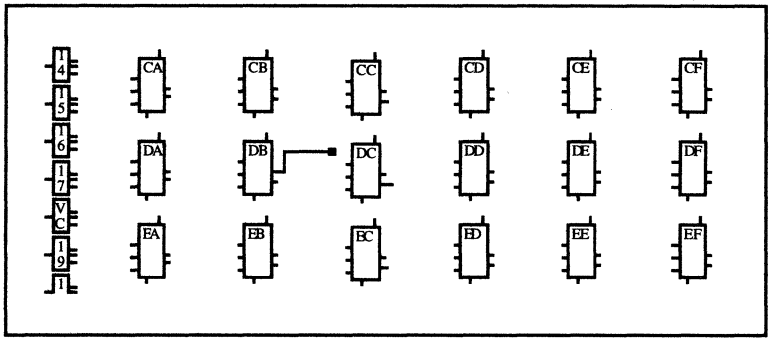
3. End the EDITNET command by selecting DONE. A warning message indicates that the net is not routed.

4. Select the ROUTE command from the NET menu, and when prompted, select the net you manually routed onto the long line. The router then completes routing that net.

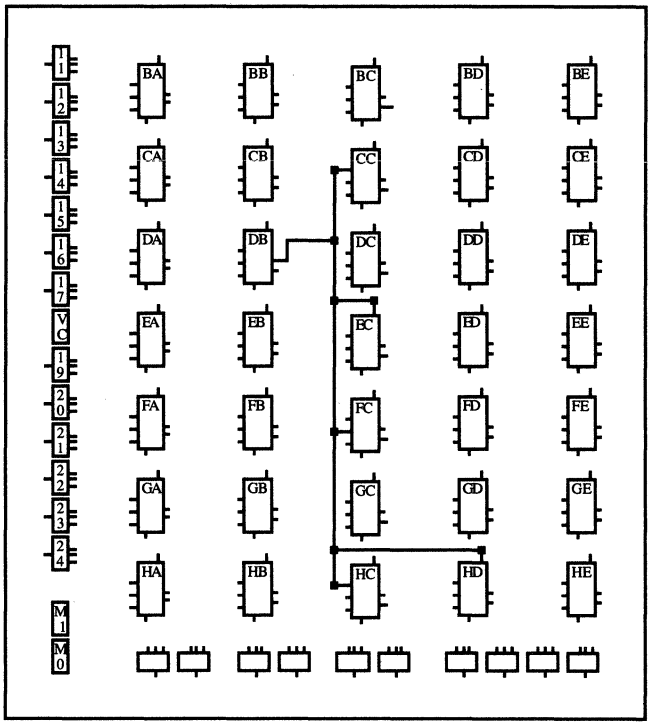
The technique above is illustrated in the two figures below. In some cases, where the destination pins are not directly accessible from the long line, the router still does not use the selected long line. In these cases you may need to use both techniques I, above, and II, below, to force the use of the long line.



Output DB.X Routed via General Interconnection



After UNROUTE Command; EDITNET has Forced Output onto Long Line



Result After Route of Signal, Using Long Line

In some cases, you add pins to a net throughout the course of a design. If you enable the XACT interactive router throughout this process, each pin is routed as you add it. The resulting net routing can become contorted and interwoven because each pin is routed independently. Extreme cases can have loops in the interconnection, or very long delays, as the source block becomes more heavily loaded and the routing more degenerate. Working with the automatic router enabled can also cause severe congestion in some areas, as the routing resources are unnecessarily consumed by the multiple routes. To help relieve this and similar multi-destination problems, you should use the following guidelines.

II. Enter the destinations in a sequence that progresses from the source location to the farthest destination. Remember, the router completes the routing from source to destinations in the order specified.

To avoid the necessity of entering destinations in location-specific sequence for large nets, you can do the following.

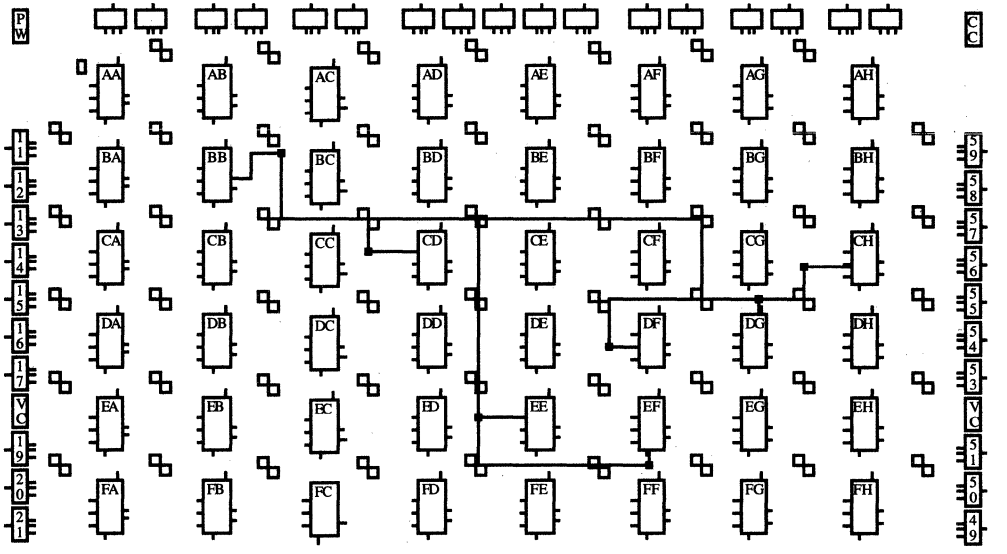
1. Enter the net into the design with the router disabled, or unroute the net as described in 1a, previously.

When you finish entering all of the destinations for the high fanout net, do the following.

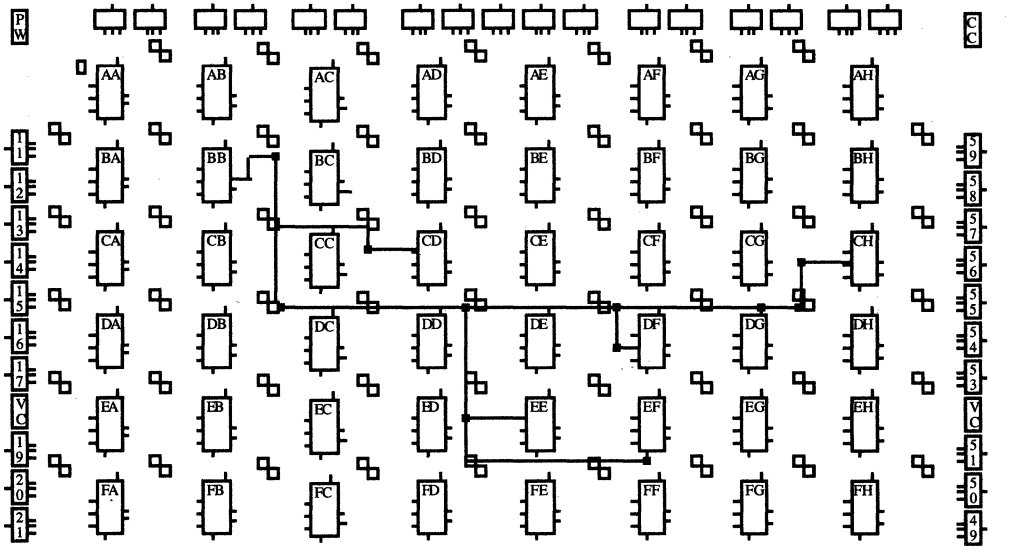
2. Use EDITNET to manually route to the destination most distant from the source. If the routing to this pin does not use a long line, you can use ROUTEPIN to accomplish the initial routing. ROUTEPIN is described under **Useful Routing Functions**, 5.4.3.3.

3. Use the ROUTE command to let the router complete routing of the other destinations in the net.

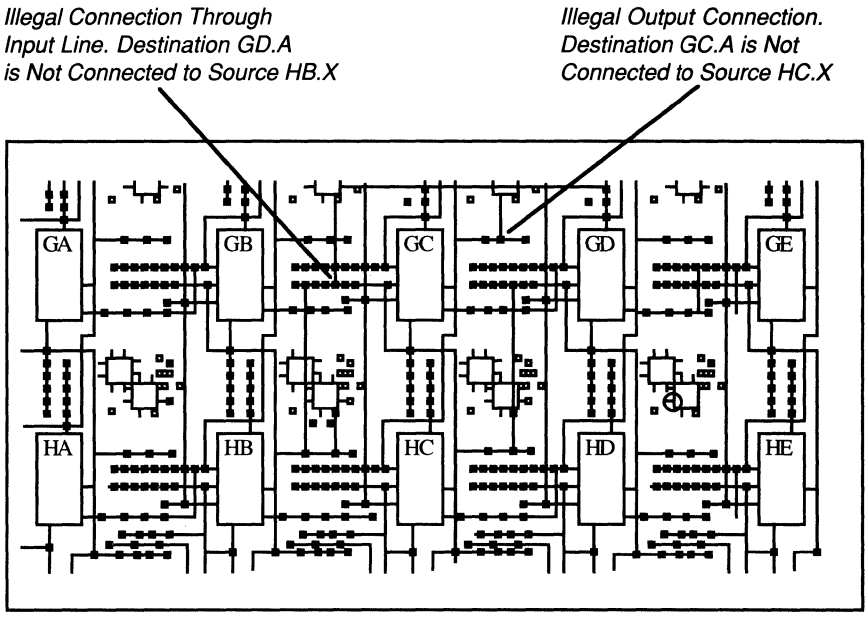
   The following figure shows the use of this technique for a net with many destinations. Another method to avoid entering destinations in the location-specific sequence is to use a text editor to modify the sequence of the destination pin specifications.



Routing Without Seeding



Routing After Seeding



5.4.3 ROUTING GUIDELINES AND FUNCTIONS

Following are some guidelines for routing inputs, outputs, and high-fanout nets, and a description of the XACT routing functions that are useful for design optimization.

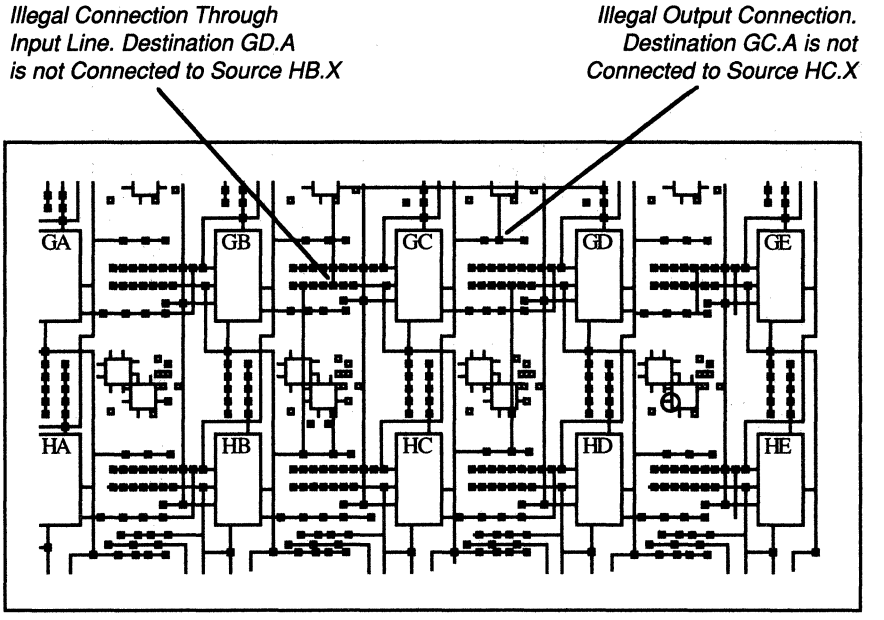
5.4.3.1 Inputs and Outputs

Although the inputs and outputs of the various blocks are shown as lines with multiple connections on them, you cannot use them as connections between parallel interconnection segments. Each input or output connection to a pin of a block is unidirectional, and only one connection per pin is allowed.

Tip: You should not route through inputs and outputs.

The EDITNET command lets you turn on multiple programmable connections to an input; however, only

the connection from the driving interconnection segment to the input pin is valid. Any additional connection points that are turned on do not connect to the driving segment, although they appear to be connected. If you execute the XACT design rule check command, DRC, it flags nets that have been routed in this way as *unrouted* and does not calculate their attendant delays. The following figure shows an improperly connected net routed through an input switch path.



Illegal Connections

Outputs of blocks can drive multiple interconnection segments, although this usually is not necessary. You can **not** use the output path switches to interconnect a net that is not driven by that block. The figure above also shows an improperly connected net using the output path switches. In both the input and output cases, you can only make these connections with the EDITNET command. To avoid these improper

connections, exercise caution when using this command.

5.4.3.2 High Fanout Nets

When you place and route a design involving high fanout nets, you often encounter congestion problems when entering it with the auto router enabled. In these instances, you probably must use alternate placements to complete a good design. You should follow this sequence of steps to route high-fanout nets.

1. Plan an initial placement on a blank LCA printout using the placement guidelines discussed above. Pay particular attention to the appropriate use of direct connection.

2. Start entering the design with the automatic router enabled. However, enter **only** the destination pins when you enter each high fanout net. Leave the source undefined, even though you know what it is. This lets XACT route more quickly, and results in a less cluttered design.

When you finish entering all of the regular nets, perform the following tasks.

3. Look at the congested areas. You can easily identify them by counting the used vertical and horizontal general interconnection segments in each column/row. A printout of the complete design, with the option SHOW USED enabled, can be helpful.

4. Save the design as a backup in case subsequent modifications fail to produce anything useful.

5. Generate a new placement by modifying the congested areas identified above. Use MOVEBLK and SWAPBLK to move the blocks to new locations. The criterion for the new placement

should be to eliminate as much congestion as possible.

6. Make the new placement with the automatic router disabled.

7. Route the high fanout net or nets using techniques 1 and 2 above. You should be able to route the high fanout nets optimally with this technique. View the design in either large or medium scale so that you can see as many blocks as necessary to find out where to locate the routing. Also highlight the high fanout net to show stubs at each of the required connections. This lets you see their physical relationships better.

8. Save the intermediate results as a backup.

9. Route all the remaining nets with the ROUTE* command, or route selected nets individually with the ROUTE command.

This iterative technique of manually routing selected nets should minimize routing problems and improve device performance. It can be applied equally well to nets with performance constraints and to those with fanout constraints.

5.4.3.3 Useful Routing Functions

There are several other useful routing-related functions you can use to optimize designs. These are SWAPSIG, CLEARPIN, and ROUTEPIN, all of which are discussed below.

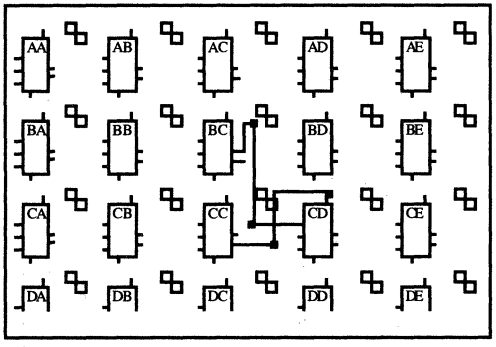
SWAPSIG

The SWAPSIG command, located in the PIN menu, is useful when you optimize the routing of a signal to a specific block. The SWAPSIG command logically interchanges the net connections of the block pins, and simultaneously changes the block function to match the new pin assignment of the signals.

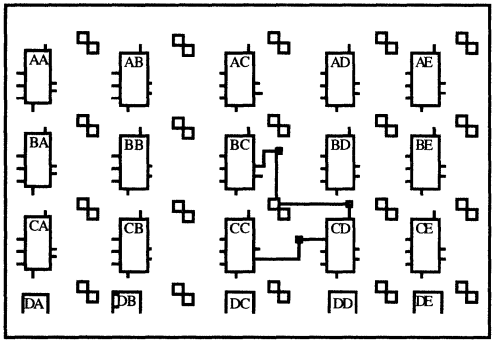
In many cases, signals are better routed to a specific block pin, in spite of the general interchangeability of the pins. The following figure shows some typical signal routes for which you can modify the choice of block pins to relieve routing congestion.

Note: You should always use SWAPSIG and not SWAPPIN when working with pins on a single block, because it modifies the internal function to match the pin swapping.

SWAPPIN is valuable for moving a net connection from one block to another.



A. Connection From CC to CD Should Use Direct Connect



B. Result After SWAPSIG of CD.A and CD.B

A. A net routed with a general interconnection. SWAPSIG easily lets you interchange the pin assignment of the destination block to use the direct connection. This frees the general segment for use in other routing.

You can also use the SWAPSIG command on block outputs to swap them for direct connection, or to drive a specific adjacent general interconnection segment. In the case of outputs, X and Y are completely interchangeable internally

so you can select and assign them entirely on their external connection usage.

B. Two pins that have been swapped using SWAPSIG, to provide more efficient use of the general interconnection. The initial connection to pin C is from a signal running in the adjacent horizontal channel; Pin D is from an adjacent vertical channel. When you swap the signals, the vertically oriented signal routes directly to C and the horizontal signal routes to D.

Note: The internal constraints on the input pins to logic blocks can limit some uses of SWAPSIG.

These constraints are flagged when the command is executed.

CLEARPIN

The CLEARPIN command lets you deconfigure the interconnection for a particular pin on a net. It also removes any spurious interconnection segments from the net.

CLEARPIN, located in the PIN menu, is useful when attempting to relieve congestion. It lets you return the interconnection from a single pin on a net to the available pool of routing resources. When you route critical or high fanout nets, you can use the freed interconnection for a particular route. Then you can route the unrouted pin either manually or with the ROUTEPIN command.

ROUTEPIN and ROUTE

When you manipulate the routing for part of a design, you often leave pins unrouted. You can route these pins with the ROUTEPIN command for the pin; however, ROUTEPIN forces you to select **all** pins to be routed. ROUTE, on the other hand, routes all pins assigned to a selected net. The figure below shows how ROUTE can be more efficient if you must route a large number of pins. ROUTEPIN operates faster than ROUTE for a single pin. ROUTE checks each pin on the net and operates on a single net at a time.

Querynet: PNRFG25B.LCA, XACT 1.3

net1	DD.X	*** DE.A
		*** CF.D
net2	DE.X	*** CG.D
		*** CE.D
net3	ED.X	*** CF.B
		*** EF.A
		*** EE.B
		*** EF.B
		*** DG.C

This report of unrouted nets indicates 9 unrouted pins.
With ROUTEPIN this requires 1 command selection and 9 location selections.
With ROUTE this requires 1 command selection and 3 location selections.

ROUTE and ROUTEPIN Comparison

5.5 TIMING ANALYSIS, DELAY CALCULATOR

After you complete the placement and routing of your LCA design, you analyze the timing of the design against the original design specifications. This discussion describes how to analyze performance.

Note: You can also perform timing simulation, as described in Chapter 2.

The LCA development system includes a unique interactive timing-**delay calculator** that shows you the worst-case delays associated with a design without having to translate and simulate the design. The delay calculator is useful for selecting placement-and-routing alternatives when tuning a design for maximum performance. The delay calculator can extract delay information for the CLBs, IOBs, and the interconnection paths, as follows.

5.5.1 CLB AND IOB DELAYS

CLB and IOB delays are perceived as fixed worst-case values based on the particular configuration of the block by the delay calculator. These delays are characterized from operating the devices at worst-case conditions and are typically constant for a particular speed grade.

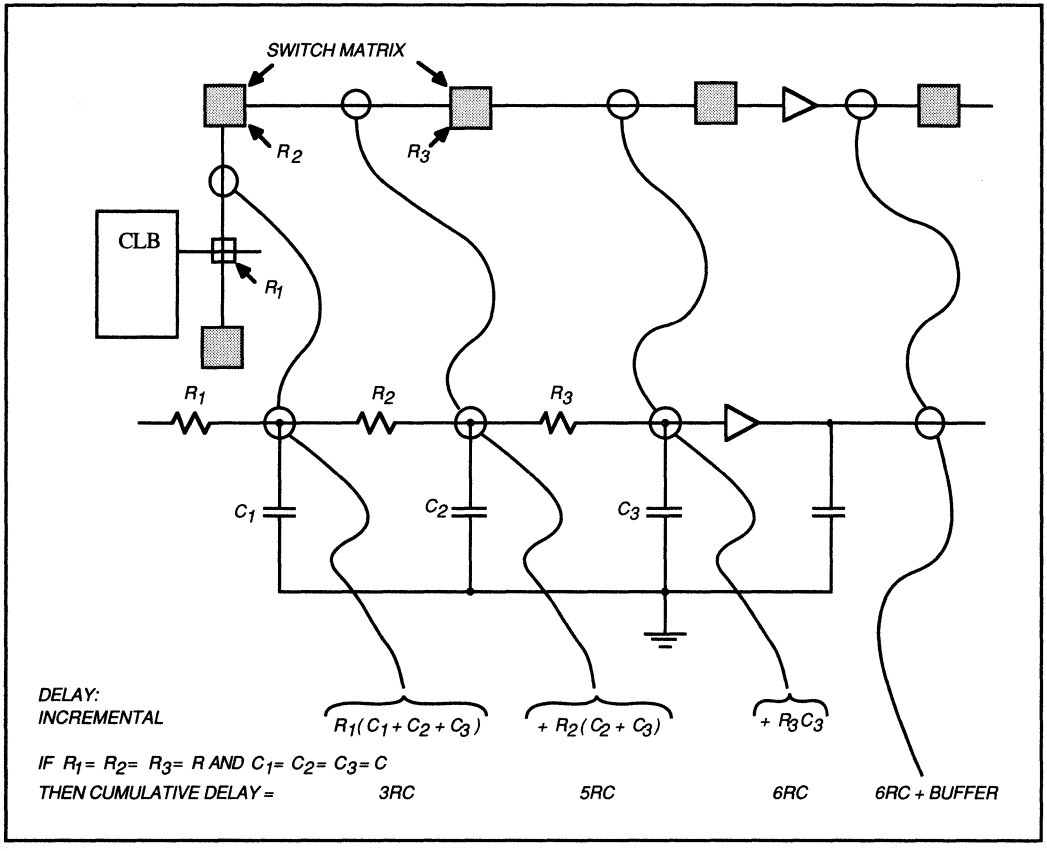
5.5.2 INTERCONNECTION DELAYS

Interconnection delays are more complex. Each interconnection segment used in a signal path represents a distributed R/C delay. Inputs to each CLB or IOB have a negligible capacitance when compared to the capacitance of the interconnection segments.

To correctly calculate the worst-case delay for interconnection, the delay calculator accounts for the accumulation of the interconnection delays. Also, each transistor switch represents a non-linear impedance that modifies the drive characteristics as viewed by downstream interconnection segments. Passing through several interconnection segments and

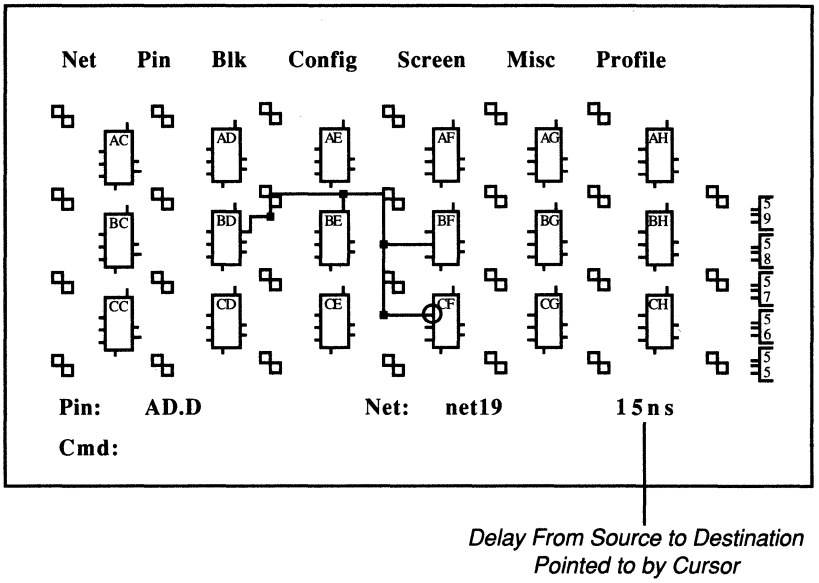
switches degrades the quality of LCA signals. In the general purpose interconnection area, the LCA device includes bidirectional buffers that re-power the signals after they pass through several segments. Each buffer also represents a delay, yet the buffer restores the initial quality. These buffer delays are accounted for in the overall delay calculation.

The next figure summarizes these delays and the elements included in the model for interconnection delay calculations.



Interconnection Delay Example

The delay calculator considers all of these elements when calculating interconnection delays. In XACT, when you position the cursor on a destination pin for a net, the worst-case delay from the net's source to that destination is shown on the information line, as shown below. As you position the cursor on each destination, the appropriate delay is shown.



Delay Calculator Result On-Screen

The delay calculator calculates the delays on a net-by-net basis because the complete net configuration must be considered to determine the delay. As you define a net, the net delay to each point is not available until you specify the source and all of the destinations. When you select the DONE option in the net specification process, interconnection delays are calculated if the net is already routed; this is typical if the automatic router is enabled. For pins not routed, a **?** is shown in the delay field.

When you perform manual routing with EDITNET or other techniques, interconnection delays are not calculated until you select the DONE option and connect a destination pin to the source pin. If you subsequently modify a net by adding other pins or interconnections, a new net delay is calculated, and the new timing information becomes available.

You can obtain interconnection delay information interactively, on the information line of the display. You can also obtain delay information in text reports, either to the screen or in printed form. The following figure shows a sample delay report printed after selecting REPORT DELAY and specifying the desired FROM and TO options. You can also get delay information, on-screen or printed , with the QUERYNET command.

```
From:  BLK  CB     (CB.X)                    :   0 ns (  0 ns)
Thru:  NET  net12  (CB.X     to  BD.D   )    :  20 ns ( 20 ns)
Thru:  BLK  BD     (BD.D     to  BD.X   )    :  35 ns ( 55 ns)
Thru:  NET  net15  (BD.X     to  CD.A   )    :   0 ns ( 55 ns)
To:    BLK  CD     (CD.A     to  SETUP  )    :  22 ns ( 77 ns)

From:  BLK  CB     (CB.X                     :   0 ns (  0 ns)
Thru:  NET  net12  (CB.X     to  CD.B   )    :  18 ns ( 18 ns)
To:    BLK  CD     (CD.B     to  SETUP  )    :  22 ns ( 40 ns)

From:  BLK  CB     (CB.X)                    :   0 ns (  0 ns)
Thru:  NET  net12  (CB.X     to  CE.A   )    :  22 ns ( 22 ns)
To:    BLK  CD     (CE.A)                    :   0 ns ( 22 ns)

From:  BLK  BC     (BC.X)                    :   0 ns (  0 ns)
Thru:  NET  net14  (BC.X)    to  CE.B   )    :  15 ns ( 15 ns)
To:    BLK  CE     (CE.B)                    :   0 ns ( 15 ns)

From:  BLK  CB     (CB.X)                    :   0 ns (  0 ns)
Thru:  NET  net12  (CB.X     to  BD.D   )    :  20 ns ( 20 ns)
Thru:  BLK  BD     (BD.D     to  BD.X   )    :  35 ns ( 55 ns)
Thru:  NET  net15  (BD.S     to  CE.C   )    :   8 ns ( 63 ns)
To:    BLK  CE     (CE.E)                    :   0 ns ( 63 ns)

From:  BLK  AE     (AE.Y)                    :   0 ns (  0 ns)
Thru:  NET  net16  (AE.Y     to  CE.D   )    :  33 ns ( 33 ns)
To:    BLK  CE     (CE.D)                    :   0 ns ( 33 ns)

From:  BLK  AE     (AE.Y)                    :   0 ns (  0 ns)
Thru:  NET  net16  (AE.Y     to  CE.B   )    :  23 ns ( 23 ns)
To:    BLK  CF     (CF.B)                    :   0 ns ( 23 ns)

From:  BLK  CD     (CLOCK   to  CD.X   )    :  35 ns ( 35 ns)
Thru:  NET  net17  (CK.S    to  BE.D   )    :   6 ns ( 41 ns)
Thru:  BLK  BE     (BE.D    to  BE.Y   )    :  35 ns ( 76 ns)
Thru:  NET  net18  (BE.Y    to  BF.B   )    :   0 ns ( 76 ns)
To:    BLK  BF     (BF.B    to  SETUP  )    :  22 ns ( 98 ns)

From:  BLK  CB     (CB.X)                    :   0 ns (  0 ns)
Thru:  NET  net12  (CB.X     to  BD.D   )    :  20 ns ( 20 ns)
Thru:  BLK  BD     (BD.D     to  BD.X   )    :  35 ns ( 55 ns)
Thru:  NET  net15  (BD.X     to  BF.C   )    :  11 ns ( 66 ns)
To:    BLK  BF     (BF.C)                    :   0 ns ( 66 ns)

From:  BLK  CC     (CC,X)                    :   0 ns (  0 ns)
Thru:  NET  net13  (CC.X     to  AD.C   )    :   9 ns (  9 ns)
To:    BLK  AD     (AD.C)                    :   0 ns (  9 ns)
```

Worst-Case Clock-To-Clock Path = 10 MHz Clock

Printed Output from Delay Calculator

5.5.3 CLOCKED SYSTEM DELAYS

In a clocked system, delay calculations are made from clock-edge to clock-edge. Because it has no knowledge of the dynamic operation of the system, the delay calculator can only consider the elements that are connected logically from one clocked device, latch or flip-flop, to the next clocked device.

Note: You must perform timing simulation to investigate the operational constraints of the clocked system. However, the delay calculator does calculate the complete clock-edge to clock-edge path, including the clock-to-output delay and the required setup time. With these complete delay paths, you can easily obtain the worst-case clock frequency: worst-case frequency = 1/(clock-to-clock delay). In the figure above, the worst-case clock-to-clock delay for Net 17 and Net 18 is calculated as 98 ns, so you could clock this circuit at 10 MHz, worst-case.

5.5.4 SPEED GRADE DELAYS

The LCA family offers multiple speed grades for different system requirements. The delay calculator can calculate all delays for a design, given different speed grades.

You select the speed grade with the SPEED command from the MISC menu in XACT and the currently available speed grades for the selected device display. You select the appropriate one. The delay calculator then re-computes and makes available all delays for that speed grade of device, either for display on the screen or in the report file available for the design.

5.5.5 SIGNAL DEGRADATION

Because of the nature of the pass transistors used to interconnect the various signal-path elements, the rise-and-fall times and general signal quality are degraded by each switch element. When taken together over a long signal path, the change in rise-and-fall times and in the signal quality can significantly degrade the predictability of the delay for a particular path. The bidirectional buffers used to re-power the signals in the general interconnection normally alleviate most of these conditions, if they are on the signal path. The combination of manual editing and the router's ability to route signals with remaining resources can create some paths with significant signal degradation.

The delay calculator flags signal paths with degraded signals with a **tilde** (~) preceding the calculated delay. These **degenerate** nets can be the result of one or more of the following factors.

A. A general interconnection segment and its associated signal drive a long line. Long lines have relatively high capacitance. This affects the signal quality, particularly when driven by a general interconnection segment and not the direct source of a signal.

B. A long line consists of one or more general interconnection segments. In general, long lines greatly decrease the drive capability of the signal source. When driving general interconnection segments, the combination of interconnection switch impedance and long line becomes a problem.

Degradation of signal quality affects the signal primarily in differences between rise and fall times. As the delay number increases, the difference between rising-signal delay and falling-signal delay also increases. For

example, consider a delay calculation of ~50. This indicates the following.

A. Falling signals, 1 to 0 transitions, occur more rapidly than indicated. Here, the falling transition can propagate in 35 to 40 ns, worst case.

B. Rising signals, 0 to 1 transitions, occur in more time than indicated. For this case, the rising transitions can require 70 ns or more, worst-case.

The percent variation between rising and falling transitions in the degenerate cases is difficult to predict, but it generally is in the range of 20 to 40% below or above the indicated value.

Caution: Be careful with degenerate nets. If these signals are timing critical, it is highly recommended that you reroute them to eliminate the tilde indication. In some cases, for example, static control, the actual delays are not critical and you can safely ignore the tilde.

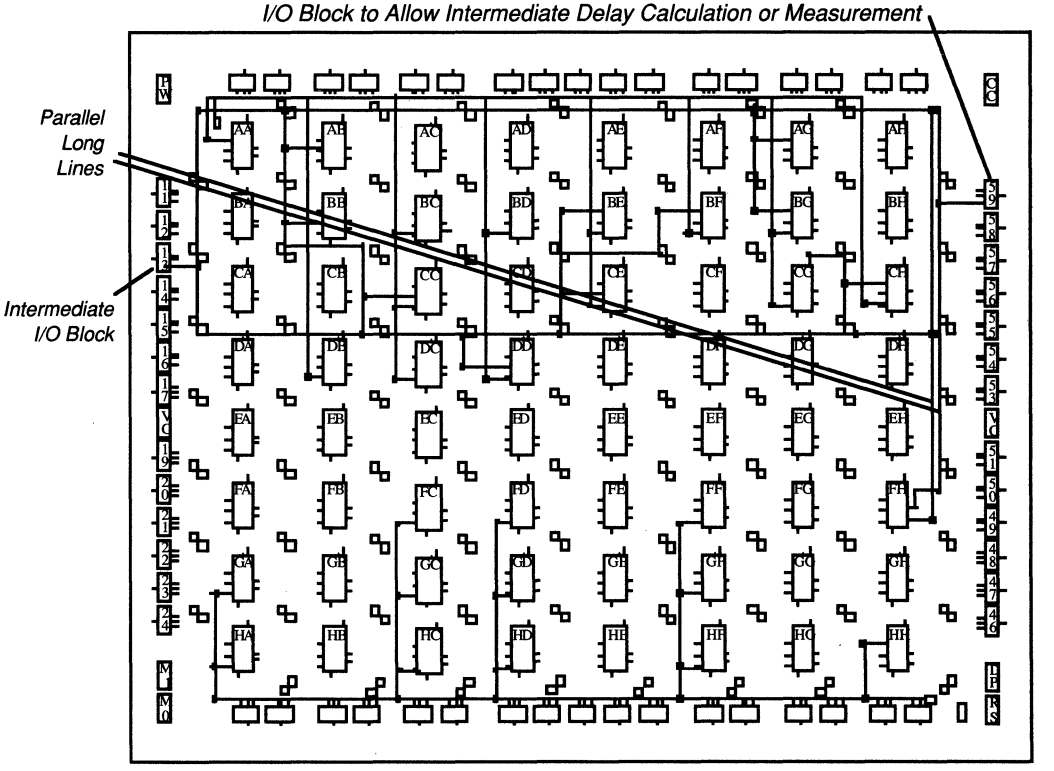
In other cases, you can compensate for the difference between the rise and fall by appropriate logic sense selection. For example, a relatively common high-fanout signal used in counter applications is a synchronous RESET generated by a terminal count detection. If the signal sense is defined as HIGH-true, or reset when 1, the critical timing edge is the rising edge.

Note: Analysis indicates that the rising edge is slower than the falling edge, so if you redefine the signal to be LOW-true, or reset on 0, you can take advantage of the quicker propagation time for falling signal transitions.

This change can improve the overall capability of the system by eliminating potential metastability or partial counter-reset problems that might otherwise occur.

5.5.5.1 Analysis of Intermediate Timing

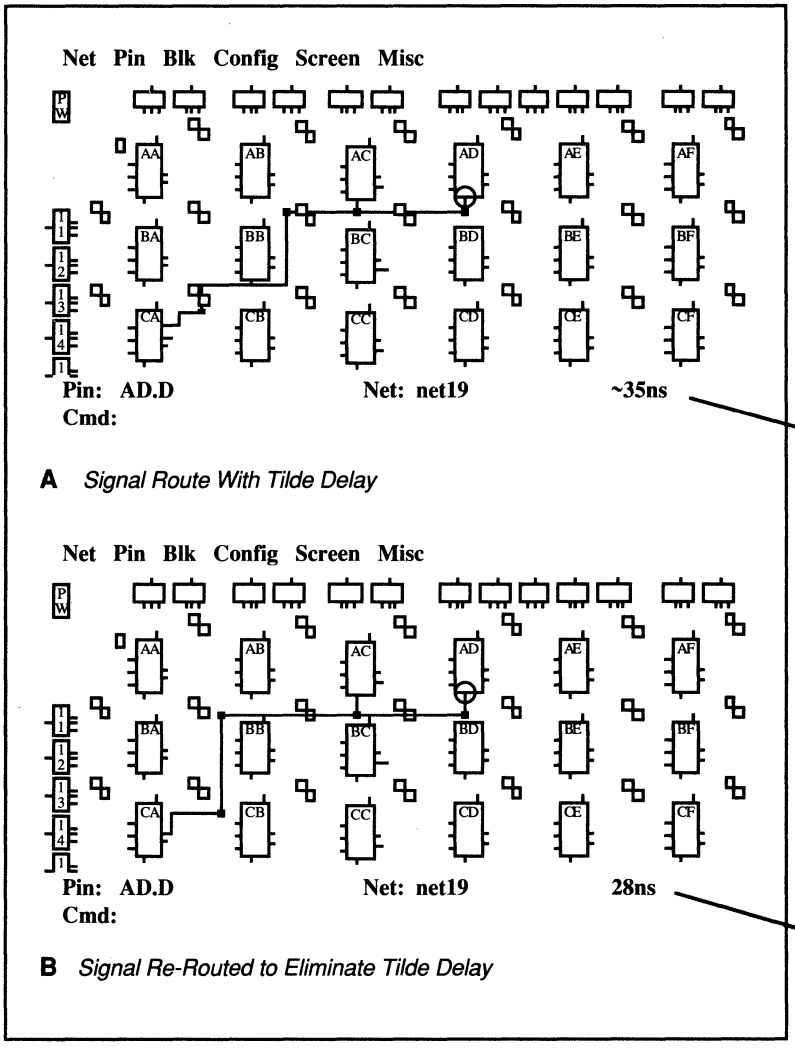
In a circuit with a long path, it is helpful to measure or predict the **intermediate delays** when considering placement and routing alternatives. One method of seeing the delay calculator results, or measuring delay differences along a path, is through **temporary I/O block connections**. The following figure shows two I/O blocks temporarily defined along the path. You can use the delay calculator to see the delay to each block; then use the differences to analyze the results of routing changes or to determine timing-skew related issues. Using the XACTOR In-Circuit Emulator you can temporarily define these I/O blocks as outputs and measure the timing differences directly.



Parallel Long Line to Provide Higher Fanout Signal Source

5.5.5.2 Examples

The example below shows a potential signal degradation problem.

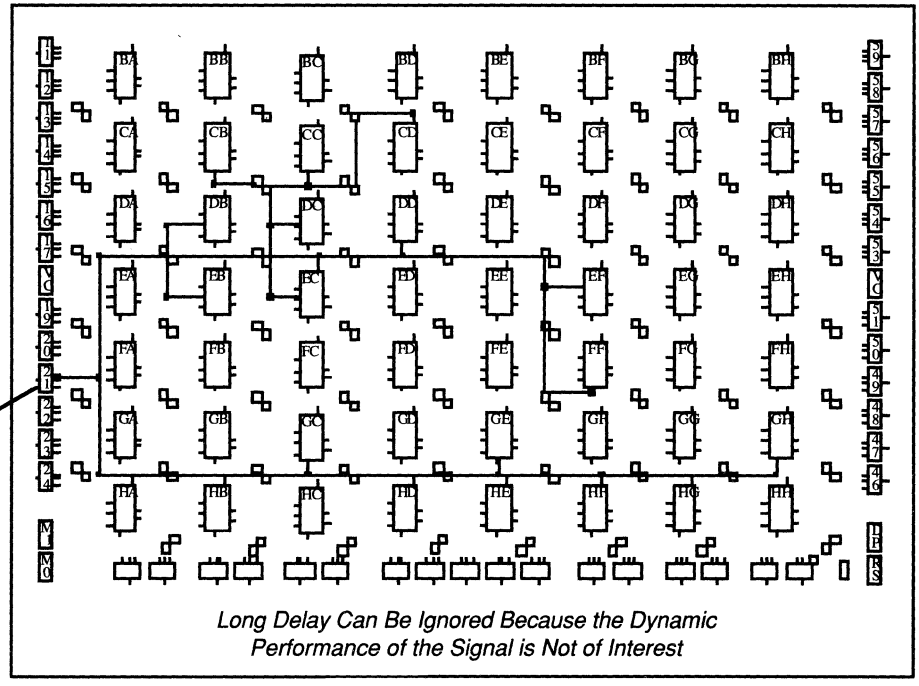


Routing with Tilde on Delay Value

A. An example with a potential signal degradation problem, with tilde delay, because the net is routed through several general interconnection segments prior to driving a long line. The timing delay calculator number in the lower-right corner of the screen is ~35 ns.

B. A routing modification that decreases the delay to 28 ns, and the tilde delay is no longer indicated.

The following figure shows an example where you can safely ignore the tilde indication. The net shown is a static input used by several blocks for function selection. In this case, the delay and signal quality are not of concern because the signal does not change, as might be the case for switch-type inputs or other interface signals generated by the user. The only concern with long delay signals of this type is that blocks using that signal must latch correctly after the transition.



Long Delay Can Be Ignored Because the Dynamic Performance of the Signal is Not of Interest

Signal with Long Delay to Final Destination

5.6 SUMMARY

Appropriate use of the LCA development system capabilities gives you powerful control over all aspects of your LCA design. You can often enter simple designs directly, without paying significant attention to the details of placement and routing. Only when you must implement complex designs, or designs with stringent performance constraints, do the issues of placement and routing require special attention. The techniques discussed here should guide you in implementing a complex design with minimum effort.

Future products for designing with LCA devices will offer improved methods of design entry and increased isolation from the implementation details of the LCA device. Regardless of the sophistication of these development systems, there will continue to be a requirement for interactive design optimization, either for performance or resource use. The LCA development system fulfills that requirement by combining simplicity of operation with quick and efficient design optimization capabilities.

CHAPTER 6

CONFIGURING THE LCA DEVICE

6 CONFIGURING THE LCA DEVICE

This chapter discusses considerations for when you configure LCA devices and read back the configuration data. The configuration techniques covered here apply to the 2064 48-pin and 68-pin packages, and to the 2018 48-pin, 68-pin, and 84-pin packages.

- The overview, 6.1, introduces LCA configuration.
- The discussion on configuration modes, 6.2, explains design considerations, pin functions, and the Slave, Peripheral, and Master modes.
- The discussion on configuring multiple LCA devices, 6.3, describes how to configure multiple devices in a daisy-chain or parallel configuration.
- The discussion on assigning multiple-function I/O pins, 6.4, explains how to handle potential I/O conflicts and unused pins.
- The discussion on the configuration data, 6.5, introduces the content and format of the LCA configuration bit stream.
- The discussion on reading back configuration data, 6.6, explains how to read back an LCA configuration bit stream for verification.

6.1 LCA CONFIGURATION OVERVIEW

When designing an LCA device, you must first identify those parts of the system-level design that can be implemented in one or more devices. Then you partition these elements into clusters of basic logic functions, composed of CLBs and IOBs. Next, you enter, place, and route the interconnection networks for each device. Finally, for each device you compile a **configuration bit stream** that defines its function.

6.1.1 CONFIGURATION BIT STREAM

You design the configuration using the LCA development system design editor, EDITLCA. You use the bit-stream generator, MAKEBITS, to convert the LCA design into a configuration bit-stream file. Then, you use the download cable with the MAKEBITS software to transfer the configuration bit stream into an LCA device in the target system.

You can verify the bit stream, described later, using the READBACK command. The bit stream determines how the LCA device functions. Discussion 6.5, Configuration Data, details the configuration format.

6.1.2 CONFIGURATION PROCESS

You verify your LCA design(s) in any of three ways. The first method is the only one that does not include a configuration process.

- Use a timing simulator to verify the logic and timing of the LCA design.
- Use the MAKEBITS software to generate a configuration bit stream and use the download cable to transfer it into the LCA device in the target system.
- Use the MAKEBITS software to generate a configuration bit stream and use the XACTOR software for in-circuit emulation.

An LCA device is best described in terms of three distinct states.

- Initialization
- Configuration
- User-operation

After a power-up delay, the LCA device powers up in the **initialization state**. In this state, its internal configuration memory is cleared, and all internal user-definable logic is held in a quiescent, or idle, state. After initialization, the LCA device checks the input logic level at the ~RESET pin. When a valid logic-1 level is detected at the pin, the device enters the **configuration state**.

During the **configuration state**, the LCA device is ready to load the configuration data. The configuration data is a serial bit-stream format. The configuration data loads as though the device is a shift register.

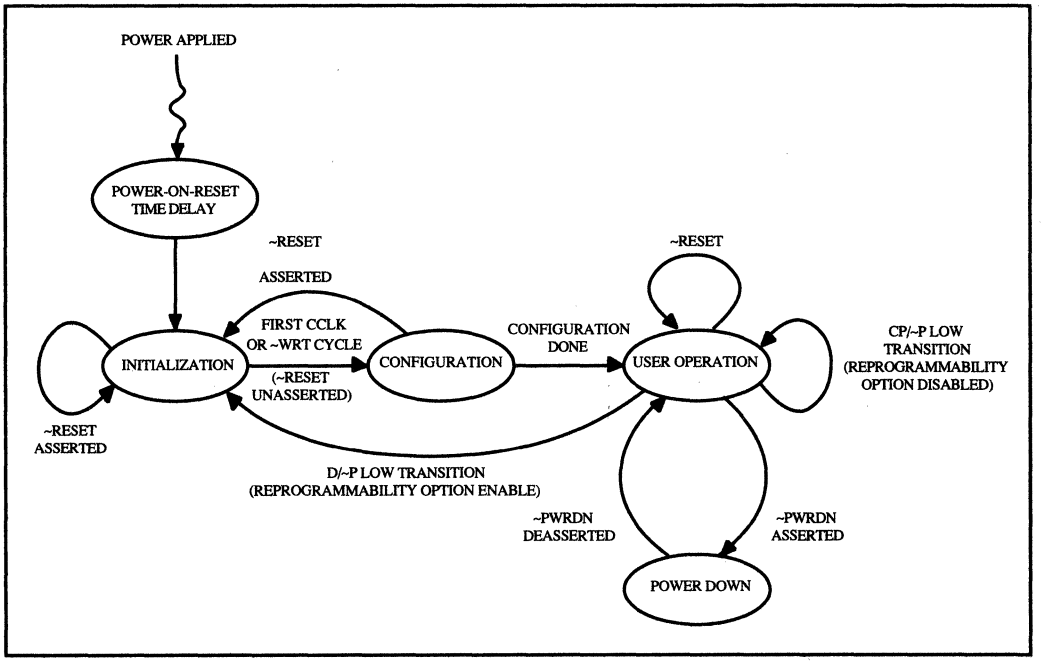
The **configuration mode**, which is the method used for loading the configuration data, depends on the logic levels of the Mode Select pins: M2, M1, and M0. These pins and their functions are explained in discussion 6.2.2.1. Although you can use any of the modes to enter the configuration data, the content and format of the bit stream are fixed for a given logic application.

The configuration bit stream contains a preamble code plus a bit field that indicates its length. When the LCA device loads the preamble code followed by the correct number of bits, as indicated by the length count, the D/~P open-drain output pin goes HIGH to indicate that configuration is complete.

Note: After the configuration process begins, it must either terminate, or abort and restart. Partial configuration is not possible.

After configuration, the LCA device enters the **user-operation state** and performs the logic functions specified in the design. During operation, however, you can return to the initialization state and repeat the configuration process. A state diagram illustrating this sequence is shown below.

Note: You can disable the reconfiguration capability by setting the appropriate bit in the configuration bit stream. If you disable the reconfiguration capability, you can change the LCA device's configuration only by removing and reapplying power to it.



LCA Configuration Diagram

Note: During the initialization and configuration states, all user I/O pins, except those used by the configuration process, have passive internal pull-up resistors that cause those pins to go HIGH when not externally overdriven.

Upon entering the user-operation state, all user I/O pins simultaneously become functional, according to your specified pin definitions.

The following discussion explains the five possible modes for loading the configuration information into the LCA device.

6.2 CONFIGURATION MODES

The LCA device supports five configuration modes, including three Master modes, for loading LCA programming information.

1. **Slave mode**
2. **Peripheral mode**
3. **Master-Serial mode**
4. **Master Parallel-Low mode**
5. **Master Parallel-High mode**

You select a configuration mode by connecting the dedicated Mode Select pins, M0 and M1, and the programmable Mode Select pin, M2, to either a logic-1 or a logic-0 signal, as indicated in the following table.

Configuration Mode Selection

Mode Select Pins	M2	M1	M0
Master-Serial mode	0	0	0
Master Parallel-Low mode	1	0	0
Master Parallel High mode	1	1	0
Peripheral mode	1	0	1
Slave mode	1	1	1

Note: The current mode selections do not use all possible combinations of M1, M2, and M3, which leaves room for future expansion of configuration options.

The following discussions describe available modes.

- Discussion 6.2.1 outlines considerations that impact selecting a configuration mode.
- Discussion 6.2.2 identifies the special LCA pin functions during the configuration process.
- Discussion 6.2.3 explains the Slave mode.
- Discussion 6.2.4 explains Peripheral mode.
- Discussion 6.2.5 explains Master modes.

6.2.1 CONSIDERATIONS

Each configuration mode has different design considerations and variations in device pin usage during configuration. The choice of a configuration mode depends on your specific system application.

The following are some questions you should ask before choosing a configuration mode.

1. Is control of the configuration process automatic or controlled externally?

    - If the configuration process is externally controlled, that is in Slave or Peripheral mode, is it controlled by software or DMA hardware?

    - If the configuration process is automatic, that is in Master mode, is the configuration

        a. shared with the microprocessor code?
        b. stored in a separate byte-wide PROM?
        b. stored in a serial memory device?

2. How much time is available for configuration?

3. For an application using multiple LCA devices, should they be configured serially as a daisy chain, or in parallel?

4. What are the I/O pin requirements?

    For example, are the I/O pins used by the target application also involved in the configuration? If so, can you assign pins to minimize or eliminate external isolation?

These considerations are discussed in more detail below.

6.2.1.1 External versus Automatic Configuration Control

The **externally-controlled** case requires that you use either **Slave mode** or **Peripheral mode** to load the configuration data into the LCA device serially. You can load the configuration data as part of the system's bootup process or you can load it on the fly.

This externally-controlled method is more flexible than automatic configuration because the **configuration bit stream can be read from a PROM, disk, or any other source accessible by a processor.** However, this method may take longer to complete than the automatic method.

Automatic configuration uses one of the three **Master modes**. With Master mode configuration, the LCA device automatically accesses (sends out to) an external PROM for the configuration bit stream. Then the LCA device configures itself using Master mode in 12 to 24 ms for the 2064, and 17 to 35 ms for the 2018.

The following table compares the configuration mode characteristics.

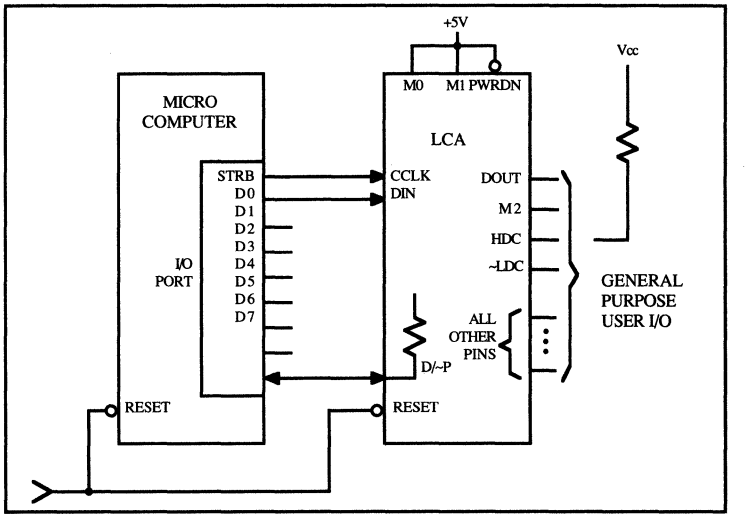
Comparison of Configuration Modes				
Configuration Mode	**Slave Mode**	**Peripheral Mode**	**Master Parallel-Low Master Parallel-High Modes**	**Master-Serial Mode**
Mode Selection Code (M2:M1:M0)	1:1:1	1:0:1	0:0:1 (Master-Low) 0:1:1 (Master-High)	0:0:0
Configuration Data	Bit-serial	Bit-serial	Byte-parallel	Bit-serial
Automatic Loading	No	No	Yes	Yes
Programming Source	User Logic or Another LCA (Note 1)	CPU Data Bus Memory	External Byte wide Memory	External Serial
Number of User I/O Pins Required	2	6	25	3
Configuration Time	Source-dependent (Note 2)	Source-dependent (Note 2)	12-24 ms (2064) 17-34 ms (2018) (Note 3)	12-24 ms (2064) 17-34 ms (2018) (Note 3)

Notes: 1. Slave mode is also used by XACTOR for In-Circuit Emulation.
2. The minimum time in any case is approximately 12 ms for the 2064 and 17 ms for the 2018.
3. This parameter depends on internal timing circuits and is manufacturing-process dependent. Therefore, it may vary from device to device within the limits shown.

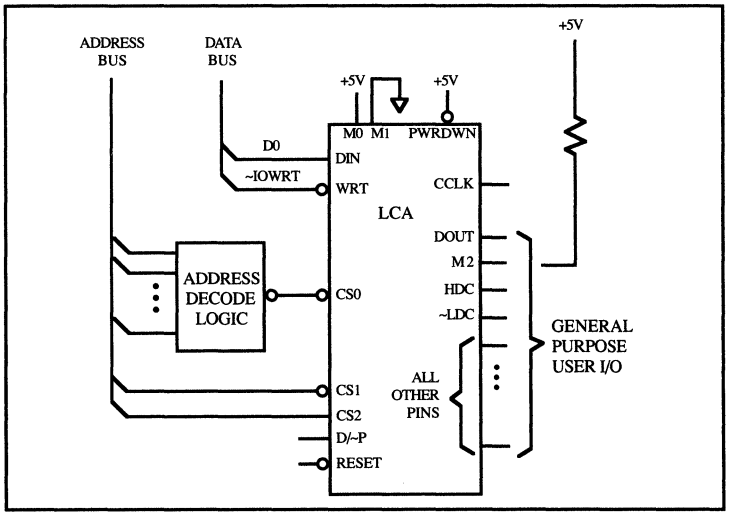
In all configuration modes, some user I/O pins are temporarily assigned configuration-related functions. The number of such pins ranges from five in Slave mode to 25 in Master mode. Once configuration is complete, these pins become general purpose I/O pins.

However, it is your **responsibility** to guarantee that no signal conflicts occur between the pin's use in the configuration state and its use in the user-operation state. Signal conflicts on these pins can create undesired side effects, such as disturbing the configuration process or other external logic. With a little care, however, you should not experience problems using these dual-function I/O pins. Although

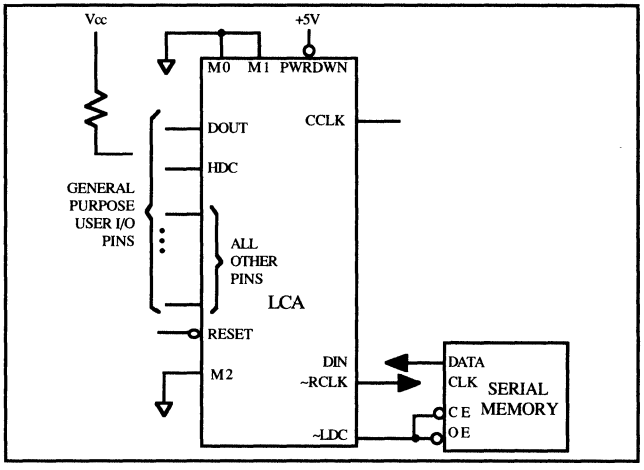
signal conflicts may be resolved by using external buffers for isolation, careful selection of the pinout assignment can usually eliminate the need for isolation. The five figures below show the LCA device's pin usage for each configuration mode.



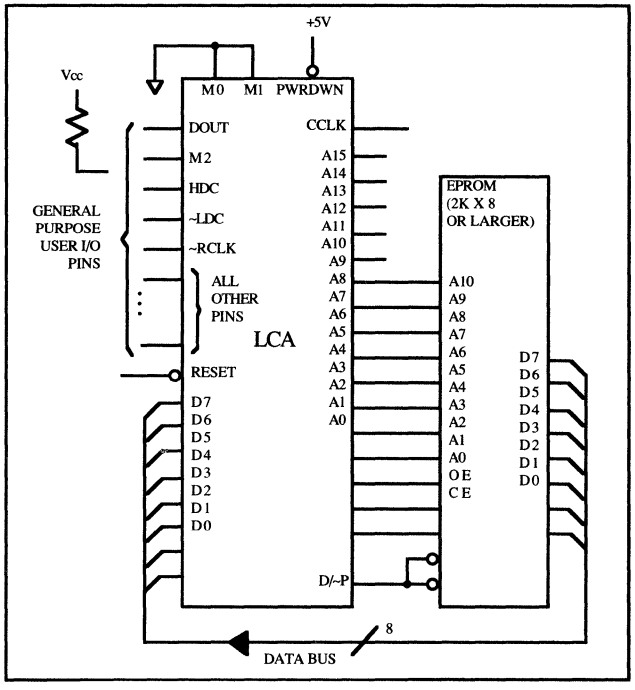
Slave Mode



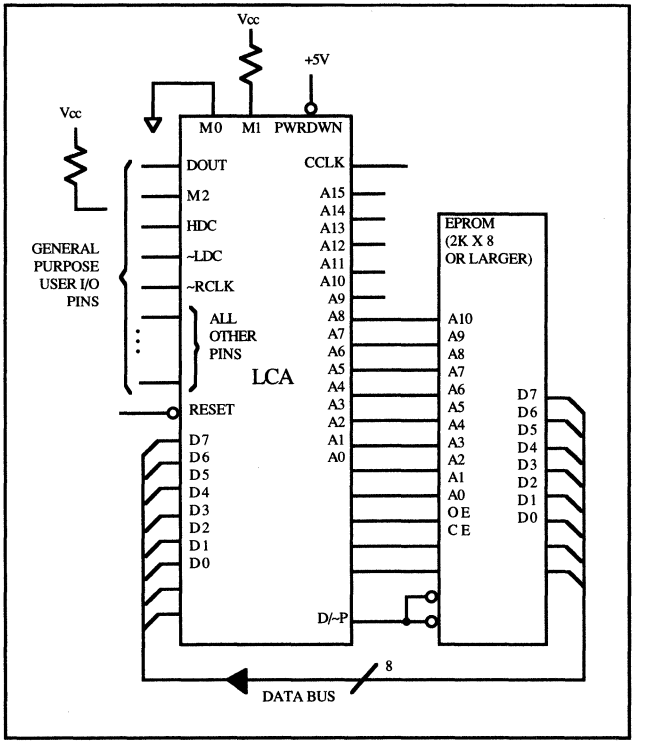
Peripheral Mode



Master-Serial Mode



Master Parallel-Low Mode



Master Parallel-High Mode

The following table summarizes the pins used in each mode. Individual pins are described in more detail below, under Configuration Pin Functions, 6.2.2. Also, refer to the example under discussion 6.4.1.

Summary of Pins Used for Configuration									
Pin Name	Applicable Config. Mode(s) S	P	MS	MH	ML	Function During Configuration		Function During User Operation	
M0	•	•	•	•	•	Mode Select 0	(I)	Readback Trigger	(I)
M1	•	•	•	•	•	Mode Select 1	(I)	Readback Data Out	(O)
M2	•	•	•	•	•	Mode Select 2	(I)	<User I/O>	
D/~P (Note 1)	•	•	•	•	•	Indicates configuration process is done	(O)	Initiates/Inhibits Re-configuration	(I)
~RESET (Note 1)	•	•	•	•	•	Abort/Restart configuration	(I)	Master Clear for all internal flip-flops	(I)
CCLK	•	•	-	•	•	Configuration Clock(Notes 1 & 2)		Readback Clock	(I)
DIN	•	•	•	-	-	Configuration Data In	(I)	<User I/O> (Note 3)	
DOUT	•	•	•	•	•	Configuration Data Out	(O)	<User I/O>	
HDC	•	•	•	•	•	Logic HIGH	(O)	<User I/O>	
~LDC	•	•	•	•	•	Logic LOW	(O)	<User I/O>	
A0-A15	-	-	-	•	•	Address Bus	(O)	<User I/O>	
D0-D7	-	-	-	•	•	Data Bus	(I)	<User I/O> (Note 3)	
~RCLK	-	-	•	•	•	Read Clock	(O)	<User I/O>	
~WRT	-	•	-	-	-	Write Strobe	(I)	<User I/O>	
~CS0	-	•	-	-	-	Chip Select 0	(I)	<User I/O>	
~CS1	-	•	-	-	-	Chip Select 1	(I)	<User I/O>	
CS2	-	•	-	-	-	Chip Select 2	(I)	<User I/O>	

Abbreviations:
S = Slave
P = Peripheral
MH = Master Parallel-High
ML = Master Parallel-Low
MS = Master-Serial
I = Input
O = Output

Notes: 1. The ~RESET, CCLK, and D/~P pins have multiple functions. See text for further details.
2. During Slave mode configuration, the CCLK pin is an input, while for all other modes, it is an output.
3. DIN and D0 are the same physical pins but are associated with different configuration modes.

6.2.1.2 Configuration Time

For some applications, the time required to configure the LCA device is an important consideration. The minimum time required to load the LCA configuration data is the same for all configuration modes, approximately 12 ms for the 2064 and 17 ms for the 2018. The processor-driven configuration techniques under software control for Slave and Peripheral mode may take longer to complete.

Unlike the user-driven Slave and Peripheral modes, the internal oscillator controls the configuration loading time for Master mode. Because the frequency of this internal oscillator depends on the LCA fabrication

process, configuration loading time can extend to twice the minimum time.

6.2.1.3 Configuring Multiple LCA Devices

In applications using multiple LCA devices, special daisy-chaining capabilities permit you to configure all the linked devices from a single data source. This is described in further detail under discussion 6.3.

6.2.2 CONFIGURATION PIN FUNCTIONS

There are two types of LCA pins used for configuration.

- **Non-programmable pins** are dedicated to the control function.
- **User-programmable pins** are available as general purpose I/O pins after configuration.

There are **six non-programmable pins** dedicated to control functions.

Non-Programmable Pins	Control Functions
M0, M1	Mode Select Pins
CCLK	Configuration Clock
~RESET	Master Reset
D/~P	Done/Program
~PWRDWN	Power-Down

In addition to the dedicated control pins, several user-programmable I/O pins have configuration functions assigned to them, regardless of which configuration mode you select. These pins, as well as the dedicated control pins, are described below. Other I/O pins are used in only one specific configuration mode and are described in the discussion of that mode.

The following are **user-programmable I/O pins** that can be used during configuration.

User-Programmable Pins	Control Functions	Notes
M2	Mode Select	(Present
DIN/D0	Configuration Data In	in
DOUT	Configuration Data Out	all
HDC	HIGH during configuration	Configuration
~LDC	LOW during configuration	Modes)
~CS0, ~CS1, CS2	Chip Selects	Peripheral mode only
~WRT	Write Strobe	Peripheral mode only
~RCLK	Read Strobe	Master modes only
A0-A15	Address Bus	Master parallel modes only
D0-D7	Input Data Bus	Master parallel modes only

Note: Except for HCD and ~LDC, all unassigned user-I/O pins not used in configuration have passive internal pull-ups to Vcc, as described in discussion 6.2.2.3. The passive internal pull-ups on all user-programmable I/O pins are removed after configuration is completed.

6.2.2.1 M2, M1, and M0

M2, M1, and M0 are Mode Select input pins that select the configuration mode the LCA device uses, as explained under 6.2, Configuration Modes. Pins **M0** and **M1** are dedicated configuration pins. The **M2** pin, unlike M0 and M1, becomes available as a general purpose user-I/O pin after configuration.

During configuration, pins M1 and M2 have internal pull-up resistors, while pin M0 does not. Except for Master-Serial mode, you should not drive pin M2 LOW during configuration. If left unconnected, it is pulled HIGH.

In applications that do not use the LCA device's read-back capability, you can tie the mode select pins directly to ground or to Vcc. Because the M1 and M2 pins are supplied with internal pull-up resistors, you may leave them unconnected after configuration is done.

Note: The mode select pins are sampled either at the conclusion of the initialization state, or with the rising edge of ~RESET if used to delay configuration. Thus, you do not need to maintain their logic levels after configuration begins.

6.2.2.2 DIN and DOUT

DIN and DOUT are used as the serial data path for configuring both a Slave mode daisy chain, and a Master-Serial mode device. DIN and DOUT are also used for other purposes, such as a data bus or Slave mode without a daisy-chain.

Refer to the descriptions of the Slave and Master-Serial modes, under 6.2.3 and 6.2.5, respectively, for more information.

6.2.2.3 HDC and ~LDC

HDC and ~LDC are user-programmable I/O pins. During configuration, the LCA device drives HDC to constant HIGH and the ~LDC to constant LOW. You use these two pins to control external logic during the initialization and configuration states. For example, you can use these pins to enable or disable various external logic circuits, depending on whether each logic circuit is required during or after configuration.

6.2.2.4 CCLK

CCLK is a dedicated control pin serving as a clock **input** during Slave mode configuration, and conversely as a clock **output** in all other configuration modes. As an input, CCLK is used during the serial loading of a configuration bit stream. As an output, CCLK serves as a clock source for configuring any Slave mode LCA devices to be daisy-chained to the master LCA device.

During operation, CCLK serves as a clock input for reading configuration data from the device in conjunction with the M0/RT and M1/RD pins. The

CCLK input is subject to a minimum time it can be held LOW. It should remain in the HIGH state when not in use. However, the LCA device can drive the CCLK input from a clock source that violates this limit, as long as deassertion of ~RESET enables configuration as soon as the clock is normal. The CCLK pin has an internal pull-up resistor that lets you disable an external clock source when configuration is done.

6.2.2.5 ~RCLK

The ~RCLK pin performs the function of a read strobe for dynamic memories in the Master-Parallel mode. For the Master-Serial mode, ~RCLK is an output pin that synchronizes the supply of serial data.

6.2.2.6 ~RESET

The ~RESET pin is an active-LOW master-reset input. Its function depends on the LCA device's state. During the **initialization** state, after power-up and prior to starting the configuration, this pin can delay the start of configuration. As soon as the **configuration** process starts, and until it completes, asserting ~RESET aborts the configuration process and returns the LCA device to the initialization state. Configuration restarts when initialization completes and ~RESET is HIGH. When configuration completes, the ~RESET pin changes function and becomes a master-reset control pin that clears all internal flip-flops and latches to the 0 state.

6.2.2.7 D/~P

The D/~P (DONE/~PROGRAM) pin is both an input and an open-drain-type output with an optional, programmable pull-up resistor. As an output, D/~P indicates the current configuration status of the LCA device.

Prior to initial configuration and during subsequent reconfigurations, the LCA device holds the D/~P pin LOW to indicate that the device is not ready for user operation. When D/~P goes HIGH, it indicates that configuration is done and that the device is in the user-

operation state. Consequently, you can use D/~P in your system reset logic to ensure that the LCA device is fully configured before the reset of the rest of the system is terminated.

The configured output pins on the LCA device become active one clock cycle before D/~P goes HIGH. This allows time for any user-I/O signals to propagate between LCA devices. You can initiate subsequent reconfigurations of the device by applying a logic-0 level to the D/~P pin, with an open-collector-type signal source. You must hold the D/~P pin LOW for several microseconds for the LCA device to recognize the LOW level. Noise is unlikely to trigger a reconfiguration.

As soon as the LCA device recognizes the LOW, it forces D/~P LOW until configuration is completed. The D/~P pin must go HIGH before it can initiate reconfiguration.

Note: By using its internal pull-up resistor option, you can leave the D/~P pin unconnected and eliminate its need for any external passive components.

Also: You can prevent the D/~P pin from going HIGH after configuration as an alternate technique for disabling the LCA reconfiguration.

6.2.2.8 ~PWRDWN

The ~PWRDWN pin is an active-LOW input that forces the LCA device into a low-power state. You can reduce Vcc to 2 V after ~PWRDWN is active. Entering the power-down state does not change or modify the configuration information stored in the LCA device; it merely reduces the device's overall power requirements by disabling its I/O pins and certain internal logic.

Power-down resets all internal storage elements, that is CLBs and IOBs but **not** memory cells, and forces all I/O pins to become high impedance. Internally, logic nodes

that were driven by inputs to the LCA device prior to power-down are electrically isolated from their pins and forced HIGH.

You must leave the ~PWRDWN pin inactive (HIGH) during initialization and configuration; only assert it while D/~P is HIGH. If your application does not use the power-down feature, you should tie the ~PWRDWN pin to Vcc.

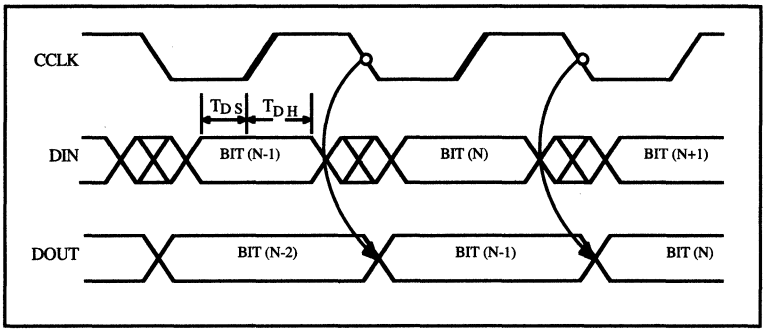
Note: All other user-I/O pins not involved in configuration have passive internal pull-ups to Vcc during configuration. The passive internal pull-ups on all user-programmable I/O pins are removed after configuration is completed.

6.2.3 SLAVE MODE

M2:M1:M0 = 1:1:1 configures the LCA device in Slave mode. This mode is simple and efficient because it uses fewer pins than any other configuration mode. It serially loads the configuration bit stream into the device.

During Slave mode configuration, each bit in the stream sequentially shifts into the DIN input on the LCA device with the rising edge of the clock applied to the CCLK pin, as illustrated below.

Note: In the following figure, T_{DS} is equivalent to T_{DCC}, which may appear in other timing diagrams in this manual, and T_{DH} is equivalent to T_{CCD}, which may appear in other timing diagrams in this manual.



Slave Mode Configuration Timing

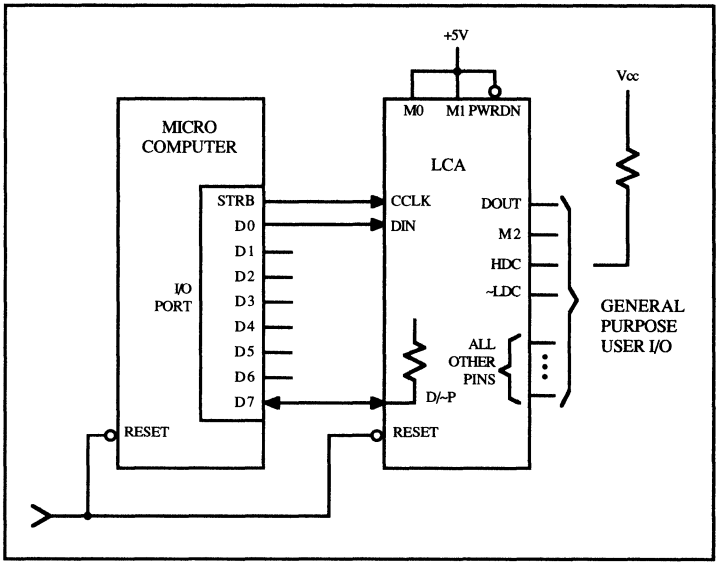
Note: In Slave mode, the CCLK pin is an input, not an output as in other modes.

After the configuration bit stream loads, completing the configuration process requires three additional clock cycles for a total of the length count plus three clocks.

Slave mode configuration is especially appropriate in applications where a host processor configures the LCA device through an I/O port. I/O instructions can drive the CCLK and DIN pins, and the system easily meets the minimum data setup and hold times.

Slave mode configuration is also useful in multiple LCA applications, where you string together the DIN and DOUT pins of several devices in a daisy-chain arrangement. This arrangement permits several LCA devices to share a common source of configuration data.

In addition to the **six non-programmable** configuration control pins, Slave mode configuration uses **five programmable** pins: M2, DIN, DOUT, HDC, and ~LDC. These become available as general-purpose user I/O pins after configuration is complete. The 53 remaining programmable I/O pins are not used during configuration, as shown in the figure and table below.



Slave Mode Pin Usage

2

Slave Mode Pin Summary

FIXED, NON-PROGRAMMABLE PINS

Pin Name	Pin Number PLCC	Pin Number DIP	Pin Type	Value During Configuration	Description
M0	26	18	Input	HIGH	Mode Select
M1	25	17	Input	HIGH	Mode Select
CCLK	60	42	Input	<Clock>	Configuration Clock
~RESET	44	31	Input	HIGH	Master Reset
D/~P	45	32	Output	LOW	Done/Program
~PWRDWN	10	7	Input	HIGH	Power-down

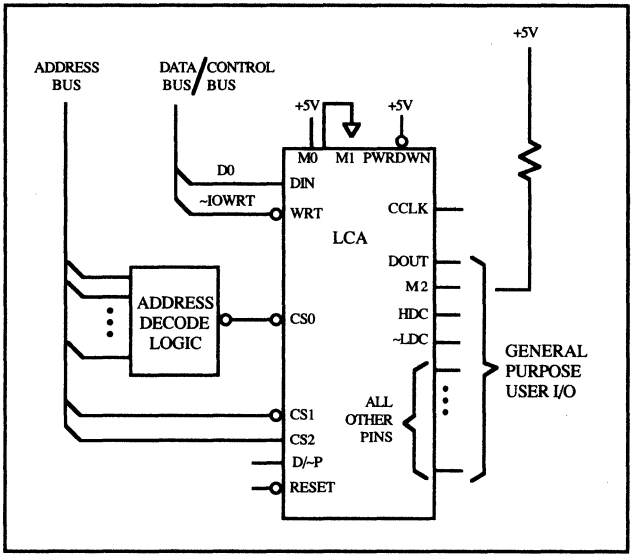
USER-PROGRAMMABLE PINS

Pin Name	Pin Number PLCC	Pin Number DIP	Pin Type	Value During Configuration	Description
M2	27	19	Input	HIGH	Mode Select
DIN	58	40	Input	<Data>	Configuration Data In
DOUT	59	41	Output	<Data>	Configuration Data Out
HDC	28	20	Output	HIGH	Constant "1" Level
~LDC	30	21	Output	LOW	Constant "0" Level

In daisy-chained LCA applications, if the first LCA device is configured in Slave mode with a free-running CCLK clock source, you must synchronize the first device with the other devices in the chain. To accomplish this synchronization, you must ensure that ~RESET is released with the proper setup and hold times relative to CCLK. This timing guarantees that all LCA devices in the daisy chain become operational simultaneously. The devices all begin configuration on the same clock cycle. You can easily ensure this timing by de-asserting ~RESET with the falling edge of CCLK.

6.2.4 PERIPHERAL MODE

M2:M1:M0 = 1:0:1 enables a host processor to load the configuration bit stream into the LCA device via the data bus. In this configuration mode, you can think of the LCA device as a one-bit-wide peripheral device because you load the configuration bit stream into it one bit at a time. Typically, you tie data bus bit 0 to the DIN pin of the LCA device. The processor then shifts each successive bit of the data byte into data bus bit 0 between load instructions to the LCA device. Peripheral mode requires the next fewest LCA device pins for configuration, as shown below.



Peripheral Mode Pin Usage

2

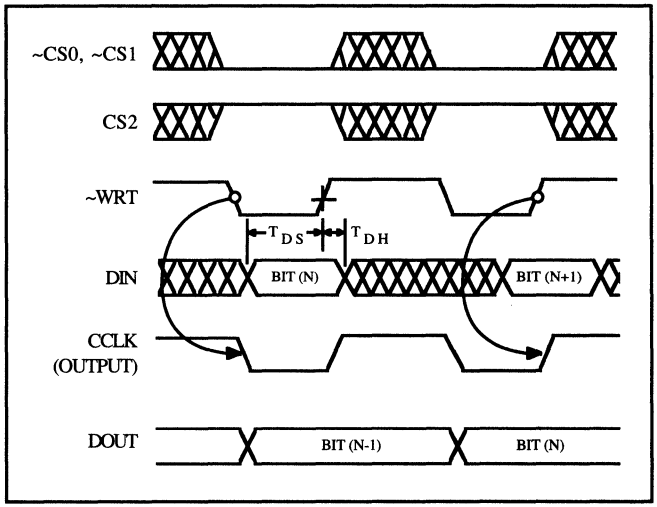
As in Slave mode, Peripheral mode loads the configuration bit stream into the LCA device bit-serially. When the correct number of bits are loaded into the device, the D/~P pin goes HIGH to indicate that the configuration bit stream is loaded. Completing the configuration process requires three additional clock cycles after the bit stream is loaded, for a total of three clocks more than the length count.

During Peripheral mode configuration, **nine** of the LCA device's **programmable** I/O pins function as configuration control pins in addition to the **six fixed**, **non-programmable** control pins. The next table shows the configuration pins used in this mode.

Peripheral Mode Pin Summary					
FIXED, NON-PROGRAMMABLE PINS					
Pin Name	**Pin Number PLCC**	**DIP**	**Pin Type**	**Value During Configuration**	**Description**
M0	26	18	Input	HIGH	Mode Select
M1	25	17	Input	LOW	Mode Select
CCLK	60	42	Output	<Clock>	Configuration Clock
~RESET	44	31	Input	HIGH	Master Reset
D/~P	45	32	Output	LOW	Done/Program
~PWRDWN	10	7	Input	HIGH	Power-down
USER-PROGRAMMABLE PINS					
Pin Name	**Pin Number PLCC**	**DIP**	**Pin Type**	**Value During Configuration**	**Description**
M2	27	19	Input	HIGH	Mode Select
DIN	58	40	Input	<Data>	Configuration Data In
DOUT	59	41	Output	<Data>	Configuration Data Out
~CS0	50	35	Input	LOW	Chip Select (Active LOW)
~CS1	51	36	Input	LOW	Chip Select (Active LOW)
CS2	54	37	Input	HIGH	Chip Select
~WRT	56	38	Input	<Strobed>	Write Enable (Active LOW)
HDC	28	20	Output	HIGH	Constant "1" Level
~LDC	30	21	Output	LOW	Constant "0" Level

The following figure shows the timing relationship between the signals on these control pins.

Note: In the following figure, T_{DS} is equivalent to T_{DC}, which may appear in other timing diagrams in this manual, and T_{DH} is equivalent to T_{CD}, which may appear in other timing diagrams in this manual.

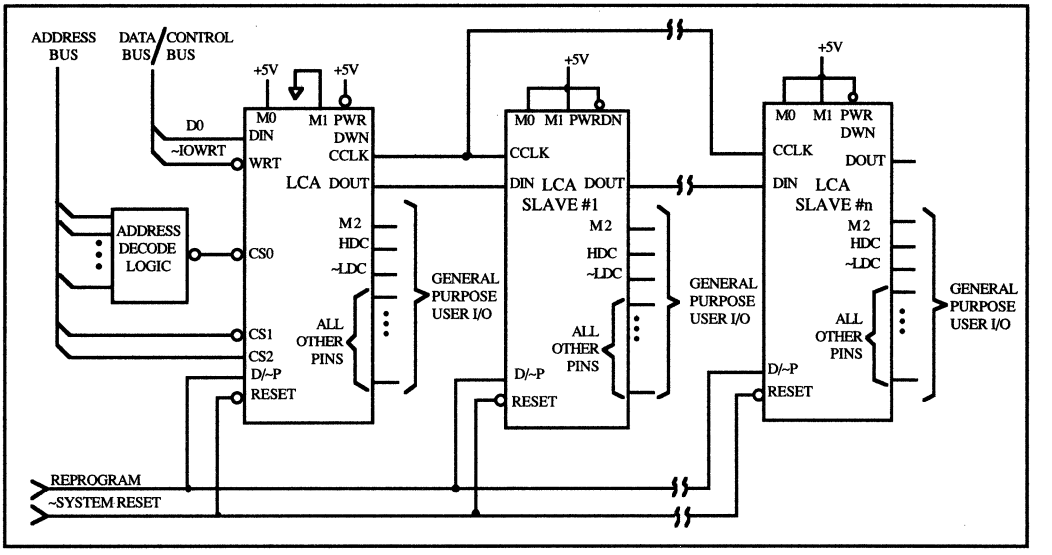


Peripheral Mode Configuration Timing

The DIN and DOUT pins function the same in Peripheral mode as in Slave mode. **Four** other pins serve as bus interface controls. Three of these pins, **~CS0**, **~CS1**, and **CS2**, become Chip Select pins, while the fourth pin, **~WRT**, becomes the Write Strobe input. The ~WRT pin serves the same function in Peripheral mode as CCLK does in Slave mode. A pulse applied to the ~WRT pin, while the three Chip Selects are asserted, shifts one bit of the stream into the DIN input of the LCA device. Each write strobe to a Peripheral mode LCA device also produces a CCLK output pulse that drives the CCLK inputs of the cascaded devices, shown below. The three Chip Selects (two active-LOW, one active-HIGH) map the LCA device to a specific I/O or memory address for configuration.

Note: The nine pins mentioned above are available as general purpose, user-programmable I/O pins when configuration is completed.

The other 49 programmable I/O pins are not used for Peripheral mode configuration.



Peripheral Mode LCA Device with Daisy Chain

6.2.5 MASTER MODES

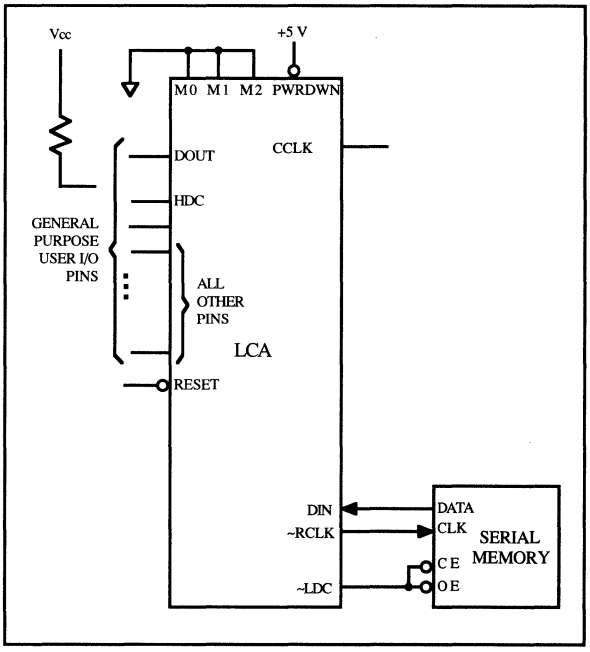
M2:M1:M0 = 0:0:0 configures the LCA device in Master Serial mode.

M2:M1:M0 = 1:0:0 configures the LCA device in Master Parallel-Low mode.

M2:M1:M0 = 1:1:0 configures the LCA device in Master Parallel-High mode.

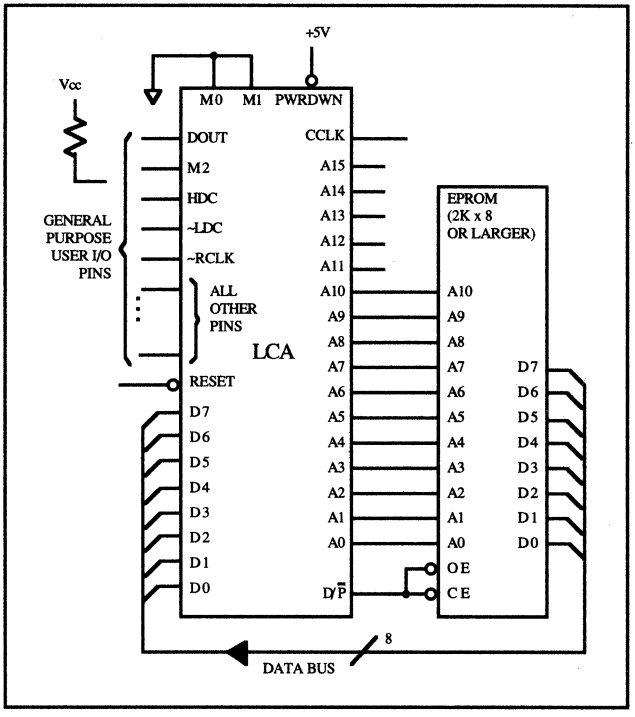
In the Master configuration modes, the LCA device automatically controls loading the configuration bit stream.

In the **Master-Serial mode**, the LCA device uses ~RCLK to synchronize the serial input data that provides the configuration bit stream, shown in the figure below.



Master-Serial Mode Pin Usage

In the **Master Parallel-Low** and **Master Parallel-High** modes, the LCA device uses on-chip control logic to address an external, byte-wide memory device. This memory device, for example an EPROM, holds the configuration bit stream. The following figure shows a typical Master Parallel-Low mode configuration.



Master Parallel-Low Mode Pin Usage

For the byte-wide or parallel modes, 16 of the LCA device's I/O pins form an address bus. Eight additional I/O pins form a unidirectional data bus. There are two types of byte-wide Master modes.

- The **Master Parallel-Low** mode addresses memory in *ascending* sequence, starting at address zero, 0.

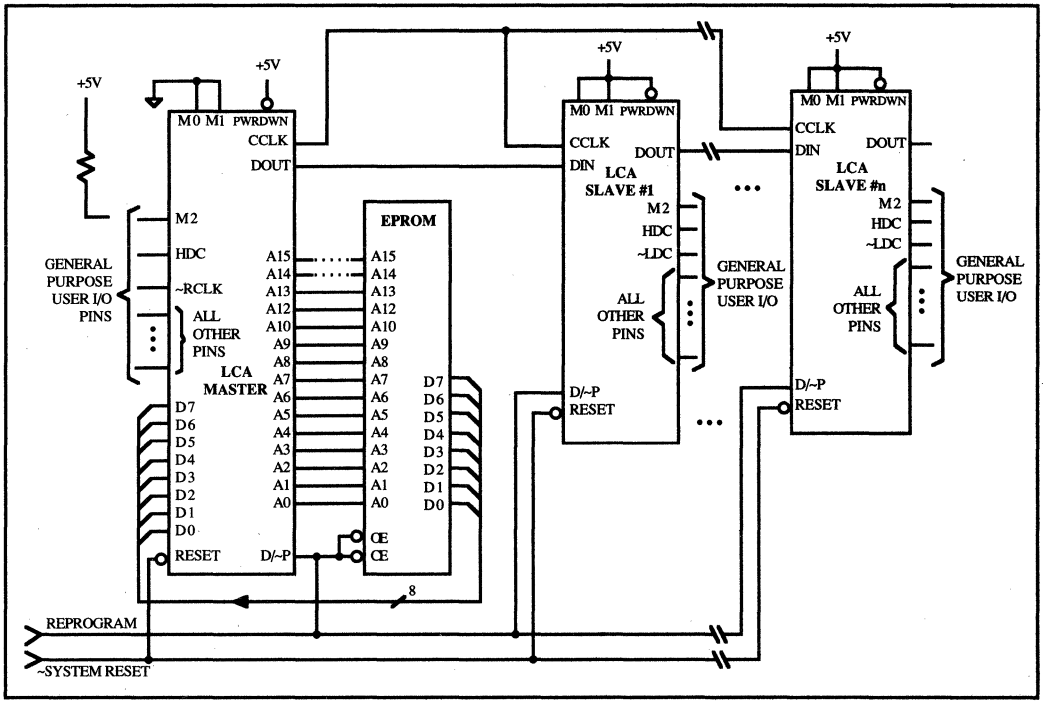
- The **Master Parallel-High** mode addresses memory in *descending* sequence, starting at hexadecimal address FFFF.

With this addressing flexibility, the configuration data can **share** space in a ROM or EPROM that typically stores a microprocessor program.

After configuration begins, memory-read cycles continue until the correct number of bits are read. The D/~P pin goes HIGH to indicate that the configuration bit stream is loaded.

Note: Bytes of data read from the external byte-wide memory are serialized on-chip, and are independent of physical byte boundaries.

In addition to using the 16 address outputs and 8 data-bus input pins, Master Parallel-Low and High modes also use several other signals. One is the ~RCLK output signal, which is active LOW, yet goes HIGH while the address bus is changing state. This allows clocked EPROMs to store configurations. Other signals are the CCLK and DOUT outputs, both of which drive cascaded or daisy-chained LCA devices, as shown in the following figure.

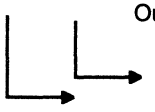


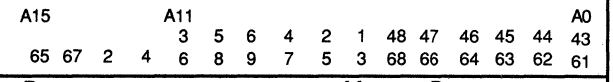
Master Mode LCA Device with Daisy Chain

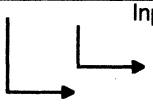
The following table summarizes each pin's function in the Master-parallel modes.

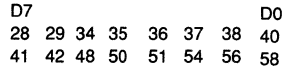
Master Low and High Modes Pin Summary

Pin Name	Pin Number PLCC	Pin Number DIP	Pin Type	Value During Configuration	Description
Fixed, Non-Programmable Pins:					
M0	26	18	Input	LOW	Mode Select
M1	25	17	Input	LOW or HIGH	(Master-LOW Mode) (Master-HIGH Mode)
CCLK	60	42	Output	<Clock>	Configuration Clock
~RESET	44	31	Input	HIGH	Master Reset
D/~P	45	32	Output	LOW	Done/Program
~PWRDWN	10	7	Input	HIGH	Power-Down
User-Programmable Pins:					
M2	27	19	Input	HIGH	Mode Select
DOUT	59	41	Output	<Data>	Configuration Data Out
HDC	28	20	Output	HIGH	Constant 1 Level
~LDC	30	21	Output	LOW	Constant 0 Level
~RCLK	57	39	Output	<Strobed>	Chip Enable Output
A0 - Axx			Outputs	<Address>	Memory Address
D0 - D7			Inputs	<Data>	Memory Data



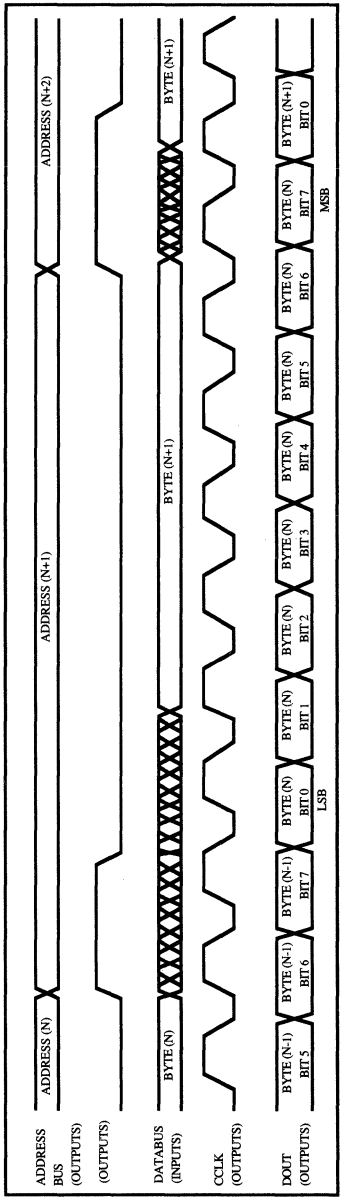






Although 16 address bits are generated in the Master parallel modes, not all 16 bits are required to configure a single LCA device. The extra addressing capacity of the LCA device lets it address multiple configuration bit streams in a single EPROM. Thus, you can configure several daisy-chained devices from a single source. The device illustrated above presents an example of a Master mode LCA device tied to a daisy chain of Slave

mode LCA devices. The timing diagram below shows the timing for the Master configuration mode.



Master Mode Configuration Timing

The Master mode device pauses briefly when powering-up, before it starts the configuration process. This ensures that it successfully configures daisy-chained LCA devices. This **power-up delay**, which is substantially longer than, and unrelated to, the initialization delay for either Slave or Peripheral mode, allows variations in the LCA device's response to V_{CC} rise times. It also ensures that all Slave mode LCA devices have time to become fully initialized and ready for configuration data. If your system requires longer delays to guarantee that all slave devices have been powered, you can use ~RESET to extend the power-up delay and to hold off the start of configuration.

The next discussion describes the configuration of multiple LCA devices in more detail.

6.3 CONFIGURE MULTIPLE LCA DEVICES

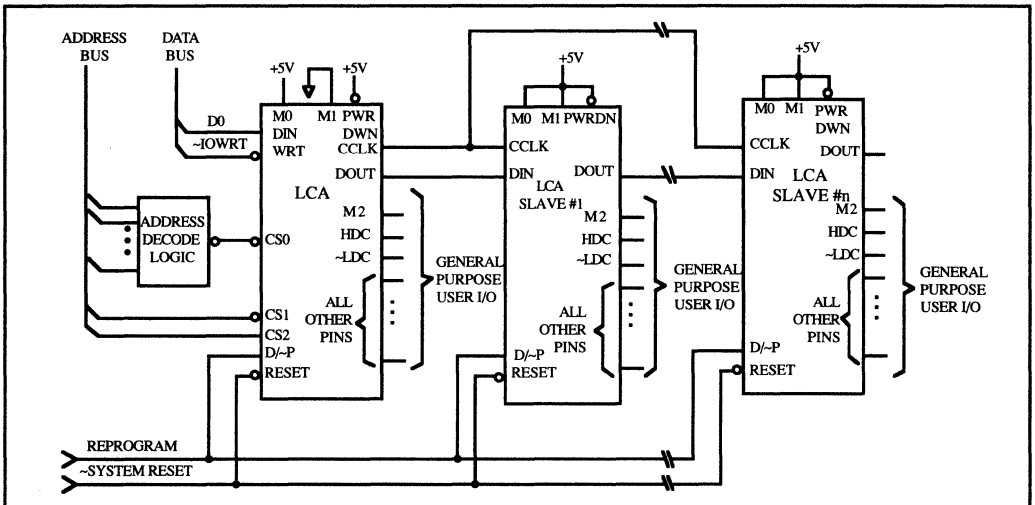
Designs using multiple LCA devices can reduce configuration overhead by logically concatenating stored configuration bit streams. Using this option, you can configure LCA devices in daisy chains or in parallel mode, as discussed below.

6.3.1 DAISY-CHAIN CONFIGURATION

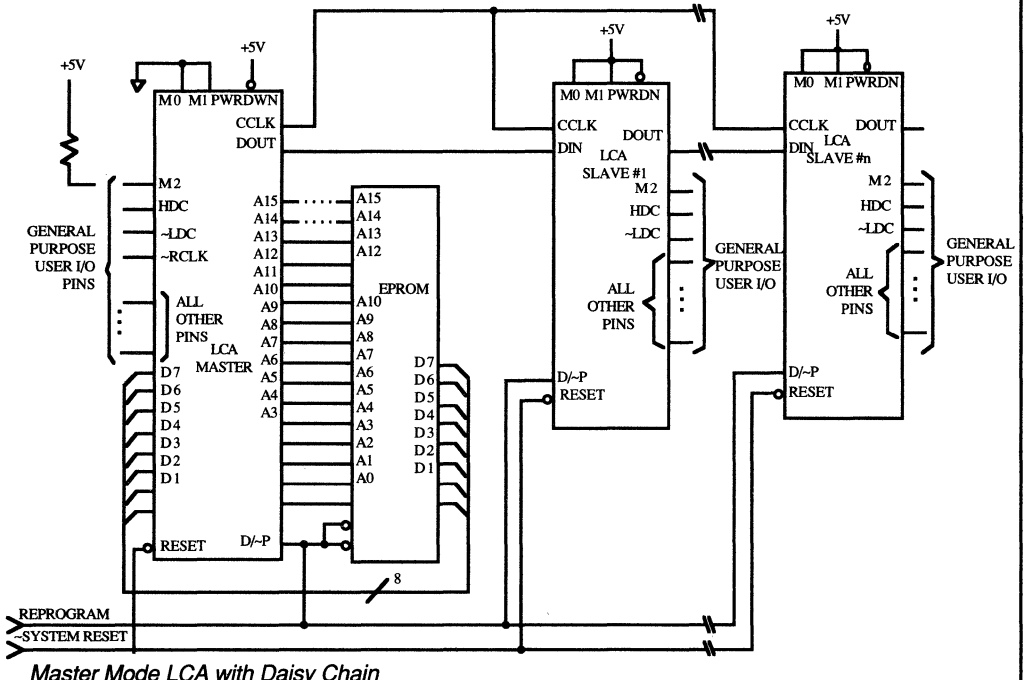
If one data source supplies the configuration bit stream for all devices in the daisy chain, then you can configure the first LCA device in the daisy chain in any configuration mode. After it is configured, you load configuration bit streams for all remaining devices in the chain using the pin-efficient Slave mode. When you cascade LCA devices in this way, you configure them one at a time in sequence, starting with the first device in the chain. You can configure virtually any length of daisy chained devices in this manner.

You daisy-chain LCA devices by connecting the DOUT pin of one device to the DIN pin of the next device. Each device in the chain supplies data to the immediately following Slave mode device. As soon as a given device in the daisy chain receives its share of the configuration data, the balance of the data simply passes through it to configure the remaining devices in the chain. The DOUT pin is HIGH until the length count is reached and configuration is completed.

Data passing through an LCA device from the DIN pin to the DOUT pin is subject to a one-clock-cycle resynchronization delay. When configuration completes for all devices, both DIN and DOUT become available as general purpose I/O pins. The following figure illustrates how you can connect multiple LCA devices into a daisy chain.



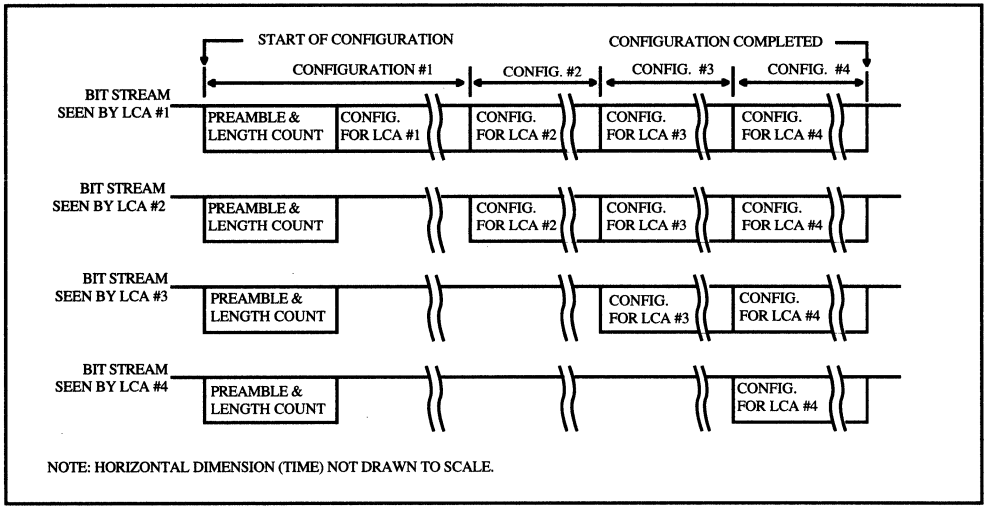
Peripheral Mode LCA with Daisy Chain



Master Mode LCA with Daisy Chain

2

The following figure shows the configuration timing for the previously discussed daisy-chained LCA device.



Timing for Daisy-Chained LCA Devices (Using Four Device Examples)

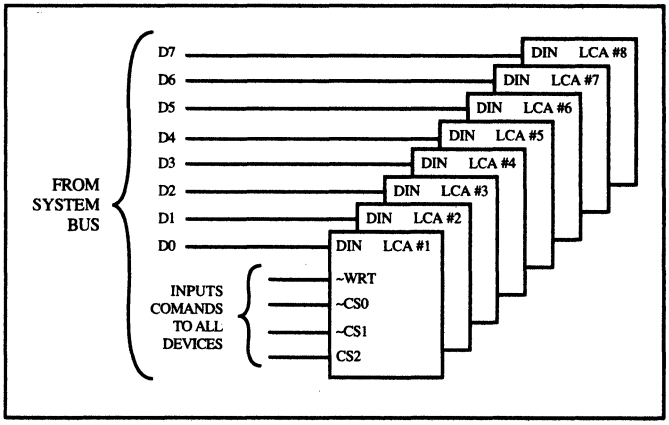
6.3.2 PARALLEL CONFIGURATION

In multiple LCA applications, you have great latitude in designing the configuration logic. The serial daisy-chain technique described above is just one method to program multiple LCA devices.

Another possibility, which takes advantage of the bit-serial nature of the Slave and Peripheral configuration modes, is simultaneous parallel configuration. You simultaneously configure multiple LCA devices, as shown in the Peripheral mode example below. Each write-cycle loads one bit into each device.

Simultaneous loading reduces the total time required to configure a group of LCA devices, to the time required to configure a **single** device. You can further improve performance by adding hardware to configure this group via DMA transfers. For example, if a processor is available, you can simultaneously configure up to eight

LCA devices in parallel, from a single disk file containing the interleaved device configuration data.



Parallel LCA Configuration Using Peripheral Mode

2

The following discussion explains considerations for assigning multiple-function I/O pins.

6.4 ASSIGNING MULTIPLE-FUNCTION I/O PINS

After selecting a suitable configuration mode, you must assign input/output functions to specific I/O pins. Typically, you base the pin assignments on

- logic block placement within the LCA device
- common I/O clock constraints, and
- I/O pin usage during configuration.

You can also use user-definable I/O pins to configure the LCA device, but these pins require careful design. For applications that require many programmable I/O pins, you should consider techniques for making efficient use of these dual-function pins, as discussed below.

6.4.1 POTENTIAL I/O CONFLICTS

Good design practice dictates that no logic signal conflicts occur during either the configuration phase or the user-operation phase. However, these conflicts may not be obvious. The directional nature of some I/O pins used for configuration changes when the LCA device completes configuration and enters the user-operation state. Your design should guarantee that pins used as outputs during configuration do not conflict with other logic sources also tied to those pins, even when they are not used in a given application.

The DOUT pin is an example of an output pin that is easily overlooked. During configuration, DOUT becomes an output, regardless of whether it drives the DIN pin of another LCA device. Other examples include the HDC and ~LDC pins, which are driven HIGH and LOW, respectively, during configuration. A design should be able to tolerate activity on these and other I/O pins used during configuration, without causing a problem if external circuits are also tied to these pins. You can usually prevent this problem by careful pinout assignment or use of isolation buffers.

The following cases describe three approaches that minimize potential signal conflicts.

- **Case 1:** I/O pins used for configuration are dedicated to that function and are not used during operation. In this case, no signal conflicts occur. However, this approach reduces the number of available I/O pins.

- **Case 2:** I/O pins used for configuration are also used during operation. However, the signals are similar in input/output sense and the system suffers no adverse effect from transitions occurring on those pins during configuration. Isolation buffers are not required.

- **Case 3:** I/O pins used for configuration are also used during operation. However, they either conflict in the input/output sense or have signal transitions during configuration that can adversely affect other system logic. You can use three-state buffers to solve this problem, perhaps with the D/~P, ~LDC, or HDC pins serving as the enable control for the buffers.

You can eliminate, or significantly reduce, external logic components in an LCA-based design by watching for the above-listed cases and carefully assigning I/O functions to actual pins. When faced with the conflicts described in Case 3, assign another pinout to eliminate the conflict. Usually, isolation buffers are not necessary because inputs and outputs are assigned without conflicts to the I/O pins used during configuration.

As an illustration, assigning output functions to pins that are already used as outputs during configuration, such as address outputs in Master mode, might obviate the need for buffering those signals. In general, any sharing of similar pin functions during and after

configuration might eliminate the need for external buffer logic.

The following cases illustrate how careful pinout assignment can reduce the number of external logic components in LCA-based applications.

- **Case 1:** When the LCA device is configured in Master mode, the final application can share pins with this mode's address and/or data buses.
- **Case 2:** When the LCA device is configured in Peripheral mode and interfaces to a CPU bus, the ~WRT, DIN, ~CS0, ~CS1, and CS2 pins are driven from this bus and, thus, can be assigned similar functions during configuration and final application.
- **Case 3:** When an application uses multiple LCA devices, and a signal passes from one device to another, you can assign the signal to the DOUT of the first device and to the DIN connection of the second device.

6.4.2 UNUSED I/O PINS

An LCA pin programmed as an input and not connected to any external logic is considered a **floating input**. As with any CMOS device, floating inputs can provide a low-impedance current path from Vcc to ground and result in permanent damage to the device. Thus, you should handle an unused LCA pin in one of the following three ways.

1. **Define it as an output,** and drive it with an internal signal, preferably a constant level 0 or 1.
2. **Define it as an input** and either
   a. drive it externally with logic, or
   b. tie it to an external pull-up or -down resistor, or
   c. tie it to Vcc or ground.

The relative advantages of defining unused pins as inputs or outputs depend on your specific application. You should try to minimize the following.

- Static and dynamic power dissipation
- Component count
- Risk of electrical damage to the device
- Future circuit board modifications

3. **The preferred method of treating unused I/O pins follows.**

   a. Externally leave the pin open or unconnected.

   b. Internally configure the pin as an output.

   c. Drive the pin internally with a constant level signal.

Typically, you select a nearby, unused CLB output, define it as a constant 1 or 0, and tie that signal to all nearby unused IOBs. If internal routing congestion precludes routing this DC signal to an IOB, your next best option is to drive the IOB's output pin with an accessible net. In this case, a net with the lowest toggle frequency is best because it results in less power dissipation.

Diagnostic **test-point outputs** are another practical use for unused LCA pins. These test points can be very valuable for monitoring internal logic nodes that would otherwise be inaccessible. Test-point-outputs aid in circuit analysis and debugging.

6.5 CONFIGURATION DATA

This discussion explains the LCA configuration bit stream format and loading.

The configuration data required to program the LCA device is a string of bits. The number of bits required to supply all the configuration information for a single device depends on the type of device, as outlined in the following table. For applications using multiple LCA devices connected as a daisy chain, the bit stream grows for each additional device.

Note: This description applies only to the bit stream generated by the LCA development system for use in EPROMs. The XACTOR in-circuit emulator uses a different version of the bit stream that is longer; the data is not packed.

Configuration File Format Shown in Binary Equivalent		
M2064LCA		
1111	Dummy Bits (4 Bits Minimum)	
0010	Preamble Code	
<24 Bit length count>	Configuration Bit Stream Listing	
1111	Dummy Bits (4 Bits Minimum)	
0 <Data frame #001> 111 0 <Data frame #002> 111 0 <Data frame #003> 111 . . . . 0 <Data frame #159> 111 0 <Data frame #160> 111	160 Configuration data frames (Each frame consists of a 0 start bit, a 71-bit data field, and 2 or more dummy bits)	Repeated once for each LCA in the daisy chain
1111	Postamable code (4 bits)	
M2018LCA		
1111	Dummy Bits (4 Bits Minimum)	
0010	Preamble Code	
<24 Bit length count>	Total Number of Bit Stream Bits	
1111	Dummy Bits (4 Bits Minimum)	
0 <Cata frame #001> 111 0 <Data frame #002> 111 0 <Data frame #003> 111 . . . . 0 <Data frame #195> 111 0 <Data frame #196> 111	196 Configuration data frames (Each frame consists of: a 0 start bit, an 87-bit data field, and 2 or more dummy bits)	Repeated once for each LCA in the daisy chain
1111	Postamable code (4 bits)	

Notes:
1. Data bits as shown in the table are shifted into the LCA device with the left-most bit of each line in the table being entered first. The bit field containing the length count is shifted in with the most significant bit first. For master-mode applications, bytes of data read from the EPROM are internally serialized so that D0 is sensed first, D7 last. Therefore, the first byte of the EPROM would read "0100 1111" in binary, or "4F" in hexadecimal notation.
2. In multiple LCA applications where a daisy chain is used for configuration, the length count reflects the total number of clock cycles for all LCA devices configured from this one bit stream.

6.5.1 CONFIGURATION FILE FORMAT

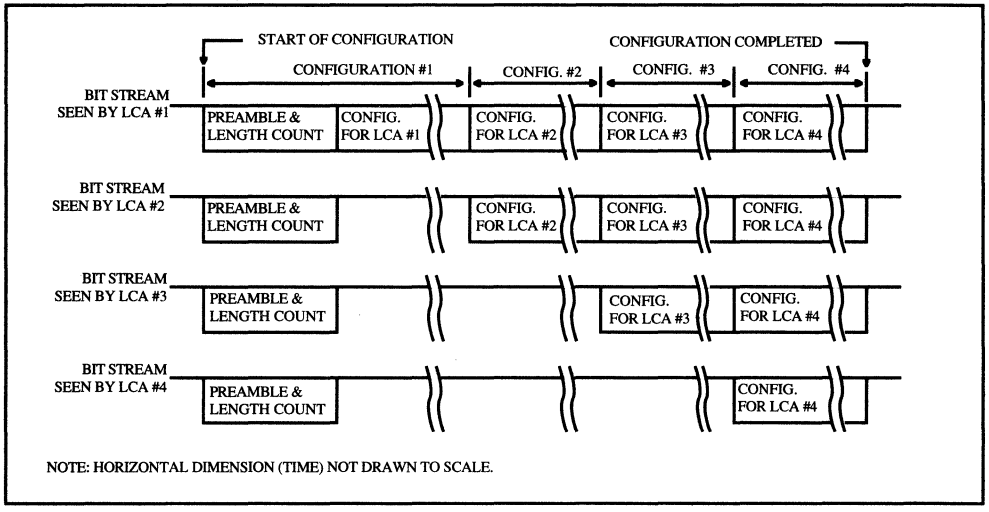
The configuration bit stream begins with several logic-1 level bits, termed **dummy bits.** These are followed by a 0010 **preamble bit pattern**, left-most bit first. Following the preamble are 24 bits that represent the **length count**. The magnitude of this count must equal or exceed a value that is two less than the total number of clock cycles required to shift in all bits in the bit stream, including the dummy bit. Length counts greater than this number (up to 2^{24}-1) are valid, and merely delay the D/~P pin from going HIGH to indicate the completion of configuration. All data associated with these additional clocks are ignored.

Note: Configuration bit streams for several LCA devices connected in a daisy chain have only a single preamble and length count.

Within the LCA device, the length-count value is held in the **length-count register** and compared to a **CCLK clock-cycle counter** to determine when the configuration process is completed. When the value of the CCLK cycle counter equals the value in the length-count register, and all required data frames are entered, configuration is done and the D/~P pin is released. Because all devices in the daisy chain start their clock cycle counters simultaneously, all LCA devices in the daisy chain complete configuration and become simultaneously operational.

The value used for the length count is a function of how many LCA devices the bit stream must configure. For example, if there are three 2064 LCA devices connected in a daisy chain, the configuration bit stream is over 36,000 bits long. The length count is included only once at the beginning of the bit stream. Several additional cycles are required to compensate for the resynchronization delay of the data at each DOUT pin.

The LCA development system **computes** the **precise value** of the length count and automatically **enters it** into the configuration file. The preamble and length-count bits are sensed by each LCA device at its DIN pin and immediately passed on to the next device in the daisy chain via the DOUT pin. Afterwards, however, each device in turn accepts its portion of the configuration bit stream before passing any subsequent data on to the next device. Refer to the following timing diagram.



Timing for Daisy-Chained LCA Devices (Example Using Four Devices)

Within the configuration bit stream, data are presented in frames that begin with a **start bit, 0**, and end with at least two dummy or **stop bits**. Between the start and stop bits of each frame, there is a data field that defines your design's logic functions. The last frame is followed by a field of **postamble bits**.

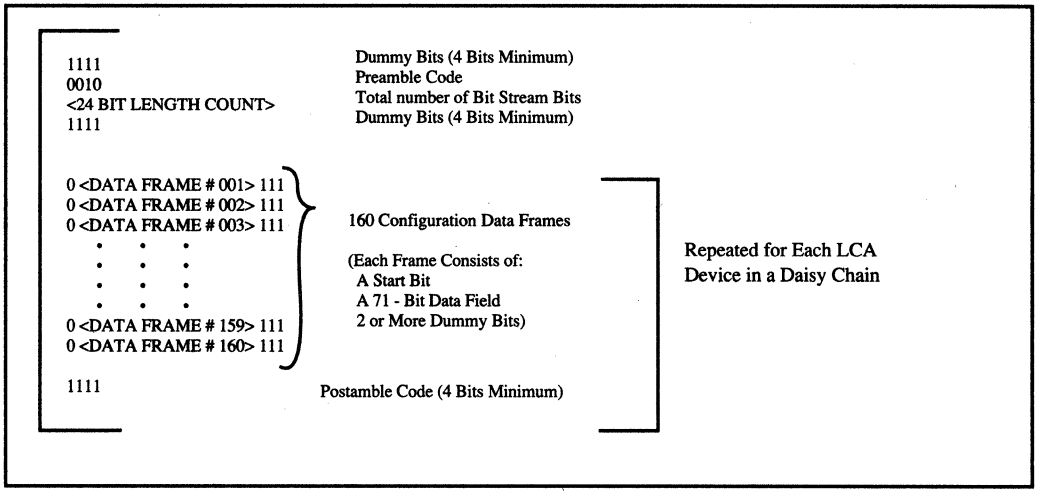
Note: AMD reserves the right to change the format, organization, and length of the bit stream used to configure the LCA device.

6.5.2 A SAMPLE EQUIVALENT CONFIGURATION FILE

This discussion assumes that a processor configures the LCA device in either Peripheral or Slave mode. The connections to the device, and the timing needed to perform the configuration, are discussed earlier in this chapter. Regardless of the configuration method, the bit stream data is the same. The bit stream for this example, created by MAKEBITS, is in a PROM file, formatted for Intel MCS86 compatibility.

The following figure summarizes the equivalent data format of the **PROM file**.

> **Recall**: The information preceding the first data field is required to initialize the configuration logic on the LCA device with the proper bit-stream length. Each subsequent data field provides configuration information for a portion of the device.



Typical Equivalent Configuration-Data Arrangement for the 2064

The beginning of the sample PROM-format configuration file for a 2064 is shown next.

The required leading 1s are in the low nibble, with the preamble in the high nibble. The bits are arranged this way to simplify the connections from an external PROM or ROM to the LCA device in Master mode.

```
: 020000020000FC
: 100000004F00F460EFFAF3F3FFF7C5FFFF7FD39CD7
: 10001000A5EBBB5975F7FFFB3F7FFEFEFEB5FCFF6E
: 10002000DFBD59AFBDBE4FFFFBFEFEDFBFBFAF5F01
: 100030007F7FF7F7FF7FFFFDFDFFFBFFBFFFFFFFA8
: 10004000CBD7FFB7FFFFFDFFFFBF3EFFFF7FFFEFF7
: 10005000FFFFBDFFFFFFB7FF7BFFFFFFFFFFDFEFEE
: 10006000FFFDFBFFFFFFDFFFFFFFFF9D7F7FFFFE29
: 10007000FCFAFD7DEFEEFFFFFFFFFFFFFFFFF7FF45
: 10008000FFFFFFFFFFFFF7F3FFFFFFFFFFFFFFFFC0
: 10009000FEF9EFEFDFFFD7BFBFFFCFDFFFFFFFBFEF
: 1000A000FFFFFF77EFEFDFDFDFBFBFBFFFF3FF7FB4
: 1000B000FFCFBFDFFFFF9DFBFAB7F7F7FFFF7FEF33
: 1000C000FAFEFEFDFD7D7B7BFBF7FFFFFFFFFFFBE5
: 1000D000FBFB3FF3F7EFE9EFFFFFFFDBB9BD5D3B54
: 1000E0005BFB77F7F7CE7777EFEFAED5D7D77B9F70
: 1000F0009F3F3F3E373E3EDE7B67FFFEFEFEEEFF4C
```

Beginning of Typical Hex PROM File

Note: The first byte of the data field, underlined in line 2, is hexadecimal 4F.

Although the LCA device's internal memory always loads serially, the Master-parallel modes read the configuration bit stream in parallel directly from an external memory device, such as a PROM. The data is serialized internally for loading into the memory cells. The PROM connections serialize the least significant bit of the byte, D0, first. Consequently, the order of the data bits in the PROM file is the reverse of the order in which they are read by the LCA device. When a processor loads the configuration data, the PROM file data is read one byte at a time and supplied to the

device one bit at a time, beginning with the least significant bit, D0.

Consider the three bytes following the first byte, 4F, of the data field in the PROM file above. The hexadecimal 00F460 represents the 24-bit binary length count of 000000000010111100000110, or 12038 decimal, which is the total number of clock cycles required to load this bit stream. Three additional clock cycles are required to complete the configuration and activate the device. The fifth byte, hexadecimal EF, contains the four pad 1s, the start bit, and the first 3 bits of the 71-bit data field.

Note: The LCA device inverts the incoming data, so the data bits stored in the memory cells are the complement of the input data.

6.5.3 CONFIGURATION LOADING

Data supplied to the LCA device during configuration is shifted into a 71-bit shift register. When the shift register is full, it is written into the internal memory cells as a single 71-bit word. In the 2064, there are 160 words of 71 bits, comprising a total of 11,360 bits of configuration data. Preamble data increases the total to 12,038 bits. For the 2018, there are 196 words of 87 bits, a total of 17,052 bits of configuration data.

The next discussion explains read-back of the configuration bit stream.

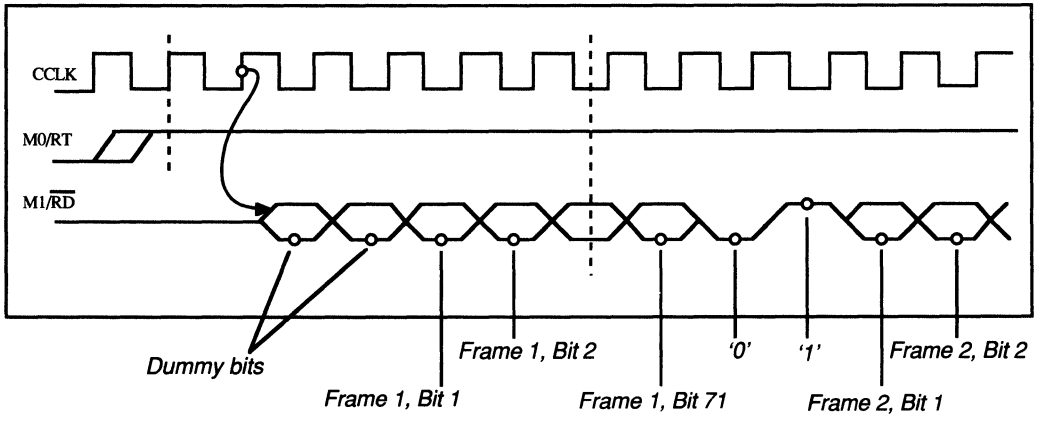
6.6 READ-BACK CONFIGURATION DATA

After you load and store the configuration bit stream, you can **read** the data **back** to confirm the configuration, as described below.

6.6.1 READ-BACK PROCESS

With the **read-back** process, you extract the configuration bit stream from the LCA device. You can use the read-back data to verify that the contents of the memory cells have not changed since the last programming cycle. The read-back data contain the state of the CLB storage elements, such as flip-flops and latches, IOB storage elements, and memory cells in the logic blocks, as well as the state of the input connection point on each I/O block.

The read-back process is accomplished without using user-programmable I/O pins. The data is read back serially by CCLK, M0, and M1. The read-back process is triggered by a LOW-to-HIGH transition on the M0/RT pin. On subsequent cycles of CCLK, internal configuration data are supplied on the M1/RD pin. The following figure illustrates the timing of this data read-back process.



Readback Control Timing

Individual data frames are read back in the same sequence as they were supplied to the device. In the read-back serial data stream, the individual bits are the **true** sense of the internally stored data bit. Recall that the bits in the programming stream are the **complement** of those stored internally.

The initial data frame of the read-back data is preceded by a dummy clock cycle and two dummy bits with an unknown state. After the first data frame, there is a stop bit, 0, and a start bit, 1, prior to the next frame. After the last frame, there is a stop bit, 0. Even when additional CCLK cycles are applied after the last data frame is read, the M1/RD output is disabled. The pin is not driven after the final stop bit.

6.6.2 READ-BACK DATA CONTENTS

After you read the configuration bit stream back, you can compare the read-back bit stream with the input data stream to determine whether the device is correctly configured. You must remove the input data dummy bits and start bits, and the read-back data start and stop bits, either as part of the programming and read-back process or after the read-back process completes.

In the configuration and read-back bit streams, some of the memory locations do **not** correspond to actual memory cells in the device. These locations may be unused during both the configuration and the read-back processes. They contain the storage elements and input block values during the read-back. You can extract and display the storage element and input block values with the XACTOR in-circuit emulator during debugging. To verify configuration, ignore these bit locations, because their contents might not be the same as the corresponding positions in the bit stream.

By using the MAKEMASK command, you generate bit positions, which are ignored in the read-back data stream. You can convert the bit file generated with

MAKEMASK into a PROM file by using the MAKEPROM software. The MAKEPROM software converts the binary bit stream to hexadecimal format. The format for the final-mask PROM is now identical to the configuration bit-stream PROM. Each data bit in the read-back data stream that should be ignored is represented by a 0 in the mask PROM file.

The following figure shows the beginning of a mask bit-stream file for a 2064 that was converted from binary format into a hexadecimal PROM-format.

Note: This file has the same preamble, length count, and pad bits for one LCA device as the standard configuration data file.

```
: 020000020000FC
: 100000004F00F4608FEDEDDD8BDBBBB78770ECEF5D
: 10001000DF1FDCBFBF3F9463FFFFFFE9FFFFFFB3BC
: 100020001DFFFFFBD7FFF7FFAFED587FDFFEFEBEE2
: 10003000FDFD2DC7FAFFFEFFFFFDFF7FB9E6FFF7CD
: 10004000BFFFEFFF7FDB35FFBFFFF77FFFFFDFEE77
: 10005000F1F3E7A7E7CFCFCF7EBFBC7C7979F9F288
: 10006000F2FAFBF575EBEBEBD6D7D7DFA7A74B4F33
: 100070004F979E9EFE3E3D5D7A7ABAF4F4F4D7ED3A
: 10008000EDDADBDBB5B7B7BD7F7FF7FEFEEEFDFD3A
: 10009000FDFDFFFFFFFFFFFFFFEFEFFBFBF5F7F7B6
: 1000A000EBEF6F7FDEDFAFBFBF5F7F7FBB7B78F89B
: 1000B000F0F0F0E1E1C1DEEFEFDBDFDFB7BFBFFE65
: 1000C0007E7FFFFEFEFEFDFDF5D7F9F9F2F3F3E5C5
: 1000D000E7E7BDCECF979F9F2F3F3FEF757FBFFED6
: 1000E000FE7EFDFD7DEFFBFBF5F7F7EBEFEF7FFF0E
: 1000F000FFEFFFFFDFFFFFFFFBFFFFFFFFFFFFFF44
```

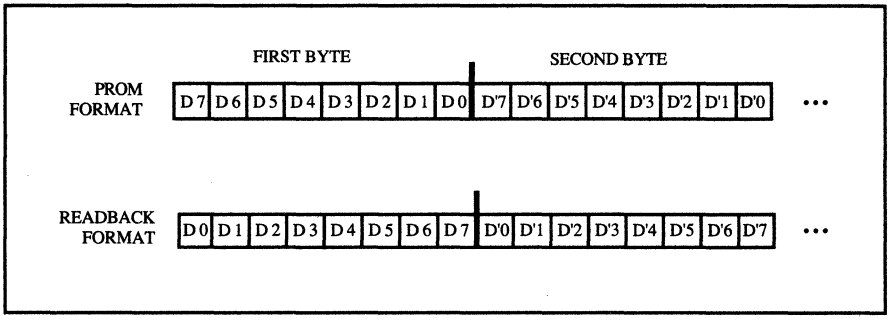
Mask PROM File

To use the mask information,

- strip off the preamble and length count information, and
- extract the appropriate data bits for each data field.

Because the PROM format has the data bits arranged with the least significant bit in the D0 position, they are reordered in the correct sequence, as shown in the figure below. For example, you could write a simple program to create the mask bit fields for each data field.

Note: You can verify the end of one data field, and the beginning of the next, by detecting the dummy bits and the start bit between each field.



Data Bit Sequence

CHAPTER 7

METASTABILITY OF LCA FLIP-FLOPS

2

7 METASTABILITY OF LCA FLIP-FLOPS

This chapter discusses flip-flop metastability in an LCA design.

- The discussion on flip-flop metastability, 7.1, defines the topic and explains the importance of considering it when implementing LCA-based designs.
- The discussion on LCA flip-flop error probability, 7.2, analyzes the flip-flop error probability due to metastability.
- The discussion on minimizing error probability, 7.3, describes the flip-flop error reduction features and provides some design techniques to minimize the error probability.

Note: The following discussions on metastability in LCA-based designs merely indicate **some** of the considerations you should take into account during your design cycle. They are not intended to be exhaustive or definitive.

7.1 FLIP-FLOP METASTABILITY

Metastability is defined as an output state between a valid logic HIGH and LOW for any digital device. It can occur for registers or latches when certain parameters, such as data setup and hold times, are violated. Data at the input of a D-type register must be established as a valid logic LOW or HIGH at some specified time t_{SU} (setup) before applying a clock input to that register. This data must also be maintained or held at the input for a specified time t_H (hold) after the clock pulse has been removed.

In a completely synchronous system, the clocking of data through registers can be synchronized to a clock edge that is generated from a local source, such as an on-board crystal oscillator. In this case, the timing is predictable and setup and hold times are adhered to when all system parts are connected. In this type of system, no metastability problems should occur. The designer can calculate propagation-delay values from published component data and ensure that no timing parameter violations occur.

With two independently clocked systems, it might not be possible to synchronize the clock frequencies or events, so when data are passed from one system output to the input of the next, setup and hold times might be violated for registered inputs to flip-flops. If the setup and hold time requirements are small relative to the sampling clock period, the probability of violating these parameters is not very high. It further decreases with a decrease in the sampling clock frequency. Also, if the setup and hold times can be reduced by using higher performance devices, then the probability of violating these parameters is further diminished. However, a small probability of a metastable state would still exist.

In digital circuits, valid data input to registers or latches are set either LOW or HIGH. The voltage level is dependent on the technology. If this valid condition is

set up prior to a clocking edge, the data is clocked to the register output and no metastable condition arises. A problem could occur if, at the time of sampling, the input signal is in transition.

In an LCA device, the following valid conditions can occur for TTL and CMOS circuits. In both cases $V_{CC} = 5$ V

TTL logic HIGH	2.0 V to 5.0 V
TTL logic LOW	0.0 V to 0.8 V
CMOS logic HIGH	3.5 V to 5.0 V
CMOS logic LOW	0.0 V to 1.0 V

For a TTL circuit, any input between 0.8 V and 2 V sampled by a clock edge is neither a LOW nor a HIGH, and represents a violation of hold and setup times. The condition of the output can **not** be guaranteed to follow a valid logic state because none was sampled at the input.

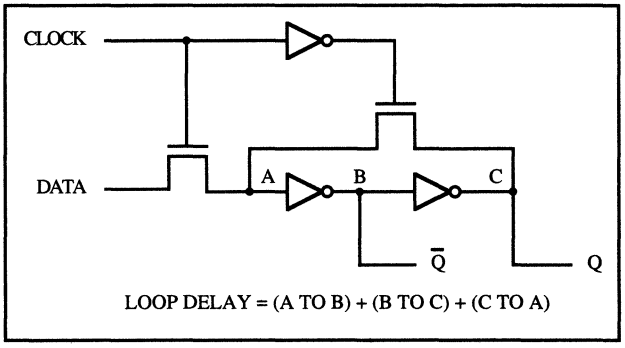
The typical gate or inverter is essentially a high-gain linear amplifier circuit. Logic HIGH or LOW outputs represent saturation conditions; further input drive does not achieve a corresponding output change.

If, during sampling, the input is transitioning between two logic states, the register or latch could be operating as a linear high-gain amplifier. The ability to recover from a metastable state is then dependent upon the characteristics of the logic device, which is not operating in a valid mode. The gain/bandwidth product of the device in this mode influences the device behavior and determines the output recovery from a metastable condition.

Attempting to characterize the metastability of LCA devices is difficult because the timing associated with different interconnections varies from application to application. Setup and hold times to register and gate

combinations depend on the routing resources used, and these depend on the CLB/IOB layout and configuration. The best way to test for metastability is to test the completed design, deriving metastability data empirically.

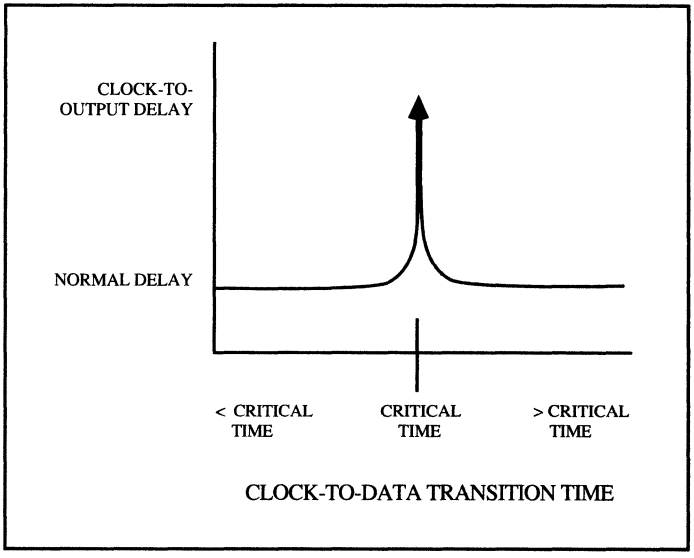
One critical element in examining metastability is the gain/bandwidth product, which gives rise to a **loop-delay** in the system. Loop-delay is the time required for a signal at any point in the flip-flop to propagate through the flip-flop circuit and reinforce the signal at its starting point. The following figure shows one type of flip-flop with the loop delay path indicated.



Loop Delay in a Typical Flip-Flop Implementation

A change in the state of a node in the flip-flop, for example, a change at the input, requires one loop delay for the flip-flop to hold the new state. In a metastable condition, an internal node, typically in the input stage, attains an intermediate level because the data signal changes while the clock is changing. This intermediate level, neither 1 nor 0, propagates around the loop and forces the output into a metastable state. The flip-flop only achieves a stable (1 or 0) output when a node at the intermediate level becomes 1 or 0, and this new value then propagates through the loop to force the output out of the metastable condition. Movement of internal nodes away from the intermediate level occurs randomly and cannot be predicted or guaranteed.

Another way to illustrate flip-flop metastability is to plot the worst-case clock to flip-flop output delay versus the delay from stable data to the clock edge. The figure below shows this type of plot for a typical flip-flop.



Flip-Flop Output Critical Timing

As the data transition approaches the clock edge, the stable output delay begins to increase. For any flip-flop type, there is a finite probability that the **loop delay** x at the critical data-to-clock relationship can be infinitely long.

The next discussion explains how error probability is determined; it also provides some sample calculations.

7.2 LCA FLIP-FLOP ERROR PROBABILITY

The two critical issues in examining metastability characteristics of flip-flops in any system are the probability of an error based on a metastable condition, and the methods of minimizing the error probability. For logic implemented with LCA devices, you have some additional control.

The probability of a flip-flop passing through a metastable region can be calculated as follows.[1]

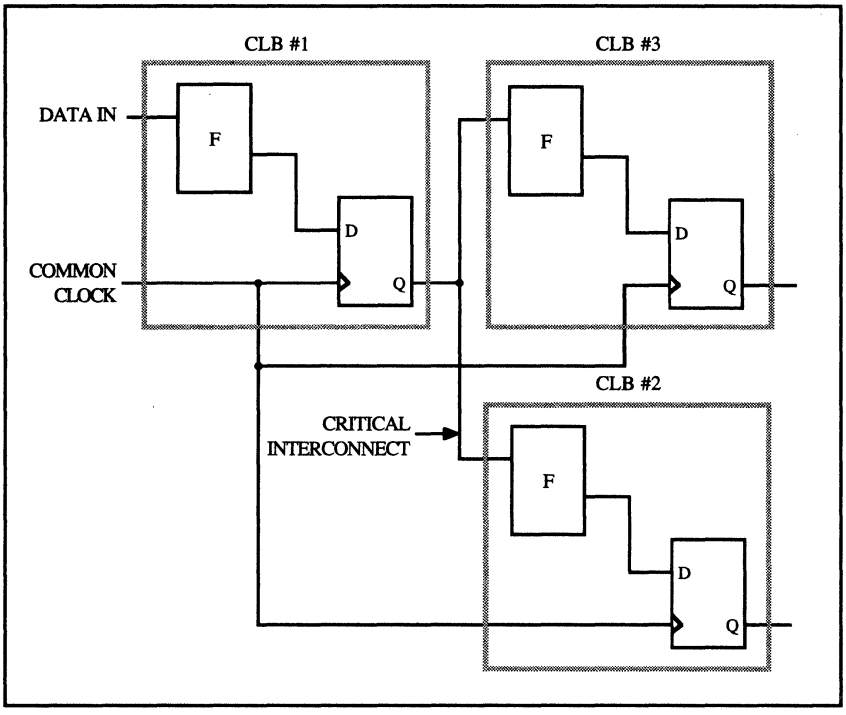
$$\text{Probability} = 1 - e^{(-\text{settling time/loop delay})}$$

The probability of a flip-flop remaining in the metastable region is then calculated as follows.

$$\text{Probability of Error} = e^{(-\text{settling time/loop delay})}$$

For the circuit shown and analyzed below, the **settling time** is the difference between the worst-case delay from a clock edge clocking the flip-flop in CLB 1 and the output propagating to the flip-flop in CLB 2; this includes the setup time and the delay from one clock edge to the next. This maximum delay path produces the lowest settling time with the highest probability of error.

1 G.R. Couranz and D.F. Wann, *Theoretical and Experimental Behavior of Synchronizers Operating in the Metastable Region*, IEEE Transactions on Computers, Vol c-24, No.6, June 1975.



LCA Implementation

Examining a specific case where the 2064 has a worst-case flip-flop loop delay of 2 ns, the critical timing parameters required to estimate the error probability for a metastable condition are shown below.

Flip-flop clock to output delay	20 ns
Interconnection delay	15 ns
Flip-flop setup time	12 ns
Flip-flop loop delay	2 ns
Clock Period (10 MHz)	100 ns

For this example, the error probability is calculated as follows.

P [Error]	= $e^{(-[100-(20+15+12)]/2)}$
	= $e^{(-53/2)}$
	= 3.1×10^{-12}

This value represents the probability of a flip-flop remaining in a metastable region beyond the given settling time for a **single event**. For **multiple events**, which represent a repetitive clock sampling of an asynchronous signal, the time between failures obeys the following relationship.

Failures Per Time Period	= Probability per Event X Events per Time

The time between failures is, then, the inverse of the failures per time-period value. For this example,

Failures per Time Period	= $(3.1 \times 10^{-12}) \times (1 \times 10^{7})$
	= 3.1×10^{-5}
Time Between Failures	= $1 / (3.1 \times 10^{-5})$
	= 3.3×10^{4}
	= 8.96 hrs

This number indicates a very short time between errors for sampling an event at 10 MHz. This is a worst-case calculation because it is based on an assumption that a potential error condition exists for **each** clock edge. In a real system this is not the case. When the input clock edge is not synchronized with the arrival of data at the input of the same register, a violation of setup and/or hold times can occur. A metastable condition can result under these circumstances. The statistical probability of these violations, however, is relatively small.

The previous type of calculation indicates several ways of improving flip-flop performance. First, the AMD LCA devices have incorporated error reduction features. Second, there are several design techniques you can follow to increase the error-free performance of the flip-flops in your LCA design. These are explained in the next discussion.

7.3 MINIMIZING THE ERROR PROBABILITY

There are several features and design techniques that can be used to minimize the error probability due to metastability in LCA-based designs. These are summarized in the following discussions.

- 7.3.1, Reducing Errors
- 7.3.2, Using Direct Interconnection
- 7.3.3, Changing the System Clock Rate
- 7.3.4, Using a Faster Device
- 7.3.5, Summary

7.3.1 REDUCING ERRORS

One critical factor in flip-flop metastability is the **length of the loop delay**. As shown in the previous error-probability calculations, the longer the loop delay, the higher the error probability.

Flip-flops in the LCA 2000 family are specifically designed to reduce the loop delay to a minimum, thus minimizing the probability of a metastability-induced error.

Another critical factor in determining the metastability characteristics of flip-flops in a device is the **loading of the flip-flop**. In virtually all other technologies, particularly gate arrays, the flip-flop outputs can be loaded differently depending on how the user connects the devices. This variation in loading can significantly complicate the analysis of the flip-flop's metastability behavior.

Note: In LCA devices, all flip-flops are immediately followed by a buffer, and then by the possible user-programmable connections. The buffer isolates the flip-flop from any variations in loading that could induce metastability. This flip-flop isolation significantly simplifies analysis of metastability effects in the system by standardizing the LCA flip-flop output loading.

7.3.2 USING DIRECT CONNECTIONS

If you use direct connections in the critical-path area of the above example, you can eliminate the 15 ns delay for that interconnection. The effect on the error probability is calculated as follows.

P [Error]	$= e^{(-[100 - (20 + 12)]/2)}$
	$= e^{(-68/2)}$
	$= 1.71 \times 10^{-15}$

For the same 10 MHz clock the error probability results in a significantly reduced mean time between failures (MTBF).

MTBF	$= 1/(1.71 \times 10^{-15}) \times (1 \times 10^{7})$
	$= 5.83 \times 10^{7}$
	= 675 days

Refer to Chapter 5 of this manual for more information on direct connections.

7.3.3 CHANGING THE SYSTEM CLOCK RATE

If you reduce the clock rate of the sampling, you can dramatically reduce the calculated failure rate. If, for example, you reduce the clock rate from 10 MHz to 5 MHz, you can analyze the error probability with the following parameters.

Clock to flip-flop output	20 ns
Interconnection delay to second block	15 ns
Setup time for second flip-flop	12 ns
Flip-flop loop delay	2 ns
Clock Period (5 MHz)	200 ns

The error probability is calculated as follows.

P [Error]	=	$e^{(-[200-(20+15+12)]/2)}$
	=	$e^{(-153/2)}$
	=	5.98×10^{-34}

And for a 5 MHz clock rate, the MTBF is calculated as follows.

MTBF	=	$1/(5.98 \times 10^{-34})(5 \times 10^{6})$
	=	3.35×10^{26}
	=	3.87×10^{21} days, or
	=	approximately 1.1×10^{19} years

7.3.4 USING A FASTER DEVICE

If you use a faster LCA device, you improve all device-related performance parameters. Moving to the next higher LCA speed grade results in the following critical parameters.

Flip-flop clock to output delay	15.0 ns
Interconnection delay	7.0 ns
Flip-flop setup time	8.0 ns
Flip-flop loop delay	1.5 ns

If the clock period remains the same, at 100 ns, then the new error probability is calculated as follows.

P [Error]	=	$e^{(-[100-(15+7+8)]/1.5)}$
	=	$e^{(-46.667)}$
	=	5.4×10^{-21}

For the 10 MHz clock rate, this error probability results in a new MTBF, as calculated below.

MTBF	=	$1/(5.4 \times 10^{-21})X(1 \times 10^{7})$
	=	1.85×10^{13}
	=	2.14×10^{8} days, or
	=	approximately 563 years

7.3.5 SUMMARY

As shown by these examples, the probability of a flip-flop failure, based both on a metastable condition and on the subsequent system MTBF, can vary widely and depends on several factors. When using the LCA device to implement system-level functions, you have significant control over some of the critical parameters necessary to provide sufficient immunity to metastable conditions.

CHAPTER 8

TESTING AND DATA INTEGRITY

8 TESTING AND DATA INTEGRITY

This chapter discusses the testing and data integrity features of the AMD LCA device.

- The discussion on LCA testability, 8.1, explains the built-in LCA test features.
- The discussion on data integrity, 8.2, explains the special LCA memory cell design that provides a high level of data integrity in the LCA device.

AMD is committed to providing LCA devices of the highest quality and reliability for its customers. Quality is best achieved by taking the necessary steps to achieve zero defects.

Comprehensive testing ensures that every LCA device is free of defects and that it conforms to data sheet specifications. The LCA memory cell design ensures the configuration integrity. Careful memory cell design also minimizes the effects of alpha-particle emission and electromagnetic radiation on the LCA device's operation.

8.1 LCA DEVICE TESTABILITY

As quality consciousness has grown among semiconductor users, awareness of the importance of semiconductor testability has increased among manufacturers. When manufacturers test standard components, including memories and microprocessors, they use carefully developed processes that exhaustively test the function and performance of each part.

For reasons explained below, most application-specific ICs (ASICs) cannot be comprehensively tested. Without complete testing, defective devices can escape detection and be installed in a system. In the best case, the failure is detected during system testing at a high cost. In the worst case, the device failure is detected only after shipping the system to a customer.

8.1.1 TEST-ABILITY FEATURES

The AMD LCA device has intrinsic testing advantages, which make it easy to test the device more comprehensively than is possible with other ASICs. The following discussion illustrates these advantages through a comparison with two other types of ASICs, Erasable Programmable Logic Devices (EPLDs) and gate arrays.

8.1.1.1 EPLDs

To test all of the EPLD's memory cells and logic paths, you must program it with many different patterns. This testing requires expensive quartz-lid packages and many lengthy program/test/erase cycles. To save time and reduce costs, you usually abbreviate this process and, therefore, you do not fully test the EPLD.

8.1.1.2 Gate Arrays

Testing a gate array is similar to testing an EPLD. Because each gate array is programmed with individual metal masks, you must test the part with software tailored to the specific design. Each new gate array design requires new test software and test vectors. And each time you create a new design, you must incorporate sufficient controllability and observability for comprehensive testing. Therefore, the design schedule for a new gate array device must include time to incorporate testability features, develop test vectors, and specify test software.

Software to test a complete gate array requires comprehensive fault simulation and test grading, and significant amounts of expensive computer time. It typically requires a series of time-consuming and expensive test iterations to reach 80% fault coverage for a typical design; the cost of greater coverage is often prohibitive. In production, many gate array vendors either limit the number of test vectors you can use, or charge for using additional ones.

When you design a gate array, you can improve your design's testability using special, testable, storage elements, called **scan cells**, in place of flip-flops and latches. Although using scan cells can reduce the production testing costs, it typically adds about 30% more circuitry, decreases performance by up to 20%, and increases design time.

8.1.1.3 LCA Devices

LCA devices are designed for ease of testing. The testability of an LCA device is similar to that of other standard products, including microprocessors and memories. LCA design and test strategies, discussed below, make LCA devices highly testable.

The LCA design strategy has the following features.

- It **incorporates testability** because you can configure each functional node and route it to I/O pads.
- It **permits repeated exercise of the part** without removing it from the tester, because only a short time is required to load new configuration data.
- It **produces a standard product** that guarantees that every valid configuration works.

The LCA test strategy is characterized by the following.

- It **reads and writes all bits** in the configuration memory, as in memory testing.
- It **uses an efficient parallel-testing scheme** that fully tests multiple configurable logic blocks (CLBs) simultaneously.
- It **is exhaustive** because all CLBs are identical.

8.1.2 TESTING PROCEDURES

As an LCA device user, you can better appreciate the LCA test procedures by examining each of the testing requirements listed below.

- LCA testing exercises and verifies all configuration memory bits using a read-back mode.
- LCA testing detects all possible process-related faults, such as short circuits. The LCA design lets you drive and observe every metal line directly from the I/O pads.
- LCA testing configurations provide good **controllability** and **observability**. The

controllability and observability requirements are feasible for LCA devices because all CLBs can be connected to I/O pads. This LCA characteristic makes CLBs **easy to control,** by testing different input combinations, and **easy to observe**, by comparing the actual outputs with expected values.

8.1.3 SUMMARY

The test features and procedures described above indicate that **the LCA device was carefully designed to be 100% testable.** This AMD test strategy uses numerous design configurations to fully test the LCA device. Furthermore, this strategy reduces your design time because testability requirements need not be considered during the design cycle.

The AMD LCA device frees you from the burden of developing a test program and generating test vectors. Also, it eliminates any questions about fault coverage and the need for fault grading.

Testability is very important in quality-sensitive applications. You can build significant added-value into your design by using the higher-quality levels of an LCA-based implementation.

The next discussion focuses on the integrity of data in an LCA device. It explains the various aspects of integrity, such as the robustness of the memory cells that contain the configuration bit stream, the effect of alpha particles, electrostatic discharge, latchup, radiation, and high temperature on the reliability of an LCA device.

8.2 DATA INTEGRITY

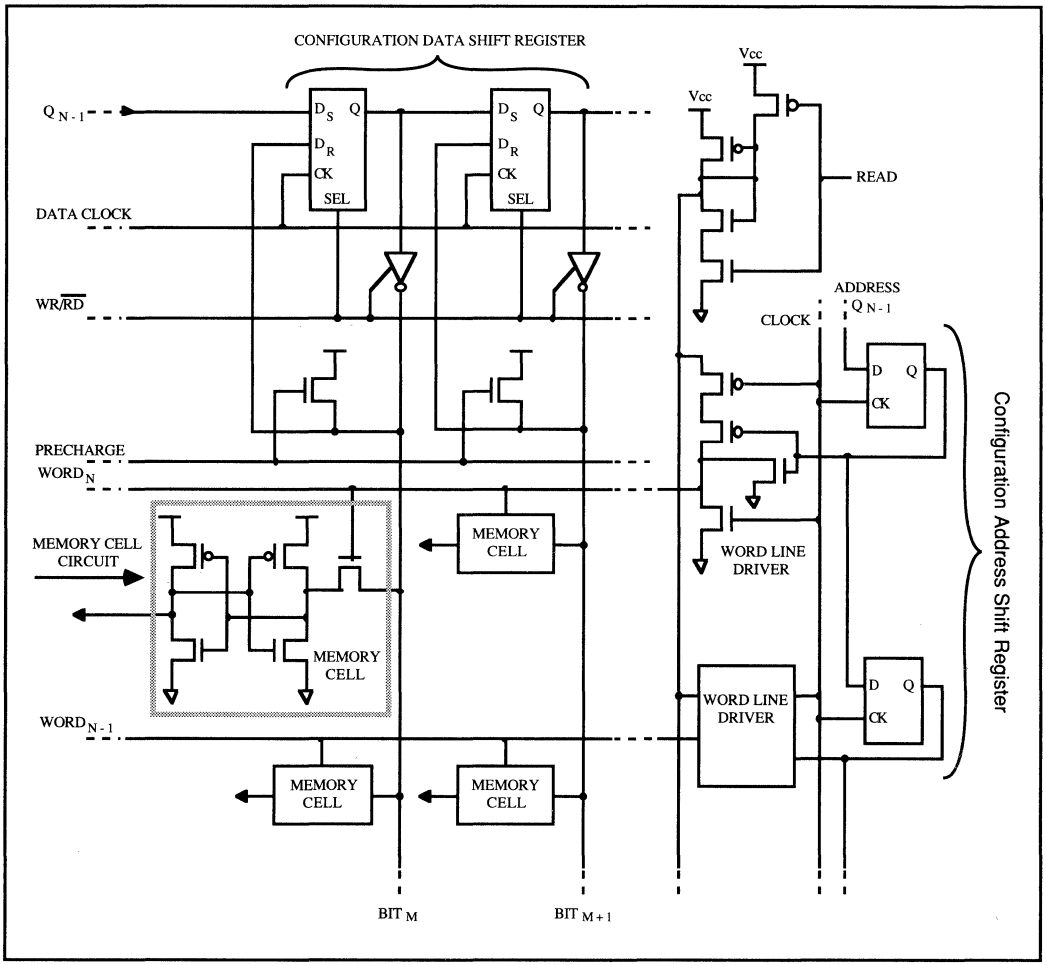
The high level of data integrity in the AMD LCA device is based on the specially designed static memory cells that store the LCA configuration data. All characteristics that ensure reliable operation are designed into the memory cells. These characteristics are described in the discussions below.

- 8.2.1, Reliability
- 8.2.2, Alpha Particle (Soft Error) Sensitivity
- 8.2.3, Electrostatic-Discharge Protection
- 8.2.4, Latchup Protection
- 8.2.5, Radiation Hardness
- 8.2.6, High-Temperature Performance

8.2.1 RELIABILITY

An important aspect of the LCA device's reliability is the robustness of the static memory cells that store the configuration data.

The basic LCA memory cell shown below is a special, single-ended, five-transistor memory element and not the typical, six-transistor, memory cell. Eliminating the sixth transistor, which usually functions as a pass transistor for the complementary output bit, technically slows down the memory cell. However, using the five-transistor, memory cell achieves a higher circuit density with no impact on normal LCA device operation, as explained below.



Basic LCA Configuration Memory Cell

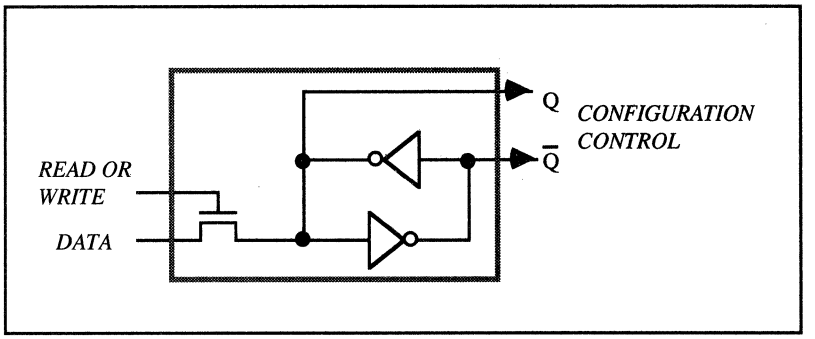
During normal operation, LCA memory cell outputs are fixed; they hold the LCA configuration. Because these outputs do not change, the slower transition time of the five-transistor configuration does not impact normal operation of the LCA device at all.

The LCA memory cell outputs do change during the configuration write and read-back operations, so using the five-transistor cell slows down these operations.

However, the write and read-back operations have no impact on LCA performance during normal operation, so the LCA user gets more functionality per unit area with no noticeable performance impact.

The design of the basic memory cell also ensures the LCA user of high data integrity in a noisy environment. Consider the following three situations.

- Normal operation
- Write operation
- Read-back operation



Configuration Memory Cell

In **normal operation**, the data in the basic memory element does not change. Because, as shown in the above figure, the two circularly linked inverters that hold the data are physically adjacent, supply transients result in only small, relative, voltages differences. Each inverter is actually a complementary pair of transistors; so a low impedance path exists to the supply rail whether the output is HIGH or LOW. This memory cell configuration results in extremely high-noise immunity. Power supply or ground transients of several volts have no effect on the stored configuration data.

For the **write operation**, the transistor driving the memory cell bit line is carefully designed so it can easily override the output of the feedback inverter whenever

the data to be written to the memory cell is the opposite sense of the stored data. The reliability of the write operation is guaranteed within the manufacturing process tolerances.

During a **read-back operation**, the bit line, which has a significant amount of parasitic capacitance, is precharged to a logic 1. The pass transistor is then enabled by driving the word line HIGH. If the stored value is 0, the line is discharged to ground. Reliable reading of the memory cell is achieved during a read-back operation by reducing the word line HIGH level to a level that ensures that the cell is not disturbed.

8.2.2 ALPHA PARTICLE SENSITIVITY

The CMOS static memory cell design is insensitive to alpha-particle emissions. The following tests verify that it achieves this design goal.

A 1-microCurie alpha-particle source (Americum 241) was placed in direct contact with the top surface of a 2064 die, allowing the die to capture at least 40% of the emissions from the radiation source. Next, the following tests were performed.

1. A complex pattern containing roughly 50% logic 1s was loaded into the 2064, at operating conditions of 25°C and 5.0V.

2. A pause of variable duration was allowed.

3. The entire contents of the 2064 were then read back and compared with the original data.

To ensure that the test setup would detect errors, validation tests were performed before and after the alpha particle tests, with the following results.

Test	Time Duration (sec)	Read-back Time (sec)	Total Time Exposed (sec)	Errors
1	10	70	80	0
2	120	70	190	0
3	300	70	370	0
4	1500	70	1570	0
Total	1920	280	2210	0

Validation Tests

The following discussion analyzes this alpha particle testing. A 1-microCurie source emits 3.7×10^4 alpha particles per second. Assuming that 40% of these are captured by the 2064 during this experiment, this rate corresponds to 5.3×10^7 alpha particles per hour.

The alpha-particle-emission rate of the molding compound used by AMD is specified to emit fewer than 0.003 alpha particles per square centimeter per hour (alpha particles/cm^2/hr). The 2064 die's surface area is less than 0.5 cm^2; thus, less than 0.0015 alpha particles per hour are captured by the 2064 in normal operation. Therefore, the error rate acceleration in this test equals

$$\frac{5.3 \times 10^7 \text{ particles/hour}}{0.0015 \text{ particles/hour}} = 3.6 \times 10^{10}$$

The 0.61 hours of test time without error correspond to 2.2×10^{10} hours, or 2.5 million years error-free operation.

Most ceramic packages are specified to emit less than 0.01 alpha particles/cm^2/hr, which is about three times more than the plastic compound. For a 2064 in a ceramic package, this still results in almost one million years of error-free operation.

The highest rate of alpha particle emission comes from the sealing glass used in cerdip packages and some ceramic packages (frit lids). For instance, KCIM glass

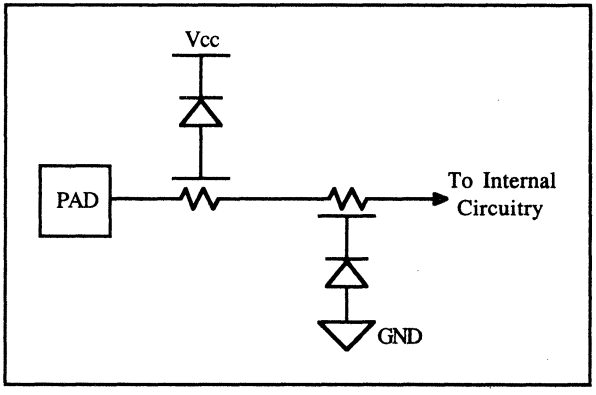
emits about 24 alpha particles/cm^2/hr. Low alpha glasses are specified at 0.8 alpha particles/cm^2/hr.

Because the glasses are used only for the package seal, they present a relatively small emitting cross section, less than 0.1 cm^2 square, to the die. Therefore, a low-alpha glass would cause fewer than 0.8 alpha particle hits per hour. The acceleration factor is then $6.6x10^8$, which translates to about 46,000 years of error-free operation.

Clearly, the AMD LCA memory cell design ensures that soft errors caused by alpha particles can be ignored.

8.2.3 ELECTRO-STATIC DISCHARGE PROTECTION

Electrostatic discharge (ESD) protection for each LCA pad is provided by a circuit that uses forward- and reverse-biased distributed resistor-diodes, as shown in the following figure.



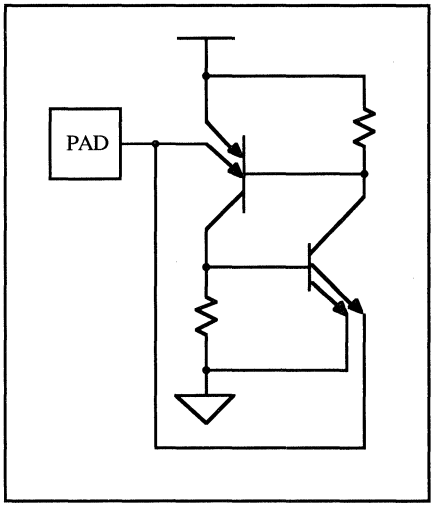
Input Protection Circuitry

Also, the inherent capacitance integrates any current spikes to give the diode and breakdown protections sufficient time to provide a low-impedance path to the power-supply rail. The LCA geometries and doping levels are optimized to provide sufficient ESD

protections for both positive and negative discharge pulses.

8.2.4 LATCHUP PROTECTION

Latchup is a condition in which parasitic bipolar transistors form a positive feedback loop; this allows the current to quickly reach levels that permanently damage the device, as shown in the following figure.



SCR Model

Latchup is caused by the formation of parasitic bipolar transistors around two adjacent CMOS transistors. A Semiconductor Controlled Rectifier (SCR), or thyristor, is formed, as shown in the diagram of the SCR model. The net effect of latchup is the static short circuit created from V_{DD} to V_{SS} as the positive feedback causes both parasitic transistors to turn on. The phenomenon of latchup can cause permanent damage, because the short circuit current overheats and ultimately, destroys the CMOS gate.

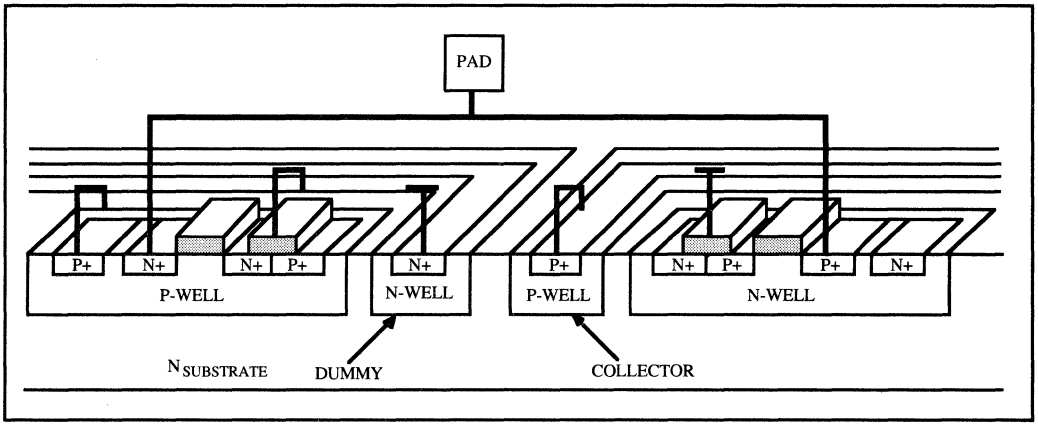
AMD uses techniques based on doping levels and circuit placement to avoid latchup. The cross section of

the transistor shown below illustrates some of these protection features.

First, reducing the gain of the parasitic transistors protects against latchup. Because the overall loop gain is minimized, the likelihood of positive feedback is reduced. This is achieved by increasing the width of the transistor bases, which lowers the parasitic transistors' beta.

Second, the butting of contacts effectively shorts the N+ and P+ regions of both wells, so a forward bias V_{BE} cannot be established to start the two parasitic transistors' conduction.

Finally, a guard ring surrounds each well. The N+ ring region is taken to V_{DD} and the P+ ring region is taken to V_{SS}. This acts as a reversed-biased diode between the two CMOS transistors, effectively isolating them.



CMOS Input Circuit Layout

8.2.5 RADIATION HARDNESS

A preliminary estimate of the circuit's hardness to withstand ionizing radiation ranges from 10,000 to 100,000 rad SI. This estimate was reached with Sandia National Labs and is based on the design and layout parameters of the LCA device.

8.2.6 HIGH TEMPERATURE PERFORMANCE

AMD guarantees parts to perform within the specifications of the datasheet. Furthermore, AMD has performed extensive high-temperature life testing at 125°C with excellent results.

CHAPTER 9

NONHERMETIC PACKAGE RELIABILITY

9 NONHERMETIC PACKAGE RELIABILITY

This chapter describes the nature and purpose of the various reliability tests that AMD performs on finished devices.

- The overview of AMD's testing software, 9.1, explains the applied testing standards.
- The discussion on AMD's test procedures, 9.2, describes the tests performed for die qualification, package integrity, and assembly qualification.
- The chapter summary, 9.3, shows test data from the initial qualification tests.

9.1 TESTING OVERVIEW

AMD is committed to delivering the highest quality, most reliable, programmable gate arrays possible. A strong quality assurance and reliability regimen begins at the initial stages of design and carries through to final shipment. The ultimate proof of AMD's success is in the performance of the LCA device in your applications.

AMD uses an extensive, continuing reliability-testing process to predict the field performance of the LCA device. These tests provide an accelerated means of emulating long-term system operation in severe field environments. From the performance of the devices during these tests, AMD predicts actual field performance.

AMD performs qualification testing of nonhermetic devices to demonstrate the reliability, both of the device die and of the materials and methods used in assembling the device. AMD testing methods are derived from, and patterned after, those specified in MIL-STD-883. However, AMD does not intend the MIL-STD-883 reference to imply that nonhermetic products comply with those requirements.

These test methods are recognized industry-wide as stringent tests of reliability. The tests are commonly used for non-military-grade semiconductor devices, as well as for fully compliant military-grade products.

The following discussion describes the various test procedures used. The summary shows the qualification test data for the 2064/2018 devices. Upon request, **AMD's Quality Assurance and Reliability Department** will make updated summaries available.

9.2 TEST PROCEDURES

The following table summarizes the reliability testing sequence for nonhermetic logic cell array device die-package qualification.

Name of Test	Test Conditions	Results/Parameters
1. High-Temperature Life	•1000 hr min. equivalent at temperature = 125° C •Max, rated operating voltage •Life test circuit equivalent to MIL-STD-883	LTPD = 5, S = 77, C = 1
2. Biased Moisture Life	•1000 hr min. exposure •T= 85° C, RH = 85% •Max. rated operating voltage. •Biased-moisture life circuit equivalent to MIL-STD-883	LTPD = 5, S = 77, C = 1
3. Unbiased Pressure Pot	•168 hr minimum exposure •T = 121° C, P = 2 atm H_20 sat	LTPD = 5, S = 77, C = 1
4. Temperature-Cycling	•MIL-STD-883, Method 1010, Cond. C •-65° C to +150° C •100 cycles	LTPD = 5, S = 77, C = 1
5. Resistance to Solvents	MIL-STD-883, Method 2015	S = 8, C = 0
6. Solderability	MIL-STD-883, Method 2003	S = 15, C = 0
7. Lead Integrity	MIL-STD-883, Method 2004	S = 15, C = 0

Notes: Lot Tolerance Percent Defective (LTPD); Minimum Sample Size (S); Maximum Acceptable Failures (C)

9.2.1 DIE QUALIFICATION

AMD performs a High-Temperature Life Test and a Biased-Moisture Life Test to check the long-term operating characteristics of the LCA die.

9.2.1.1 High-Temperature Life Test

AMD performs the High-Temperature Life Test to evaluate the long-term reliability and life characteristics of the die. It is defined by the Military Standard (from which it is derived) as a **die-related test**, and it is contained in the Group C Quality Conformance Test. Because of the time-acceleration factor induced by higher-temperature testing, AMD can accumulate, in a reasonable period of time, data representing a large number of equivalent hours at a normal temperature of 70° C.

9.2.1.2 Biased Moisture Life Test

AMD performs the Biased-Moisture Life Test to evaluate the reliability of the die under conditions of long-term exposure to severe, high-moisture environments that could cause corrosion. Although it clearly stresses the package as well, this test is typically grouped under the category of die-related tests. AMD operates the device at 5.0 VDC, and exposes it to 85° C and 85% relative humidity throughout the test.

9.2.2 PACKAGE INTEGRITY AND ASSEMBLY QUALIFICATION

AMD performs the following tests to check the quality and integrity of both the package and the assembly.

- Unbiased Pressure-Pot Test
- Temperature Cycling Test
- Resistance-to-Solvents Test
- Solderability Test
- Lead-Integrity Test

9.2.2.1 Unbiased Pressure-Pot Test

AMD performs the Unbiased Pressure-Pot Test at 21° C and two atmospheres of saturated steam to evaluate the ability of the plastic encapsulating material to resist water vapor. Moisture penetrating the package could induce corrosion of the bonding wires and bonding pads of the LCA die. Under extreme conditions, moisture could also cause drive-in and corrosion under the bonding pads. Although it is difficult to correlate this test to actual field conditions, it provides a well-established method for comparison of plastic packaging materials and assembly and molding techniques.

9.2.2.2 Temperature-Cycling Test

AMD performs the Temperature-Cycling Test to evaluate the long-term resistance of the package to damage from alternate exposure to extremes of temperature or from intermittent operation at very low temperature. The temperature range for the test is –65° C to +150° C. The transition time is longer than that in the Thermal Shock test but the test is conducted for many more cycles.

9.2.2.3 Resistance to Solvents Test

AMD performs the Resistance to Solvents Test to evaluate the integrity of the package marking during exposure to a variety of solvents. This is an important test because an increasing number of board-level assemblies are subjected to the severe conditions of automated cleaning before system assembly. AMD performs this test according to MIL-STD-883, Method 2015.

9.2.2.4 Solderability Test

The Solderability Test evaluates the solderability of the leads under conditions of low-soldering temperature following exposure to the aging effects of water vapor.

9.2.2.5 Lead Integrity Test

The Lead Integrity Test is performed to evaluate integrity of the package leads. AMD performs this test according to MIL-STD-883, Method 2004.

A summary follows.

9.3 SUMMARY

The following test data summarizes the performance of LCA devices during AMD's qualification tests. These test results demonstrate the reliability and expected long life of the AMD nonhermetic product line. This continuing series is part of the AMD reliability testing of nonhermetic devices.

Device Type: 2064/2018			Process/Technology: 1.5 and 2.0 Micron Double-Layer Metal CMOS			
Die Attach Method : Silver Epoxy			Package Type: 68 lead PLCC			
Molding Compound: Sumitomo 6300K, H			Date: 2Q, 1987			
1. High-Temperature Life Test T=125° C	Combined Sample 850	Failures 4	Equivalent Mean hrs/Device at T=125° C 2332	Equivalent Device hrs at T=125° C 2033967	Equivalent Failure Rate in %/1000hrs at T=125° C 0.19	Equivalent Failure Rate in %/1000 hrs at T=70° C 0.0027
2. Biased-Moisture Life Test T = 85° C; RH = 85%	Combined Sample 486	Failures 2	Mean hrs per Device 1000	Total Device hrs 486000		
3. Unbiased Pressure-Pot Test +121° C, 2 atm sat. steam	Combined Sample 488	Failures 1	Mean hrs per Device 139	Total Mean hrs 67832		
4. Temperature Cycling Test -65° C + 125° C, 1000 cy. (min)	Combined Sample 233	Failures 1	Mean Cycles per Device 812	Total Device Cycles 189196		
5. Temperature Cycle Test 0° C to + 150° C Method 1009,Cond. A	Combined Sample 50	Failures 0	Mean hrs per Device 24	Total Device hrs 1200		
6. Resistance-to-Solvents Test MIL-STD-883, Method 2105	Combined Sample 16	Failures 0				
7. Solderability Test MIL-STD-883, Method 2003	Combined Sample 30	Failures 0				

GLOSSARY

Active Edge — A low-to-high or high-to-low signal transition that initiates an action.

Active High — A high-voltage active sense.

Active Low — A low-voltage active sense.

Active Pod — The pod with setup information that is displayed on XACTOR's screen. The LED lights to indicate it is active.

Active Sense — The voltage level, either high or low, associated with the active state, which is a logical 1 or 0.

Adder — A digital circuit that adds numbers.

Analysis Tool — Software that automatically or interactively determines circuit characteristics.

AND — A logic function that is true if all variables are true, false if at least one variable is false.

Application Specific Integrated Circuit — ICs that are at least partially tailored for a particular application. Includes field programmable devices such as PALs and LCA devices, and factory programmable devices such as gate arrays, standard cells, and completely customized ICs.

APR — See Automatic Placement and Routing.

ASCII — The American Standard Code for Information Interchange is an eight-bit code used for sending information over a data line.

ASIC

See Application Specific Integrated Circuit.

Assert

To cause a signal to change from its inactive to its active state.

Assertions

In timing analysis, designer-provided information that describes certain timing relationships between signals in a portion of the circuit.

Asynchronous Logic

Logic that is not clocked by, or synchronized with, the system clock.

Automatic Placement and Routing

A software tool that automatically positions logic blocks and calculates wiring paths or signal interconnections during circuit design.

Back Annotation

The process of attributing delay times to a design after it has been partitioned. This is only valid if the partitioned design is equivalent to the original schematic design.

Base Configuration

The arrangement of logic within a CLB. The base configuration can be either two functions of up to three variables each, one function of up to four variables, or two functions of up to three variables, multiplexed by the B input.

Bidirectional

A circuit element, or primitive, that passes signals or data in either direction. This contrasts with the more traditional unidirectional design elements, which only pass input signals from input ports to output ports.

Bit

One unit of binary data. A logical 1 or 0.

Bit Stream

The configuration data required by an LCA device to determine the functions of CLBs, IOBs, and interconnections.

Block

A CLB, IOB, clock buffer, or oscillator.

Block Editor

Part of the LCA design editor that performs commands to configure CLBs and IOBs. See LCA Design Editor.

Bottom-up Hierarchical Design

A structured approach to organizing circuit design data in which the designer begins with a definition of the lowest primitives, or elements, then proceeds to build higher-level functions using those elements. As the hierarchical definition process concludes, the designer expresses the overall circuit structure in terms of high-level functional blocks and their interconnections.

Buffer

An amplifier that increases drive capability and temporarily stores digital data.

Byte

Eight bits.

Cerdip

The ceramic IC dual in-line package that uses a frit-sealed process.

Circuit

A combination of electrical and electronic components that perform specific functions.

Circuit Simulation

A software breadboard to verify design functionality and performance. Software that logically emulates a circuit's functions to ensure proper design.

CLB

See Configurable Logic Block.

Clear

To force to a logical 0. See Reset.

Clock

A register whose content changes at regular intervals, generating periodic signals for synchronization and causing one logic decision to be made for each signal transition.

CMOS

See Complimentary Metal Oxide Semiconductor.

Combinational Logic Gate

A combination of gate networks that generates a specific function and has no storage capability.

Command File

A text file consisting of XACT LCA Editor commands and run by the EXECUTE command.

Comparator

A digital device that compares the magnitudes of two digital quantities and indicates their relationship.

Complimentary Metal Oxide Semiconductor

A widely-used IC technology with low-power consumption, high noise immunity, and moderate speed.

Configurable Logic Block

An LCA sub-unit that contains configurable combinational-logic and data-storage circuitry.

Configuration Data

The data required by an LCA device to determine the functions of CLBs, IOBs, and interconnections.

Congestion

A high concentration of interconnections routed in one area.

Control Signals

Signals that control functional elements through the data signals in a digital circuit.

Controllability

The degree to which signals in a part of a circuit can be made to take on specific values through manipulation of primary inputs; used in testability analysis.

Counter

A digital circuit capable of counting electronic events, such as pulses, by progressing through a sequence of binary states.

Critical Path Evaluation

The identification and analysis of signal paths whose delays could limit the speed of the circuit.

Current Block

The selected CLB or IOB to which all Block Editor commands apply until another block is selected. See Block Editor.

Current Design File

The design file that was designated by the user to be loaded when invoking an XACT program. See Design File.

Cursor

A screen symbol controlled by the mouse. Used for selecting and positioning menu items and objects on the screen. See Selection.

D Flip-Flop

A type of bistable multivibrator in which the output follows the state of the D input. See Flip-Flop.

Daisy Chain

Several devices connected in such a way that configuration data move serially from one device to the next.

DBK File

A temporary file that stores all commands entered during an (LCA Editor) editing session. After the session, the temporary file is saved with the extension .DBK. See Log File.

De-assert

To cause a signal to change from its active to its inactive state.

Decoder

A digital circuit that converts coded information into a recognizable form.

Default Value

The value used when none is specified by the user.

Delay

The timing interval between the occurrence of an event at one point in a circuit and the corresponding occurrence of a related event at another point.

Design Analysis

A set of tools and techniques used to study the operations of a circuit to determine, among other things, how a particular implementation compares to the design specification.

Design Approach

A method used to construct a system design. Possibilities include LCA devices, PALs, gate arrays, standard cells, silicon compilers, full custom ICs, and standard products.

Design Cycle

The sequence of phases through which a circuit design evolves. During each phase, the design is further developed by a combination of analysis, synthesis, and verification techniques, to create a more detailed representation of the circuit.

Design File

A file containing information about a specific LCA configuration. Design files contain the LCA suffix in their names.

Design for Testability

A design process to ensure that a product can be thoroughly tested with minimum effort, and that the test results are reliable.

Design Methodology

An approach or technique that solves certain circuit design problems.

Design Productivity

A measure of the rate at which circuit designs are produced, usually expressed in terms of design cost.

Design Rule Checking

Part of the design process that checks for conformance to electrical rules.

Design Rules

Circuit development guidelines that focus upon conformance to basic electrical engineering and logic design principles.

Design Synthesis

A design discipline that fills in the detail of an abstract design; for example, creating a Boolean equation.

Design Verification

The process of confirming or validating that the design meets the required specifications. There are many ways to verify a design, ranging from formal proofs or in-circuit emulation, to informal, tailored methods. The most common verification method is simulation.

Design Verification Test Data

Sequences of input stimuli applied to a design to determine that the circuit functions as intended.

DFT

See Design for Testability.

Dice

Multiple die. See Die.

Die

A tiny piece of the silicon wafer that, when packaged, becomes an IC.

Die Size

The rectangular area of silicon, the measurement of which is expressed in square microns or mils.

DIP

See Dual In-line Package.

Direct Interconnection

Dedicated connection of adjacent CLBs, IOBs, and outputs. It provides the shortest propagation delay and uses minimal interconnection networks.

DMA transfer

A method of accelerated loading of configuration information into multiple LCA devices by means of direct memory access hardware.

DRC

See Design Rule Checking.

Dual In-line Package

Type of packaging widely used for IC chip assembly. It has two parallel rows of connection pins, usually on 0.100 inch (100 mil) centers.

Edge-Triggered Flip-Flop

A flip-flop for which the input and output data appear on the same clock edge. See Flip-Flop.

EEPROM (E^2 PROM)

See Electrically Erasable and Programmable ROM.

Electrically Erasable and Programmable ROM

Similar to an EPROM, but it stores the charge on a floating gate. Newer EEPROMs can erase individual data bytes.

Electrostatic Discharge

The neutralizing action of two oppositely charged materials.

Enable

Allows a circuit to respond to an input. For example, a clock enable signal lets a circuit respond to its clock input.

EPLD — Erasable Programmable Logic Device, such as an EPROM.

EPROM — See Erasable Programmable ROM.

Encoder — A digital circuit that converts data into coded format.

Erasable Programmable ROM — Usually refers to the UV erasable, 2764 device type. Generally, EPROMs are erased by shining ultraviolet light on the chip through a quartz window on the package.

ESD — See Electrostatic Discharge.

Exclusive-OR — A logic function that is true if one of the variables is true and the other is false.

Executive — The XACT software that sets initial values for, and invokes all other, XACT software modules.

Fall Time — The time interval between the 10 percent and the 90 percent points on the negative-going edge of a pulse.

Fanout — The number of equivalent gate inputs, or unit leads, that a logic gate can drive.

Fault Grading — The process of measuring a test vector's effectiveness in locating defects in an IC.

Field Programmable Logic — Standard products that the user can configure to a specific application, such as PALs, FPLAs, and LCA devices.

Flip-Flop — A digital circuit used to store one bit of information, either 0 or 1. Also known as a toggle.

Floppy Disk — A magnetic storage device. A flexible mylar disk.

Full Adder — A digital circuit that adds two binary digits and an input carry to produce a sum and an output carry.

Function — The specific purpose of an entity or its characteristic action.

Function Key — A programmable key on the IBM PC keyboard. Typically, the software defines the function.

Functional Primitives — Design building blocks such as adders, shifters, decoders, and memory. A functional primitive differs from a gate-level primitive only in the individual element's degree of functional complexity.

Gate — A digital logic element. The binary value of the output depends on the values of the inputs according to some logic rule.

Gate Array — A digital IC with a configuration of uncommitted elements that are interconnected by one or more routing layers.

General Purpose Interconnections — Horizontal and vertical metal segments joining LCA switching matrices.

Glitch — An unwanted voltage or current spike, typically of short duration.

Half Adder — A digital circuit that adds two bits to produce a sum and an output carry. It cannot accommodate input carries.

Hard Macro — A macro with a predetermined physical layout.

Hold Time — The amount of time, after the clock transisiton, that the data input to a flip-flop must be stable for proper operation.

In-Circuit Emulation — A design system that uses software and hardware to test and debug LCA devices in a target system.

Initial Value — The preset value for an option when the software is invoked. After the software starts, the option's value can be changed.

Initialize

The process of establishing an initial condition or starting state. For example, setting logic elements in a digital circuit, or the contents of a storage location, to known states so that subsequent application of digital test patterns drive the logic elements to another known state. Initialization sets counters, switches, and addresses to zero or other starting values at the beginning of, or at prescribed points in, a computer routine.

Input Line

A line on the XACT screen that displays the characters typed at the keyboard. Pressing the <Return> key signals that the content of the input line is complete.

Input Loading

The number of unit loads that one output can drive.

Input Stimuli

The parameters in a software program used to test a circuit or simulated circuit.

Input/Output Block

An IOB is a sub-unit that can be configured to connect the LCA device's internal circuitry to external pins.

Instance Name

A name assigned to each occurrence of a macro. The instance name is used to create unique net and block names. See Macro.

Interactive Simulator

A circuit-simulation tool that lets the designer halt circuit emulation at any point. This allows detailed investigations of localized signal relationships, changes to simulated input signals, or changes to simulator directives.

Interconnection

A sub-unit of an LCA device that can be configured to connect two or more internal points.

Inverter

A logic device that complements a logic variable.

Invoke

To load and run software on a computer. This action includes the setting of initial values.

IOB

See Input/Output Block.

J-K Flip-Flop

A flip-flop that operates in the set, reset, no change, and toggle modes. See Flip-Flop.

Johnson Counter

A state machine based on shift registers using the absolute minimum amount of feedback. The output of the final register in the chain is inverted and fed into the least significant register in the chain.

Karnaugh Map

An arrangement of cells representing combinations of variables in a Boolean expression. Used for a systematic simplification of the expression.

Large Scale Integration

An IC equivalent of 100 to 10,000 logic gates.

Latch

A logic device that transfers the data of its input to its output when load enable is active; it retains its value when load enable is inactive.

Latchup

A condition in which the output of a circuit has become fixed near one of two voltage extremes. The circuit no longer reacts to changes in the input signal. This is a common problem with CMOS gates.

Layout

That portion of the design cycle during which the logical or electrical definition of the circuit is transformed, via placement and routing, into a physical equivalent that can be manufactured.

LCA

See Logic Cell Array.

LCA Design Editor

The XACT software used to create and edit the LCA design file.

LCC

See Leadless Chip Carrier.

Leadless Chip Carrier

A high-density IC package with J-bends suitable for socket or surface mounting.

Length Count

Part of the configuration software that specifies the number of Configuration Clock (CCLK) cycles from the start of configuration to the start of operation.

LFSR

See Linear Feedback Shift Register.

Library

A collection of predefined elements, primitives, macros, and larger functional blocks.

Linear Feedback Shift Register

A synchronous counter. The input bit is computed by XORing several bits of the current state.

Load Pin

The pin to which signals flow. See Source Pin.

Log File

A file containing all commands entered during an editing (LCA Editor) session. Each time an LCA design file is edited, a new log file replaces the existing one. Log files have the extension .LOG.

Logic Cell Array

A device that can be configured or programmed to perform a range of functions. LCA devices may replace circuits that range in size from SSI to VLSI devices.

Logic Element

A sub-unit of a CLB that contains combinational-logic circuitry.

Logic Simulation

The process of building and functionally validating a digital circuit model on a computer. A logic simulator computes signal values as a function of time in an arbitrary circuit, given the initial state and input sequence to the circuit.

Logical Primitive

Building blocks such as a NAND, NOR, inverter, or SSI/MSI TTL package.

Long Line

An interconnection that routes a signal to several destinations in an LCA device. This type of interconnection runs the length or width of an LCA device and is often used for clock signals.

LSI

See Large Scale Integration.

Macro

Explicitly identifiable portions of a design that can be described as entities. Design elements may be primitives, interconnections of primitives, or interconnections of larger elements.

Macro Library

A collection of macros provided by a vendor and used as building blocks in chip design. Organized or classified by function, macro libraries consist of files stored in directories. See Macro.

Master Mode

A configuration mode in which configuration data is transmitted in parallel, byte-wide form from an external memory to the LCA device. The LCA device generates read-addresses and automatically serializes the data for internal storage.

MB/Mb

Megabyte/megabit.

Medium Scale Integration

Traditional, standard, digital logic components having approximately 10 to 100 functional logic gate equivalents.

Menu

A list of items that appears on the display. Each item can be input to the computer by selecting it with the mouse. See Selection.

Message Line

The display line on an active workstation through which commands can be entered and on which status messages or prompts appear.

Metastability

Metastability is defined as an output state between a valid logic HIGH and LOW for any digital device. It can occur for registers or latches when certain parameters, such as data set up-and-hold times, are violated.

Micron (μ)

One micrometer; one millionth of a meter. The basic unit of measure for VLSI geometries.

MS-DOS A common IBM-PC operating system. The LCA development system runs under MS-DOS.

MSI See Medium Scale Integration.

Multiplex To put information from several sources onto a single line or transmission path.

Multiplexer A digital circuit capable of multiplexing digital data.

Mux Abbreviation for multiplexer.

NAND A logic function that is true if at least one variable is false or false if all variables are true. This is equivalent to an inverted AND function; i.e., Not AND.

Net A collection of nodes that are interconnected by wires. Also known as a signal network or signal net.

Netlist A list of circuit elements and their interconnections.

Netlist Formatting Extracting netlist data from a database and putting it into a format that can be used by another software tool.

Nibble Four bits, or half a byte.

Node An identifiable point in a design that must be electrically connected to other nodes by wires. A node can be associated with a specific device or geographic location.

Nominal Delay The mean time signals take to propagate through a logic element or a wire. The effect of an input change to an element on the output does not occur until after the nominal delay.

Non-Recurring Engineering Costs Costs charged to the user for development of tooling and other services during semicustom chip design and production.

Non-Volatile — Memory, such as PROMs and magnetic bubble memorie modules that retain data even after power-off.

NOR — A logic function that is true if all variables are true, false if at least one variable is false. This is equivalent to an inverted OR function; that is, Not OR.

NRE — See Non-Recurring Engineering costs.

Observability — In testability analysis, a measure of the degree to which the operation of signals in a certain part of a circuit can be monitored.

OR — A logic function that is true if at least one variable is true, false if all variables are false.

Pad — An area on the chip used as a contact point by the IC's package pins.

Peripheral Mode — A configuration mode in which the LCA device acts as a peripheral device. An external device, such as a processor, that loads the configuration data bits serially into the LCA device.

PGA — See Pin Grid Array. Also ascribed to programmable gate array.

Physical Interconnection Editor — The LCA Editor that configures interconnections and the locations of configured blocks. See LCA Editor.

PIE — See Physical Interconnect Editor.

Pin — The point on a device at which an electrical connection can be made.

Pin Grid Array — A high pin density ceramic package.

PIP — See Programmable Interconnection Point.

PLA — See Programmable Logic Array.

Plastic Leader Chip Carrier

An IC package with J-bend leads suitable for socket or surface mounting.

PLCC

See Plastic Leaded Chip Carrier.

Power-Down State

An idle current condition when the LCA device's power supply requirement can be reduced to a minimal level. Under this condition, circuit activity is suspended and all configuration data are preserved.

Preamble Nibble

A specific series of four data bits that signals the start of the configuration data for LCA devices.

Primitive

The basic building block of design entry. Primitives are not expanded during design entry but are used as single blocks. There are several levels of primitives. Basic systems only use combinational gates and Boolean operators. More complex systems include sequential elements such as latches, flip-flops, delay lines, and monostables in their primitive sets. Some systems include higher-level functions such as counters, shift registers, ROM and RAM as primitives.

Product-of-Sums

A Boolean expression that is the ANDing of ORed terms.

Profile

A set of commands stored in a file with a .PRO extension to define options that configure the PC environment.

Programmable Interconnection Point

A configurable connection between interconnection segments.

Programmable I/O

Buffers that the user can configure as input, output, or both.

Programmable Logic Array

A rectangular array of AND and OR gates used to generate a group of control functions in sum-of-products form. See also LCA.

Programmable Read-Only Memory

A ROM that can be programmed by the customer.

PROM

See Programmable Read-Only Memory.

Propagation Delay

The time difference between the change of an input signal, or clock, and the change of the output.

P-SILOS

A software breadboard that tests a hardware system to verify functionality before the actual hardware is programmed.

Pull-Down Menu

A menu that appears when you select its name using the device that drives the cursor. The menu usually overlaps other elements on the screen.

Race

In asynchronous digital logic, the concurrent change of two or more feedback lines. If the circuit's final state does not depend on the order in which the variables change, the race is noncritical; otherwise it is critical.

RAM

Random access memory.

Read

The process of retrieving information.

Register

A digital circuit capable of storing, or moving through, shifting binary information. Usually used as a temporary storage device.

Reset

The state of a flip-flop, register, or counter when only 0s are stored. Equivalent to the CLEAR function.

Ripple Counter

A digital circuit made up of a series of flip-flops in which the contents are continuously recirculated.

Rise Time

The time required for the pulse's positive-going edge to go from 10 percent of its full value to 90 percent of its full value.

Rise/Fall Delay

A circuit simulation feature that lets different output rise (0 to 1) and fall (1 to 0) times to be specified. This more closely approximates the actual timing parameters of a device, as compared to nominal delay models that assume equal rise and fall times.

ROM

Read only memory.

Route

To configure the interconnection between pins.

Routing

The process of configuring interconnections between pins.

Scan Test

A test method whereby data is shifted into SRLs and results are shifted out. Additional circuitry typically must be included in the design to support this type of testing.

Schematic Capture

The process of entering a circuit diagram into a computer system using a CAE editor.

Schematic Diagram

A circuit diagram in which components are represented by standard, simple, easily drawn symbols.

Schmitt Trigger

A circuit with snap action that produces a sharp, single transition from slowly changing inputs. Positive feedback shifts the threshold as soon as it is first reached. If the signal is diminished due to noise, the output remains steady.

Schottky TTL

An improved version of TTL logic in which saturation of the transistors is prevented and impedance of the output circuit is reduced. Thus, 74S logic gates have a maximum delay of 7 ns compared to 15 ns for TTL without Schottky. Power dissipation, however, is about twice that of TTL. Low-power Schottky TTL, or 74LS series, has a maximum delay of 28 ns and requires 80% less power than a normal TTL device.

Selection

The action of moving the cursor to a displayed item and pressing the mouse button. The selected item is received by the computer as input.

Semicustom IC Setup Information

The information that configures the XACTOR environment. Setup information may be stored in a file for use at any time for a given XACTOR environment.

Set

To force one or more storage elements to logical 1.

Setup Time

The amount of time, before the clock transition, that the data input to a flip-flop must be stable for proper operation.

Shift Register

A linear chain of storage elements that shifts the contents of the storage elements one position for each clock transition.

Simulation

The use of computer tools that imitate a proposed circuit design to validate its functionality. Timing and performance considerations are ignored during functional simulation.

Simulator

A software breadboard that tests a hardware system to verify functionality before the actual hardware is programmed.

Slave Mode

A configuration mode in which the LCA device receives serial configuration data and all control signals from another device, typically another LCA device.

Small Scale Integration

Refers to traditional, standard, digital logic components having one to approximately ten logic gate equivalents.

Soft Error

Refers to dropping or picking up bits in a non-repeatable manner. Often caused by alpha particles.

Soft Macro

A macro with no predetermined physical layout.

Source Pin

A pin from which a signal originates. See Load Pin.

Specification (Spec) — A document defining the ground rules for circuit design. It contains requirements, such as functionality, performance, cost, temperature, and interface.

Spike — The output condition where the inputs are being manipulated faster than the element's propagation delay.

SR flip-flop — Set-reset flip-flop. See Flip-Flop.

SRL — Shift Register Latch. See Shift Register and Latch.

SSI — See Small Scale Integration.

Standard Cell — A predefined mask-level design of a logic function or simple functional element. A library of such standard cells usually has a common height and a common power distribution protocol. Used in the design and layout of standard cell semicustom ICs.

Standard Cell Semicustom IC — Very large scale integrated circuits that are laid out using predefined mask level definitions of standard logic function cells. The standard cells are placed in rows or columns so that they share power rails and have open routing channels between adjacent row/column pairs.

Standby Mode — Also called power-down. A mode whereby volatile memory can retain data at a voltage level lower than normal operation.

State — The condition of one flip-flop or a set of flip-flops.

State Machine — A set of flip-flops for which the next state and the outputs are functions of its current state and a set of inputs.

Stimulus — Any physical or electrical input applied to a device intended to produce a measurable response.

Storage Element — A sub-unit of a block that can store data.

Sum-of-Products — A form of Boolean expression that is the ORing of ANDed terms.

Switching Matrix — An intersection of three or more segments of local interconnections. Two segments are connected by one switching-matrix connection.

Switching Time — A signal output's delay time from a defined HIGH to a defined LOW, or vice versa. This delay is calculated as the time between the specified reference points on the voltage waveforms.

Synchronous Logic — Clocked logic in which all logical signal changes are keyed to clock transitions.

T Flip-Flop — A type of flip-flop that toggles or changes state on each clock pulse. See Flip-Flop.

Tag — A user-definable label or flag that can be applied to sub-units of an LCA device.

TDM — See Time Division Multiplexing.

Test — A procedure or action that determines, under real or simulated conditions, the capabilities, limitations, characteristics, effectiveness, reliability, or suitability of a material, device, system, or method.

Test Vector — An array of inputs used to check for defects in an circuit.

Testability — A design characteristic that reliably determines in a timely fashion the status of a system or of any of its subsystems, i.e. operable, inoperable or degraded. One measure of the quality of a circuit design is the relative ease with which it may be tested.

Testability Analysis — A tool to measure the extent to which a design is testable.

Testability Measure

Attributes such as controllability and observability, which give a quantitative measure of the extent to which a design is testable.

Three-State

A mode in which the circuit functions as an output, input, or no connection (high impedance). Also called Tristate.

Time Division Multiplexing

Sending several signals during different time slots over the same channel.

Timing Analysis

Evaluation of the behavior of a circuit design that takes into account signal delay time but ignores logical functionality. Timing analysis determines whether signals arrive at their intended destinations in time.

Timing Verification

A tool for checking that the various rules, such as clock skews, are met and that the circuit functions at the required speed.

Toggle

To change to the opposite state.

Toggle rate

The maximum clock frequency at which a flip-flop storage element will toggle properly.

Top-Down Hierarchical Design

A design method by which increasing detail is added as the design progresses. Top-down is typically used at the beginning of the design cycle.

Transistor-Transistor Logic

A logic family that uses multitransistor emitters to perform logic functions. The gates have a 15 ns maximum delay and require +5 V power of about 2 mA per gate. Logic levels are normally +0.2 and +3.4 V, and input threshold is nominally 1.2 V.

TTL

See Transistor-Transistor Logic.

Unit Delay

A simulation technique used to verify functionality. In this technique all the delays of the elements are set to one time unit.

Unit Delay Simulation

A simulation that assumes equal propagation delays.

Unit Load

Under specific conditions, a measure of impedance presented to an input or output. One gate input represents a unit load to a gate output within the same logic family.

Unknown Level

A simulated signal value (or an electrical net) that is not a one or a zero. This could be caused by an error or by the node not being initialized.

Unknown State

Most memory elements used in sequential circuits are bistable devices. When the power is turned on, they are normally designed to have symmetrical configuration; i.e., the initial state becomes unpredictable. Such memory devices begin operation with internal memory signal valuation designated as unknown. Unknown states can also be encountered during circuit operation as the result of critical races, oscillations, or ambiguous delays.

VDD

The most negative power supply voltage level in an MOS circuit.

Verification

See Design Verification.

Very Large Scale Integration

A circuit with more than 10,000 transistors.

VLSI

See Very Large Scale Integration.

Volatile

The attribute of losing stored information when power is removed from a memory device.

VSS

The most positive power supply voltage level in a MOS circuit.

Waveform Input

The input data for a simulated circuit respresented as waveforms.

Wildcard — A symbol that can represent any character or characters.

Window — A user-specified part of the LCA device that is displayed on the LCA Editor screen and that can be manipulated.

Wired-AND — An arrangement of logic circuits in which the gate outputs are physically connected to form an "implied" AND function.

Wired-OR — Logic gates wired to implicitly create a logical OR function.

Wiring Congestion — See Congestion.

Workstation — A single-user, stand-alone work aid. In the engineering community, it consists of a microprocessor-based system with a high-resolution screen and software to carry out design functions. Popular workstation vendors include OrCAD, Daisy, FutureNet, Mentor, and Valid.

World View — The entire LCA design that can be displayed on the LCA Editor screen. See Window.

Write — The process of storing information in memory.

XACT — Software that lets you specify the configuration of an LCA device using interactive computer graphics.

XACTOR — A software/hardware package for performing in-circuit emulation.

Zero Delay Simulator — A simulator used for functional validation. The signals have no delay.

Zero/Unit Delay Simulator — Combination of the zero delay and unit delay simulation concepts. A circuit is modeled with zero delay elements and with unit delay circuits inserted into the feedback lines and memory elements.

INDEX

D

E

F

G

H

I

M

N

O

P

Q

R

S

T

U

V

W

X-Y-Z

Advanced
Micro
Devices

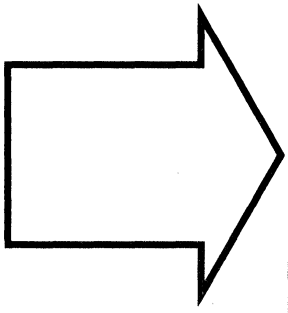


Table of Contents Section 3
Applications

Advanced
Micro
Devices

AN-182

Configuring the LCA Device

By Ed Valleau

3

Logic Cell™ Array is a trademark of XILINX Inc.

Configuring the LCA Device

By Ed Valleau

Introduction

The Logic Cell™ Array device from Monolithic Memories is a new revolution in programmable logic. It offers the user gate array logic density while still providing the versatility of a programmable logic device (PLD). Its architecture is based on an array of static RAM cells, which must be configured when power is initially applied to the circuit. This gives the LCA device a high degree of flexibility since the functional logic contained in the device may be changed during development, thus shortening prototyping cycles. To update existing equipment, logic circuits may be modified with new configuration data. It is also possible to reconfigure the LCA device "on the fly" without the need for hardware changes. This would prove specifically useful in systems that need to change their protocols of operation dynamically.

Configuration data is stored in a nonvolatile source and read by the LCA device at "power up". The time it takes to perform a configuration is variable, from 12 to 34 ms depending on device type and mode of configuration. A typical storage medium for the LCA's configuration data would be an EPROM or EEPROM, which would be connected to the the LCA and accessed immediately after "power up". When the configuration process is complete and the device is fully functional, the EPROM or EEPROM may be put into a three-state condition and left deselected until the next configuration cycle.

There are a number of different modes of LCA device configuration, each mode being better suited to a particular operating environment or application than another mode. The following information outlines these various modes.

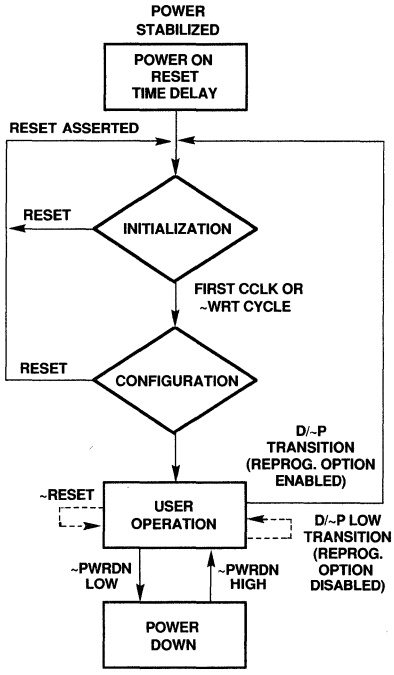


Figure 1. Configuration Sequence

Overview of the Configuration Process

The LCA device is comprised of three distinct programmable elements: Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs) and Programmable Interconnect. Because the device architecture is based on static RAM cells, the configuration of the CLBs, IOBs and Programmable Interconnect is random and indeterminate when power is first applied to the device. The LCA device goes through two states prior to being functionally operational. The first is a general initialization state, followed by a configuration sequence. (See Figure 1.)

The device uses a number of dedicated input/output pins to control the loading process. Configuration sequentially programs the RAM cells, ultimately creating functionally operational and interconnected logic blocks. The entire procedure is comparable to programming a PLD except that a conventional PLD is nonvolatile, and maintains its logic functionality when power is removed from the circuit.

The smaller of the two LCA devices, the M2064, is a 64-CLB device which requires exactly 12,040 binary bits of information to complete the configuration process. The M2018 is a 100-CLB device which needs 17,880 bits of information.

Table 1 shows the different modes that can be selected via the three dedicated mode input pins M0, M1 and M2. These will usually be hardwired to select a single mode. The LCA device reads the digital code applied to the mode pins prior to configuration, then enters one of five specific modes before becoming a functional logic system. The exact number of bits for successful configuration must be read into the LCA device, partial configuration is not possible.

CONFIGURATION MODE	SLAVE MODE	PERIPHERAL MODE	MASTER-HIGH MODE MASTER-LOW MODE	MASTER SERIAL MODE
Mode Selection code (M0:M1:M2)	1:1:1	1:0:1	0:0:1 (Master-Low) 0:1:1 (Master-High	0:0:0
Configuration data	Bit-serial	Bit-serial	Byte-parallel	Bit-serial
Automatic loading	No	No	Yes	Yes
Programming source	User Logic or Another LCA (Note 2)	CPU Data Bus Memory	External Byte-wide Memory	External Serial Memory
Number of user I/O pins required	2	6	25	3
Configuration time	Source Dependent (Note 1)	Source Dependent (Note 1)	12-24 ms (M2064) 17-34 ms (M2018) (Note 3)	12-24 ms (M2064) 17-34 ms (M2018) (Note 3)

Notes:

1. The minimum time in any case is approximately 12 ms for the M2064 and 17 ms for the M2018.
2. Also used by Monolithic Memories' XACTOR for In-Circuit Emulation.
3. This parameter depends on internal timing circuits and is manufacturing process-dependent. Therefore it may vary from device to device within the limits shown.

Table 1. Comparison of Configuration Modes

Choice of Configuration Mode

The choice of a configuration mode is influenced by the actual operating environment. For example, questions that might arise are: is the LCA being used as a standalone logic unit, or with a microprocessor? Can it be configured by a serial link? Other considerations include whether pins used for configuration are also used for functional operation, and whether or not these pins should be isolated from activating external logic during configuration. The designer has a choice of configuration modes for multiple LCA designs, and can use either parallel or serial support for a master mode LCA device which in turn can configure a slave or number of slave LCA devices.

The Configuration Pins

The pins used to configure the LCA device come in two forms: dedicated configuration pins which are used exclusively for configuring the LCA device, and multifunction configuration pins which can be used as general-purpose I/Os after configuration has been completed.

The Six Dedicated Configuration Pins

PIN	DESCRIPTION
M0,M1	Mode Select Pins
CCLK	Configuration Clock
~Reset	Master Reset
D/~P	Done/~Program
~PWRDN	Power Down

The Multi-Function Configuration Pins

PIN	DESCRIPTION	USAGE
M2	Mode Select	Present in all configuration modes
Din	Configuration Data In	
Dout	Configuration Data Out	
HDC	High During Configuration	
LDC	Low During Configuration	
~CS0,~CS1,CS2	Chip Selects	Peripheral Mode Only
~Wrt	Write Strobe	
~RCLK	Read Strobe	Master Mode Only
A0-A15	Address Lines	
D0-D7	Input Data Lines	

Pin Names and Functional Description

Done/Program Pin D/~P

The Done/~Program D/~P pin performs a dual function. It is an open-drain output with an internal programmable pull-up resistor. During configuration the pin is an output that is driven LOW by the LCA and can be monitored by external logic to determine whether or not the LCA is ready for functional use. When configuration is completed this pin is pulled HIGH by an internal programmable pull-up resistor of 3KΩ value. In a multi-LCA environment, the D/~P pin goes HIGH one clock cycle before the configuration is complete allowing time for user I/O signals to propagate between other LCA devices before entering the functional mode of operation. The LCA may be reconfigured at any time by pulling the D/~P input LOW with an external logic open-drain (open-collector) driver. When the internal logic recognizes a LOW input it drives the D/~P output LOW forcing a reconfiguration cycle.

There is a time delay of several microseconds before an active LOW input is recognized, so random triggering due to short noise bursts is highly unlikely. If the D/~P input is prevented from going HIGH immediately after configuration, then further reconfigurations will be suppressed and functional operation will not commence until that input is permitted to go HIGH. In multiple LCA designs the D/~P pins are wire-OR 'ed, so the last LCA device to be configured will prevent other, already configured devices, from functioning. When the last device is configured the composite wire-OR line goes HIGH and all LCA devices enter a functional mode simultaneously.

~RESET

The ~RESET input is active LOW and is used to start the initialization process before or during configuration, but not during functional operation. Figure 1 shows the initialization and configuration procedure as a flow diagram. Asserting the ~RESET line will put the LCA device into an initialization mode, aborting the current cycle of configuration.

Once configuration is complete the ~RESET input takes on a different role. It may be used to reset the device as if the device were a functional logic system. When asserted, internal registers and/or latches are reset. The reset condition may then be relaxed and functional operation resumed without the generation of a reconfiguration cycle.

To initiate reconfiguration the D/~P pin should be used instead of the ~RESET input. When ~RESET is LOW at the initialization cycle, its LOW-to-HIGH transition will sample the logic condition at the mode input pins. The logic state sampled determines the mode of configuration. Once configuration has started these pins are not required to be held. However, it is recommended that these pins remain unchanged until the configuration phase is completed.

MODE CONFIGURATION INPUTS M0, M1, and M2

M0, M1 and M2 are the Configuration Mode Select Pins. They force the LCA device into the selected configuration mode on the rising edge of ~RESET. Pins M0 and M1 are dedicated mode inputs having no general-purpose I/O capability. They will usually be hardwired to logic HIGH or LOW conditions. For all modes except the master serial mode, M2 is HIGH and is pulled HIGH internally during configuration. After configuration this pin can be used for general I/O functions. M2 and M0 both have internal pull-up resistors. M1 should be tied to VCC or GND. Table 2 lists five of the modes of operation. The only mode that requires a logic LOW on the input to M2 is the Master Serial Mode which is used with a dedicated serial port EPROM.

M0	M1	M2	MODE SELECT
0	0	0	Master Serial Mode
0	0	1	Master LOW Mode
0	1	1	Master HIGH Mode
1	0	1	Peripheral Mode
1	1	1	Slave Mode

Table 2.

PWRDWN~

The PWRDWN~ input is an active LOW power-down pin that can be used to reduce power consumption of the LCA device when it is not being used. The LCA device is then considered off-line because all I/O lines are disabled while internal configuration is maintained. The VCC input may be reduced to 2.0 Volts and configuration data will not be lost. With internal logic disabled and I/O pins in a high impedance condition, the D/~P pin is forced LOW and all internal storage elements become cleared. It is essential that this pin is used only while D/~P is HIGH after initialization and configuration. It is common to tie this input to a valid logic HIGH if the power-down feature is not required.

Non-Dedicated Pins used in all Configuration Modes

DIN and DOUT

DIN (configuration Data IN) is used as a serial data port into the LCA device. Individual data bits are applied to this input and clocked into the device by rising clock edges applied to the CCLK input. DIN is used in both slave and peripheral modes.

DOUT (configuration Data OUT) is used to configure multiple LCA devices in a daisy chain. The DOUT signal from the preceding LCA device drives the DIN of the succeeding device and is synchronized to rising edge of the clock pulse on the CCLK line.

CCLK

CCLK (Configuration CLocK) as described above is used to synchronize the serial data stream into the data input DIN pin. The CCLK can be an input pin or output pin depending on the mode of configuration. It is an output for every mode of configuration except for the slave mode. The CCLK reverts to an input in the slave mode, and can be driven either by an external source or the CCLK output of another LCA device configured in the master mode. In the operational mode the CCLK is an input to enable configuration data to be read back from the LCA via the M0 and M1 pins, which have the dual function of Readback Trigger (RT) and Readback Data (RD), respectively. CCLK has an internal pull-up resistor which allows the input to be pulled HIGH when not in use.

RCLK

The Read CLocK is used as an output signal which goes LOW when the LCA device expects to see valid input data, and HIGH when the address bus contents are changing states. It can be used to enable the external EPROM, and in the master serial mode can be used to clock an external address counter. This pin is used to clock the CLK input of the serial EPROM, a support device for the LCA.

HDC and LDC

High During Configuration (HDC) and Low During Configuration (LDC) are driven HIGH and LOW, respectively, for the duration of the configuration process. They are used to control external logic during configuration. When configuration has been completed HDC and LDC become general-purpose input/output pins. All of the other I/O pins not involved in the actual configuration process are connected to internal pull-ups to VCC, which are removed after configuration.

CHIP SELECT INPUTS ~CS0 ~CS1 CS2 ~WRT

~CS0, ~CS1, and CS2 are the chip select pins used during the Peripheral Configuration Mode only. The three enables can be used to map the LCA device to a specific address. Active LOW inputs to ~CS1 and ~CS0, and an active HIGH input to CS2 will select the LCA device as if it were a memory or peripheral location mapped into the address space of a microprocessor system.

~WRT in the Peripheral Configuration Mode is the same as CCLK in the slave mode. A single data bit is transferred from the data bus into the LCA device on each rising edge of ~WRT.

A0-A15 and D0-D7

The address and data pins are used in the master parallel modes only, and are converted to general-purpose input/output pins after successful configuration. A0-A15 are the address lines used to access the external EPROM. In the Master Low mode, address location 0000 hexadecimal is read first, and the addresses increment each time a data byte is read until all of the configuration data is loaded into the LCA device. The D/~P then goes HIGH indicating that the device is loaded. The address counter outputs convert to general-purpose I/O, unless the D/~P pin is held LOW by the wire-OR function of a D/~P output from a slave device. In that case the counters will continue to count, downloading the slave's configuration data. Counting will continue for more than one slave. In the Master High Mode, address FFFF hexadecimal is accessed first, and the address lines are decremented. This allows the LCA device to share addressing space with an EPROM or EEPROM used by the system. D0-D7 are the data lines connected to the external memory device.

Slave Configuration
<M0 M1 M2> = <1 1 1>

The Slave Configuration Mode (Figure 2a) is simple to implement requiring only three pins, two lines to perform handshake and synchronization and one line for data transmission. Slave configuration is used in the XACT Development System to download data to the support hardware. In this mode data is presented to the LCA device as a serial bit stream, which is transferred to the device as if it were a very large shift register. The data is presented to the DIN pin and sequentially clocked into the device using the CCLK input. When the device is completely loaded, the D/~P pin goes HIGH to confirm that configuration is complete and that the device is ready to function as a programmed unit.

In Figure 2a the LCA device is shown connected to a microcomputer/microprocessor port with the data line (D0) connected to the DIN input. The rising edge of the strobe output will clock valid data on D0 into the LCA. The process continues until the D/~P confirms that configuration is complete to the microcomputer via the data input (D7). The microcomputer can poll D7 to

PIN NAME	APPLICABLE CONFIGURATION MODE(S)					FUNCTION DURING CONFIGURATION	FUNCTION DURING USER OPERATION
	S	P	MH	ML	MS		
M0	•	•	•	•	•	Mode select 0 (I)	Readback trigger (I)
M1	•	•	•	•	•	Mode select 1 (I)	Readback data out (O)
M2	•	•	•	•	•	Mode select 2 (I)	<User I/O>
D/P (Note 1)	•	•	•	•	•	Indicates when configuration process is done (O)	Initiates/Inhibits Reconfiguration (I)
RESET (Note 1)	•	•	•	•	•	Abort/Restart configuration (I)	Master clear of all internal Flip-Flops (I)
CCLK	•	•	•	•	—	Configuration clock (See Notes 1 and 2)	Readback clock (I)
DIN	•	•	—	—	•	Configuration data in (I)	<User I/O> (Note 3)
DOUT	•	•	•	•	•	Configuration data out (O)	<User I/O>
HDC	•	•	•	•	•	Logic HIGH (O)	<User I/O>
LDC	•	•	•	•	•	Logic LOW (O)	<User I/O>
A0-A15	—	—	•	•	—	Address bus (O)	<User I/O>
D0-D7	—	—	•	•	—	Data bus (I)	<User I/O> (Note 3)
RCLK	—	—	•	•	•	Read clock (O)	<User I/O>
WRT	—	•	—	—	—	Write strobe (I)	<User I/O>
CS0	—	•	—	—	—	Chip select 0 (I)	<User I/O>
CS1	—	•	—	—	—	Chip select 1 (I)	<User I/O>
CS2	—	•	—	—	—	Chip select 2 (I)	<User I/O>

Notes:

1. The $\overline{\text{RESET}}$, CCLK, and D/$\overline{\text{P}}$ pins have multiple functions. See text for further details.
2. During Slave mode configuration, the CCLK pin is an input, while for all other modes, it is an output.
3. DIN and D0 are the same physical pins but are associated with different configuration modes.

Abbreviations:

S = Slave
P = Peripheral
I = Input
O = Output
MH = Master high
ML = Master low
MS = Master serial

Table 3. Summary of Pins Used for Configuration

determine the configuration status, or set input D7 to generate an interrupt. Another application for the slave mode would be in a multiple LCA design where one LCA device can serially load subsequent slave LCA devices arranged in a daisy chain. Using LCA devices connected as master and slave or slaves is discussed more fully in a later section.

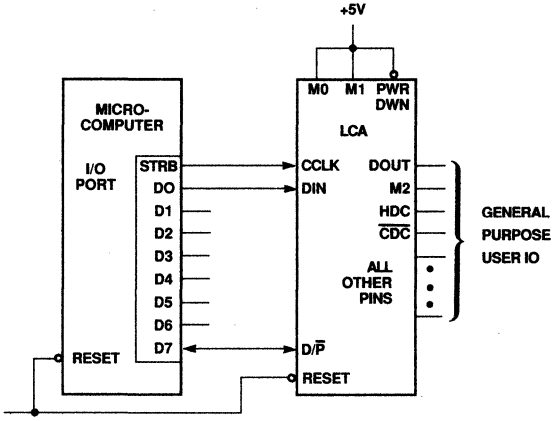


Figure 2a. Slave Mode

PIN NAME	PIN NUMBER		PIN TYPE	VALUE DURING CONFIGURATION	DESCRIPTION
	PLCC	DIP			
Fixed, Non-programmable Pins					
M0	26	18	Input	HIGH	Mode Select
M1	25	17	Input	HIGH	Mode Select
CCLK	60	42	Input	<Clock>	Configuration Clock
RESET	44	31	Input	HIGH	Master Reset
D/$\overline{P}$	45	32	Output	LOW	Done/Program
PWRDWN	10	7	Input	HIGH	Power-down
User-Programmable Pins					
M2	27	19	Input	HIGH	Mode Select
DIN	58	40	Input	<Data>	Configuration Data In
DOUT	59	41	Output	<Data>	Configuration Data Out
HDC	28	20	Output	HIGH	Constant "1" Level
LDC	30	21	Output	LOW	Constant "0" Level

Figure 2b. Slave Mode Pin Summary

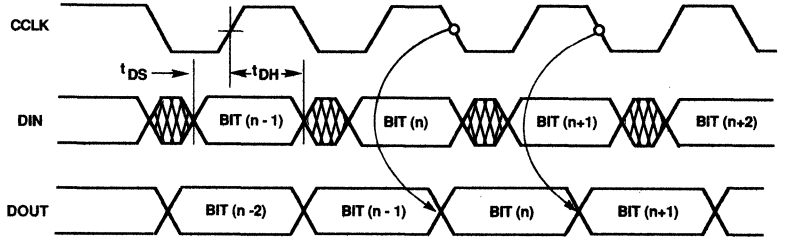


Figure 2c. Slave Mode Configuration Timing

Peripheral Mode Configuration <M0 M1 M2> = <1 0 1>

The Peripheral Mode (Figure 3a) is similar to the Slave Mode in that the data is presented in a bit serial form. In the Peripheral mode, however the LCA device is configured as a microprocessor-compatible peripheral device. The microprocessor can access the LCA device directly through its internal address bus, and map it through chip select logic. Control of the write operation is direct to the LCA device. Less hardware is required to connect the LCA to a microprocessor in this mode, and the loading of data is serial over a single data line. Synchronization is achieved by the falling edge of a ~WRT input, while data is set up on the DIN line and strobed by the falling edge of a microprocessor-generated write signal.

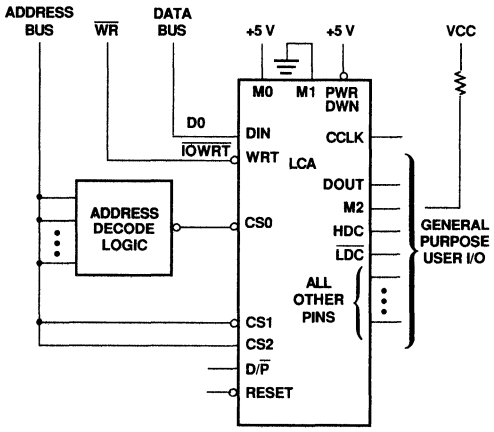


Figure 3a. Peripheral Mode

PIN NAME	PIN NUMBER		PIN TYPE	VALUE DURING CONFIGURATION	DESCRIPTION
	PLCC	DIP			
Fixed, Non-programmable Pins					
M0	26	18	Input	HIGH	Mode Select
M1	25	17	Input	LOW	Mode Select
CCLK	60	42	Output	<Clock>	Configuration Clock
$\overline{\text{RESET}}$	44	31	Input	HIGH	Master Reset
D/$\overline{\text{P}}$	45	32	Output	LOW	Done/Program
$\overline{\text{PWRDWN}}$	10	7	Input	HIGH	Power-down
User-Programmable Pins					
M2	27	19	Input	HIGH	Mode Select
DIN	58	40	Input	<Data>	Configuration Data In
DOUT	59	41	Output	<Data>	Configuration Data Out
$\overline{\text{CS0}}$	50	35	Input	LOW	Chip select (Active LOW)
$\overline{\text{CS1}}$	51	36	Input	LOW	Chip select (Active LOW)
CS2	54	37	Input	HIGH	Chip select
$\overline{\text{WRT}}$	56	38	Input	<Strobed>	Write enable (Active LOW)
HDC	28	20	Output	HIGH	Constant "1" Level
$\overline{\text{LDC}}$	30	21	Output	LOW	Constant "0" Level

Figure 3b. Peripheral Mode Pin Summary

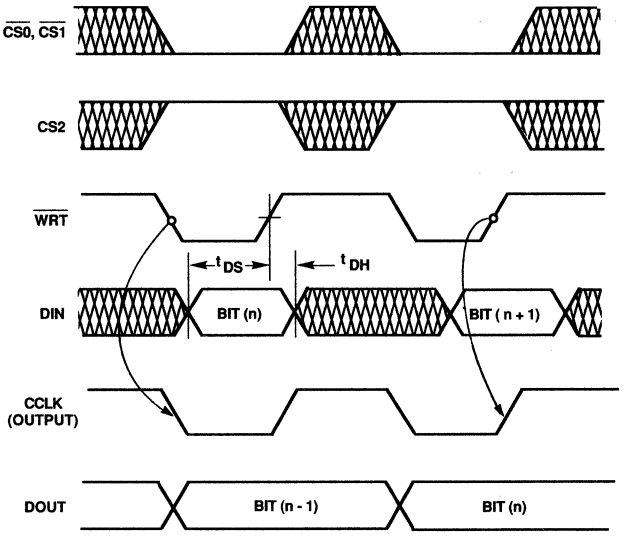


Figure 3c. Peripheral Mode Configuration Timing

Parallel Master Mode Configuration

MASTER MODE LOW <M0 M1 M2> = <0 0 1>

MASTER MODE HIGH <M0 M1 M2> = <0 1 1>

In the Master Low Parallel mode (Figure 4a) data is loaded in parallel at a rate determined by an on-chip timer in the LCA device. Configuration data is stored in an EPROM or EEPROM, and a parallel data path is connected from the memory output data lines D0-D7 to dedicated data inputs on the LCA device. There are sixteen address outputs, A0(LSB)-A15(MSB), connected to the EPROM. At the start of configuration in master mode low the LCA device sends an incrementing address starting at address 0000 hexadecimal and sequentially selects the configuration bytes. Data is serialized when loaded into the LCA device.

When configuration is complete the address and data lines become general-purpose I/O blocks. The D/~P pin is used to provide external logic, an indication that the LCA is currently functional or configuring.

The D/~P pin can be used to select the EPROM/EEPROM during configuration. An active LOW output applied to the memory's Chip Select input will select the device to read the configuration data and deselect it when configuration is complete. The method shown in Figure 4a is suggested for logic systems that have no host processor available to perform the configuration. A drawback is that address and data lines might require isolation from external logic circuits while the LCA device is configuring. The logic designer must determine which signals need isolation and which do not. Using IOBs which are not address and data lines might be a solution to avoid any conflict that might occur when external logic is driven during configuration. Also, the designer must avoid loading address and data lines with capacitive or inductive components. For example general IOBs should be used for circuits such as relaxation oscillators which require capacitive loads. Loading address or data lines with capacitors might lead to a violation of setup and hold times, causing erroneous configuration information to be read.

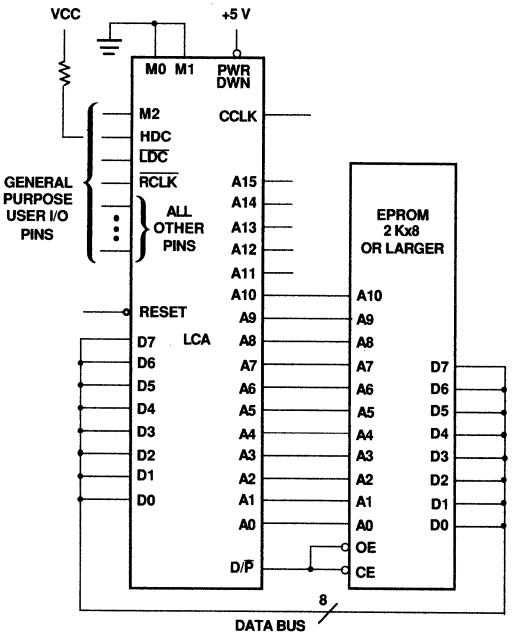


Figure 4a. Master Parallel Mode

In the Master Mode HIGH the initial starting address is hexadecimal FFFF, and the address decrements after each data byte is read. For both Master Mode HIGH and Master Mode LOW, the LCA device will set the D/~P pin HIGH after enough bits have been read to configure the one master LCA device. This permits configuration data to be stored in an EPROM that also holds microprocessor firmware. Only 1505 (5E1 hex) bytes are required for an M2064 LCA device and 2235 (8BB hex) bytes for the M2018, so large EPROMs can store configuration data for a number of LCA devices. Configuration information is concatenated in one EPROM and read by the master device, then sent to slave LCA devices as outlined in the section describing the configuring of multiple LCA devices.

Bytewide Master Mode Pin Summary

PIN NAME	PIN NUMBER		PIN TYPE	VALUE DURING CONFIGURATION	DESCRIPTION
	PLCC	DIP			
Fixed, Non-programmable Pins					
M0	26	18	Input	LOW	Mode Select
M1	25	17	Input	LOW or HIGH	(Master-low mode) (Master-high mode)
CCLK	60	42	Output	<Clock>	Configuration Clock
$\overline{\text{RESET}}$	44	31	Input	HIGH	Master Reset
D/$\overline{\text{P}}$	45	32	Output	LOW	Done/Program
$\overline{\text{PWRDWN}}$	10	7	Input	HIGH	Power-down
User-Programmable Pins					
M2	27	19	Input	HIGH	Mode Select
DOUT	59	41	Output	<Data>	Configuration Data Out
HDC	28	20	Output	HIGH	Constant "1" Level
$\overline{\text{LDC}}$	30	21	Output	LOW	Constant "0" Level
$\overline{\text{RCLK}}$	57	39	Output	<Strobed>	Chip enable output
A0-Axx	—	—	Outputs	<Address>	Memory address bus
				A15 ... A11 ... A0	
48 DIP				3 5 6 4 2 1 48 47 46 45 44 43	
68 PLCC				65 67 2 4 6 8 9 7 5 3 68 66 64 63 62 61	
D0-D7	—	—	Inputs	<Data>	Memory data bus
				D7 ... D0	
48 DIP				28 29 34 35 36 37 38 40	
68 PLCC				41 42 48 50 51 54 56 58	

Table 4b. Master Mode Pin Summary

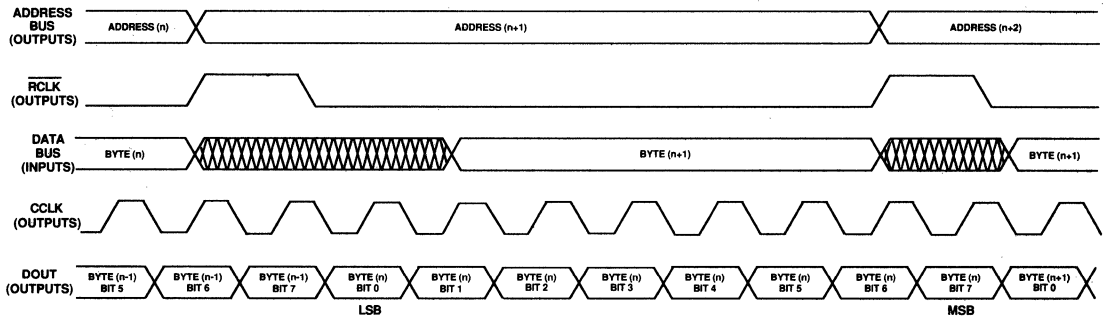


Figure 4c. Master Mode Configuration Timing

Serial Master Mode
<M0 M1 M2> = <0 0 0>

The Master Serial Mode (Figure 5) uses a Serial Data PROM (SDPROM) which has an internal address counter and a serial data port for the conversion of parallel data from the SDPROM to a single data output. Figure 5a shows the schematic diagram required for this mode. Configuration data is stored in the PROM section and downloaded to the LCA device in the most pin-efficient way, avoiding large numbers of address and data interconnections. The SDPROM is connected to the LCA device using only three pins. In this mode the LCA generates a clock signal which is used to increment the address counter built into the SDPROM. The SDPROM then sends data to the DIN pin on the LCA device. This mode is very similar to the Slave and Peripheral Modes in that data input is through the DIN pin. However, the SDPROM requires a clock input to increment its internal address counter which is supplied by the RCLK output of the LCA device. The configuration time is therefore determined by the LCA devices on chip counter. The D/~P input terminates the configuration process by disabling the SDPROM. One SDPROM can hold enough configuration data for three M2064 or two M2018 devices, and they can be cascaded for larger configuration arrays.

The data to be loaded into an LCA device is developed using the XACT Development Software, and is stored in one of two modes: a serial bit stream to be used in the Peripheral and Slave modes, or a 1500-byte PROM file for use in the Master Mode.

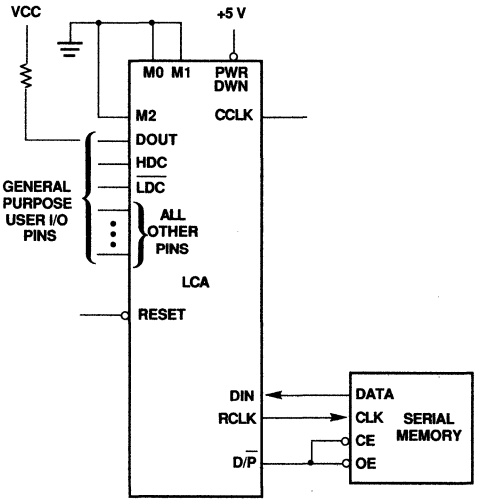


Figure 5a. Serial Mode

PIN NAME	PIN NUMBER		PIN TYPE	VALUE DURING CONFIGURATION	DESCRIPTION
	PLCC	DIP			
Fixed, Non-programmable Pins					
M0	26	18	Input	LOW	Mode Select
M1	25	17	Input	LOW	Mode Select
CCLK	60	42	Output	<Clock>	Configuration Clock
$\overline{\text{RESET}}$	44	31	Input	HIGH	Master Reset
D/$\overline{\text{P}}$	45	32	Output	LOW	Done/Program
PWRDWN	10	7	Input	HIGH	Power-down
User-Programmable Pins					
M2	27	19	Input	LOW	Mode Select
DIN	58	40	Input	<Data>	Configuration Data In
RCLK	57	39	Output	<Strobed>	SDPROM Clock
HDC	28	20	Output	HIGH	Constant "1" Level
$\overline{\text{LDC}}$	30	21	Output	LOW	Constant "0" Level

Figure 5b. Master Serial Configuration Mode Pin Summary

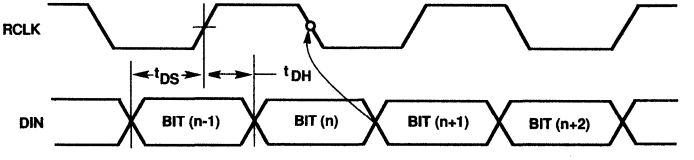


Figure 5c. Master Serial Mode Configuration Timing

Configuring Multiple LCAs

Recognizing that multiple LCAs could be used in a system, the designers added a feature which makes configuration easier by allowing several LCAs to be connected together in a daisy chain. In the Master mode, the first LCA reads data from the EPROM in parallel until it has received all of its configuration data. At this point its D/~P pin would normally be pulled passive HIGH but it is wire-ORed to the remaining LCA devices configured in the slave mode and is held LOW. The first LCA, although configured will not start to function, but continue to pass configuration data to the next LCA device. The data out line (DOUT) drives the DIN line of the first slave, and the CCLK input clocks the configuration data through until the next LCA is configured. The wire-OR action holds the D/~P input LOW preventing the configured LCAs from entering a functional mode. The sequence continues until the last LCA in the slave chain is configured and allows the D/~P line to go HIGH. All the configured devices in the chain start to function simultaneously because the composite wired-OR function goes HIGH on all LCAs simultaneously.

Due to the current sink capability of the D/~P input being able to handle only one pull-up load satisfactorily, only one of the LCA devices in the daisy chain should be configured with 'pull ups'. The first device in the chain can be configured in any of the four modes. All other devices in the chain would use the slave mode.

The daisy chain configuration mode is the easiest and most pin efficient, but the total configuration time increases linearly for each LCA added to the chain. It is also possible to configure LCAs in parallel in the slave or peripheral modes as shown in Figure 7. The total configuration time is then reduced to the configuration time of a single device.

Connecting parallel LCA's in the peripheral mode also allows the use of DMA transfer techniques to decrease configuration time.

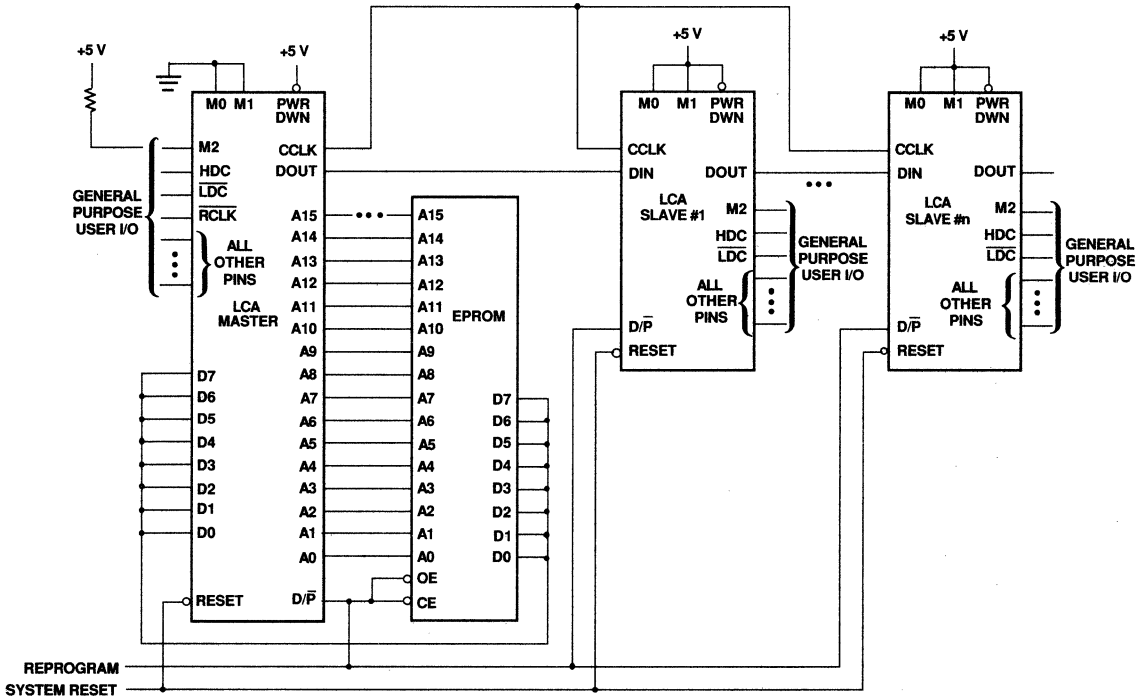


Figure 6a. Master Mode LCA with Daisy-Chain

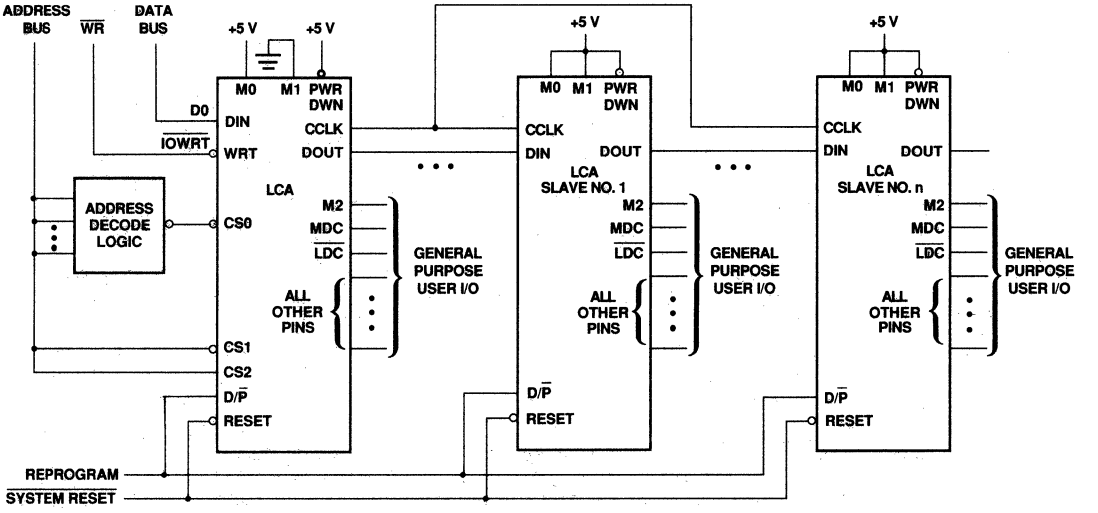


Figure 6b. Peripheral Mode LCA with Daisy-Chain

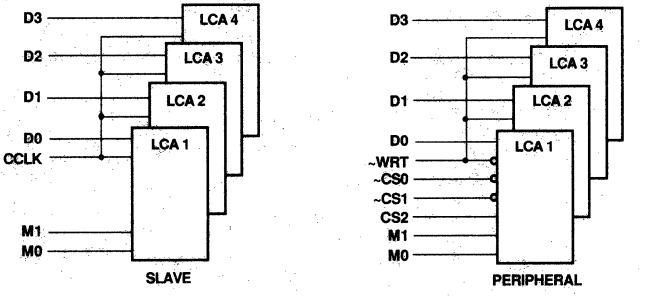


Figure 7. Parallel Loading of LCAs in Slave and Peripheral Modes

Use of the Multiple I/O Pins

When designing with the LCA, careful consideration must be given to which signals are connected to the multiple-function I/O pins. Conflicts can arise when a pin normally used as an input to the LCA suddenly becomes an output during the configuration cycle. Similarly, outputs which are normally under tight control by the LCA logic can transition erratically during configuration, causing adverse effects to the circuitry connected to them.

By simply assigning LCA outputs to pins which become outputs during configuration, and LCA inputs to pins which become inputs during configuration, the designer assures that conflicts can be held to a minimum. If all conflicts cannot be resolved using this method, then external buffers may be added to eliminate the the possibility of any bus contention. The signals HDC (High During Configuration) and LDC (Low During Configuration) can be used to enable or disable the buffers at the appropriate times. See Figure 8.

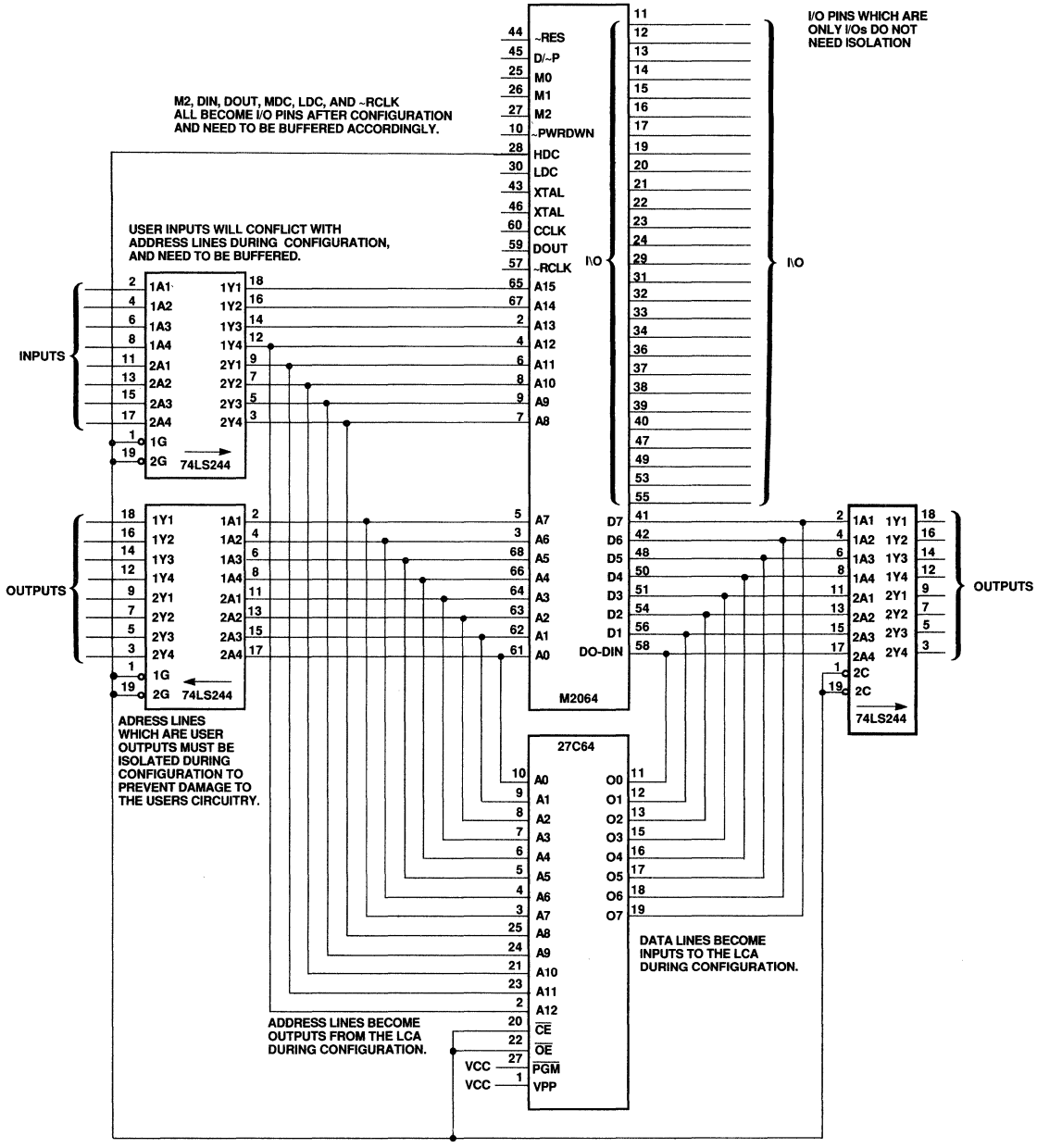


Figure 8. Isolation of I/O Pins During Master Mode Configuration

M2018 Provides Decoding for Six-Digit, Seven-Segment Liquid Crystal Display

Chris Jay

Abstract

The Logic Cell™ Array (LCA) from Monolithic Memories is a high density programmable device capable of supporting true VLSI logic functions. Its unique properties can be viewed as providing the benefits of Programmable Logic Devices (PLDs) while approaching the high functional density of a gate array. Combining these features makes the LCA device a product capable of bridging the gap between PLDs and Gate Arrays.

The LCA structure differs from the conventional concept of the PLD because it is based on a CMOS RAM cell architecture. Its internal logic circuits, input/output (I/O) resources plus the interconnect are all programmable. In comparison to the traditional fuse array of a PLD which is linked to a fixed logic structure. The efficiency of this logic structure reduces as functional density increases.

The low-power CMOS RAM locations of the LCA device must be initialized and configured immediately after power has been applied to the circuit. Once configured, functionality is maintained by the continued application of power to the device. To ease configuration, the device can reside alongside a low-cost EPROM which holds the configuration data. The LCA device can automatically load itself and be reprogrammed any number of times. This feature of reconfigurability can be used for system development, modifying existing designs, or supporting multiple system choice, with very little actual hardware modification. In many instances this can be achieved dynamically or "on the fly."

The following applications note describes the design of a six-digit, seven-segment decoder/driver for Liquid Crystal Displays (LCDs). Design methodology is outlined and the use of the XACT™ software is discussed in the role of a CAD design tool for the LCA device. The final designs are available to readers of this applications note on request.

M2018 Provides Decoding for Six-Digit, Seven-Segment Liquid Crystal Display

Chris Jay

Introduction

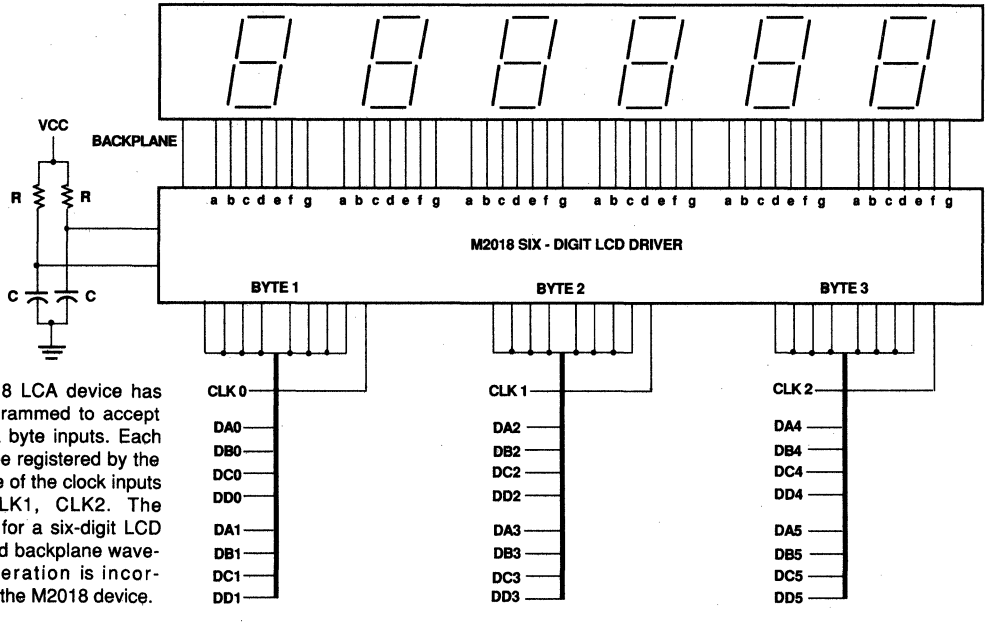
There are two types of low-voltage, seven-segment displays available as indicator panels of multimeters, frequency counters, tachometers and other digital instruments: the light emitting diode displays (LEDs), and Liquid Crystal Displays (LCDs). The advantages of the LCD are mainly associated with very low power consumption and ease of readability. Instruments and portable equipment can benefit from low-power CMOS technology and use these displays. The problem with LED displays is that a much higher operating current is required to illuminate the segments, virtually precluding their application in portable battery-operated equipment. Also, LED display outputs tend to "wash out" in sunlight, so they are very unsuitable in bright daylight, or where the ambient presence of light is high. In the display instrumentation of an automobile, for example, an LED display panel might be "washed out" by high levels of reflected sunlight. LCD displays are preferable because the visual quality is good even in brightly lit environments.

Despite the good visual quality and low-power consumption of the LCD display, an LED type can easily be driven from a multiplexed bus output. The output pin requirement of a seven-segment LED driver can be reduced considerably by a multiplexed output arrangement. In a six-digit display, each anode (or cathode) drive is selected in synchronism with its multiplexed seven-segment output. A total of thirteen pins are required, seven pins to drive the segments and six pins to control the anode (or cathode) of each display. If each digit is multiplexed at high speed with a proportionally larger pulse current per display, then the readability of the output is unaffected. With an LCD display, multiplexing is virtually impossible for two reasons. First, the LCD display has a slow response time and can not be multiplexed to give a good visual output. Second, the display backplane has to be pulsed. Any beating of a multiplexing frequency with the backplane bias frequency can occur, resulting in an unreadable output.

LCD displays have drawbacks because any driver circuit will need one pin per segment, plus a backplane driver pin for the whole display unit. Seven segments for each digit, in a six-digit display requires forty-two outputs pins, plus the one backplane pin. The input pin requirement is twenty-four, four pins for each of the six digits. The total pin count is sixty-seven. Additional input circuitry to support register enables and oscillator inputs would use up even more I/O resources and pins.

Block Function of the LCD Decoder/Driver

Figure 1 shows a schematic block diagram of the system as designed into the M2018 Logic Cell Array. Additional pins to synchronize the clocking of data into the internal data registers take three more pins. Since a backplane oscillator is required, two pins are configured to a resistor/capacitor (RC)



The M2018 LCA device has been programmed to accept three data byte inputs. Each byte can be registered by the rising edge of the clock inputs CLK0, CLK1, CLK2. The decoding for a six-digit LCD display and backplane waveform generation is incorporated in the M2018 device.

Figure 1. Liquid Crystal Display Decoder Driver

network. The RC combination generates a square wave which is used as a backplane bias to the display. The pin count is seventy-two.

Binary data applied to byte 1 is registered after the rising edge of CLK0. The registered data is decoded to illuminate the relevant segments forming the hexadecimal display. Decoded data is then routed to the two least significant digits of the six-digit LCD. Byte data applied to inputs 2 and 3 are loaded by a clock rising edge, CLK1 and CLK2, respectively. A backplane oscillator output drives the LCD's backplane with a low-frequency square wave, which is determined by the RC time constants of a resistor/capactitor network. To display a segment the decoder circuitry will drive the selected segment with a square wave that is 180 degrees phase-shifted from the backplane reference. If a segment is driven from an in-phase signal, then it is blanked.

The advantage of the M2018 Logic Cell Array in this application is that it is a CMOS device, and consumes low power at low operational frequencies. Its power consumption is dependant on the internal clocking frequency and also on the percent usage of on-chip logic. There are one hundred individually configured logic blocks in the M2018, and any logic blocks that remain unused will consume only static substrate leakage current. Since the M2018 contains seventy-four general-purpose user I/O blocks, there is ample programmable logic and I/O capability to perform decoding, registering and storage of data for six seven-segment digits.

LCA Configuration

Since the LCA device is based around a programmable array of volatile RAM cells, it must recieve configuration information at "power up." In this application it is programmed from an EPROM which holds the configuration data. Configuration takes place after power has been applied to the circuit, and it has been successfully reset. The LCA device enters a configuration mode, sequentially reads the data from the EPROM into its RAM cells, and becomes functionally operational in approximately twenty-five milliseconds. It disables the EPROM after configuration. The volatility of the M2018 is a distinct advantage in many applications. In this case, different display decode operations may be selected at "power up'. Higher order address lines may be "hardwired" to select different configuration patterns from separate pages in the EPROM. So a small amount of deferred design may be added to the system.

The M2018 has many advantages over custom VLSI circuits. Here, designers have control over how they wish to configure the circuit. Different display fonts can be programmed, for special characters. Most of the I/O pins can be reconfigured to ease printed circuit board layout.

Figure 2 shows the circuit connections of an 84-pin M2018 in a PLCC to an 8 Kbyte-wide EPROM. Approximately 2.5 Kbytes are required for configuration. Three configuration patterns may be cascaded into one 8 K x 8 EPROM, but in this application, two configuration patterns may be stored, one at base location of 0K and one at base location of 4 K. Address line A12 may be configured to a DIP switch or hardwire link, to select one of two configuration patterns. Two designs were developed, one for decoding binary data to a hexadecimal display and one for binary coded decimal. Both configuration patterns could be programmed into the EPROM and selected at a later time.

The configuration mode chosen for the LCA device was the Master Low mode. Data is sequentially read from the EPROM during the configuration cycle. After power has been applied and the RESET input has been deasserted, the M2018 will output a series of incrementing addresses to the memory starting at the initial address of 0000. The DONE/~PROG (D/~P), which is an output during configuration, is driven active LOW. It is tied to the CS and OE inputs of the EPROM. When configuration is finished, D/~P is pulled passive HIGH by an internally-configured "pull-up" resistor in the LCA device . The EPROM is then deselected. The normal data flow into the device is via dedicated input lines D0-D7 during configuration, and afterwards these become general-purpose I/O lines. The total time for configuration can vary between 17 to 34 ms.

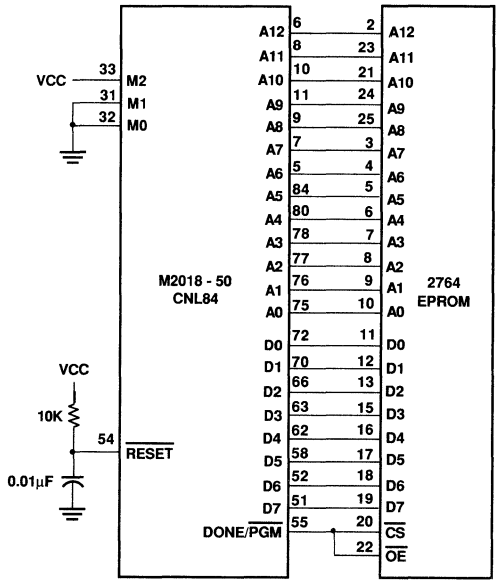


Figure 2. Liquid Crystal Display Decoder Driver

Design of the Backplane Oscillator

To establish a backplane bias frequency, an R/C network is used in conjunction with a logic block that is designed to function as a relaxation oscillator. The Configurable Logic Block (CLB) is connected to two Input Output Blocks (IOBs) which in turn are connected to two external R/C networks, R1,C1 and R2 and C2 as shown in Figure 3a. The calculation of the R/C values to create a low-frequency backplane oscillation of even mark space ratio is given below. For an even mark/space ratio R = R1 = R2 and C = C1 = C2. The time constants for period t are given by the following formula:

T1 = 0.35(C X R X 2) ;for TTL voltage ;thresholds.

T2 = 0.75(C X R X 2) ;for CMOS voltage ;thresholds.

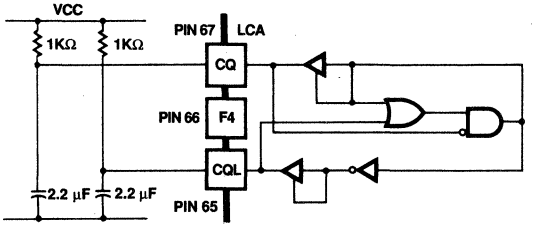
The general expression for the calculation becomes:

T = N X [(R1 X C1) + (R2 X C2)].....(1)

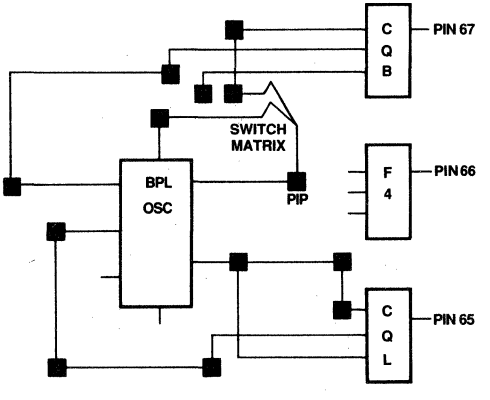
Where N is the TTL, or CMOS multiplying factor of 0.35 for TTL or 0.75 for CMOS. The LCA device can be programmed to be compatible with either technology.

The values of 1 Kohm and 2.2 microfarad gave a backplane oscillation frequency of about 80 Hz.

Figure 3a shows the schematic arrangement of the backplane oscillator as it would be configured in the LCA device. Two R/C



a. Logic Schematic Diagram of GOSC Macro



b. CLB and IOB Placement of GOSC Macro

Figure 3. Liquid Crystal Display Decoder Driver

networks of 1 Kohm and 2.2 microfarads are connected to pins 67 and 65 respectively. The placement of the blocks is shown in Figure 3b as it would be seen on the color monitor of a PC when the XACT Design Editor is used. Pins dedicated to configuration functions must not be loaded with components such as capacitors because they could prevent the LCA device from reading valid data. Pin 66, which is associated with data line D2 was therefore assigned to drive one of the LCD segments.

XACT Editor and Macro Support for Quick Design Entry

The logic design is accomplished by using the XACT Design Editor running in an IBM® PC/AT™ or PC/XT™. This support software allows users to edit their designs via a mouse interface and a keyboard input. The CLB and IOB elements may be configured and connected to form complete logic subsystems through the programmable interconnect. The XACT Editor allows the user to view a section of the LCA device on the display unit of the PC, for CLB and IOB placement, and distribution of interconnect. Supported by the zoom-in and zoom-out facility, and a "world view" feature (this allows the user to see the entire LCA device layout) enables a designer to "hook" subsystems of the design together. Menus are displayed at the head of the video display, and a designer may move the cursor with a mouse to select options. These include BLOCK commands enabling CLB/IOB editing; CONFIG to enable configuration of blocks; and NET and PIN to assign pins to nets, so creating the overall interconnection. This could be analagous to interconnections on a printed-circuit card, making the electrical connection between pins of 74LSXX or other types of logic devices.

The XACT editor is equipped with a support MACRO library. This library allows widely used functions to be invoked without redesign. The relaxation oscillator, used for the backplane generator, is supplied as an existing design configuration in the MACRO. The oscillator MACRO may be called by downloading the GOSC MACRO file.

When using the XACT editor the application of already existing MACRO support files can dramatically reduce the overall design cycle time. The six-digit decoder display driver is basically repetition of the same six logic subsystems, one for each digit as shown in Figure 4. The CUTMACRO feature is useful as a support facility in avoiding repetitive design operations on individual IOBs or CLBs. Once a CLB has been designed, it can be stored as a MACRO, and given a user-defined name. The designer may call that MACRO wherever the IOB or CLB needs configuration and placement. The CUTMACRO feature also supports multiple CLBs, IOBs and interconnects of subsystems. A decode section identical to the one shown in Figure 4 can be stored as a MACRO and given a user-defined name. The design for a decoder section needs only to be performed once. The remaining five sections may be called and placed using the user-defined MACRO support facility.

Widely Used XACT Functions

As described,the XACT software is menu-driven and operations supported by the design editor are listed at the head of the screen with a schematic representation of a portion of the device shown below. A mouse interface permits the designer to scan a cursor over the schematic, or choose a functional operation by "clicking" on the selected option. Following are the available options from left to right:

NET PIN BLK CONFIG SCREEN MISC PROFILE

When selected, a heading will list its menu options. The BLK command gives nine options. The most commonly used options are EDITBLK and ENDBLK, which allow the designer to select a specific logic block and enter a logic description into it. The latter command will terminate the editing process and return the designer to a world view of the LCA schematic. Details on editing CLBs are described in more detail in the CLB Configuration Section. Other commonly used commands in the BLK menu include COPYBLK, to copy a repetitive logic design into other unconfigured blocks, DELBLK to erase unwanted designs from the network; CLRBLK, to clear the contents of a block but leave its input and output pins attached to the nets; MOVBLK, move a configured block with its associated nets to another block location; and NAMEBLK, to permit the assignment of user-defined names. Each time a function is invoked, the XACT editor prompts for a response at the bottom of the screen. For example, EDITBLK would invoke a "SELECT

BLOCK" invitation. The user then "clicks on" the block he/she wishes to configure.

The CONFIG command has six subcommands which relate to the editing of a CLB. The BASE command directs which logic arrangement is to be selected: one Boolean function of four-input variables, or two functions of three-input variables, or multiplexed input selecting either one choice of two three-input variables. EQUATE is commonly used, and allows a designer to generate a Boolean equation from keyboard input and configure it to an ouput. Other choices include ORDER, to establish blank truth tables for Karnaugh map entry; CLEAR, to delete unwanted functions; and CDATA (see data), for a text description of the block configuration.

The creation of configured blocks must have the support of interconnections. The NET and PIN commands are invoked to establish the connection of CLB outputs to CLB or IOB inputs. The NET menu may be invoked to create a net onto which input pins and one output pin is listed. The ADDNET command allows a designer to add a net to the netlist, and ADDPIN allows the user to add pins to a selected net. The NAMENET subcommand in the NET menu allows the default value of netx (where x is an XACT default-assigned integer) to be overruled by a user-assigned name. This is similar to the NAMEBLK command for BLK functions. Nets can be routed, unrouted, deleted and merged by the ROUTE, UNROUTE, DELNET and JOINNET commands, respectively. Manual editing of the net is invoked by the EDITNET command. A net is established and manually interconnected to Programmable Interconnect Points (PIPs) to create a circuit. The other NET options include FLAGNET, which assigns critical or non-critical status to a net; and HIGHLIGHT/UNHIGHLIGHT commands which "bright up" or "clear" net distribution on the "world view."

The PIN commands are: ADDPIN for adding pins to nets, and CLEARPIN for removing them. The SWAPPIN allows pins to be switched without switching functionality, where SWAPSIG exchanges the logic functionality inside the CLB without switching the pins. MOVEPIN will move a pin from one location to another, and ROUTEPIN will establish an interconnection on a net.

These and other menu functions are described in greater detail in Monolithic Memories' XACT Development System manual.

Design Methodology

Figure 4 shows a block schematic of one seven-segment decoder driver. Seven logic blocks decode the four data inputs DA, DB, DC, and DD, where DA is the least significant data input. Each block has the capability of storing the data in a register inside the logic block. The backplane oscillator drives the LCD backplane for all segments. The individual segments are driven from an exclusive-'OR" (XOR) gate, which is programmed to invert the backplane waveform for an illuminated segment, and not invert for a blanked segment. The segment decoder circuits, SA0, SB0 to SG0, provide a logic LOW output for an active segment drive. Therefore, the backplane input to the XOR gate is inverted in a second logic block and provides the correct phase control for the segment driver. The configuration was repeated six times in the LCA device to support a total of six digits.

Figure 5 shows the truth table for decoding four binary inputs into a hexadecimal segment drive. DA is the least significant binary input and DD is the most significant. The hexadecimal weighting of the binary input is also shown in the table. To illuminate a zero digit, all the segments are driven active except for segment "g," for digit "one"; segments "b" and "c" are active, and for digit "two" segments "a," "b," "d," "e" and "g" are driven. The complete decode arrangement is shown at the head of Figure 6.

The Karnaugh maps, derived from the truth table shown in Figure 5 are given in Figure 6, and provide the decode equations for all the segments. Logic "one" entries represent illuminated segments and logic zeros represent extinguished segments. A greater reduction efficiency of minterm entries was achieved using reduction techniques applied to map locations containing logic zeros. This required an inverted input to the exclusive OR (XOR), polarity control gate, as shown in Figure 4. The logic equations derived from the

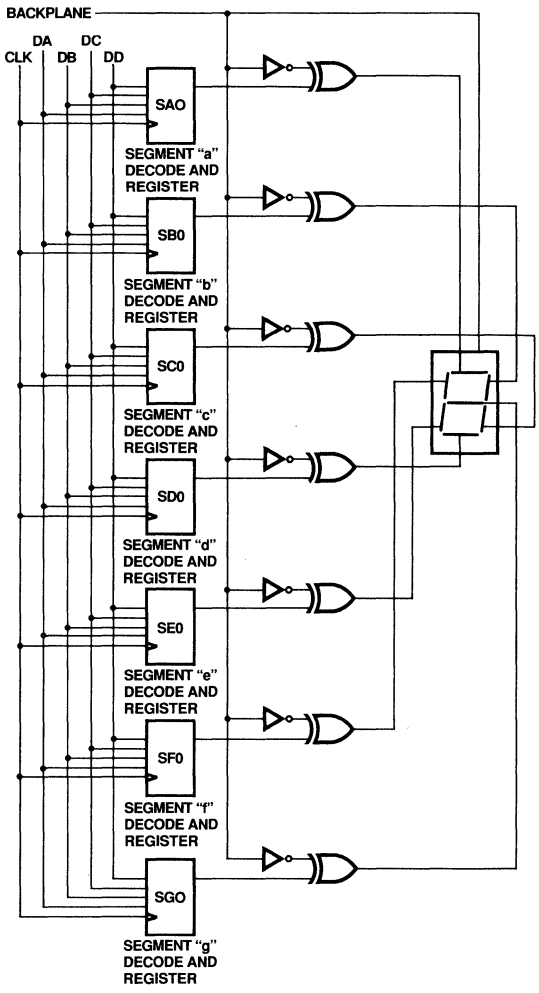
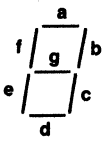


Figure 4. Liquid Crystal Display Decoder Driver



BINARY ENCODED INPUT					SEGMENT LABEL						
	LSB			MSB							
HEX.	DA	DB	DC	DD	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	1	0	0	0	0	1	1	0	0	0	0
2	0	1	0	0	1	1	0	1	1	0	1
3	1	1	0	0	1	1	1	1	0	0	1
4	0	0	1	0	0	1	1	0	0	1	1
5	1	0	1	0	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	1	1	1	0	1	1	1	0	0	0	0
8	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
A	0	1	0	1	1	1	1	0	1	1	1
B	1	1	0	1	0	0	1	1	1	1	1
C	0	0	1	1	1	0	0	1	1	1	0
D	1	0	1	1	0	1	1	1	1	0	1
E	0	1	1	1	1	0	0	1	1	1	1
F	1	1	1	1	1	0	0	0	1	1	1

Figure 5. Liquid Crystal Display Decoder Driver

Karnaugh maps in Figure 6 could be used for Boolean design entry, the active LOW equation for segment "a" is given as:

~a = ~A*~B*C*~D + A*~B*~C*~D + A*B*~C*D + A*~B*C*D

where the inputs to the CLB are A, B, C and D. The tilde sign (~) represents signal inversion, the * and + signs represent product and sum terms, respectively. The Boolean entry also supports the function XOR by the @ symbol, the function ~A*B + A*~B is condensed to A@B.

The conventional logic gate configurations are shown in Figure 7. Seven "sum of product", combinational circuits were derived from the Karnaugh maps shown in Figure 6. Logic designers familiar with PLD designs will recognize a sum of products architecture. Having generated Karnaugh maps and equations, the design can be implemented in the LCA device by using the facilities of the XACT Design Editor running in a PC/AT or PC/XT.

Binary to Seven-Segment Decoder

The display font shows the decoding function required for a binary to seven-segment hexadecimal display. A logic one represents an illuminated segment, from the table shown in Figure 5.

LCD Seven-Segment Display Driver in LCA

D,C \ B,A	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	C	D	F	E
10	8	9	B	A

For binary input ABCD the hexadecimal value of that binary weighting is entered in the correct location of the Karnaugh map.

D,C \ B,A	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	1	0	1	1
10	1	1	0	1

~a = ~A*~B*C*~D + A*~B*~C*~D + A*B*~C*D + A*~B*C*D

(a)

D,C \ B,A	00	01	11	10
00	1	1	1	1
01	1	0	1	0
11	0	1	0	1
10	1	1	0	1

~b = ~A*B*C + A*B*D + ~A*C*D + A*~B*C*~D

(b)

D,C \ B,A	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	0	1	0	1
10	1	1	1	1

~c = ~A*C*D + B*C*D + ~A*B*~C*~D

(c)

D,C \ B,A	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	1	1	0	1
10	1	1	1	0

~d = A*B*C + ~A*B*~C*D + ~A*~B*C*~D + A*~B*~C*~D

(d)

D,C \ B,A	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	1	1	1	1
10	1	0	1	1

~e = A*~B*~C + A*~D + ~B*C*~D

(e)

D,C \ B,A	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	1	0	1	1
10	1	1	1	1

~f = B*~C*~D + A*~C*~D + A*B*~D + A*~B*C*D

(f)

D,C \ B,A	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	0	1	1	1
10	1	0	1	1

~g = ~B*~C*~D + A*B*C*~D + ~A*~B*C*D

(g)

Figure 6. Liquid Crystal Display Decoder Driver

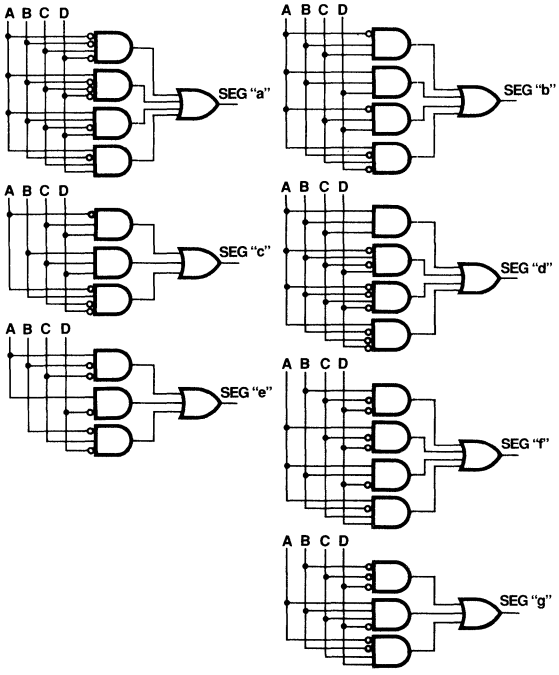


Figure 7. Liquid Crystal Display Decoder Driver

The decode logic for each segment is shown in the conventional sum of products form and was derived from the Karnaugh Maps. It is possible to encode these logic configurations in to one "CLB" for each segment. The "CLB" is capable of producing one output that is a Boolean function of up to a maximum of four inputs.

Editing the Design

The XACT Design Editor was used to create the design, and generate the configuration data, suitable for programming into an EPROM. Three programmable elements have to be considered; the IOBs, CLBs and the interconnection of IOB and CLB blocks.

CLB Configuration

Figure 8 shows the exploded view of a CLB that has been configured as the segment "a" decoder. The XACT Design Editor provides a mix of text and diagram editing supported by a keyboard and mouse interface. The logic equation is shown as text entry at the bottom of the diagram. The combinational block, AA (user defined as SA0 by invoking the NAMEBLOCK feature in XACT), has been configured as the "a" segment decoder. Each CLB can be configured as a single output function of four Boolean input variables, or two output functions of three-input variables each. Both configurations were used in this design example.

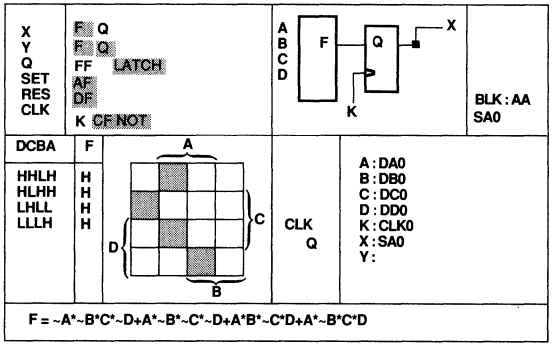
The Karnaugh map reflects the equation entered from the keyboard. An alternate way of entering configuration information may be achieved by using the mouse to select individual locations in the Karnaugh map. These locations may be "clicked" on or off, to select or deselect minterm entries. The equation relevant to the Karnaugh map entry is updated after modification. If designers have generated Karnaugh maps during the design cycle of the project, they could use this visual aid to enter and check the logic integrity of the design. Also, a truth table is available for verification purposes.

An unconfigured CLB will not display a Karnaugh map so to generate a four-input map, the instruction:

Config(Order(F(A(B(C(D))))))

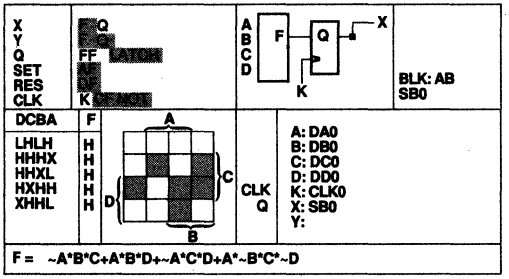
is typed to create one output F that is a function of four Boolean inputs. A blank Karnaugh map will be displayed ready for editing.

The top left-hand portion of Figure 8 indicates how the CLB is configured. The Q output from the register is connected to the X output of the CLB. To activate this path, the mouse is used to select the Q option. The register could be bypassed if the F output was selected. Either register or combinational configurations may be realized. The path from the register output to the CLB X output is established by activating the PIP represented by the block tying Q to X shown in Figure 8. The alternate output, Y has the same assignment options of registered or combinational outputs. There are two configurations available to the storage element; either register or transparent latch. The register's output will change as a result of a transition being applied to its clock input. For a latch, the control input is level-sensitive requiring a logic HIGH or LOW level to distinguish between storage or a transparent mode of operation. In the CLB the clock input polarity can be selected by the NOT function in the CLK submenu. Two features that were not used in this design are the asynchronous SET and RESET functions. If required, the SET option can be invoked by selecting either the A input, or choosing a combinational output from F, which can also perform RESET. A single input to RESET the register or latch is the D input. To the bottom right of the CLB configuration diagram in Figure 8, all the inputs to, and outputs from, the CLB are listed. The net to which each CLB input or output is associated is shown after the colon. The A input is tied to the net DA0; B, C and D inputs are connected to nets DB0, DC0, and DD0, respectively. The user can assign meaningful names to nets in the design process. In this example, net SA0, is segment "a" of the least significant digit and is driven from the X output of the CLB. DA0-DD0 are the four binary inputs for the least significant digit. The clock input is assigned to the K terminal of the CLB, thus, the decoded output may be registered after a rising edge of the clock has been applied to the net CLK0.

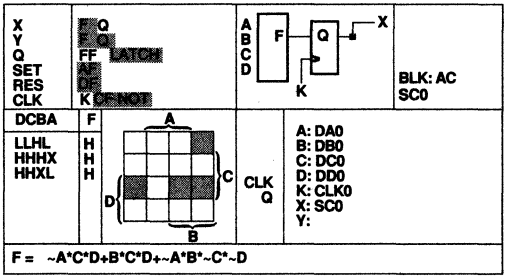


CLB configuration for segment 'a' decoder.

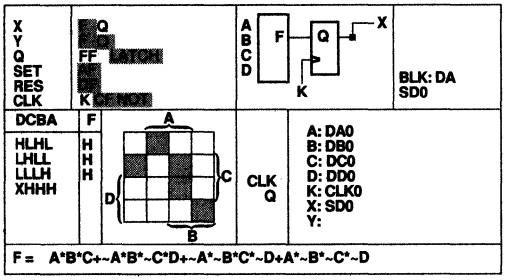
Figure 8. Liquid Crystal Display Decoder Driver



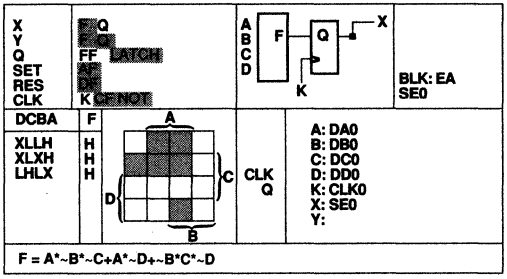
CLB configuration for segment 'b' decoder.



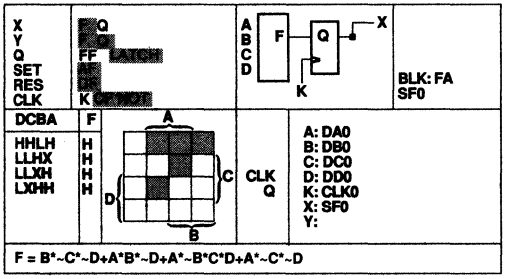
CLB configuration for segment 'c' decoder.



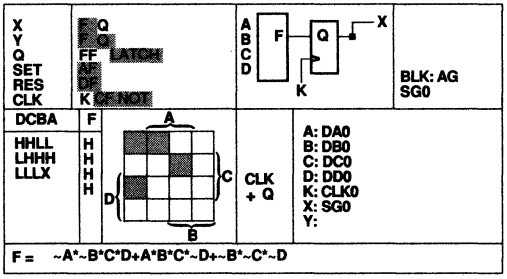
CLB configuration for segment 'd' decoder.



CLB configuration for segment 'e' decoder.



CLB configuration for segment 'f' decoder.



CLB configuration for segment 'g' decoder.

Figure 9. Liquid Crystal Display Decoder Driver

Like nets, individual blocks may be given meaningful names. The CLB situated on a grid location "AA" of the LCA device has been named SA0. Naming nets and blocks can help in debugging the final design. Figure 9 shows the configuration of the CLB elements for segments "b," "c," "d," "e," "f" and "g."

Figure 10 shows the configuration of a CLB as a combinational logic circuit which provides two outputs at X and Y. The backplane waveform is fed to the A input where it is inverted. This signal will pass through the X and Y outputs, appearing either in phase with the backplane reference or shifted by 180 degrees from it. The control inputs come from the segment decoder circuitry. In Figure 10, SA0 and SB0 are the interconnecting nets that convey control signals from the segment "a" and "b" decoders. The configuration used was two Boolean outputs as functions of three-input variables, so outputs F and G were selected.

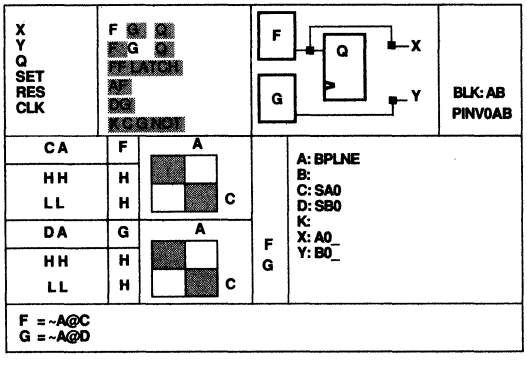
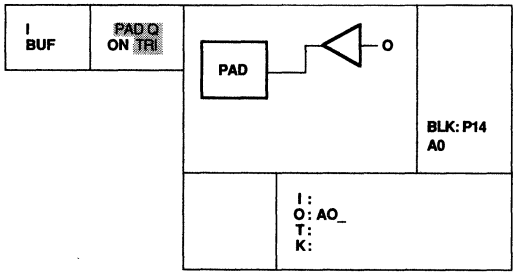


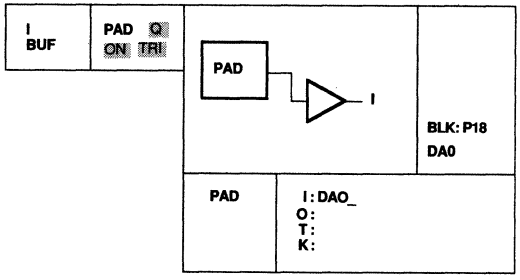
Figure 10. Liquid Crystal Display Decoder Driver

IOB Configuration

The Input/Output blocks must be configured to drive the segments, backplane, and to receive data for decoding. The segment and backplane IOBs are configured as outputs. Inputs are configured for data ports DA0-DD0, DA1-DD1, up to DA5-DD5. The IOBs are considerably less complex than the CLBs, having interface functionality rather than Boolean logic functionality. Each IOB is capable of being programmed as input, output, bidirectional, with HIGH-Z output, and registered input. Individual or composite I/O functions may be selected. Figure 11a shows an output buffer, the input of which is driven from net A0_. Net A0_ is driven from the X output of the CLB featured in Figure 10. The designer has some control over the placement of IOBs around the perimeter of the LCA device, and can use the BLKMOVE commands to optimize pad layout for the best printed circuit design. Figure 11b shows an IOB that has been configured to receive data. The buffer is an input driving net DA0_.



a. Liquid Crystal Display Decoder Driver



b. Liquid Crystal Display Decoder Driver

Figure 11. Input/Output Block Configuration

The IOB has been configured as an output buffer to drive the segment "a" of the least significant digit. The block has been assigned the name A0. the output buffer has been turned on and the input to that buffer has been configured to net AO_. All the segments and the backplane of the LCD have been assigned an output buffer.

The data path to the LCA device for binary data is via IOBs configured as input buffers. The binary inputs DA0, DB0, DC0, and DD0 for the least significant digit and the configured IOBs are assigned user-defined names DA0, DB0, DC0, and DD0. The input nets are DA0_, DB0_, DC0_, and DD0_ respectively. This is repeated for digits 1, 2, 3, 4, and 5.

XACT Supports Design Rule Checking

During the design of the LCD decoder driver circuit the Design Rule Checker (DRC), was invoked to verify that no fundamental design rules were being violated. Errors can easily occur, especially for the uninitiated user. Errors such as two CLB outputs driving one net are usually caught dynamically. Also, a net listing of inputs that are not driven by an output would constitute an error. A general DRC run will list all design violations, errors and warnings. Warnings such as an assigned CLB output that is not connected to a net. This information can be sent to a line printer and the resulting list of Errors and Warnings can be used for design correction. Invoking DRC will check blocks, interconnect and nets. Also, prior to using the MAKEBITS command (for the eventual MAKEPROM) software, the DRC is invoked to trap any possible design violations. Of course, the design rule checker does not screen the design for logic function integrity. An additional software package is available in P-SILOS™ as a logic, and timing simulator. Inputs may be activated by HIGH, LOW, HIGH-Z levels etc., while output logic levels may be listed during the simulation run.

Conclusion

The design was modified for a decimal decoder display driver. Figure 12 shows the modifications necessary to each segment decoder. The choice of a latched version of the display driver was developed as an alternative to the registered type. In each registered CLB, the option was changed for a latched option. By removing the backplane oscillator and making the net BPLNE an input which is driven HIGH or LOW, high efficiency LED displays of common anode or cathode may be driven. The designs developed are as follows:

XDES01.LCA REGISTERED HEX DECODER DRIVER.

XDES02.LCA REGISTERED DEC DECODER DRIVER.

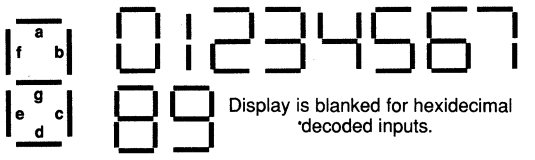
XDES03.LCA LATCHED HEX DECODER DRIVER.

XDES04.LCA LATCHED DEC DECODER DRIVER.

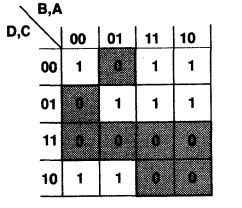
These designs are available as bit patterns for programming EPROMs on request.

The Print World of the design DES01.LCA shows the placement of CLBs, IOBs and the routing for the final design. This can be used as a reference for wiring the device into a circuit. Pin 1 is shown at the top center of the diagram and is the GND connection. The pins are numbered counter-clockwise from that reference pin. For example DB3 is connected to pin 2 and D2 connected to pin 3 etc.

LCD Seven-Segment Display Driver in LCA Binary Coded Decimal

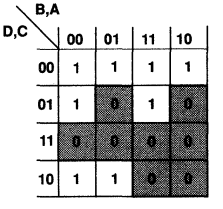


Display is blanked for hexidecimal decoded inputs.



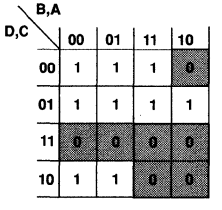
~a = C*D + B*D + ~A*~B*C*~D + A*~B*~C*~D

(a)



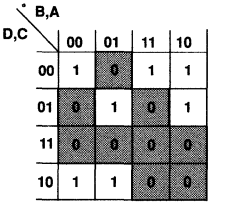
~b = C*D + B*D + ~A*B*C* + A*~B*C

(b)



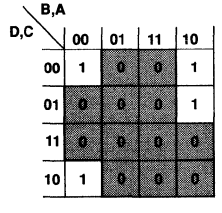
~c = C*D + B*D + ~A*B*~C

(c)



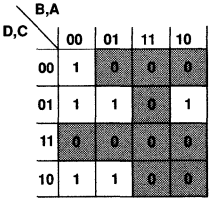
~d = C*D + B*D + A*B*C + ~A*~B*C + A*~B*~C*~D

(d)



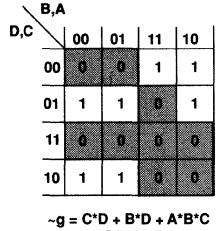
~e = A + ~B*C + B*D

(e)



~f = B*D + A*B + C*D + B*~C + A*~C*~D

(f)



~g = C*D + B*D + A*B*C + ~B*~C*~D

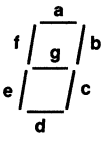
(g)

Karnaugh maps for binary coded decimal drivers. A logic zero represents an extinguished segment.

Figure 12. Liquid Crystal Display Decoder Driver— For Binary Coded Decimal Output

BCD to Seven-Segment Decoder

The table below shows the decoding function required for a BCD to seven-segment hexadecimal display. A logic one represents an illuminated segment.



BINARY ENCODED INPUT					SEGMENT LABEL						
	LSB		MSB								
HEX.	DA	DB	DC	DD	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	1	0	0	0	0	1	1	0	0	0	0
2	0	1	0	0	1	1	0	1	1	0	1
3	1	1	0	0	1	1	1	1	0	0	1
4	0	0	1	0	0	1	1	0	0	1	1
5	1	0	1	0	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	1	1	1	0	1	1	1	0	0	0	0
8	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
A	0	1	0	1	0	0	0	0	0	0	0
B	1	1	0	1	0	0	0	0	0	0	0
C	0	0	1	1	0	0	0	0	0	0	0
D	1	0	1	1	0	0	0	0	0	0	0
E	0	1	1	1	0	0	0	0	0	0	0
F	1	1	1	1	0	0	0	0	0	0	0

Figure 12. Liquid Crystal Display Decoder Driver

Print World: DES01.LCA (2018PC84-50), XACT 1.30, 15:12:14 JUL 1, 1987 Print World: DES01.LCA (2018PC84-50), XACT 1.30,

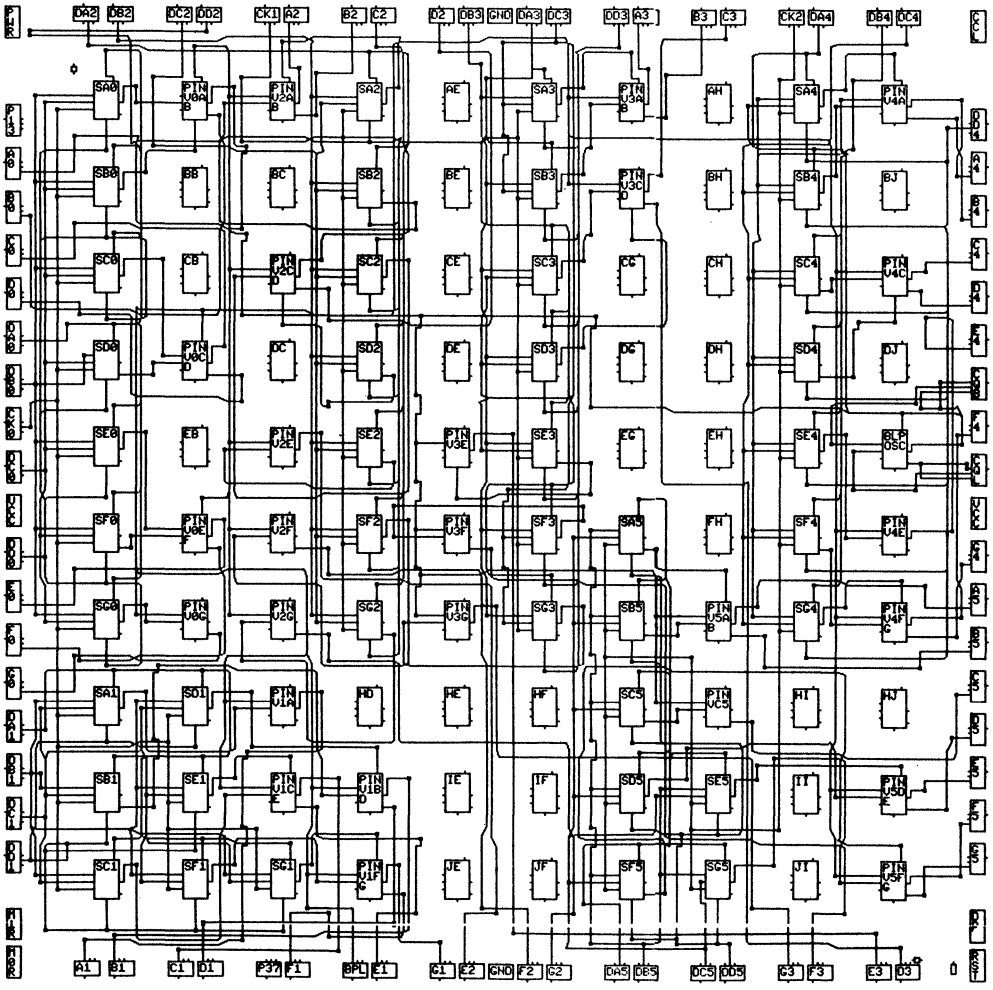


Figure 12. Liquid Crystal Display Decoder Driver

Advanced
Micro
Devices
AN-180

LCA Counter Applications

Chris Jay and Karen Spesard

Abstract

Counters are used in many logic systems to control and synchronize events. Essentially, counters have one thing in common, they are all state machines. State machines, unlike basic combinational functions, require registers with feedback. The design creates an output that is a function of the previous state of the registers in the machine, and in some cases a function of other combinational inputs as well. Listed below are some of the systems that would use various types of counters.

- Direct Memory Access (DMA) Controllers
- Video RAM Refresh
- Dynamic RAM Control Refresh Counters
- Event timing
- Sequence Controllers
- Disc Controllers
- Status Sequencers

The type of counter chosen will depend upon the application. The simplest type of counter would be a free-running type as used for refreshing DRAM or Video RAM. Counter types vary, depending on application from the simple free-running type to loadable binary up/down counters which would be used in applications such as DMA control.

To design efficient counters in the LCA device, designers must consider the desired performance while keeping in mind the available logic and interconnection resources.

Familiarization with the different counter types and their characteristics will enable designers to choose the best counter for a specific application.

LCA Counter Applications

Chris Jay and Karen Spesard

Introduction

The LCA device has an architecture that is very well suited to the development of some counters. Designers already familiar with state machine applications in Programmable Logic Devices (PLDs) will probably be familiar with how to design counters in Programmable Array Logic (PAL®) and/or Programmable Logic Arrays (PLA). The differences between the structure of the PLA or PAL device and the Monolithic Memories' Logic Cell™ Array (LCA) impose both restrictions and freedoms in certain types of state machine design.

Ample width of the PLD "AND" gate means that counter depth can be limited only by the number of registers in the device. This applies especially in the PAL24X family that is specifically designed for applications in high-performance binary counting. In the LCA device, the maximum number of Boolean inputs to any CLB is four. This limiting factor is compensated by the high number of CLBs present in the device architecture. The design philosophy used in an LCA device for developing state machines is different from that of a PLD, but not less effective. Parallel counter architecture, and lookahead-carry techniques can be employed to create machines capable of medium performance.

Counters Using Shift Registers

The type of state machine well suited to an LCA device is based on a shift register. These types may be subdivided into different categories. In each state machine, there are "n" general registers, and the number of states that can be reached vary with counter type. The number of useable states are listed below with the counter type:

1) Johnson counters, 2n, states.

2) Linear feedback shift registers, with $2^n - 1$ states.

3) Modified linear feedback shift register, with 2^n states.

These state machines are based on shift registers using the absolute minimum amount of feedback, and this is applied only to the least significant register in the chain. The CLBs may be located at close proximity in the LCA device and benefit from the smallest propagation delays provided by local interconnection. The disadvantage of the shift register structure is the inability to count through the conventional binary sequence. This might or might not be of any concern, depending on the system application.

The Johnson Counter

All of the states of a six-bit Johnson counter are listed in Table 1. The "D"-type register chain shown in Figure 1 illustrates the schematic diagram of the circuit. The output from Q0 feeds directly to D1, and Q1 to D2, and so on. If these registers are configured in adjacent CLBs, direct interconnect can be used to link each one. The output of the final register Q5 is inverted and fed back to the D0 input of the first register. If the entire structure is configured in a column or row in the LCA device, a long line should be used to convey the feedback signal back to the D0 input of the least significant register. Minimum propagation delay is achieved by using long line interconnect. The advantage gained with the small propagation delay of direct short interconnections between adjacent CLBs is not lost by the use of this long feedback path.

The counter design shown in Figure 1 uses six registers and the maximum number of states reached is 12, two times the number of registers. A binary counter with six registers could reach sixty-four individual states but would need combinational feedback to each register. Feedback propagation delay could degrade the potential clocking speed of the counter. The Johnson counter, with its decreased number of states, should be used whenever maximum operational performance is needed.

STATE	Q0	Q1	Q2	Q3	Q4	Q5	HEX
0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1
2	1	1	0	0	0	0	3
3	1	1	1	0	0	0	7
4	1	1	1	1	0	0	F
5	1	1	1	1	1	0	1F
6	1	1	1	1	1	1	3F
7	0	1	1	1	1	1	3E
8	0	0	1	1	1	1	3C
9	0	0	0	1	1	1	38
10	0	0	0	0	1	1	30
11	0	0	0	0	0	1	20

Table 1. Truth Table of the Johnson Counter Counter Applications

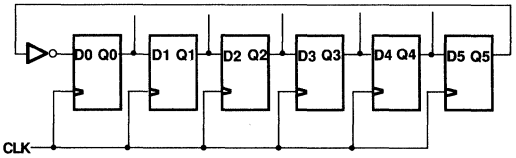


Figure 1. Johnson Counter

PAL® is a registered trademark of Monolithic Memories.
Logic Cell™ and XACT™ are trademarks of Xilinx, Inc.
P-Silos™ is a trademark of SimuCad Corporation.
IBM® is a registered trademark of International Business Machines Corporation.
PC™, PC-AT™, and PC-XT™ are trademarks of International Business Machines Corporation.

Linear Feedback Shift Register

Figure 2 shows a modification to the Johnson counter. In this three-bit counter, six states are reached. The eXclusive NOR feedback from register outputs Q1 and Q2 gives a HIGH input to D0 when both are HIGH or LOW. Again the design is based on a shift register, and this time the feedback is applied to the least significant register in the chain. The Linear Feedback Shift Register or LFSR shown in Figure 2 is implemented in three CLBs for small state machine designs. CLB placement and routing could be optimized for speed and performance. The LFSR counter would have a "stuck" state if all of the registers were set to a logic HIGH. The feedback input to D0 would remain HIGH and the XNOR inputs from Q1 and Q2 would remain stuck at a logic HIGH. The device has $2^n - 1$ states because in normal counting the "stuck" state cannot be entered. Table 2a shows the sequence and truth table of a three-register, free-running LFSR.

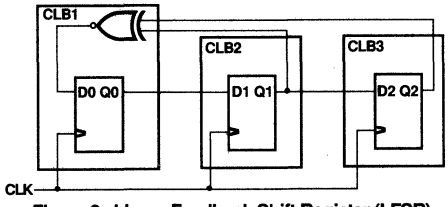


Figure 2. Linear Feedback Shift Register (LFSR)

Modified Linear Feedback Shift Register

The "stuck state" is included in the truth table shown in Figure 2b. If after the count of 3, the state machine could be modified with gating to accommodate the "stuck state" of 7, then an additional state could be used. Shifting a HIGH rather than a LOW into the least significant register would generate state 7, and all of the possible states could be reached. Table 2b shows the new entry in the truth table, and the Karnaugh map in Figure 3b shows the state assignment entry. For example, the map location Q0 = 0, Q1 = Q2 = 1 is the binary code 6. The State Diagram of the counter (Figure 3a), which is derived from the truth table (Table 2b), shows the sequential flow of each state into the next. From information in the State Diagram and State Assignment Map, a State Excitation Map may be developed as shown in Figure 3c. The next state, or excitation state from STATE 6 in Figure 3a is STATE 5. The corresponding entry of 6 in the Assignment Map is replaced by 5 in the Excitation Map and represents the transition from 6 to 5. Figure 3c is developed into Figure 3d by converting the entries to binary notation and placing the condition of the least significant bit into the Karnaugh map shown in Figure 3d. Minimization gives a Boolean equation:

$$Q0 := \sim Q1^*\sim Q2 + Q0^*\sim Q2 + \sim Q0^*Q1^*Q2 \qquad (1)$$

Equation (1) is the functional input to the least significant register in the chain. The other registers in the chain require only the direct inputs from the preceding registers, so no additional minimization is required. Figure 4 shows the final gate implementation which can be incorporated into CLB 1 replacing the XNOR gate.

The advantages of using every possible state in the counter avoids the possibility of a stuck state which, if entered, will prevent the entire design from functioning.

A very important consideration of the LFSR in CLB-intensive designs is the ability to use IOB registers in the counter. While the IOBs have no logic functionality they do have registers. The LFSR is based on the shift register, so if CLB resources run out, IOBs could be used.

COUNT	Q0	Q1	Q2
0	0	0	0
1	1	0	0
3	1	1	0
6	0	1	1
5	1	0	1
2	0	1	0
4	0	0	1

Table 2a. Truth Table of a Linear Feedback Shift Register

COUNT	Q0	Q1	Q2
0	0	0	0
1	1	0	0
3	1	1	0
7	1	1	1
6	0	1	1
5	1	0	1
2	0	1	0
4	0	0	1

Table 2b. Truth Table of a Modified Linear Feedback Shift Register

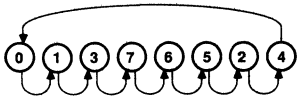


Figure 3a. State Diagram for Modified Linear Feedback Shift Register

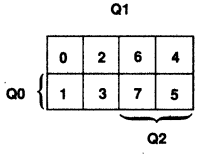


Figure 3b. State Assignment Map

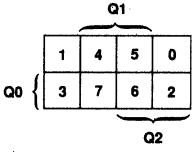
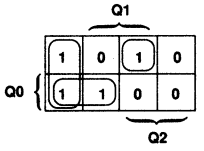


Figure 3c. State Excitation Map




```
Q0 := ~Q1*~Q2 + Q0*~Q2 + ~Q0*Q1*Q2
```

Figure 3d.

From the state excitation map, the condition of the least significant register Q0 is entered. Minimization gives the equation required for Q0 IN as a Boolean function of Q0, Q1 and Q2. When implemented, the modified LFSR will count through all states.

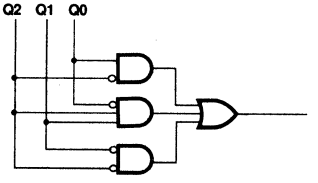
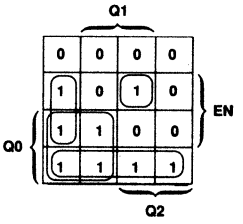


Figure 4.

If this circuit is used to replace the exclusive NOR gate in CLB1 of Figure 2 then the count shown in Table 2b will be realized enabling a count through 2^n states.

Provisions for Deeper Counting

Deeper counters may be made from cascading 3-bit LFSRs. The same three-bit module may be repeated any number of times but the original circuit needs modification. When cascading counters it is essential to use an ENABLE or CARRY input to the next stage. The Karnaugh maps, shown in Figures 3b and 3c, represent State Assignment and State Excitation, respectively. The Assignment Map can be considered as identifying HOLD conditions while the Excitation Map indicates count activity. The Karnaugh map in Figure 5 merges the information contained in both Figures 3b and 3c, and an additional ENable input qualifying COUNT and HOLD activities. The registers will count if EN is HIGH and HOLD when it is LOW. The Karnaugh map in Figure 5 was developed to derive the Boolean equation for the least significant register in the chain. The other registered cells need modifying to incorporate an enable input. Table 3 shows a truth table of the current register contents Qn with the EN input B and the input from the next least significant register A. The required output Q_{n+1} is given as a HOLD condition if B = LOW, and when HIGH, data from the A input will be clocked into the register. For each of the three registers, an EN input will enable count activity to allow cascading of 3-bit counter modules.

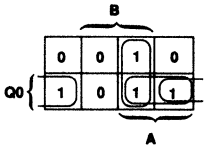



```
Q0 := Q0*~EN          ;HOLD FUNCTION.
    + Q0*~Q2*EN       ;COUNT
    + ~Q1*~Q2*EN      ;COUNT
    + Q2*Q1*~Q0*EN    ;COUNT.
```

Figure 5. Incorporating a Count Enable Input

Q_n	A	B	Q_{n+1}
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	0
1	0	1	0
0	1	1	1
1	1	1	1

Table 3.



General shift register with count enable. A = input from the next least significant register and B = enable count. Q is the state of the internal register. This Karnaugh Map has been developed from Table 3.

$$Q := Q^*\sim B + Q^*A + A^*B$$

Figure 6.

Propagation Carry

To link counter modules together, it is necessary to pass a count enable signal from one module output to the next input. Figure 7 shows the configuration of CLBs 1 to 3 linked to CLB 4, which is configured as a carry generator providing an enable output to the next modified LFSR. The next counter module will not increment until it receives an assertion on its count enable input. Figure 8 shows the Karnaugh map from which the lookahead-carry output was generated. The penultimate count in the sequence, as shown in Figure 2a is two. This count is decoded in the CLB 4 where the registered output is passed to the next counter module and delayed by one clock cycle. The concept of decoding the penultimate count, and delaying it by one clock cycle provides a fast method for propagating the lookahead carry signal. If the ultimate count is decoded by a combinational circuit, the next counter module will have to wait for the carry to propagate through logic gates before the next clock pulse can be applied. Registering the next-to-last count allows premature carry propagation, providing the carry enable to be set up prior to the next clock edge.

Figure 9 shows two identical LFSRs linked by a carry generator's CLBs 5,6 and 7 and a propagate carry configuration in CLB 8. The propagate carry circuit is a combinational circuit that enables the generated carry output, from CLB 4 to the ENB output, as shown in Figure 9. A third set of modified LFSRs may be added if a synchronous nine-bit counter is required.

Applications for a nine-bit counter might include a refresh counter for 256 K Dynamic RAMs. The EN input to CLB 1 can be used to hold off count increment activity. If an LFSR with $2^n - 1$ states were used, then one row in the DRAMs would never get refreshed, so the modified LFSR would be the appropriate choice. Moreover, the count sequence is not important because it does not have to be a binary code, just as long as the DRAMs are refreshed at regular intervals for every row.

Another application might be as a counter in a video controller. The system could count through the required states, and additional gating could be used to generate line and frame sync pulses at certain states during the count sequence.

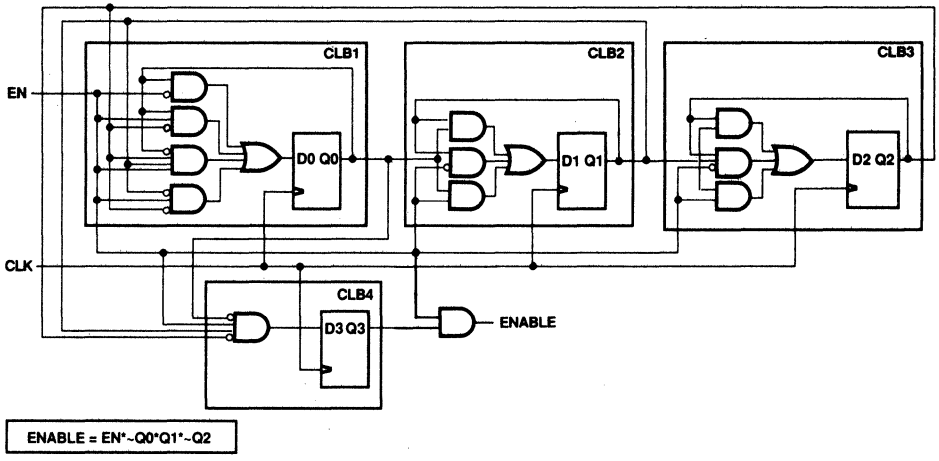


Figure 7.

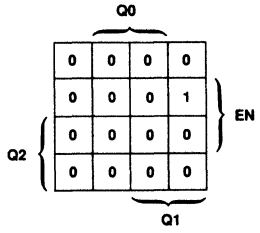


Figure 8. To Enable a Second Stage as a Synchronous Counter a Carry Output Must Be Generated from the First Three Registers. The Enable Output from CLB 4 Is Decoded from the Penultimate Count and Clocked Through the Register. When the Ultimate Count Is Reached the Enable Input to the Next Stage Is Asserted.

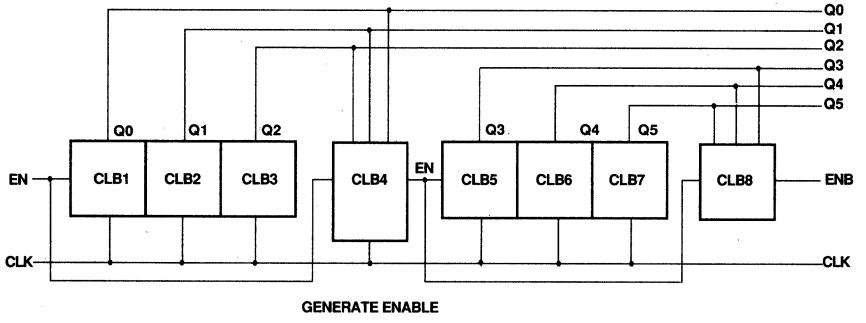


Figure 9. Two LFSRs Are Linked by the Propagate Enable Logic Contained in CLB4. CLB5, 6 and 7 Contain the Same Logic as CLB1, 2 and 3 to Create a Modulo 6 Counter with Sixty-four States. Propagate Enable is Decoded from the Ultimate Count of the Second LFSR Stage. It Can Be Used as the Enable Input to a Third LFSR Stage to Make a 9-Bit Counter.

Design of a Four-Bit LFSR

If a four-bit free-running counter was required for standalone count applications, then a design can be realized using four CLBs. The truth table for a modulo four modified LFSR is shown in Table 4. The state assignment map in Figure 10 is developed into the state excitation map shown in Figure 11. The least significant bit of the hexadecimal data is entered into the Karnaugh map as shown in Figure 12. Minimization is performed to develop the equation for the least significant register input. The actual circuit implementation is complete in four CLBs as shown in Figure 13.

	Q	Q	Q	Q	
STATE	0	1	2	3	CODE
0	0	0	0	0	0
1	1	0	0	0	1
2	1	1	0	0	3
3	1	1	1	0	7
4	1	1	1	1	F
5	0	1	1	1	E
6	1	0	1	1	D
7	0	1	0	1	A
8	1	0	1	0	5
9	1	1	0	1	B
10	0	1	1	0	6
11	0	0	1	1	C
12	1	0	0	1	9
13	0	1	0	0	2
14	0	0	1	0	4
15	0	0	0	1	8

Table 4.

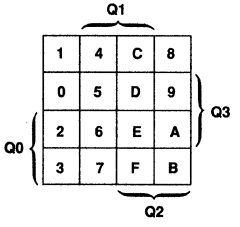


Figure 10. State Assignment Map

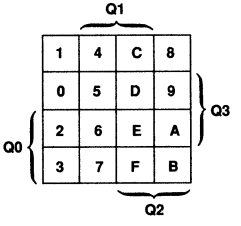


Figure 11. State Excitation Map

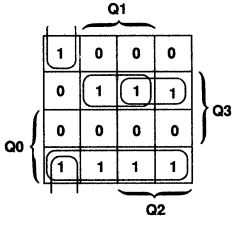


Figure 12. State Excitation Map for the Least Significant Register

```
Q0 := Q0*~Q3 +~Q0*Q1*Q3 +~Q0*Q2*Q3 +~Q1*~Q2*~Q3
```

The Binary Counter

The number of states available in a binary counter is 2^n, making it one of the most register-efficient types of counters. The three main categories of binary or weighted counters are ripple, ripple-carry, and lookahead-carry. As the amount of "look-ahead" logic decoded at each counter stage increases, the performance of the binary counter also increases. As a result, lookahead-carry counters are the fastest of the three categories of binary counters. On the other hand, the more "look-ahead" logic there is, the more decoding is needed. Increased decoding requires extra logic and routing resources which may be disadvantageous if these resources become scarce.

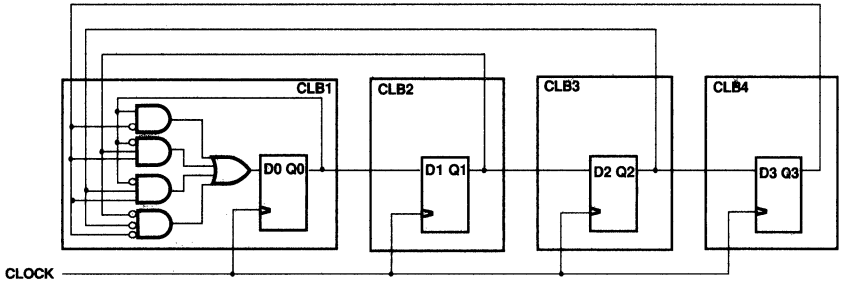


Figure 13. Modified LFSR Modulo 4 Counter

The Ripple Counter

Ripple counters are asynchronous in nature and do not generate carry signals. When the output of each counter stage clocks the next stage on the negative clock transition, a ripple effect is induced (thus the name). A schematic representation of a six-bit ripple counter is shown in Figure 14. One of the benefits of a ripple counter is that it requires few resources. Only one CLB per counter bit is needed to implement the counter and routing is simple regardless of the counter length. The tradeoff for this simplicity, however, is that ripple counters cannot be modified to be loadable or up/down which restricts their operation to be free-running, and they have lower performance.

The overall performance of a ripple counter degrades with each counter bit by one CLB delay time. This can be shown by the equation below:

$$\text{Ripple Counter} = N * \frac{(\text{Clock to Output Delay})}{\text{Clock Period}} \quad (2)$$

where N = the number of ripple counter flip-flops

The overall counter clock period must therefore be greater than or equal to the total cumulative CLB delay.

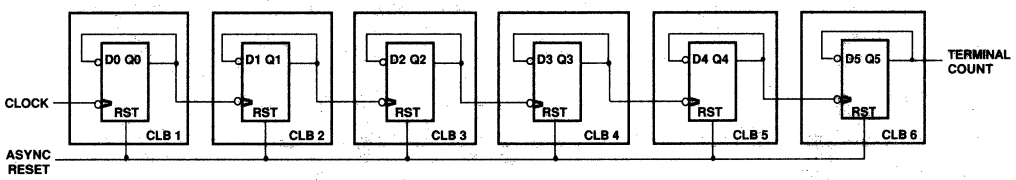


Figure 14. Schematic of a 6-Bit Binary Ripple Counter. Ripple Counters Are Easy to Design and Are Cascadable to Nearly Any Length. They Are, However, Asynchronous and Are Not Recommended for Most Designs.

The Ripple-Carry Counter

The ripple-carry counter is similar to the ripple counter. The ripple-carry counter has carry signals, however, whereas the standard ripple counter does not. Each carry signal propagates to the next counter stage to produce the counting sequence. This cascaded connection is called ripple-carry.

The ripple-carry counter differs from the ripple counter in the respect that it is synchronous in operation. Because of this, the ripple-carry counter provides more reliable operation and better performance. Performance still degrades with each additional counter stage though, due to the inclusion of combinational logic from the previous counter stage. Like the ripple counter, the ripple-carry counter requires only one CLB per counter bit and is easily cascadeable.

One example of a four-bit binary ripple-carry counter is shown in Figure 15. Here, a modulo-16 counter with count enable (CE) and reset is built with four CLBs and three levels of ripple logic. The same counter could be designed with two levels of ripple logic, if the two-input AND gate in CLB2 becomes a three-input AND gate, and the signals CE and Q0 from CLB1 were carried over to CLB2 and ANDed with Q1.

Another four-bit counter, this time with parallel enable (load), count enable, and reset, is shown in Figure 16. Here, just two ripple levels of logic were designed in, using two C8BCP MACROS from Monolithic Memories' Macrocell Library. Six CLBs, two more than the previous four-bit counter, were used in this design because of the addition of parallel enable circuitry and the reduction of ripple-carry logic levels. The Karnaugh map corresponding to the registered CLBs is shown in Figure 17.

Sometimes a designer will want to carry-over a counter implementation he or she was previously using to the LCA device. Consider an eight-bit ripple-carry counter with reset, built with two typical 74-series TTL devices: two 74-161's shown in Figure 18. Notice that the data bus as well as the LD and CET pins are not being utilized. This is wasteful. Obviously, the two 74-161 macros, available from Monolithic Memories' LCA Device Macrocell Library, requiring sixteen CLBs would also be inefficient. An equivalent counter, shown in Figure 19, requires only ten CLBs. It is built with two C8BC-rd MACROS and one C4BC-rd MACRO and contains three ripple-carry delay levels. It can be seen that implementing just the counter functions desired, instead of including additional functions not required in the counter, allows designers more control over their design. The designer can then better optimize the design for speed and performance while minimizing CLB and routing resources.

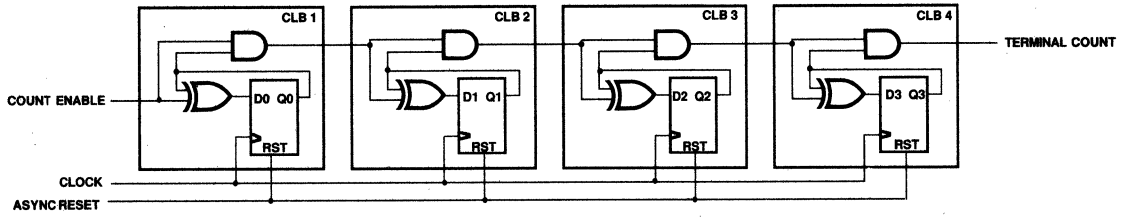


Figure 15. Schematic of a 4-Bit Ripple-Carry Counter. Although the Ripple-Carry Counter Is Synchronous, the Carry Input to the Next Stage Is Conveyed Through Combinational Logic. The Propagation Delay Due to this Logic Must be Taken into Account.

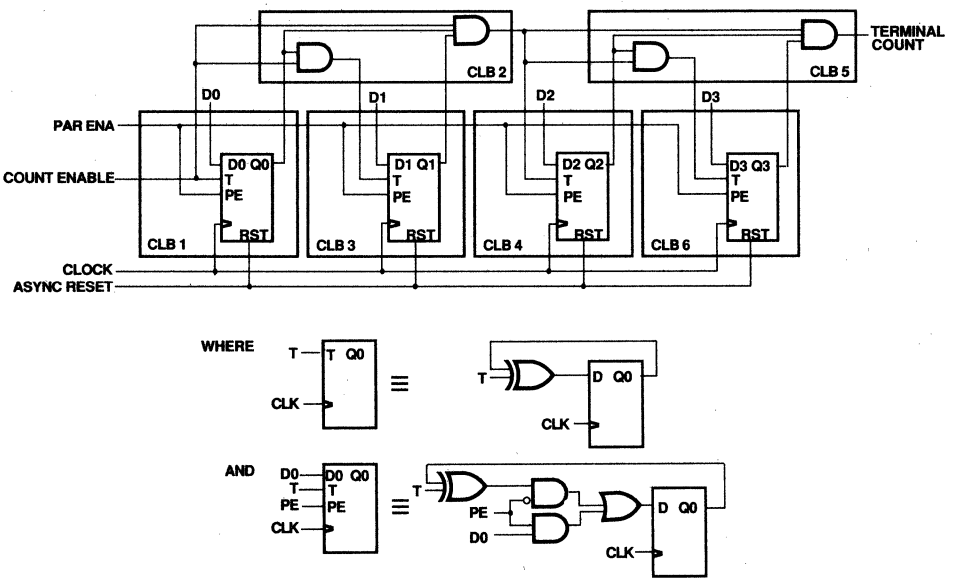


Figure 16. Schematic of a 4-Bit Ripple-Carry Counter. With Parallel Enable, Count Enable, and Reset, this Counter Only Contains Two Levels of Ripple-Carry Logic.

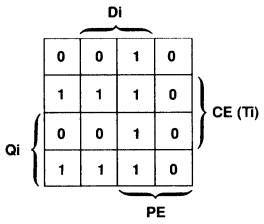


Figure 17. Karnaugh Map for the Registered CLBS in Figure 16.

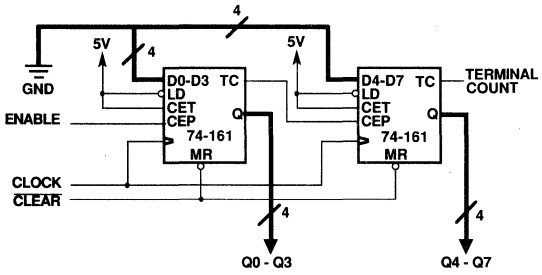


Figure 18. Schematic of an 8-Bit Counter with Reset Using Two 74-161 TTL Devices. Since All of the Available Logic Is Not Utilized, the Counter Should Be Designed More Efficiently for Use in the LCA Device.

The Lookahead-Carry Counter

When the performance of binary ripple and ripple-carry counters is not adequate, synchronous binary lookahead-carry counters can be a solution. A lookahead-carry counter incorporates all the previous counter outputs into a single lookahead-carry signal for each counter stage. This logic reduction minimizes the overall delays of the design which makes the performance of lookahead-carry counters the highest among binary counters.

With lookahead-carry counters, the complexity of the design increases with each counter bit as the decoding inputs become ever wider. As a result, these counters usually need more CLBs and routing resources to implement than other binary counters. The equations which characterize an n-bit lookahead-carry counter with count enable and parallel enable are listed below:

Q0 := ((/PARENA * CE) @ Q0) + (PARENA * DO)

Q1 := ((/PARENA * CE * Q0) @ Q1) + (PARENA * D1)

Q2 := ((/PARENA * CE * Q1) @ Q2) + (PARENA * D2) . . .

and

Qn := ((/PARENA * CE * Q1 * ... * Qn-1) @ Qn)
+ (PARENA * Dn)

To decrease the complexity of the counter, one or more of the control signals can be removed.

For a ten-bit lookahead-carry counter with reset, a minimum of fourteen CLBs are needed as shown in Figure 20. The logic for the counter was partitioned as follows:

CLB 1 Q0 := CE @ Q0

CLB 2 Q1 := (CE * Q0) @ Q1

CLB 3 Q2 := (CE * Q0 * Q1) @ Q2

CLB 4 Q02 = Q0 * Q1 * Q2 * CE

CLB 5 Q3 := Q02 @ Q3

CLB 6 Q4 := (Q02 * Q3) @ Q4

CLB 7 Q5 := (Q02 * Q3 * Q4) @ Q5

CLB 8 Q35 = Q3 * Q4 * Q5

CLB 9 Q6 := (Q02 * Q35) @ Q6

CLB 10 Q36 = Q3 * Q4 * Q5 * Q6

CLB 11 Q7 := (Q02 * Q36) @ Q7

CLB 12 Q8 := (Q02 * Q36 * Q7) @ Q8

CLB 13 Q38 = (Q36 * Q7 * Q8)

CLB 14 Q9 := (Q02 * Q38) @ Q9

The first seven bits of the counter were decoded in a similar fashion to the counter in Figure 19. The next three bits continue to minimize the amount of ripple-carry logic implemented to make it a lookahead-carry counter.

When routing a design such as this in the LCA device, it is best to place the CLBs lengthwise. Then, if the high fan-out output signals for Q02 and Q36 CLBs can be routed through "long line" interconnects, the routing-dependent delays can be reduced allowing the counter to perform at its maximum potential speed.

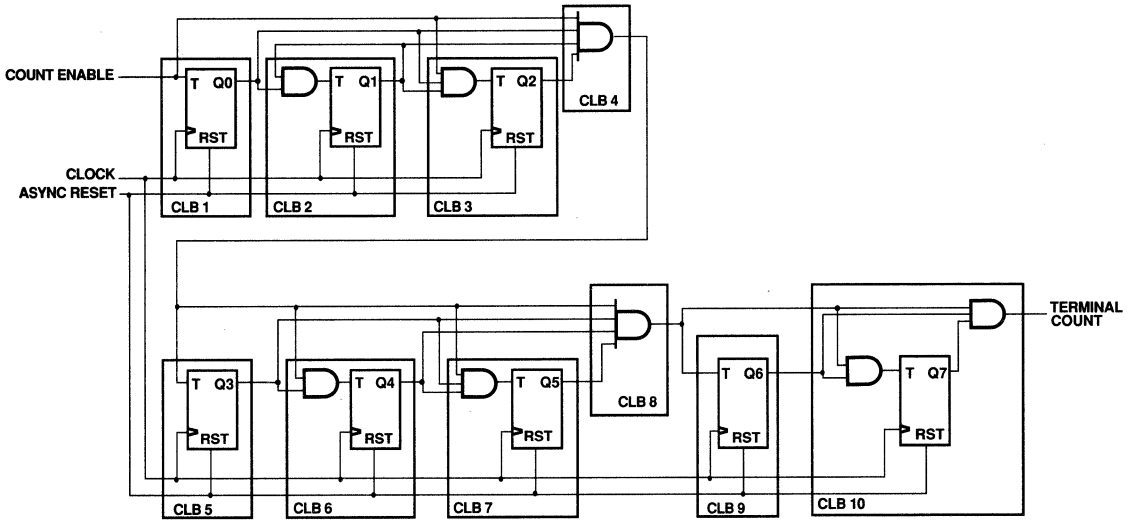


Figure 19. An Efficient Implementation of an 8-Bit Counter with Reset. This Alternative Performs Only the Desired Function While Maximizing CLB and Routing Utilization.

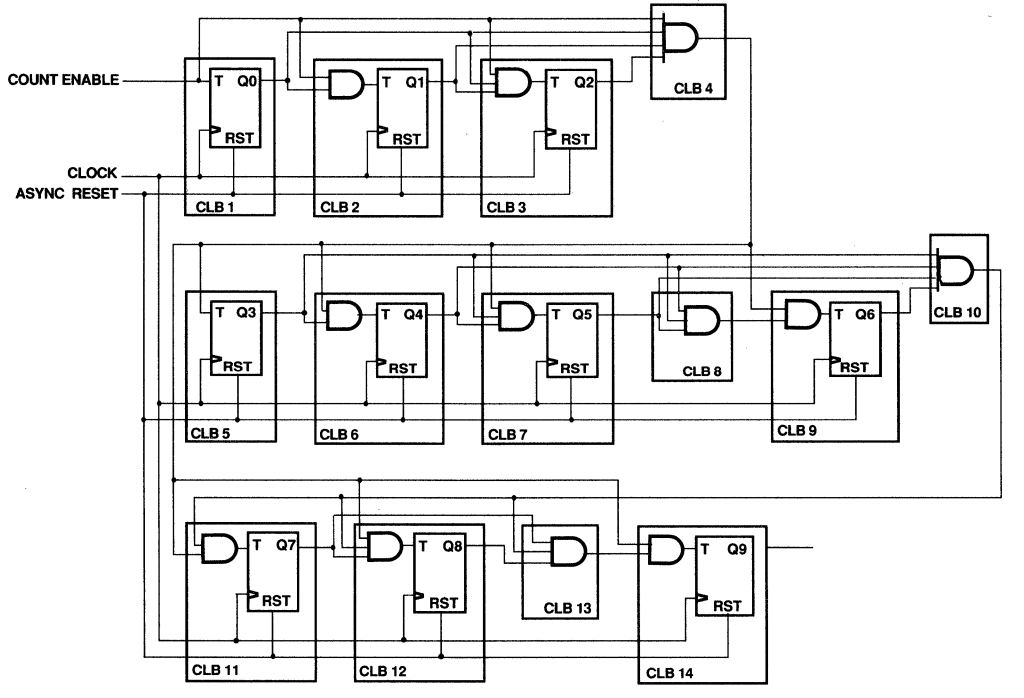


Figure 20. Schematic of a Synchronous Counter with Reset as in Figure 19, but with Lookahead-Carry Logic. As More Counter Bits Are Added, the Design Becomes Increasingly Complex Due to Wider Gating.

The Up/Down Counter

The up/down counter can be a very useful device. For instance, status counters or address counters employed in DMA systems, dual slope integrators, and delta modulation systems usually use up/down counters. In operation, an up/down counter will count UP when all previous counter bits are HIGH and will count DOWN when all previous counter bits are LOW. Thus, in an up/down counter, each register output in a CLB will toggle when:

- Q0 through Qn-1 are HIGH and the counter direction is UP, or
- Q0 through Qn-1 are LOW and the counter direction is DOWN.

This translates to:

Qn := Qn @ ((Qn-1 * Qn-2 * ... * Q1 * Q0 * UP) +
(/Qn-1 * /Qn-2 * ... * /Q1 * /Q0 * /UP)). (3)

Design of the counter can become more complicated by adding new features to it such as count enable, reset, or load (parallel enable). Because of this, it is best to limit the number of control signals which would require more complex logic. For example, a counter which only needs to be reset during initialization will not need additional reset capability since all registers are reset upon initialization of the LCA device.

Simple ripple-carry binary up/down counters require 2 * N - 4 CLBs for implementation where N is the number of counter bits. For example, eight CLBs would be needed for a six-bit up/down counter and twenty-eight CLBs would be needed for a 16-bit up/down counter. An example of a six-bit ripple-carry up/down counter configured in CLBs is given in Figure 21.

The general equations used for this ripple-carry counter design were derived from equation 3 and are as follows:

Running Total of CLBs	Equations
1	Q0 := /Q0 C2X = UP * Q0 * Q1
2	Q1 := Q1 @ ((UP * Q0) + (/UP * /Q0)) C2Y = /UP * /Q0 * /Q1
3	Q2 := Q2 @ (C2X + C2Y)
4	Q3 := Q3 @ ((Q2 * C2X) + (/Q2 * C2Y))
5	C3X = C2X * Q2 C3Y = C2Y * /Q2
6	Q4 := Q4 @ ((Q3 * C3X) + (/Q3 * C3Y))
7	C4X = C3X * Q3 C4Y = C3Y * /Q3
8	Q5 := Q5 @ ((Q4 * C4X) + (/Q4 * C4Y))

To build a simple ripple-carry 16-bit up/down counter, this algorithm would be continued:

Qn := Qn @ ((Qn-1 * Cn-1X) + (/Qn-1 * Cn-1Y))

where:

Cn-1X = Cn-2X * Qn-2
Cn-1Y = Cn-2Y * /Qn-2
Cn-1X, Cn-1Y are > or = C2X, C2Y

and

C2X = UP * Q0 * Q1
C2Y = /UP * /Q0 * /Q1.

The test file that was used to verify the logic of the six-bit up/down counter in Figure 21 is shown in Figure 22. It was generated by running the SIMGEN program included with the XACT Development System. This file, with a .DAT extension, was modified to include the desired input test vectors which would be executed. Once in the P-Silos™ simulator, "IN <filename>.DAT" is entered. This command automatically inputs the netlist to the simulator. Finally, a time period is entered which informs the simulator of the length of time the simulation should run. For instance, "SIM 0 10000" could be entered. The outputs will be plotted in a table format and will list the signals specified.

An alternative method for designing up/down counters is to use lookahead-carry logic, based on equation 3. This performance-driven method, however, is quite CLB intensive due to the wide gating of input signals required for implementation. Thus, for a six-bit lookahead-carry up/down counter, sixteen CLBs would be needed, whereas for the six-bit ripple-carry up/down counter, ten CLBs were needed. Therefore, it is advantageous to use the lookahead-carry method when maximum performance is needed.

A design of an n-bit ripple-carry up/down counter that can synchronously RESET and LOAD new values into the counter also becomes more difficult. From the general expression below, derived from state tables and Karnaugh maps, any up/down counter can be designed.

Qn = (/RESET * LOAD * Dn)
@ ((/RESET * /LOAD * UP * Qn-1 * ... * Q1 * Q0)
+ (/RESET * /LOAD * /UP * /Qn-1 * ... * /Q1 * /Q0))

where RESET and LOAD are active HIGH.

There are many ways to design up/down counters. The lookahead carry method, though it consumes a great deal of resources, usually enhances performance. The best way to design the up/down counter for systems that do not require high speed, is to use the ripple-carry method. This method is usually the most straightforward and requires the least number of resources.

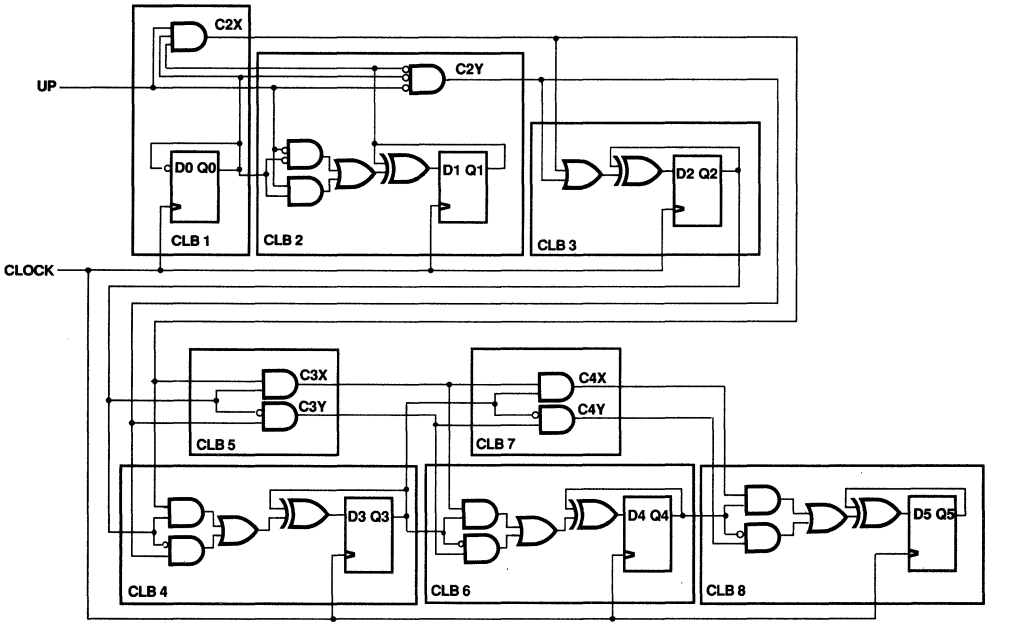


Figure 21. Schematic of a 6-Bit Up/Down Counter. Only Eight CLBs are Required for Implementation.

```
$
$ Simulation file for design 'FIG21.LCA' type '2064NL68-70'
$ Created by XACT Ver. 1.30 at 14:14:31 JUL 21, 1987
$
!INPUT FIG21.sim

$ INPUTS:
GLOBALRESET- .CLK 0 S0 1 S1 $ Initial pulse to reset latches
CLK          .CLK 0 S0 1000 S1 2000 S0 .REP 0
UP           .CLK 0 S0 128000 S1 256000 S0 270000 S1 280000 S0

 .MONITOR CLK ; UP ; Q0 Q1 Q2 Q3 Q4 Q5
 .TABLE   CLK ; UP ; Q0 Q1 Q2 Q3 Q4 Q5
```

Figure 22. Input File to P-Silos

Summary

Many ways exist for implementing counters in the LCA device. The ideal counter design for a specific application depends on the desired performance and the available logic and interconnection resources. See Table 5. For example, three main types of counters, Johnson, Linear Feedback Shift Register, and binary counters were discussed. Each of these counters utilizes different features of the LCA device but each can be optimized to perform a required function. Moreover, it was seen that by implementing only the logic necessary for a required function, resource efficiency was increased and by implementing more lookahead-carry logic or using LFSRs, performance was increased.

COUNTING METHOD	COUNTER TYPE	MODULO	CLB EFFICIENCY	ROUTING EFFICIENCY	PERFORMANCE	ADVANTAGES/ DISADVANTAGES
Non-binary	Johnson	2n	Poor	Excellent	Excellent with low modulo	High performance, easy routing Glitch-free decoding Low-register efficiency
	Linear feedback shift register or modified LFSR	$2^n - 1$ or 2^n	Good	Very good	Very good decreases with increasing modulo	Good performance at high modulos
Binary	Ripple	2^n	Excellent	Excellent	Poor	Low resources, easy routing but slow and asynchronous
	Ripple-carry	2^n	Very good	Good	Good	Good general binary counters
	Lookahead carry	2^n	Good but decreases with increasing modulo	Good but decreases with increasing modulo	Good for high-performance binary counters	Highest performance binary counter requires more CLB and routing resources

Table 5. Summary of Counter Types and Their Characteristics

Advanced
Micro
Devices

AN-161

Time Division Multiplexing with the LCA Device

C.B. Lee and Theresa Shafer

Abstract

High-speed data communication lines are efficiently utilized when low-speed signals are time division multiplexed onto high-speed lines. Time division multiplexing requires data buffering, rate adaption and data selection. The data selection fits into a single CMOS Logic Cell™ Array (LCA device). The LCA device is designed and optimized using the FutureNet® schematic capture package, Monolithic Memories' automatic place and route software, and the XACT™ Design Editor System.

Time Division Multiplexing with the LCA Device

AN-161

Introduction

The Logic Cell Array (LCA device) implements a multiplexer and counter used in time division multiplexing. With the device's flexible I/O pins, the multiplexer and counter are implemented into a single CMOS device. This application note covers time division multiplexing and the design of a multiplexer and counter. Additional information on how to design with an LCA device can be found in the LCA design methodology chapter in Monolithic Memories' LCA Design and Applications Handbook. For programming the LCA device, refer to "Configuring the LCA Device", Monolithic Memories' Application Note 182.

Principles of Multiplexing

Multiplexing efficiently utilizes data communication lines by combining multiple low-speed signals into a single, high-speed line. The two methods of performing multiplexing are Time Division Multiplexing (TDM) and Frequency Division Multiplexing (FDM). TDM divides the transmission bandwidth into equal time slots where each input signal is assigned one time slot per time cycle. Once assigned, that time slot is not used by any other input. Figure 1 shows a multiplexer combining three low-speed signals into one high-speed line. Terminal A transmits during the first time slot, B transmits during the second time slot, and C transmits during the third time slot. This sequence is repeated every time cycle. FDM, on the other hand, divides the frequency spectrum among logical channels where each channel has full bandwidth of its assigned frequencies. Analog systems usually use FDM.

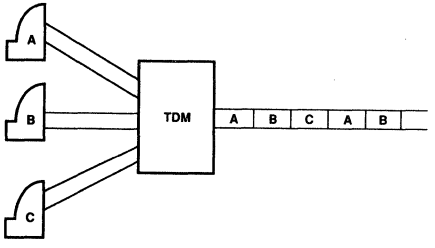


Figure 1. Time Division Multiplexing

There are many classes of TDMs including bit interleave, character interleave, statistical TDM, and T1 multiplexers. In bit interleaved TDM, each input is assigned to one time slot. Each time slot's length is one bit. Character interleaved TDM is similar to bit interleave, except that each time slot represents one character. Statistical multiplexing assigns the bandwidth into unequal slots where only the active incoming lines are assigned slots. The slots are of variable length, so each input is assigned the amount of bandwidth needed. The T1 standard specifies twenty-four 64 kbps channels multiplexed on a 1.544 Mbps high-speed link.

Data Selection/Time Slot Assignment

To transmit data from terminal A in the first time slot as shown in Figure 1, data buffering, rate adaption and data selection must be performed. Data buffering and rate adaption are required since the rates of the lines to be multiplexed are slower than the high-speed link. A serializing FIFO or shift register can implement the required buffering. Rate adaption is performed by a clock/shift scheme tailored to the exact application. Once buffered, the data must be selected in the proper order. Several methods can be used for data selection; one method is a 32-to-1 multiplexer. To implement sequential selection of input lines, an on-board counter selects the multiplexer's output. However, for random selection needed in statistical multiplexing, more flexibility is needed. A 2-to-1 multiplexer provides this flexibility by selecting between the counter and random selection inputs.

Comparison of Design Methodologies

The 32-to-1 multiplexer and counter design can be implemented in several ways. Briefly, we will evaluate discrete logic, PAL and LCA device implementations.

For this design, thirty-eight inputs, five registers, and one output are required. Conventional PLD devices do not meet this large I/O requirement. A single PLD device is not suited for this specific application because flexible I/O and buried registers are not supported. Instead, the design must be divided into four sections. The counter fits in a PALC16R6Z device and the 32-to-1 multiplexer requires three PALC20L8Z devices.

Since a large 32-to-1 multiplexer is not available as a discrete part, it must be built using smaller multiplexers. The total chip count, using discrete logic, is five devices: two 74AS850s, two 74ALS257s and one 74AS867. A combination of PAL devices and discrete logic devices is possible, however, it is also a multiple chip solution.

Monolithic Memories' LCA device is a high-density programmable CMOS circuit with flexible I/Os. It has forty pins which are user-defined as either inputs, outputs, or bidirectional. The LCA device meets the I/O requirement for this design. Since the LCA circuit allows multiple logic levels, implementing the counter does not use up output pins. Moreover an LCA device enables the 32-to-1 multiplexer with counter to be implemented in a single low-power device.

Detailed LCA Design

The M2064 LCA device in a 48-pin DIP is used in this design, although both PLCC and PGA packages are available. Eight of the forty-eight pins are reserved for programming, power, and ground, and the remaining forty I/O pins are available to the user. Table 1 shows the efficient use of the package pinout.

PIN	DESCRIPTION	TOTAL PINS
ADDR(4:0)	External Address Inputs	5 inputs
CK	External Clock Input	1 input
LD	Counter Load Input	1 input
D(31:0)	32-to-1 Multiplexer Data Inputs.	32 inputs
OUT	Output	1 output
	TOTAL I/O	40 pins

Table 1. 32-to-1 Multiplexer Pins

Figure 2 shows the block diagram of the 32-to-1 multiplexer and the counter. The select lines of the 32-to-1 multiplexer are either the address lines or the output of the counter. When LD is deasserted, the 5-bit counter sequentially selects each input. Asserting the LD signal loads the counter with the external addresses.It also selects the external address for the 32-to-1 multiplexer's select lines. This permits random input selection from the external address lines which supports statistical multiplexing.

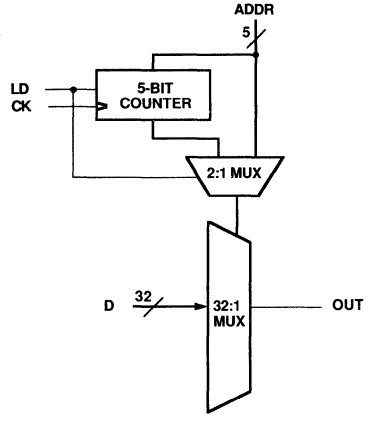


Figure 2. Block Diagram

The select signals for the 32-to-1 multiplexer are determined by the LD signal. When LD = 0, the counter selects the multiplexer's output. When LD = 1, the counter is loaded and the external address selects the output. Also, the LD asynchronously loads the 5-bit counter. Table 2 summarizes the device operation.

INPUTS	OUTPUT	FUNCTION
LD ADDR(4:0)	OUT	
0 XXXXX	D(CNT)	Counter as Select
1 ADDR(4:0)	D(ADDR)	Address as Select and Load Counter

Table 2. 32-to-1 Multiplexer Function Table

The design was entered with FutureNet's DASH schematic capture package using high-level macros. Then the schematic capture file was translated into an LCA design file. The translation software, called PIN2LCA, partitions the design into CLBs and generates the equations for the CLBs' configuration. The translation software produces an unrouted LCA design file. This design file requires final placement of CLBs and interconnect routing which was performed using Monolithic Memories' APR software, an automatic place and route program. After the design is placed and routed, it was optimized with Monolithic Memories' XACT Design Editor System. The XACT system provides a graphic interface to manually optimize the CLB logic functions and connections.

The logic functions and interconnects of an LCA device are established with CMOS memory cells, so the array is never physically altered, although it is physically programmed. A programmable LCA device can be reconfigured for the prototype. In addition, it can be reprogrammed any number of times in the target system. With reconfigurable devices such as this one, the array can also be programmed on power-up, or whenever the design details need to be changed. For volume production, a hard-array version will be available.

Entering the Design with Futurenet

The schematic entered in FutureNet's DASH is comprised of different components called from the macro library supplied by Monolithic Memories. The M8-1 and M4-1 macros are the multiplexers which implement the 32-to-1 multiplexer as shown in Figure 3. The 5-bit counter was built from INV, C16BPRD, XOR2, and FDM macros. Since the C16BPRD macro is only a 4-bit parallel-load binary counter, the XOR2 and FDM macros were added to form another bit which increased the length to five bits.The GMUX macro, a 2-to-1 multiplexer, selects between the counter and the external address lines.

To translate from FutureNet to LCA, the PIN2LCA program is invoked. This program generates an unrouted <filename>.LCA from the FutureNet's <filename>.PIN. The PIN2LCA software partitions the design into CLBs and generates each CLB's configuration.

The PIN2LCA performs logic reduction by eliminating unused logic to maximize CLB utilization. With this design, for example, C16BPRD's reset logic is not used, and PIN2LCA eliminates all the reset logic in the C16BPRD macro. Another means to maximize CLB utilization is by combining macro logic. At times, this combined logic results in a single CLB which implements logic from two different macros. For designs not utilizing all the CLBs, this logic reduction is not required and may actually hinder placement and delay tradeoffs. If it does produce delay problems, it is possible to edit the CLBs in XACT.

At this stage in the design flow, the PIN2LCA software can generate a P-SILOS simulation file. Since the <filename>.SIM was generated from an unrouted LCA design file, the simulation only has unit delays and does not have actual routing delays. The simulation verifies the logic, but does not give any timing information. It may be necessary to make changes in DASH™ depending on the simulation results.

Place and Route Using APR

Once the logic is finalized, the LCA design must be placed and routed. Monolithic Memories' APR software performs the final placement of CLBs and interconnect routing. The input to APR is an unrouted <filename>.LCA, and a routed LCA design file and report document are the outputs.

At first glance, the APR program seems cumbersome to apply to every application. However, it can be used to create an efficient design. While the software uses random placement, it does allow tailoring for specific requirements.

Initially, the multiplexer and counter design did not route completely. To achieve good placement and 100% routing completion, several things were tried. Some worked well while others did not. An overview of how 100% routing completion was achieved is explained below.

Since this design utilizes only 53% of the LCA device, delay is a larger concern than CLB utilization. In this case, placement is critical to achieve an efficient design with the desired delays. The ideal placement would be for the 5-bit counter and 2-to-1 multiplexer to be placed near the ADDR(4:0), CK, and LD signals. The four M8-1 macros should be near the 8-bit cluster of inputs. The remaining M4-1 macro should be placed near the OUT signal.

In FutureNet, the I/Os are assigned pin numbers so that the signals are clustered into groups. These I/O assignments restrict the APR's placement of the signals and are found in the <filename>.SCP. Other restrictions and options are specified when invoking the APR software. The options, "-a", "-g", "-e", and "-k", tailor the placement for this specific design.

The APR's "-a" option specifies the depth of CLB logic levels which are considered connected. In this design, "-a2" was selected because the C16BPRD and M8-1 macros are implemented in two CLB logic levels.

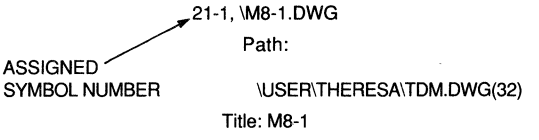
The "-g" and "-e" options are related. The "-g" option specifies the number of CLBs that should be grouped together. The "-e" option determines the number of CLBs in a group which should be evaluated for all possible placements. Since the C16BPRD and M8-1 macros contain four to six CLBs, the options, "-e4" and "-g4", were used. This particular value was selected because it was the largest possible value which maintained the e>=g relationship. Avoiding values where e<g insures as much as possible that the CLBs within each group are placed in the best possible arrangement. This maintains the groups' integrity.

The "-k" option specifies the number of shapes per group. By trial and error, "-k6" works well for this design. With "-k6", six of the best arrangements or shapes were saved for each group of CLBs. Every combination of shaped group is tried with all six other grouped shapes and evaluated for best placement. Generally, the larger values result in better placements, however, run-time grows exponentially. While larger values of "-k" were tried, they did not improve the placement significantly.

Running APR with these options and restrictions did not result in a 100% routed design. Therefore, more application-specific information was required. Noting that the APR software was not fully utilizing the high-level macro information, placement could still be optimized. The APR's report file shows that the CLBs which comprise C16BPRD or the M8-1 macros were not grouped together. Instead, the CLBs were randomly spread over the device. In a constraint file, <filename>.CST, CLBs from the high-level macros could be grouped together (see Appendix A). If necessary, the exact CLB placement could be specified in the same file.

To generate the constraint file, CLBs which implement each high-level macro were identified using the cross-reference file generated by PIN2LCA, <filename>.CRF. The <filename>.CRF file is organized into three cross-reference sections: macros, CLBs, and IOBs.

The macro cross-reference section identifies and assigns a hierarchical symbol number to each macro. The symbol number is used to generate the default CLB names.



The CLB cross-reference section of the <filename>.CRF contains the CLB names. For macros with multiple CLBs, the system assigns default names. The default CLB name consists of two parts. The first is the macro symbol number as specified in the macro section and the second number is the signal name. By grouping CLBs with related symbol numbers, all the CLBs for a given macro will be placed together.

```
CLB BD Name = '21-D03':
    F = signal '21-D03', contains:
        symbol '26-OR(2)', output signal = '21-D03'
        symbol '26-AND(3)', output signal = '26-S0'
        symbol '27-OR(2)', output signal = '21-D01'
        symbol '26-AND)1)', output signal = '26-S1'
```

The IOB cross-reference section shows the IOB assignments. For this design, the assignments are those specified in the FutureNet's schematic.

```
IOB P30: Name = 'D13', Symbol = 'PIN(40)'
    I = signal 'D13'
```

Optimizing the Design with Monolithic Memories' XACT Design Editor System

Monolithic Memories' XACT Design Editor System can increase resource utilization and performance by manually modifying the placement and routing. The XACT system provides a graphic interface to specify the CLB design. All CLB logic functions and connections can be optimized for the designs' needs. With XACT, the designer can partition the design and optimize the placement of logic blocks to gain the utilization and performance required.

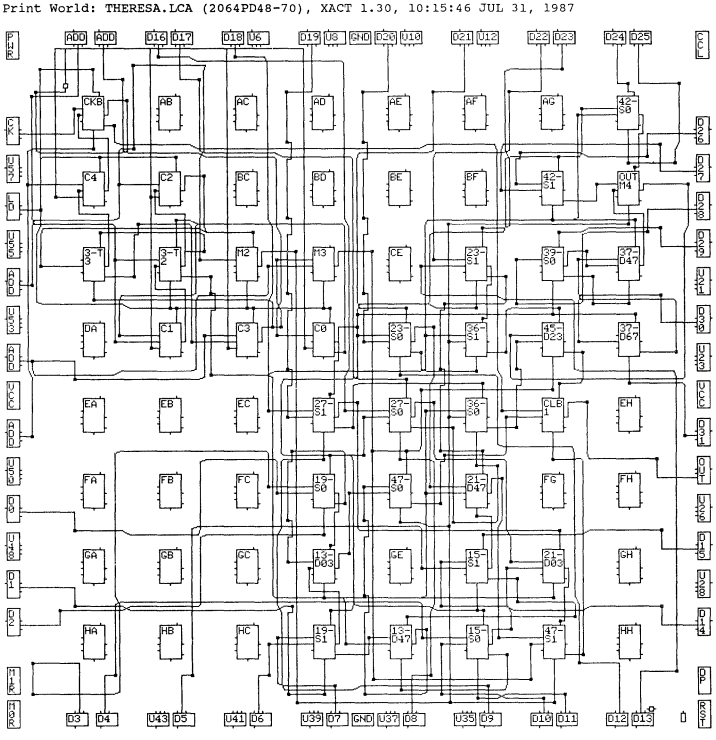
The clock line routing was optimized to reduce clock skew. Using the clock buffer resources, the common clock is driven by the global clock buffer. To optimize internal routing, "long lines" were used for signals with large fanouts such as the select lines for the multiplexers and the LD signal. Use of the "long lines" minimizes the delay for these control signals. The "long lines" were selected by manually routing the signal net through the programmable interconnect points (PIPs).

On the device itself, implementing the 5-bit counter and 2-to-1 multiplexers requires ten CLBs; and the 32-to-1 multiplexer requires twenty-four CLBs, giving a total of thirty-four CLBs. This design utilizes 53% of the LCA device.

Summary and More Information

For time division multiplexing systems, data selection can be implemented in a single CMOS device. FutureNet's DASH schematic capture with Monolithic Memories' LCA macro library was used to implement a design in an LCA device. Placement of CLBs and signal routing were performed by Monolithic Memories' APR software. Optimization, which may be required to improve performance, can be done using Monolithic Memories' XACT Design Editor System. This is useful for placing and routing critical signal nets such as clock or control signals. More information can be found in P-SILOS, FutureNet, and LCA user guides and handbooks.

This design, developed with an LCA device, is available upon request. The FutureNet drawing, LCA design and bit pattern files can be provided for programming the LCA device in an EPROM. Please ask for design XDES07.LCA.



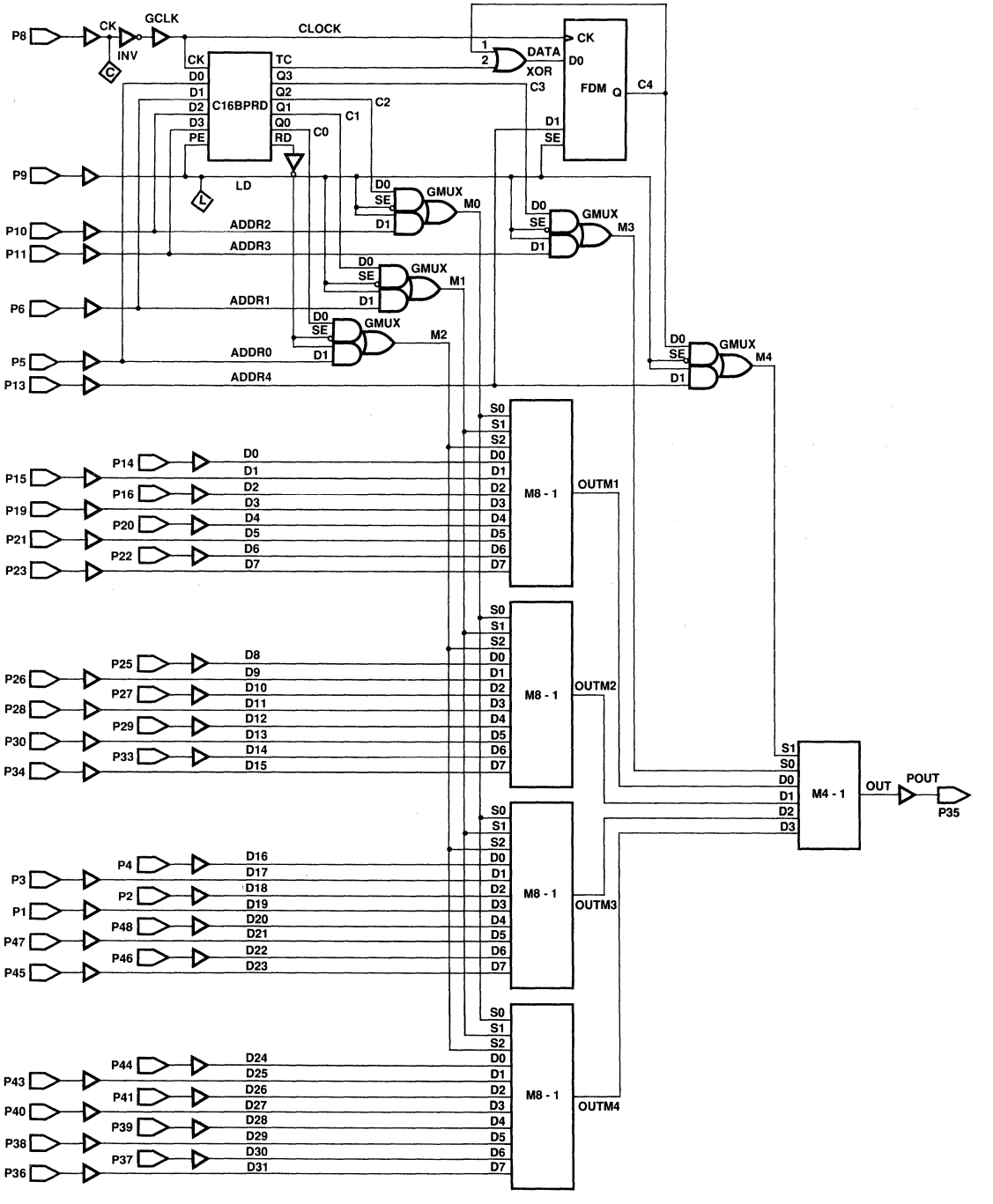


Figure 3. FutureNet Schematic

Appendix A> Constraint File

```
; CONSTRAINT FILE GROUPING MACROS IN BLOCKS

DEFINE GROUP CNT_GROUP
        3-T3 3-T2 CKB
        C4 C2 C3 C1 C0
        M3 M2;

DEFINE GROUP OUT_GROUP
        CLB1 45-D23;

DEFINE GROUP M1_GROUP
        47-S0
        19-S0 19-S1
        13-D03 13-D47;

DEFINE GROUP M2_GROUP
        47-S1
        15-S0 15-S1
        21-D03 21-D47;

DEFINE GROUP M3_GROUP
        27-S0 27-S1
        23-S0 23-S1
        36-S0 36-S1;

DEFINE GROUP M4_GROUP
        OUTM4
        42-S0 42-S1
        37-D47 37-D67
        39-S0;
```

```
$
$ Simulation file for design 'TDM.sim', type '2064N48-50'
$ Created by PIN2LCA Ver. 1.01x at 10:58:00 JUL 14, 1987
$
!INPUT TDM.sim

GLOBALRESET-      .CLK 0 S0 1 S1 $ Initial pulse to reset latches
CK.PAD            .CLK 0 0 500 1 1000 0 .REP 0
LD.PAD            .CLK 0 0 40000 1
ADDR0.PAD         .CLK 0 0 40000 0 41000 1 42000 0 .REP 40000
+ 78000 0
ADDR1.PAD         .CLK 0 0 40000 0 42000 1 44000 0 .REP 40000
+ 78000 1
ADDR2.PAD         .CLK 0 0 40000 0 44000 1 48000 0 .REP 40000
+ 78000 1
ADDR3.PAD         .CLK 0 0 40000 0 48000 1 56000 0 .REP 40000
+ 78000 0
ADDR4.PAD         .CLK 0 0 40000 0 56000 1 72000 0 .REP 40000
+ 78000 1
D0.PAD            .CLK 0 1  1000 0          40000 0 41000 1
D1.PAD            .CLK 0 0  1000 1  2000 0 40000 1 41000 0 42000 1
D2.PAD            .CLK 0 0  2000 1  3000 0 40000 1 42000 0 43000 1
D3.PAD            .CLK 0 0  3000 1  4000 0 40000 1 43000 0 44000 1
D4.PAD            .CLK 0 0  4000 1  5000 0 40000 1 44000 0 45000 1
D5.PAD            .CLK 0 0  5000 1  6000 0 40000 1 45000 0 46000 1
D6.PAD            .CLK 0 0  6000 1  7000 0 40000 1 46000 0 47000 1
D7.PAD            .CLK 0 0  7000 1  8000 0 40000 1 47000 0 48000 1
D8.PAD            .CLK 0 0  8000 1  9000 0 40000 1 48000 0 49000 1
D9.PAD            .CLK 0 0  9000 1 10000 0 40000 1 49000 0 50000 1
D10.PAD           .CLK 0 0 10000 1 11000 0 40000 1 50000 0 51000 1
D11.PAD           .CLK 0 0 11000 1 12000 0 40000 1 51000 0 52000 1
D12.PAD           .CLK 0 0 12000 1 13000 0 40000 1 52000 0 53000 1
D13.PAD           .CLK 0 0 13000 1 14000 0 40000 1 53000 0 54000 1
D14.PAD           .CLK 0 0 14000 1 15000 0 40000 1 54000 0 55000 1
D15.PAD           .CLK 0 0 15000 1 16000 0 40000 1 55000 0 56000 1
D16.PAD           .CLK 0 0 16000 1 17000 0 40000 1 56000 0 57000 1
D17.PAD           .CLK 0 0 17000 1 18000 0 40000 1 57000 0 58000 1
D18.PAD           .CLK 0 0 18000 1 19000 0 40000 1 58000 0 59000 1
D19.PAD           .CLK 0 0 19000 1 20000 0 40000 1 59000 0 60000 1
D20.PAD           .CLK 0 0 20000 1 21000 0 40000 1 60000 0 61000 1
D21.PAD           .CLK 0 0 21000 1 22000 0 40000 1 61000 0 62000 1
D22.PAD           .CLK 0 0 22000 1 23000 0 40000 1 62000 0 63000 1 73000 0
D23.PAD           .CLK 0 0 23000 1 24000 0 40000 1 63000 0 64000 1
D24.PAD           .CLK 0 0 24000 1 25000 0 40000 1 64000 0 65000 1
D25.PAD           .CLK 0 0 25000 1 26000 0 40000 1 65000 0 66000 1
D26.PAD           .CLK 0 0 26000 1 27000 0 40000 1 66000 0 67000 1
D27.PAD           .CLK 0 0 27000 1 28000 0 40000 1 67000 0 68000 1
D28.PAD           .CLK 0 0 28000 1 29000 0 40000 1 68000 0 69000 1
D29.PAD           .CLK 0 0 29000 1 30000 0 40000 1 69000 0 70000 1
D30.PAD           .CLK 0 0 30000 1 31000 0 40000 1 70000 0 71000 1
D31.PAD           .CLK 0 0 31000 1 32000 0 40000 1 71000 0 72000 1

 .MONITOR CK.PAD LD.PAD ; ADDR4.PAD ADDR3.PAD ADDR2.PAD ADDR1.PAD ADDR0.PAD ; ;
+ C4 C3 C2 C1 C0 ;
+ D0.PAD D1.PAD D2.PAD D3.PAD D4.PAD D5.PAD D6.PAD D7.PAD ;
+ D8.PAD D9.PAD D10.PAD D11.PAD D12.PAD D13.PAD D14.PAD D15.PAD ;
+ D16.PAD D17.PAD D18.PAD D19.PAD D20.PAD D21.PAD D22.PAD D23.PAD ;
+ D24.PAD D25.PAD D26.PAD D27.PAD D28.PAD D29.PAD D30.PAD D31.PAD ; OUT.PAD

 .TABLE CK.PAD LD.PAD ; ADDR4.PAD ADDR3.PAD ADDR2.PAD ADDR1.PAD ADDR0.PAD ; ;
+ C4 C3 C2 C1 C0 ;
+ D0.PAD D1.PAD D2.PAD D3.PAD D4.PAD D5.PAD D6.PAD D7.PAD ;
+ D8.PAD D9.PAD D10.PAD D11.PAD D12.PAD D13.PAD D14.PAD D15.PAD ;
+ D16.PAD D17.PAD D18.PAD D19.PAD D20.PAD D21.PAD D22.PAD D23.PAD ;
+ D24.PAD D25.PAD D26.PAD D27.PAD D28.PAD D29.PAD D30.PAD D31.PAD ; OUT.PAD
```

```
*  P - S I L O S   1U.3  *   OUTPUTS      10:26:22      07-23-87

                CL AAAAA  CCCCC DDDDDDDD DDDDDDDD DDDDDDDD DDDDDDDD O
                KD DDDDD  43210 01234567 89111111 11112222 22222233 U
                .. DDDDD        ........ ..012345 67890123 45678901 T
                PP RRRRR        PPPPPPPP PP...... ........ ........ .
                AA 43210        AAAAAAAA AAPPPPPP PPPPPPPP PPPPPPPP P
                DD .....        DDDDDDDD DDAAAAAA AAAAAAAA AAAAAAAA A
                   PPPPP                   DDDDDD DDDDDDDD DDDDDDDD D
                   AAAAA
                   DDDDD
          TIME
             0  00 00000  00000 10000000 00000000 00000000 00000000 1
           500  10 00000  00000 10000000 00000000 00000000 00000000 1
          1000  00 00000  00000 01000000 00000000 00000000 00000000 1
          1500  10 00000  00001 01000000 00000000 00000000 00000000 1
          2000  00 00000  00001 00100000 00000000 00000000 00000000 1
          2500  10 00000  00010 00100000 00000000 00000000 00000000 1
          3000  00 00000  00010 00010000 00000000 00000000 00000000 1
          3500  10 00000  00011 00010000 00000000 00000000 00000000 1
          4000  00 00000  00011 00001000 00000000 00000000 00000000 1
          4500  10 00000  00100 00001000 00000000 00000000 00000000 1
          5000  00 00000  00100 00000100 00000000 00000000 00000000 1
          5500  10 00000  00101 00000100 00000000 00000000 00000000 1
          6000  00 00000  00101 00000010 00000000 00000000 00000000 1
          6500  10 00000  00110 00000010 00000000 00000000 00000000 1
          7000  00 00000  00110 00000001 00000000 00000000 00000000 1
          7500  10 00000  00111 00000001 00000000 00000000 00000000 1
          8000  00 00000  00111 00000000 10000000 00000000 00000000 1
          8500  10 00000  01000 00000000 10000000 00000000 00000000 1
          9000  00 00000  01000 00000000 01000000 00000000 00000000 1
          9500  10 00000  01001 00000000 01000000 00000000 00000000 1
         10000  00 00000  01001 00000000 00100000 00000000 00000000 1
         10500  10 00000  01010 00000000 00100000 00000000 00000000 1
         11000  00 00000  01010 00000000 00010000 00000000 00000000 1
         11500  10 00000  01011 00000000 00010000 00000000 00000000 1
         12000  00 00000  01011 00000000 00001000 00000000 00000000 1
         12500  10 00000  01100 00000000 00001000 00000000 00000000 1
         13000  00 00000  01100 00000000 00000100 00000000 00000000 1
         13500  10 00000  01101 00000000 00000100 00000000 00000000 1
         14000  00 00000  01101 00000000 00000010 00000000 00000000 1
         14500  10 00000  01110 00000000 00000010 00000000 00000000 1
         15000  00 00000  01110 00000000 00000001 00000000 00000000 1
         15500  10 00000  01111 00000000 00000001 00000000 00000000 1
         16000  00 00000  01111 00000000 00000000 10000000 00000000 1
         16500  10 00000  10000 00000000 00000000 10000000 00000000 1
         17000  00 00000  10000 00000000 00000000 01000000 00000000 1
         17500  10 00000  10001 00000000 00000000 01000000 00000000 1
         18000  00 00000  10001 00000000 00000000 00100000 00000000 1
         18500  10 00000  10010 00000000 00000000 00100000 00000000 1
         19000  00 00000  10010 00000000 00000000 00010000 00000000 1
         19500  10 00000  10011 00000000 00000000 00010000 00000000 1
         20000  00 00000  10011 00000000 00000000 00001000 00000000 1
         20500  00 00000  10011 00000000 00000000 00001000 00000000 1
         21000  00 00000  10100 00000000 00000000 00000100 00000000 1
         21500  10 00000  10101 00000000 00000000 00000100 00000000 1
         22000  00 00000  10101 00000000 00000000 00000010 00000000 1
         22500  10 00000  10110 00000000 00000000 00000010 00000000 1
         23000  00 00000  10110 00000000 00000000 00000001 00000000 1
         23500  10 00000  10111 00000000 00000000 00000001 00000000 1
         24000  00 00000  10111 00000000 00000000 00000000 10000000 1
         24500  10 00000  11000 00000000 00000000 00000000 10000000 1
         25000  00 00000  11000 00000000 00000000 00000000 01000000 1
         25500  10 00000  11001 00000000 00000000 00000000 01000000 1
         26000  00 00000  11001 00000000 00000000 00000000 00100000 1
         26500  10 00000  11010 00000000 00000000 00000000 00100000 1
         27000  00 00000  11010 00000000 00000000 00000000 00010000 1
         27500  10 00000  11011 00000000 00000000 00000000 00010000 1
         28000  00 00000  11011 00000000 00000000 00000000 00001000 1
         28500  10 00000  11100 00000000 00000000 00000000 00001000 1
         29000  00 00000  11100 00000000 00000000 00000000 00000100 1
         29500  10 00000  11101 00000000 00000000 00000000 00000100 1
         30000  00 00000  11101 00000000 00000000 00000000 00000010 1
         30500  10 00000  11110 00000000 00000000 00000000 00000010 1
         31000  00 00000  11110 00000000 00000000 00000000 00000001 1
         31500  10 00000  11111 00000000 00000000 00000000 00000001 1
         32000  00 00000  11111 00000000 00000000 00000000 00000000 1
         32500  10 00000  00000 00000000 00000000 00000000 00000000 0
         33000  00 00000  00000 00000000 00000000 00000000 00000000 0
         39000  00 00000  00110 00000000 00000000 00000000 00000000 0
         39500  10 00000  00111 00000000 00000000 00000000 00000000 0
         40000  01 00000  00111 01111111 11111111 11111111 11111111 0
         40500  11 00000  01000 01111111 11111111 11111111 11111111 0
         41000  01 00001  01000 10111111 11111111 11111111 11111111 0
         41500  11 00001  00001 10111111 11111111 11111111 11111111 0
         42000  01 00010  00001 11011111 11111111 11111111 11111111 0
         42500  11 00010  00010 11011111 11111111 11111111 11111111 0
```

```
*  P - S I L O S   1U.3  *   OUTPUTS      10:26:22      07-23-87

            CL AAAAA  CCCCC DDDDDDDD DDDDDDDD DDDDDDDD DDDDDDDD O
            KD DDDDD  43210 01234567 89111111 11112222 22222233 U
            .. DDDDD        ........ ..012345 67890123 45678901 T
            PP RRRRR        PPPPPPPP PP...... ........ ........ .
            AA 43210        AAAAAAAA AAPPPPPP PPPPPPPP PPPPPPPP P
            DD .....        DDDDDDDD DDAAAAAA AAAAAAAA AAAAAAAA A
               PPPPP                    DDDDDD DDDDDDDD DDDDDDDD D
               AAAAA
               DDDDD
 TIME
 43000  01 00011  00010 11101111 11111111 11111111 11111111 0
 43500  11 00011  00011 11101111 11111111 11111111 11111111 0
 44000  01 00100  00011 11110111 11111111 11111111 11111111 0
 44500  11 00100  00000 11110111 11111111 11111111 11111111 0
 45000  01 00101  00000 11111011 11111111 11111111 11111111 0
 45500  11 00101  00101 11111011 11111111 11111111 11111111 0
 46000  01 00110  00101 11111101 11111111 11111111 11111111 0
 46500  11 00110  00110 11111101 11111111 11111111 11111111 0
 47000  01 00111  00110 11111110 11111111 11111111 11111111 0
 47500  11 00111  00111 11111110 11111111 11111111 11111111 0
 48000  01 01000  00111 11111111 01111111 11111111 11111111 0
 48500  11 01000  01100 11111111 01111111 11111111 11111111 0
 49000  01 01001  01100 11111111 10111111 11111111 11111111 0
 49500  11 01001  01001 11111111 10111111 11111111 11111111 0
 50000  01 01010  01001 11111111 11011111 11111111 11111111 0
 50500  11 01010  01010 11111111 11011111 11111111 11111111 0
 51000  01 01011  01010 11111111 11101111 11111111 11111111 0
 51500  11 01011  01011 11111111 11101111 11111111 11111111 0
 52000  01 01100  01011 11111111 11110111 11111111 11111111 0
 52500  11 01100  01000 11111111 11110111 11111111 11111111 0
 53000  01 01101  01000 11111111 11111011 11111111 11111111 0
 53500  11 01101  01101 11111111 11111011 11111111 11111111 0
 54000  01 01110  01101 11111111 11111101 11111111 11111111 0
 54500  11 01110  01110 11111111 11111101 11111111 11111111 0
 55000  01 01111  01110 11111111 11111110 11111111 11111111 0
 55500  11 01111  01111 11111111 11111110 11111111 11111111 0
 56000  01 10000  01111 11111111 11111111 01111111 11111111 0
 56500  11 10000  10100 11111111 11111111 01111111 11111111 0
 57000  01 10001  10100 11111111 11111111 10111111 11111111 0
 57500  11 10001  10001 11111111 11111111 10111111 11111111 0
 58000  01 10010  10001 11111111 11111111 11011111 11111111 0
 58500  11 10010  10010 11111111 11111111 11011111 11111111 0
 59000  01 10011  10010 11111111 11111111 11101111 11111111 0
 59500  11 10011  10011 11111111 11111111 11101111 11111111 0
 60000  01 10100  10011 11111111 11111111 11110111 11111111 0
 60500  11 10100  10000 11111111 11111111 11110111 11111111 0
 61000  01 10101  10000 11111111 11111111 11111011 11111111 0
 61500  11 10101  10101 11111111 11111111 11111011 11111111 0
 62000  01 10110  10101 11111111 11111111 11111101 11111111 0
 62500  11 10110  10110 11111111 11111111 11111101 11111111 0
 63000  01 10111  10110 11111111 11111111 11111110 11111111 0
 63500  11 10111  10111 11111111 11111111 11111110 11111111 0
 64000  01 11000  10111 11111111 11111111 11111111 01111111 0
 64500  11 11000  11100 11111111 11111111 11111111 01111111 0
 65000  01 11001  11100 11111111 11111111 11111111 10111111 0
 65500  11 11001  11001 11111111 11111111 11111111 10111111 0
 66000  01 11010  11001 11111111 11111111 11111111 11011111 0
 66500  11 11010  11010 11111111 11111111 11111111 11011111 0
 67000  01 11011  11010 11111111 11111111 11111111 11101111 0
 67500  11 11011  11011 11111111 11111111 11111111 11101111 0
 68000  01 11100  11011 11111111 11111111 11111111 11110111 0
 68500  11 11100  11000 11111111 11111111 11111111 11110111 0
 69000  01 11101  11000 11111111 11111111 11111111 11111011 0
 69500  11 11101  11101 11111111 11111111 11111111 11111011 0
 70000  01 11110  11101 11111111 11111111 11111111 11111101 0
 70500  11 11110  11110 11111111 11111111 11111111 11111101 0
 71000  01 11111  11110 11111111 11111111 11111111 11111110 0
 71500  11 11111  11111 11111111 11111111 11111111 11111110 0
 72000  01 00000  11111 11111111 11111111 11111111 11111111 0
 72500  11 00000  00100 11111111 11111111 11111111 11111111 1
 73000  01 00001  00100 11111111 11111111 11111101 11111111 1
 73500  11 00001  00001 11111111 11111111 11111101 11111111 1
 74000  01 00010  00001 11111111 11111111 11111101 11111111 1
 74500  11 00010  00010 11111111 11111111 11111101 11111111 1
 75000  01 00011  00010 11111111 11111111 11111101 11111111 1
 75500  11 00011  00011 11111111 11111111 11111101 11111111 1
 76000  01 00100  00011 11111111 11111111 11111101 11111111 1
 76500  11 00100  00000 11111111 11111111 11111101 11111111 1
 77000  01 00101  00000 11111111 11111111 11111101 11111111 1
 77500  11 00101  00101 11111111 11111111 11111101 11111111 1
 78000  01 10110  00101 11111111 11111111 11111101 11111111 1
 78500  11 10110  10110 11111111 11111111 11111101 11111111 0
 79000  01 10110  10110 11111111 11111111 11111101 11111111 0
 79500  11 10110  10110 11111111 11111111 11111101 11111111 0
 80000  01 10110  10110 11111111 11111111 11111101 11111111 0
```

Advanced
Micro
Devices

AN-172

Dual 32-bit Serial CRC Error Detection in an LCA Device

Karen Spesard

Abstract

The transmission and reception of digital data over local area networks (LANs) is more popular than ever. To transmit reliable information from one system to another over a LAN, an efficient error-detection technique is necessary. The error-detection scheme chosen by the IEEE for the Ethernet, a local area network developed by the Xerox Corporation, uses a 32-bit cyclic redundancy check (CRC).

A Dual 32-bit Serial CRC transmitter-receiver for the Ethernet is implemented in one Logic Cell™ Array (LCA device). The LCA device is a RAM-based CMOS circuit which is electrically programmable. It allows the designer to make design changes as necessary to accommodate other data communication protocols as well as other applications. This applications note describes the Ethernet CRC and how it is implemented in the versatile LCA device.

Dual 32-bit Serial CRC Error Detection in an LCA Device

Karen Spesard

Introduction

In transferring digital information from computer to computer, or from computer to peripheral devices, it is possible that errors in transmission may be introduced. The Cyclic Redundancy Check, or CRC, developed for the data communications industry will detect most of these errors. With one LCA device, a dual 32-bit serial CRC transmitter-receiver was implemented for the Ethernet, one of the industry's most popular local networks. This article describes the methodology used for designing the Dual 32-bit Serial CRC with an LCA device.

Overview of the CRC

Central to a serial hardware CRC system is a set of n-bit linear feedback shift registers, or LFSRs. These registers are designed to represent a fixed generator polynomial G(x). The generator polynomial is the divisor in equation 1,

$$x^n \frac{D(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}, \quad (1)$$

and D(x) is the data polynomial, which is transmitted and checked for errors. The data polynomial is prescaled by x^n, the length of the generator polynomial, to insure the remainder is always different from the data itself.

The data is shifted in the LFSR and subsequently divided by G(x). The division generates the remainder polynomial R(x) but ignores the quotient polynomial Q(x). The remainder polynomial or redundancy check-bits remain in the registers after all of the data has been shifted in. The check-bits are then shifted out and appended to the data-bits to produce the encoded data. The encoded data becomes D(x) + R(x).

The encoded data is received in a similar LFSR with the same G(x). Because addition and subtraction operations are identical in modulo-2 arithmetic, equation (1) becomes:

$$Q(x)G(x) = x^nD(x) + R(x). \quad (2)$$

Since D(x) was prescaled, and appending the remainder to the data is equivalent to adding it, the new encoded polynomial should be exactly divisible by G(x). Thus, if the remainder of this operation is zero, all of the original transmitted data-bits are assumed unaltered. If the remainder is anything other than zero, an error occurred and an error flag is set.

Generating CRC Bits and Checking Data

An example of the LFSR hardware needed to generate CRC bits serially for a three-bit G(x) is shown in Figure 1. In this example, the output of the last register is XORed, or shift subtracted, with the data-bit before feeding back to registers X0 and X2. The feedback term positions in the LFSR correspond to all but the highest power of x in the generator polynomial. In this case, the feedback positions are determined from x^2 and x^0.

The redundancy check-bits can be calculated from Table 1 for $G(x) = x^3 + x^2 + 1$ where the internal state of each register in the LFSR after every shift is shown. Therefore, if the data is 10011, the redundancy check-bits are D5 = 0, D6 = 1, and D7 = 0 after the last data-bit is shifted in (5th shift). This is because D0 = 1, D1 = 1, D2 = 0, D3 = 0, and D4 = 1. ($D(x) = 1 + 0x + 0x^2 + 1x^3 + 1x^4$ and the least significant bit, LSB, is the first bit transmitted.) The check-bits are shifted out from the LFSR by

REGISTERS	X0	X1	X2	COMMENTS
Initialize	0	0	0	If registers initialized to 0
1st shift	D0	0	D0	
2nd shift	D0⊕D1	D0	D0⊕D1	
3rd shift	D0⊕D1⊕D2	D0⊕D1	D1⊕D2	
4th shift	D1⊕D2⊕D3	D0⊕D1⊕D2	D0⊕D2⊕D3	
5th shift	D0⊕D2⊕D3⊕D4	D1⊕D2⊕D3	D1⊕D3⊕D4	Redundancy check bits
6th shift	D1⊕D3⊕D4⊕D5	D0⊕D2⊕D3⊕D4	D2⊕D4⊕D5	
7th shift	D2⊕D4⊕D5⊕D6	D1⊕D3⊕D4⊕D5	D0⊕D3⊕D5⊕D6	
8th shift	D0⊕D3⊕D5⊕D6⊕D7	D2⊕D4⊕D5⊕D6	D0⊕D1⊕D4⊕D6⊕D7	Zero remainder

⊕ = EXOR

Table 1. Calculation of Redundancy Check-Bits for $G(X) = X^3 + X^2 + 1$

disabling the feedback terms of the LFSR. The check-bits can be verified by long division as in Figure 2.

The encoded data then becomes 010 10011. To check for data integrity, the encoded data is shifted in another LFSR with the same G(x). This time the last bit (8th bit) is shifted in, and if the message was unaltered, the bits residing in the registers are zero. If the redundancy bits are not zero, an error occurred. This can also be verified by long division as in Figure 3.

In summary, CRC modifies the data polynomial so that it is exactly divisible by a fixed polynomial G(x). When the modified polynomial is received, it is checked for the exact division by G(x). If an error occurred, the RDYFLG is set.

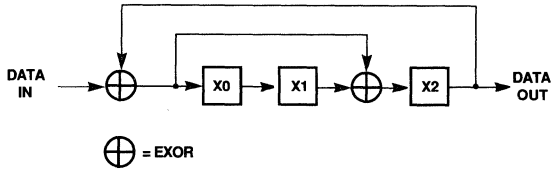


Figure 1. LFSR for G(X) = $X^3 + X^2 + 1$

```
        1001
1101 | 11001000   PRESCALED DATA
       1101       WITH 3 ZERO BITS
       -----
       11000      REMAINDER
       11010
       -----
         010
```

Figure 2. Long Division Example - With Pure Data

```
        1001
1101 | 11001010   ENCODED DATA
       1101
       -------
       011010
        11010
       -------
           0      NO REMAINDER
```

Figure 3. Long Division Example - With Encoded Data

Why Use LCA Devices in CRC?

The 32-bit serial CRC, with transmission and reception sections, can be implemented in several ways. One way would be to acquire standard CRC chips. These chips, however, only exist with specific G(x) and therefore can be inflexible. Another method would be to design the transmission and reception sections of the CRC with traditional PLD devices. This would require eight PLD devices for the dual 32-bit LFSRs (eight bits in each PLD) even without additional control logic. A third method would use an LCA device, as it would take only one to implement the dual 32-bit CRC and with more flexibility than a dedicated custom part.

In addition, the LCA device can provide designers with more control after design release than the alternative solutions. For example, the designer can change the generator polynomial or the control logic "on-the-fly" by merely reprogramming the configuration data from separate sections of an EPROM. A large EPROM, 8Kx8 or 16Kx8, can typically store three and six configuration patterns, respectively.

32-Bit CRC Design

CRC error detection is one of the best methods for checking the validity of large frames of information. It can detect all errors within n successive bits, all errors with an odd number of bits in error for even G(x), and, of course, all error patterns that are not divisible by G(x).

The IEEE-802.3/Ethernet Local Area Network Standard defines a 32-bit CRC code. This code works with data ranges from 46 to 1500 bytes, and is also used in the Autodin-II Network.

The generator polynomial used for the Ethernet 32-bit CRC is defined as:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1,$$

where the coefficients of the generator polynomial correspond to the feedback positions in the LFSR.

Figure 4 shows the block diagram of the Dual 32-bit Serial CRC. As the diagram illustrates, the transmitter portion of the CRC is controlled separately from the receiver portion.

The standard specifies that on the transmitter end, the shift registers are preloaded to ones with INITA, and CONTROLA is held HIGH while the incoming data bits are shifted in to generate the CRC. When all of the data bits have been entered, CONTROLA is held LOW and the complemented CRC is shifted out for transmission.

On the receiver end, the data bits and the complemented check bits are sent. When the end of the frame is reached, the remainder is checked. If no error occurs, the final contents of the shift register are:

X31 X0

11000111 00000100 11011101 01111011

This remainder is not 0 because the check-bits are complemented before being transmitted to the receiver.

CRC Logic Implementation

Figures 5 and 6 show the complete design for the 32-bit CRC with the standard generator polynomial described above. The generator polynomial is implemented with thirty-two registers in each LFSR. The feedback terms (Q31 xor DIN) are at registers Q0, Q1, Q2, Q4, Q5, Q7, ... Q23, and Q26 and correspond to the Ethernet generator polynomial, G(x) given above.

The registers in the LFSRs are D-type flip-flops and are clocked synchronously. In the transmitter portion, two pins - CONTROL and INIT - are respectively added for initializing the registers and controlling the feedback terms. When INIT is HIGH, all registers are initialized to HIGH. When CONTROL is HIGH, the multiplexer shifts out data bits and generates check-bits. When CONTROL is LOW, the feedback term is disabled and inverted redundancy check-bits are shifted out from the LFSR.

The five-bit synchronous ripple carry counter is enabled with the control line LOW and reset with the control line HIGH. When the five count bits reach all ones, the thirty-two check bits are shifted out of the LFSR, and the RDYFLAG is set.

The receiver LFSR is essentially the same as the transmitter LFSR. However, after data and the thirty-two check bits are received, the data is checked for errors. If an error occurred, the error flag is set.

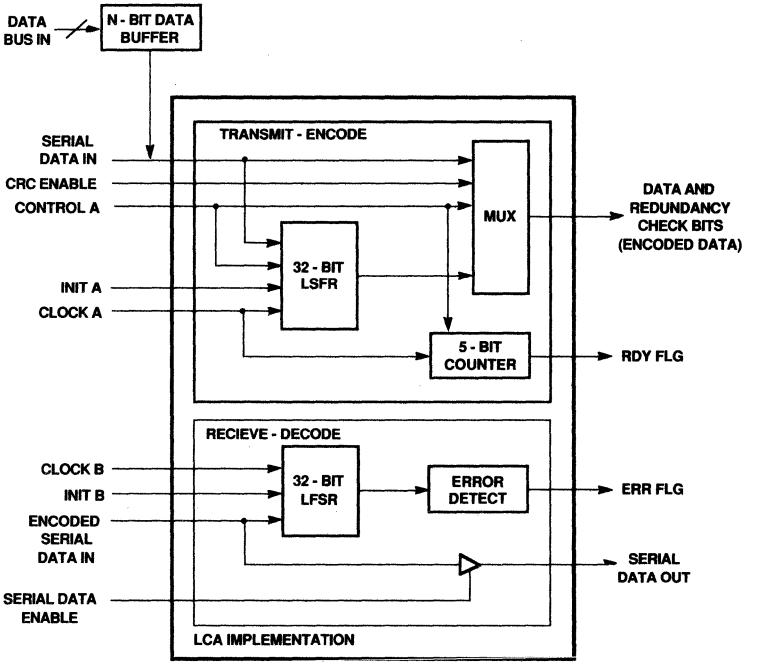


Figure 4. Block Diagram of the Dual 32-Bit Serial CRC

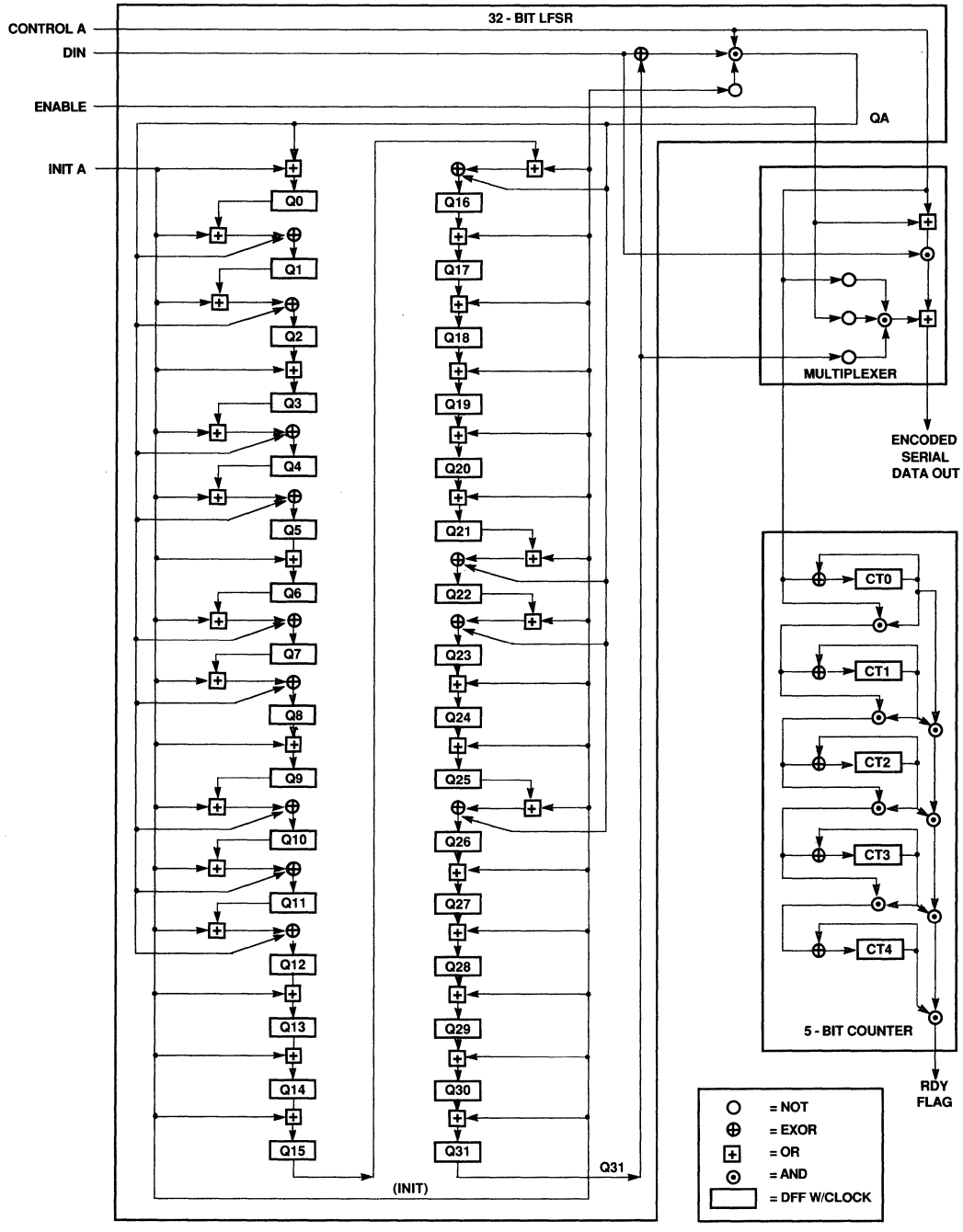


Figure 5. Logic Implementation of Transmitter-Encoder Section

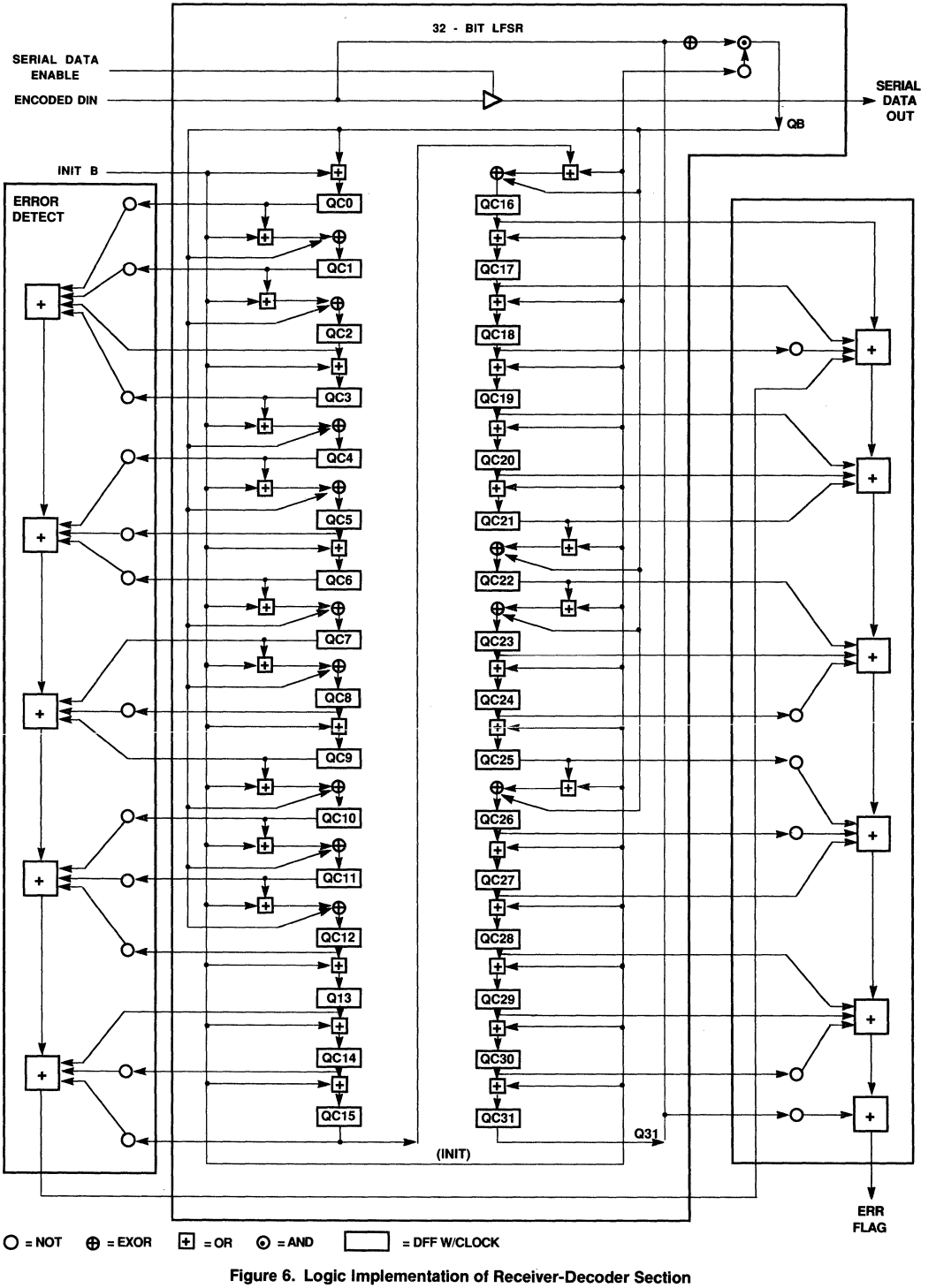


Figure 6. Logic Implementation of Receiver-Decoder Section

Logic Cell Array Implementation

The Dual 32-bit Serial CRC circuit described above is implemented in an M2018 LCA device. It contains one hundred configurable logic blocks (CLBs) and seventy-four user-configurable input/output blocks (IOBs) providing a total of 1800 usable logic gates. The logic is partitioned and placed in a graphics environment on the IBM® PC-XT™ or PC-AT™ using the Monolithic Memories' XACT™ LCA Editor.

The Monolithic Memories' XACT LCA Editor provides the tools from which the logic and interconnects (CLBs, IOBs, and programmable interconnect points) are constructed. The configuration of specified logic blocks can be made with the XACT "Edit Block" command. Once in the "Edit Block" menu, a combinational, latched, or registered equation can be selected with the mouse. The clock input polarity, set and reset functions can also be selected as required. The CLB can be configured as one output function of four Boolean input variables, or two functions of three Boolean input variables. The logic equations can be input from the PC keyboard using Boolean algebra, or from the mouse using a Karnaugh map. For an IOB, the "Edit Block" menu can be used to select pad and/or latched inputs, buffered and/or three-state control outputs, or bidirectional I/Os.

The programmable interconnect points (PIPs), are configured using the XACT "NET" and "PIN" menus. The features used from the "NET" and "PIN" menus are summarized below.

Command	Function
Name Net	Names a Net
Add Net	Names and Adds a Net
Add Pin	Adds Net and Automatically Routes a Net
Edit Net	Manually Routes a Net
Route Net	Routes Named Nets

When the nets are routed, the PIPs are programmed automatically.

An example of a CLB is shown in Figure 7. Here the output of the CLB is combinational, and the CLB is configured as a multiplexer. The multiplexer selects the serial data when CONTROL or ENABLE are HIGH and selects $\overline{Q31}$ when CONTROL and ENABLE are LOW. The Karnaugh map and the Boolean equation in the figure illustrate the logic used to implement this function. The output of the multiplexer is named ENCDATAOUT.

Figure 8 shows the configuration of the CLBs for the modulo-2 addition of the feedback signal with the data, when INIT is LOW and CONTROL is HIGH. The output again is combinational and is named QA.

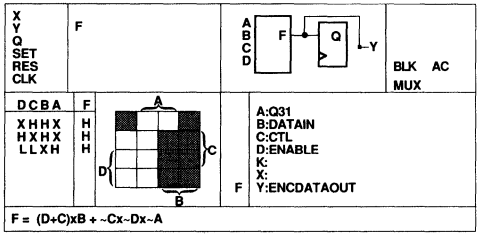


Figure 7. Mux for Data/Check Bit Select

Figures 9 to 11 show the configuration used to implement the logic for the first few shift registers, corresponding to equations in Tables 2 and 3. Note that these shift registers are configured as registered equations. Figure 12 illustrates the signal configuration of the RDYFLG in the transmitter section

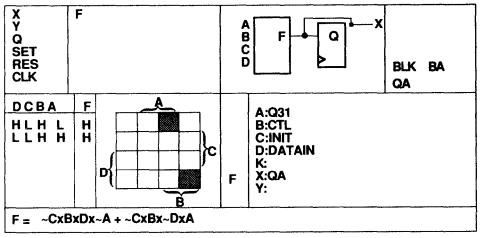


Figure 8. Modulo-2 Addition of Q31 and DATAIN

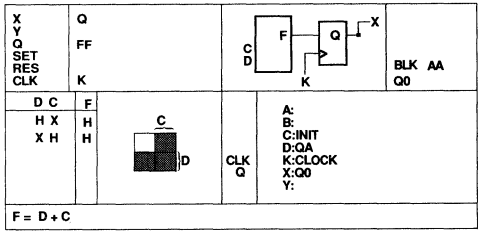


Figure 9. The 1st Bit of the LFSR

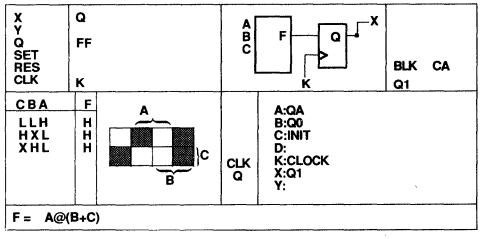


Figure 10. The 2nd Bit of the LFSR

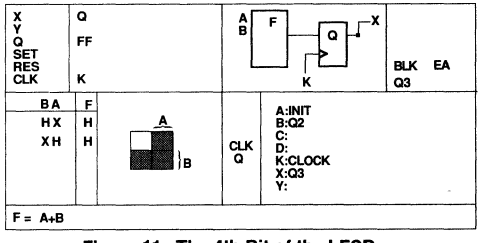


Figure 11. The 4th Bit of the LFSR

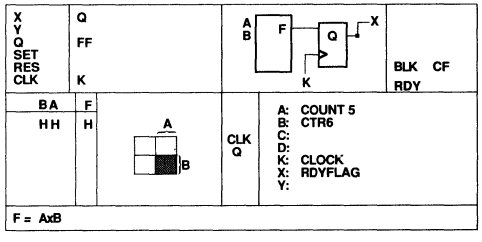


Figure 12. The RDY Flag Signal

of the LFSR. It is also configured as a registered equation. The RDYFLAG is set when COUNT5 is HIGH and CTR14 is HIGH. CTR14 is an AND gate of input bits 1 - 4.

The placed and routed Dual 32-bit CRC design requires sixty-four CLBs for the two LFSRs and two CLBs for the feedback controls. It also requires one CLB for the multiplexer, five for the counter, two for the RDYFLG and eleven for the ERRFLG.

The design, therefore, requires eighty-five CLBs and utilizes 85% of the LCA device. A 68-pin package, 2018NL68, was chosen for this implementation. This 2018NL68 has a speed grade of 70 MHz, the flip-flop toggle rate. A copy of the Monolithic Memories' XACT EDITLCA file is shown in Figure 13 with all of the CLBs and IOBs labelled.

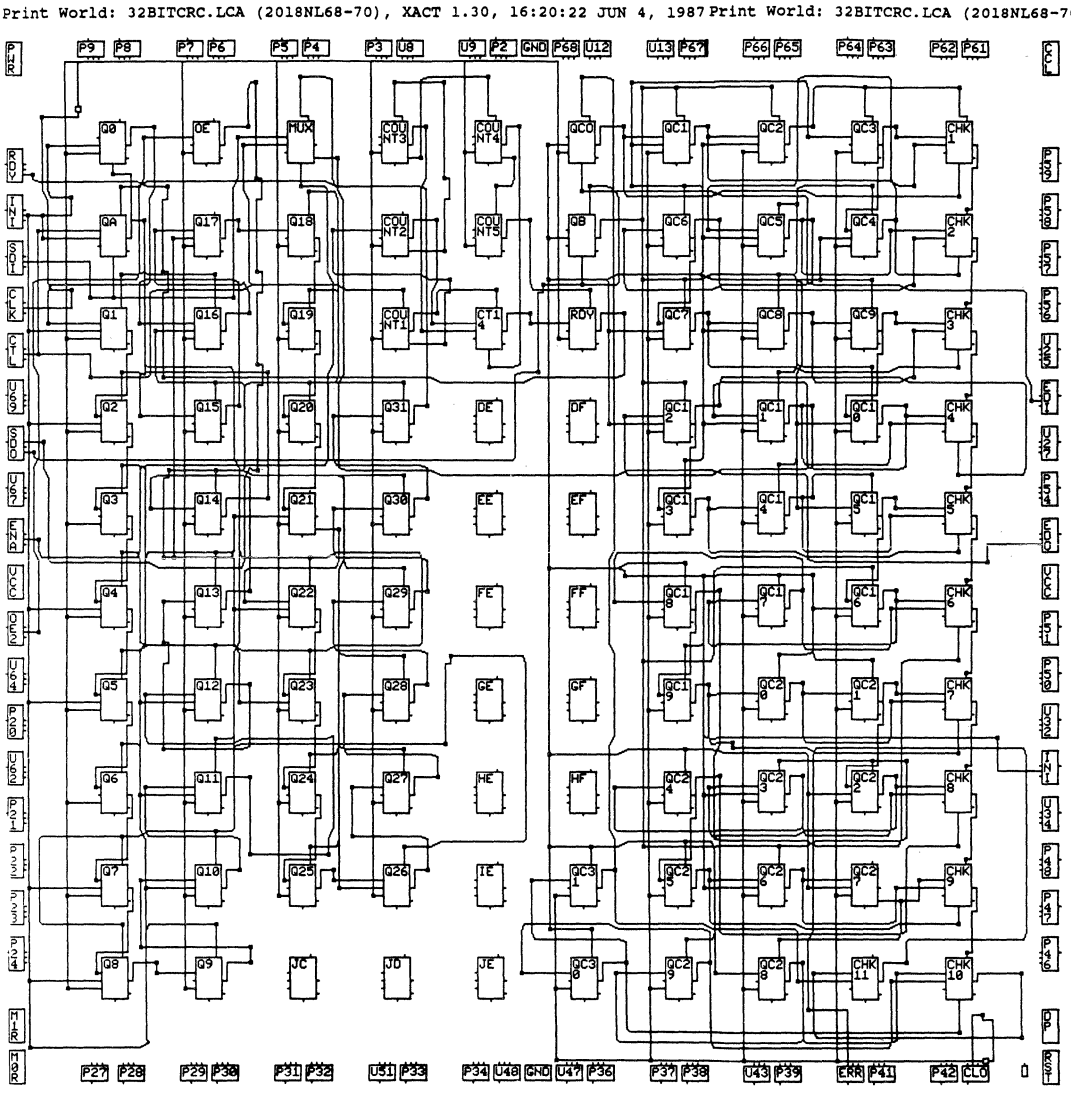


Figure 13. Monolithic Memories' XACT "EDITLCA" File for the Dual 32-Bit Serial CRC

```
                EQUATIONS FOR 32-BIT SERIAL CRC
                    TRANSMISSION SECTION

QA  =  CTL*/INIT(Q31@DIN)       ;MOD 2 ADDITION, IF CTL*/INIT
Q0  := QA+INIT                  ;SHIFT QA OR INITIALIZE
Q1  := (Q0+INIT)@QA             ;SHIFT Q0@QA OR INITIALIZE
Q2  := (Q1+INIT)@QA             ;SHIFT Q1@QA OR INITIALIZE
Q3  := Q2+INIT                  ;SHIFT Q2 OR INITIALIZE
Q4  := (Q3+INIT)@QA             ;SHIFT Q3@QA OR INITIALIZE
Q5  := (Q4+INIT)@QA             ;SHIFT Q4@QA OR INITIALIZE
Q6  := Q5+INIT                  ;SHIFT Q5 OR INITIALIZE
Q7  := (Q6+INIT)@QA             ;SHIFT Q6@QA OR INITIALIZE
Q8  := (Q7+INIT)@QA             ;SHIFT Q7@QA OR INITIALIZE
Q9  := Q8+INIT                  ;SHIFT Q8 OR INITIALIZE
Q10 := (Q9+INIT)@QA             ;SHIFT Q9@QA OR INITIALIZE
Q11 := (Q10+INIT)@QA            ;SHIFT Q10@QA OR INITIALIZE
Q12 := (Q11+INIT)@QA            ;SHIFT Q11@QA OR INITIALIZE
Q13 := Q12+INIT                 ;SHIFT Q12 OR INITIALIZE
Q14 := Q13+INIT                 ;SHIFT Q13 OR INITIALIZE
Q15 := Q14+INIT                 ;SHIFT Q14 OR INITIALIZE
Q16 := (Q15+INIT)@QA            ;SHIFT Q15@QA OR INITIALIZE
Q17 := Q16+INIT                 ;SHIFT Q16 OR INITIALIZE
Q18 := Q17+INIT                 ;SHIFT Q17 OR INITIALIZE
Q19 := Q18+INIT                 ;SHIFT Q18 OR INITIALIZE
Q20 := Q19+INIT                 ;SHIFT Q19 OR INITIALIZE
Q21 := Q20+INIT                 ;SHIFT Q20 OR INITIALIZE
Q22 := (Q21+INIT)@QA            ;SHIFT Q21@QA OR INITIALIZE
Q23 := (Q22+INIT)@QA            ;SHIFT Q22@QA OR INITIALIZE
Q24 := Q23+INIT                 ;SHIFT Q23 OR INITIALIZE
Q25 := Q24+INIT                 ;SHIFT Q24 OR INITIALIZE
Q26 := (Q25+INIT)@QA            ;SHIFT Q25@QA OR INITIALIZE
Q27 := Q26+INIT                 ;SHIFT Q26 OR INITIALIZE
Q28 := Q27+INIT                 ;SHIFT Q27 OR INITIALIZE
Q29 := Q28+INIT                 ;SHIFT Q28 OR INITIALIZE
Q30 := Q29+INIT                 ;SHIFT Q29 OR INITIALIZE
Q31 := Q30+INIT                 ;SHIFT Q30 OR INITIALIZE

ENCDATAOUT = (ENABLE+CTL)*SERDATAIN+(/ENABLE*/CTL*Q31)
RDYFLG := COUNT5*CTR14
```

Table 2.

```
                EQUATIONS FOR 32-BIT SERIAL CRC
                    RECEPTION SECTION

QB    =  /INIT*(Q31@DIN)          ;MOD 2 ADDITION, IF /INIT
QC0  := QB+INIT                   ;SHIFT QB OR INITIALIZE
QC1  := (QC0+INIT)@QB             ;SHIFT QC0@QB OR INITIALIZE
QC2  := (QC1+INIT)@QB             ;SHIFT QC1@QB OR INITIALIZE
QC3  := QC2+INIT                  ;SHIFT QC2 OR INITIALIZE
QC4  := (QC3+INIT)@QB             ;SHIFT QC3@QB OR INITIALIZE
QC5  := (QC4+INIT)@QB             ;SHIFT QC4@QB OR INITIALIZE
QC6  := QC5+INIT                  ;SHIFT QC5 OR INITIALIZE
QC7  := (QC6+INIT)@QB             ;SHIFT QC6@QB OR INITIALIZE
QC8  := (QC7+INIT)@QB             ;SHIFT QC7@QB OR INITIALIZE
QC9  := QC8+INIT                  ;SHIFT QC8 OR INITIALIZE
QC10 := (QC9+INIT)@QB             ;SHIFT QC9@QB OR INITIALIZE
QC11 := (QC10+INIT)@QB            ;SHIFT QC10@QB OR INITIALIZE
QC12 := (QC11+INIT)@QB            ;SHIFT QC11@QB OR INITIALIZE
QC13 := QC12+INIT                 ;SHIFT QC12 OR INITIALIZE
QC14 := QC13+INIT                 ;SHIFT QC13 OR INITIALIZE
QC15 := QC14+INIT                 ;SHIFT QC14 OR INITIALIZE
QC16 := (QC15+INIT)@QB            ;SHIFT QC15@QB OR INITIALIZE
QC17 := QC16+INIT                 ;SHIFT QC16 OR INITIALIZE
QC18 := QC17+INIT                 ;SHIFT QC17 OR INITIALIZE
QC19 := QC18+INIT                 ;SHIFT QC18 OR INITIALIZE
QC20 := QC19+INIT                 ;SHIFT QC19 OR INITIALIZE
QC21 := QC20+INIT                 ;SHIFT QC20 OR INITIALIZE
QC22 := (QC21+INIT)@QB            ;SHIFT QC21@QB OR INITIALIZE
QC23 := (QC22+INIT)@QB            ;SHIFT QC22@QB OR INITIALIZE
QC24 := QC23+INIT                 ;SHIFT QC23 OR INITIALIZE
QC25 := QC24+INIT                 ;SHIFT QC24 OR INITIALIZE
QC26 := (QC25+INIT)@QB            ;SHIFT QC25@QB OR INITIALIZE
QC27 := QC26+INIT                 ;SHIFT QC26 OR INITIALIZE
QC28 := QC27+INIT                 ;SHIFT QC27 OR INITIALIZE
QC29 := QC28+INIT                 ;SHIFT QC28 OR INITIALIZE
QC30 := QC29+INIT                 ;SHIFT QC29 OR INITIALIZE
QC31 := QC30+INIT                 ;SHIFT QC30 OR INITIALIZE

IF /OE2, SERDATAOUT := ENCDATAIN

ERRFLG := /Q0+/Q1+Q2+/Q3+/Q4+/Q5+/Q6+Q7+/Q8+Q9+/Q10+/Q11+/Q12+Q13
          +/Q14+/Q15+Q16+Q17+/Q18+Q19+Q20+Q21+A22+Q23+/Q24+/Q25
          +/Q26+Q27+Q28+Q29+/Q30+Q31
```

Table 3.

Configuration

The LCA device is manufactured with a programmable RAM-based CMOS process and must be configured upon power-up. This requires the loading of a configuration program into the LCA device. A number of methods for doing this exist. The Master Mode LOW was selected in this application.

In this mode, the LCA device is configured from an external EPROM containing the configuration data as shown in Figure 14. After an initial power-up delay and with the $\overline{\text{RESET}}$ pin HIGH, the LCA device reads the byte-wide configuration data from the EPROM starting at address 0000 and loads it into pins D0-D7. When the DONE pin goes HIGH, configuration is complete and the LCA device is free to perform as designed. The configuration process, lasts approximately 25 ms.

Summary

In this applications note, the principles of designing a Dual 32-bit Serial CRC in a Monolithic Memories' LCA device was described. The device implementation has many advantages which include its user-configurable logic functions, interconnects and I/O pins. This flexibility facilitates the design process significantly as standards in the data communications industry continually change. Thus, the LCA device is a viable solution for designers who wish to gain more control over their designs.

The Dual 32-bit CRC design developed for the LCA device is available upon request. The bit pattern and .LCA file will be provided for programming the LCA device in an EPROM. Please ask for design XDES09.LCA.

References

1. Nadia Sachs, *"Cyclic Redundancy Check using PAL® Devices."* AN-105, Monolithic Memories Inc., 2175 Mission College Blvd., Santa Clara, CA 95054-1592
2. Vivian Kong, *"Implementation of Serial/Parallel CRC Using PAL Devices."* AN-125, Monolithic Memories Inc., 2175 Mission College Blvd., Santa Clara, CA 95054-1592
3. IEEE Std 802.3-1985 *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications,* The Institute of Electrical and Electronics Engineers, Inc., Wiley-Interscience, 1985
4. Robert Swanson, *"Understanding Cyclic Redundancy Codes,"* Computer Design, Nov. 1975.

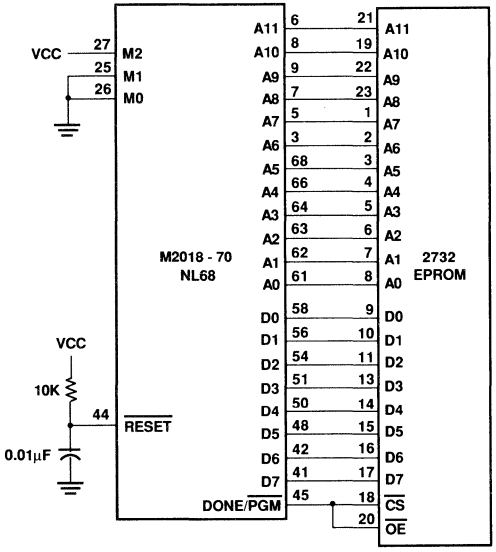


Figure 14. Master Mode Low Configuration for the Dual 32-Bit Serial CRC

Advanced
Micro
Devices

AN-178

LCA Device Implements an 8-Bit Format Converter in a PBX Switching Module

Raj Paripatyadar and C.B. Lee

Abstract

In Private Branch Exchanges, thousands of voice channels are multiplexed using time division multiplexing (TDM) techniques. Many of these voice channels may be multiplexed onto either serial or parallel channels. This application note describes an eight-bit format converter (parallel-to-serial and serial-to-parallel) circuit which transposes eight serial TDM highways into a single eight-bit parallel bus. A user-programmable Monolithic Memories' Logic Cell™ Array (LCA device) implements this circuit very efficiently using ninety-nine Configurable Logic Blocks (CLBs) out of one hundred CLBs available. Each CLB has a combinational logic section and a storage element. The circuit operates at up to 18.5 MHz when implemented in a 70-MHz CMOS, LCA M2018 device

LCA Device Implements an 8-Bit Format Converter in a PBX Switching Module

AN-178

Introduction

In a Private Branch Exchange (PBX) or Central Office (CO), incoming digitized voice circuits are "switched" or routed to their appropriate destination. The "switching" takes place while the data is in serial or parallel format. An eight-bit format converter circuit, widely used in a variety of PBX and CO architectures, translates the serial data streams into parallel data streams and vice-versa.

The eight-bit format converter circuit requires a large number of register elements. These elements, namely serial shift registers, can be implemented efficiently in an LCA device due to its high register content and flexible structure. The LCA M2018 programmable device contains one hundred configurable register elements, ninety-nine of which are used in the implementation of the converter circuit. This circuit is capable of running at 12.5 MHz and 18.5 MHz in a 50-MHz and 70-MHz LCA device, respectively.

Powerful software tools and circuit uniformity allowed the design to be laid out and simulated in just two days. However, if different support logic becomes necessary, PBX vendors are able to generate a new configuration program in the LCA reprogrammable device. Since these devices are manufactured in a CMOS technology, a complete PBX system with several LCA devices can help keep the active power consumption down. (Please refer to the LCA Design and Applications Handbook-Reference 3 for basic information regarding structure and programming aspects.)

Overview of a PBX

Central Offices (CO) and Private Branch Exchanges (PBX) provide both telephone and data services. A PBX may be considered as a localized telephone exchange serving the interconnection requirements of users in office environments. Outgoing calls are directed out of the PBX and routed through the local CO to the far destination. In general terms, a PBX or a CO consists of major blocks shown in Figure 1. Racks of peripheral equipment modules containing line cards provide access to telephone units, or to data adapter modules which provide data communication services. Trunk cards provide high-speed links to host computers, or other PBXs. Network switching modules interconnect calling parties to answering parties via digital multiplexed links. Control modules containing a Central Processor Unit (CPU), mass storage and local memory, perform the "setup" and "tear-down" of each circuit connection, as well as the monitoring of PBX activities. Central resources are modules of hardware and software which operate in a time-shared manner. The central resources include facilities such as ringing tone generator and message recording modules.

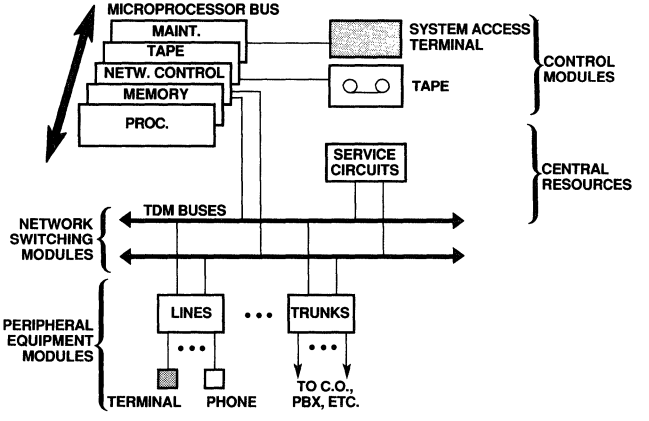


Figure 1. PBX Communication and Control Architecture

Hierarchy of Channel Multiplexing in a PBX

The main function of switching modules is to interconnect circuits between two or more parties such as in a conference call. A digital telephone unit, or a line card, contains a Coder/Decoder (CODEC) device. The CODEC device digitizes voice and transmits it into a Pulse Code Modulated (PCM) data stream on a common serial data highway at appropriate time intervals, which are assigned at the start of the call.

Analog voice is sampled at 8 KHz and passes through an eight-bit analog-to-digital converter. The digitized sample (eight-bit binary word) is transmitted serially to produce a serial data stream of (8 x 8 =) 64 Kbps. Thirty-two of these samples or CODECs are normally connected to a common highway operating at (32 x 64 =) 2.048 Mbps. Each CODEC is assigned "Serial Time Slots" of eight-bit duration, and sends its eight bits of data at the rate of 2.048 Mbps every 8 KHz. Another time slot is available on a separate highway for the receive data stream.

Therefore, a 2.048 Mbps highway can support thirty-two channels or voice connections in one direction. However, there are several ways to combine serial highways for additional voice connections. One way is to combine four highways into one high-speed highway of 8.192 Mbps. Further integration is possible, but the aggregate data rate becomes high and leads to implementation problems including timing, signal reflection, and radiation, to name a few.

Another way to integrate channels without increasing the clock rate is to convert eight serial data highways into a single eight-bit parallel bus. With this method, the serial clock rate and the parallel clock rate are the same, 2.048 Mbps. This means that data throughput on the eight serial highways of 16.384 Mbps is maintained in the eight-bit parallel bus operating at 2.048 Mbps. This works by careful alignment of the incoming parallel time slots to serial time slots on the serial highways. An eight-bit format converter scheme, sometimes called a "Corner Bender", is widely used in PBX or CO switching modules. Four of these parallel buses are combined (see Figure 2) to form a single parallel system bus that is thirty-two bits wide, yet still operates at 2.048 Mbps. The multiplexing architecture shown in Figure 2 is used in the Harris 20-20™ Integrated Network Switch (see Reference 1).

Space switching and/or time slot switching to interconnect different parties is performed either in the serial highway format or in the parallel data stream format.

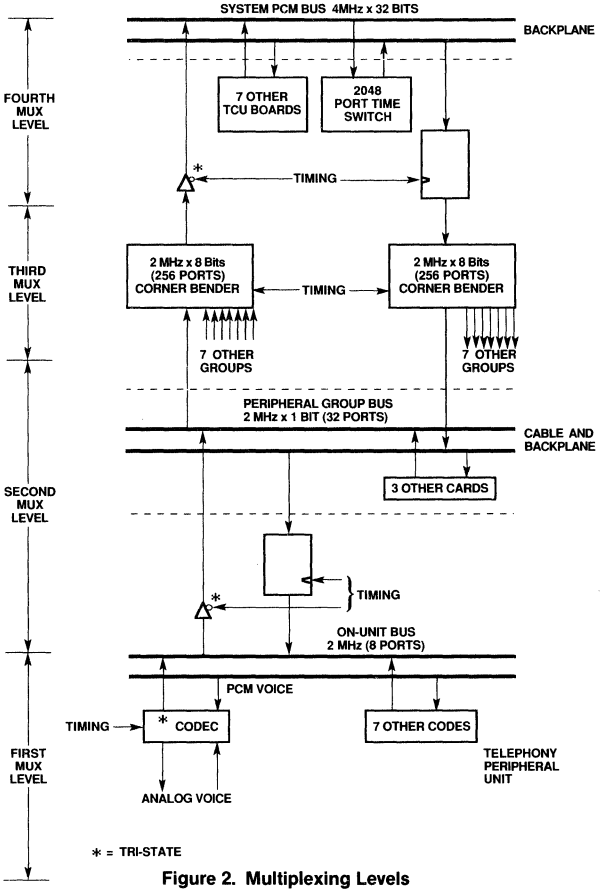


Figure 2. Multiplexing Levels

Description of the "Corner Bender" Circuit

The circuit consists of eight parallel-to-serial shift registers, and additional shift registers which provide proportional delays to synchronize the outputs. The principle of operation is shown in Figure 3, and the circuit is shown in Figure 4. Parallel data from the parallel bus is loaded with each clock into one of the eight shift registers in a rotating pattern. The data then shifts out of the shift register with each subsequent clock. After all eight registers have been loaded, the first register will be empty on the next clock. New data is loaded into this register at the same clock edge as the last bit shifts out to the next stage. Thus, a continuous flow of data through the registers is insured. The data out of the eight shift registers is skewed in relation to each other as shown in Figure 3b. The various space slots on the highways need to be aligned to keep synchronization. This is achieved by proportionally delaying the outputs. Channel 1 output is delayed by seven clocks (using a seven-bit serial shift register), channel 2 is delayed by six clocks, etc. Channel 8 output does not go through any additional delay. The final outputs of the delay registers are aligned and data is clocked out in a synchronous fashion. See Figure 3c.

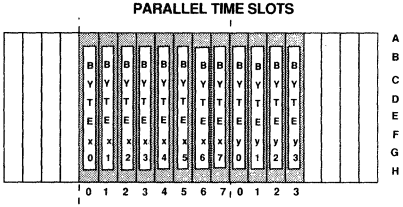


Figure 3a. Parallel Bus (8 Bit), 2.048 Mbyte/sec

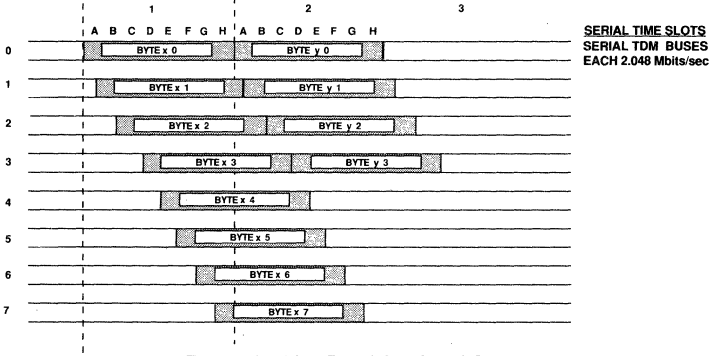


Figure 3b. After Parallel to Serial Conversion

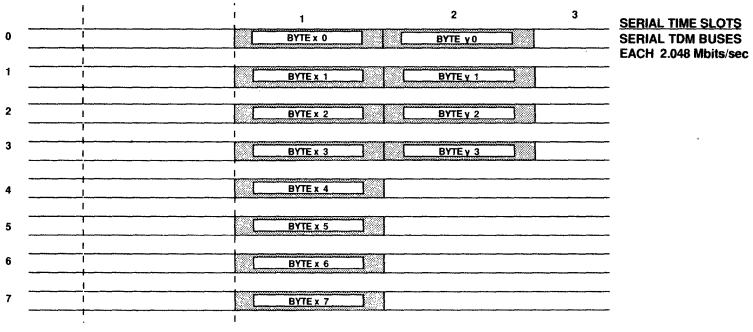


Figure 3c. After Appropriate Time Delay to Synchronize

Figure 3. Principles of "Corner Bending"

Figure 4 also shows the additional circuitry needed to control the shift registers. A preloadable three-bit counter keeps count of eight clock pulses. The parallel-to-serial bus conversion can be "programmed" to start in any register by setting the appropriate binary value on the counter preload inputs and applying a pulse to the sync input. If the loading sequence produced by the counter is not required, it can be disabled by connecting the "clock" to "sync" input. At each positive clock edge, the register loaded will depend upon the data at the counter inputs on the previous positive clock edge. The 3-to-8 decode circuit produces load pulses to latch parallel data into the shift registers. Figure 5 shows the parallel-to-serial conversion data matrix.

The description above shows the conversion of parallel data into eight streams of serial data. However, the same circuit also performs serial-to-parallel conversion. A serial eight-bit data stream on one of the eight inputs appears as an eight-bit parallel word on the eight outputs. Successive parallel words appearing at the eight outputs correspond to the serial data on each of the eight inputs in rotation. See Figure 5.

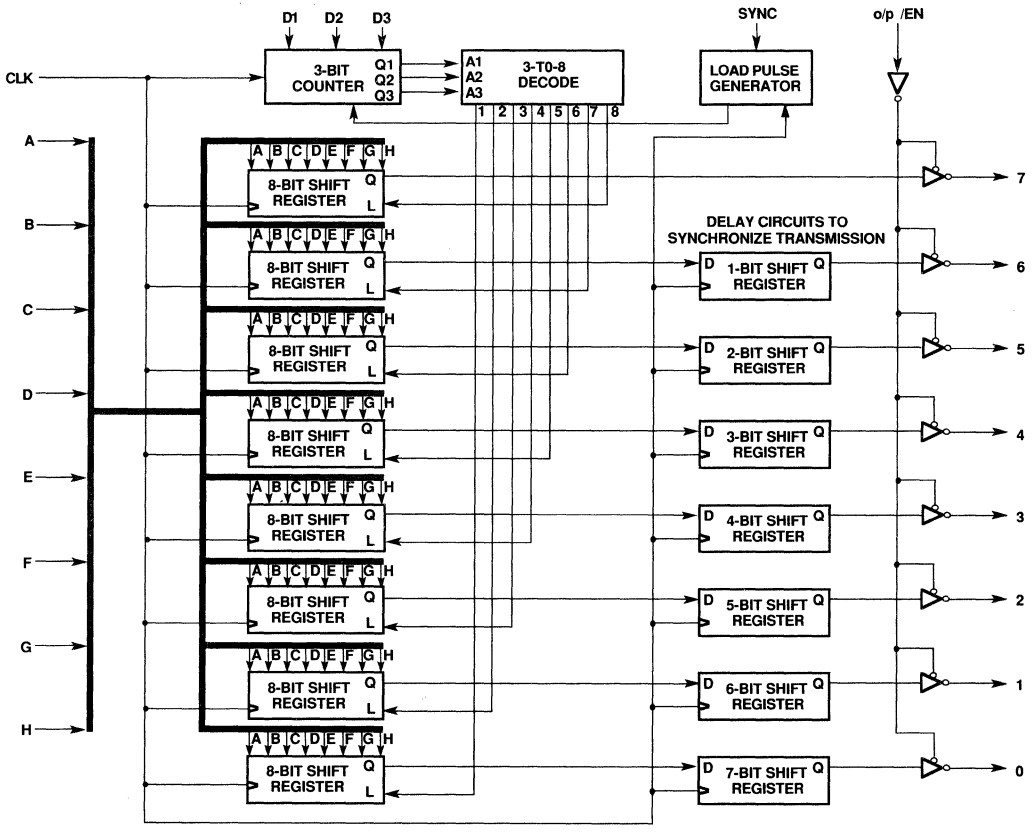
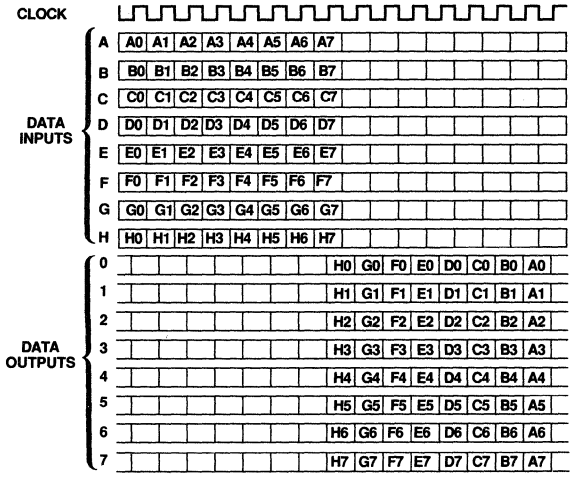


Figure 4. 8-Bit Format Converter ("Corner Bender") Circuit



PARALLEL TO SERIAL CONVERSION
A0, B0, C0, D0, E0, F0, G0, H0 = IN(I)0, IN(I)1, IN(I)2, IN(I)3, IN(I)4, IN(I)5, IN(I)6, IN(I)7
A1, B1, C1, D1, E1, F1, G1, H1 = IN(II)0, IN(II)1, IN(II)2, IN(II)3, IN(II)4, IN(II)5, IN(II)6, IN(II)7

SERIAL TO PARALLEL CONVERSION
A0, A1, A2, A3, A4, A5, A6, A7 = IN(I)0, IN(I)1, IN(I)2, IN(I)3, IN(I)4, IN(I)5, IN(I)6, IN(I)7
B0, B1, B2, B3, B4, B5, B6, B7 = IN(II)0, IN(II)1, IN(II)2, IN(II)3, IN(II)4, IN(II)5, IN(II)6, IN(II)7

*** IN(I)X ARE THE INPUT 8-BIT WORDS**

Figure 5. Data Conversion

Comparison of Implementation Techniques

The "Corner Bender" circuit can be implemented in several ways. The efficiency of an LCA design can be compared to that of a design in SSI/MSI logic devices or Erasable Programmable Logic devices (EPLDs). Table 1 contains the comparison of package counts with the different alternatives.

The circuit implemented in 74LSxx devices requires nineteen packages. Alternatively, sixteen simple PLD devices (such as a PAL20R8 or a PAL20R4) are required to implement the same circuit. Since the circuit is register intensive and not designed for high speed, PLDs are not chosen for this application.

The functionally larger EPLD devices, EP1200 and EP1800, are more register intensive and contain twenty-eight and fourty-eight register elements, respectively. Nevertheless, four EP1200 devices, or two EP1800 plus a smaller PLD device, is required to implement this design.

The entire Corner Bender circuit fits neatly into a single LCA 2018 device. It uses ninety-nine out of the one hundred available internal macrocells and demonstrates efficient implementation of shift registers and small counters. An LSI device from Plessey (Reference 2) is available to implement the same circuit. However, it is an NMOS device and limited to speeds of under 2 MHz.

EQUIVALENT CIRCUIT IN SSI/MSI DEVICES		PACKAGES
Parallel in serial out shift Registers	74LS165	8
Serial in Parallel out shift Registers	74LS164	7
4-bit counter	74LS161	1
3 to 8 decoder	74LS138	1
Three-state buffers	74LS241	1
EQUIVALENT CIRCUIT IN EPLDs		
a) EP1200	28 macro cells each	4
b) EP1800	48 macro cells each	2
plus PAL device e.g., 16R4		1
EQUIVALENT CIRCUIT IN MMI LCA DEVICE		
LCA 2018 device	100 macro cell	1

Table 1. Implementation Alternatives for the 8-Bit Format Converter Circuit "Corner Bender"

"Corner Bender" Design in an LCA Device

A count of the register elements in the circuit shows a need of sixty-four elements for the eight-shift registers, twenty-eight elements for the delay registers, three elements for the counter, and four elements for the 3-to-8 decode circuit. Each configurable logic block (CLB) in the LCA device can produce two outputs. Hence, only four CLBs are required for the 3-to-8 decode circuit. Since, each CLB contains one register element, the total count of CLBs is therefore ninety-nine. The LCA 2018 contains one hundred CLBs. Twenty-two input/output blocks (IOBs) were used. Figure 6 shows the layout of the design in the LCA device. The circuit fits into a 68-pin PLCC package.

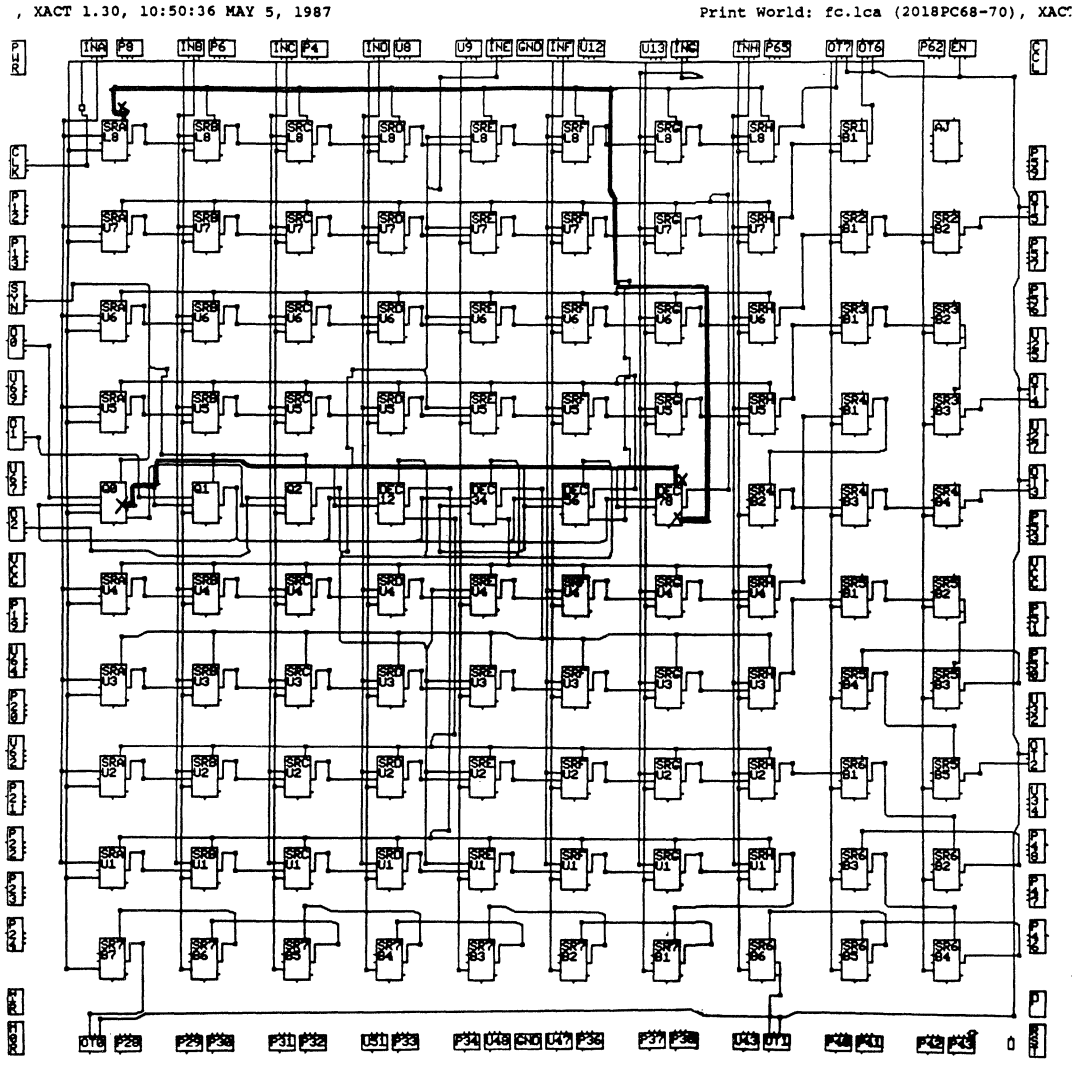


Figure 6. Layout Diagram of "Corner Bender" Circuit on LCA Device of Size 10x10

The Monolithic Memories' XACT™ Design Editor is used to create the design by implementing the appropriate logic functions in CLBs and IOBs. An example of a CLB for one bit of the eight-bit shift register in this design is shown in Figure 7. Input A is the decoder input and it is ANDed with input B, the data to be shifted. To implement the eight-bit shift registers from the one-bit shift register, eight of the one-bit shift registers were linked together so that the output of each register became an input to the next register. This is shown in Figure 8 where each CLB represents one bit of the eight-bit shift register.

The three-bit counter and the 3-to-8 decoder in this design were implemented in CLBs as shown in Figures 9 and 10. The counter is a synchronous binary counter with ripple carry and parallel load. The decoder is a standard 3-to-8 decoder. The outputs of the counter become inputs to the decoder, whereas the outputs of the decoder are used to decode the eight 8-bit shift registers.

With the XACT Development System, the designer can optimally arrange the logic blocks on the LCA device in order to minimize net delays between each block. With this in mind, the layout for the design is described below:

Four of the eight-bit parallel-to-serial shift registers were placed starting from the top left-hand edge of the LCA device (see Figure 7). The three-bit counter (three CLBs) and the 3-to-8 decoder (four CLBs) were then placed on the following row. The next four rows contained the last four parallel-to-serial shift registers. This allowed the shift register select lines to have minimal delay spread when accessing all eight shift registers. The seven serial-to-serial shift registers were placed in the remaining CLBs as uniformly as possible.

The optimum placement and distribution of configured blocks in the array is influenced by the performance needs of the system. Blocks placed in close proximity can use local interconnection resources which incur short signal propagation delays, whereas blocks placed further apart must use either "long lines" or other interconnection resources. Manual optimization using the delay efficient "long lines" was performed for the most critical net connections. After routing completion, the longest delay between two clock pulses was the delay for the counter to change state, the state which is decoded via the 3-to-8 decoder and selects the appropriate shift register to load the parallel data (see Figure 7). This delay was 54 ns, 79 ns, and 106 ns, respectively for the 70-, 50-, and 33-MHz versions of the device. The delays were measured using the XACT simulation package by invoking the timing delay calculator. This translates to a maximum circuit operating speed of about 18.5 MHz for the 70-MHz version of the device, or 9.43 MHz for the 33 MHz version.

Although fabricated in a CMOS technology, the inputs to the LCA device can be made either TTL or CMOS compatible. For high fan-out CMOS or LS TTL-compatible loads, the output buffers of the LCA device are capable of driving 4 mA. Moreover, each output buffer can be put into a HIGH-Z state for bus-driving applications. This features was also used in the design of the eight-bit format converter.

More information about entering a design with the XACT Development System is included in the LCA Design and Applications Handbook (Reference 3). Information about configuring the LCA device is described in the "Configuring the LCA Device" applications note.

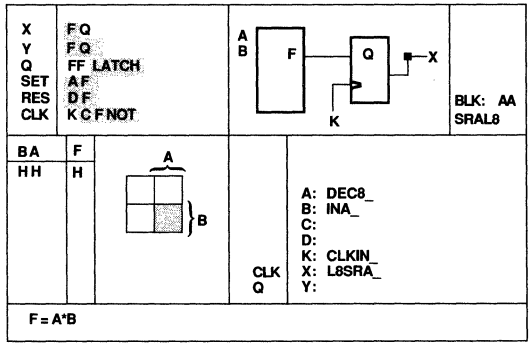


Figure 7. One Bit of an 8-Bit Serial - Parallel Shift Register

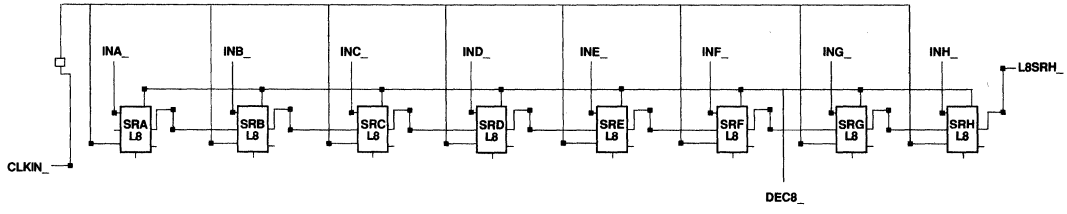


Figure 8. CLB Schematic Output for the 8-Bit Shift Register

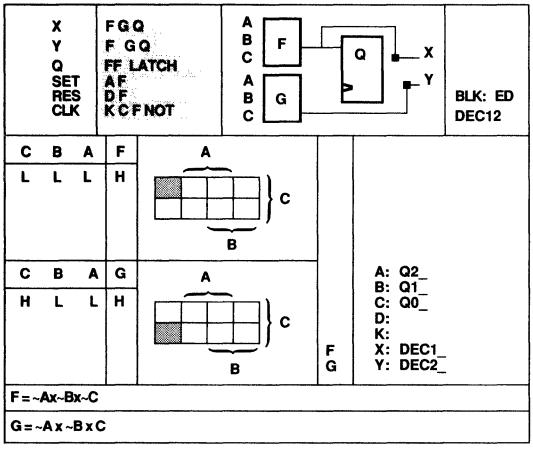


Figure 9a. Decode Outputs 1 and 2 of 3-to-8 Decoder

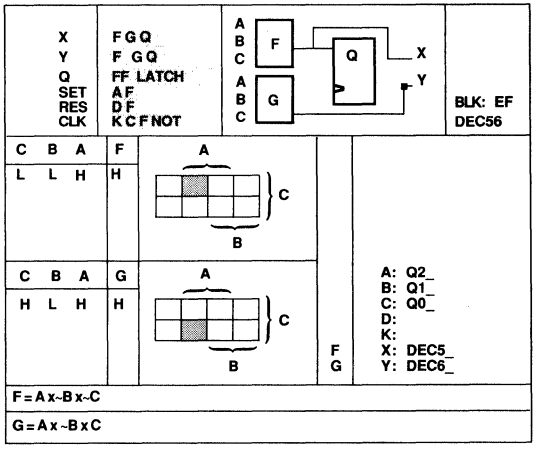


Figure 9c. Decode Outputs 5 and 6 of 3-to-8 Decoder

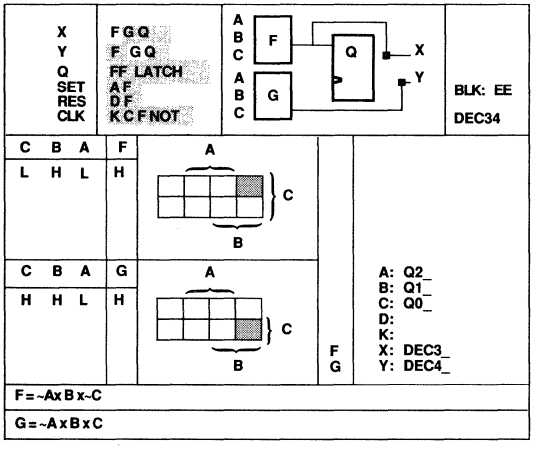


Figure 9b. Decode Outputs 3 and 4 of 3-to-6 Decoder

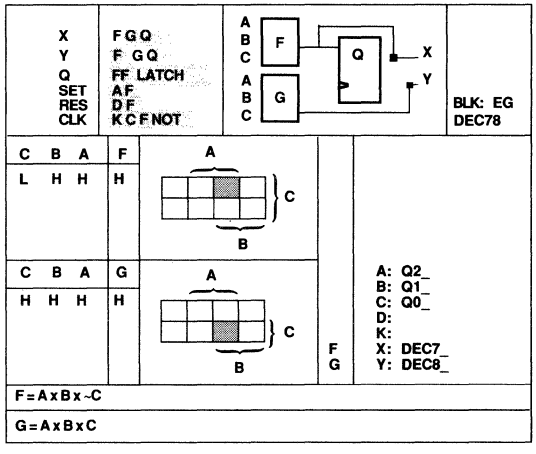


Figure 9d. Decode Outputs 7 and 8 of 3-to-8 Decoder

Figure 9. CLB Configuration of a 3-to-8 Decoder

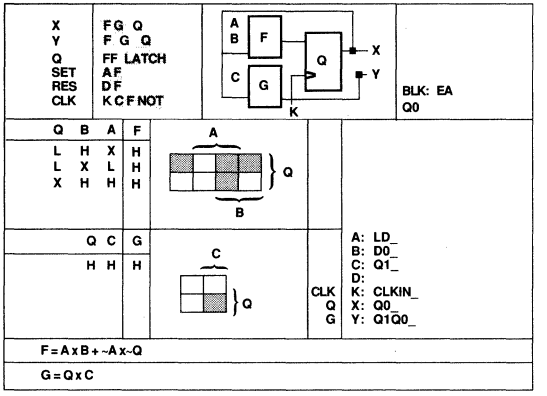


Figure 10a. 1st Bit of a 3-Bit Counter

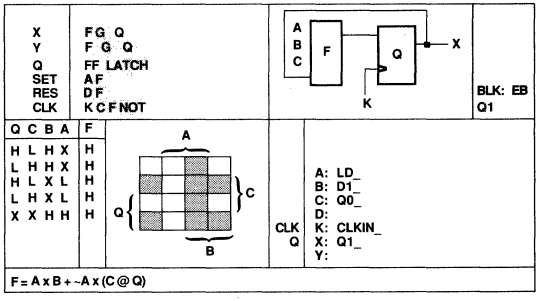


Figure 10b. 2nd Bit of a 3-Bit Counter

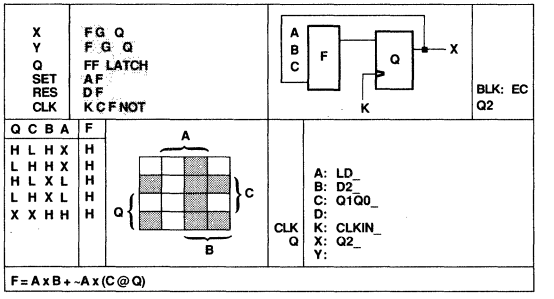


Figure 10c. 3rd Bit of a 3-Bit Counter

Figure 10. CLB Configuration of a 3-Bit Binary Counter with Ripple Carry

Summary

The LCA device has achieved a good solution to the principles used for multiplexing digitally-encoded voice channels in both serial and parallel hierarchies. These hierarchies can accommodate thousands of voice circuits in a PBX switching module. The detailed design of the eight-bit (parallel-to-serial and serial-to-parallel) format converter circuit, or Corner Bender, developed in Monolithic Memories' LCA M2018 device was shown to be efficiently implemented. Despite the fast development of the system using the XACT Design Editor, the final design was programmed and bench tested to verify functional integrity.

The Corner Bender design is available from Monolithic Memories upon request. The bit pattern and .LCA file will be provided for programming the LCA device in an EPROM. Please ask for design XDES05.LCA

References

1) *"Harris 20-20 Integrated Network Switch"*, A. Jackson, IEEE SAC-3, No. 4 July 1985, p561-568.

2) MJ1410, 8-bit Format Converter, Data Sheet, Plessey Semiconductors.

3) Logic Cell Array Design and Applications Handbook, Monolithic Memories, 1987.

Advanced
Micro
Devices
AN-174

Reconfigurable Programmable Devices (LCA) Simplify Digital TDM Line Transcoder

C. B. Lee and Cindy Lee

Abstract

The Logic Cell™ Array (LCA) is a high-density CMOS programmable integrated circuit. Its reprogrammability and reconfigurability make complex design of custom LSI devices easy and flexible.

This application note describes the design of a universal transcoder on trunk transmission lines. Any one of three different transcoders (B8ZS, B6ZS, and HDB3) can be implemented in a single LCA device (M2018). Because it is reconfigurable, any one of these different transcoders may be selected. Other common communication features can be implemented simply by reconfiguring the Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs), or interconnect.

Reconfigurable Programmable Devices (LCA) Simplify Digital TDM Line Transcoder

C. B. Lee and Cindy Lee

TDM Hierarchies

Recently, the International Telegraph and Telephone Consultative Committee (CCITT) agreed on transmission hierarchies based on different national standards. Two major hierarchies are in use today. The first hierarchy, known as the T-carrier, is used in North America and Japan and multiplexes twenty-four voice channels together using Time Division Multiplexing (TDM) technique. T-carrier is based on a primary rate of 1.544 Mbps and multiples up to 274.176 Mbps (see Figure1).

The other hierarchy, used in Europe, Africa, and South America, multiplexes thirty voice channels and two signaling channels together. This TDM hierarchy is based on a primary rate of 2.048 Mbps and multiples up to 139.264 Mbps (see Figure 2) .

CCITT recommendation G.703 defines line coding for both TDM hierarchies with different orders. Table 1 shows a 1.544 Mbps primary rate T-carrier hierarchy with four different orders. Alternate Mark Inversion (AMI) or Binary 8 Zeros Substitution (B8ZS) is the proper line coding in the first order. In the second order, if the transmission medium is coaxial pair cables, then B8ZS is used. Otherwise, Binary 6 Zeros Substitution (B6ZS) is used for symmetrical pair cables. In the third order, Binary 3 Zeros Substitution (B3ZS) is used, while polar binary NRZ is used in the fourth order.

Similarly, Table 2 shows line coding based on a 2.048 Mbps primary rate with four different orders. High Density Bipolar (HDB3) is used in the first, second, and third orders. Finally, Code Mark Inversion (CMI) is used in the fourth order.

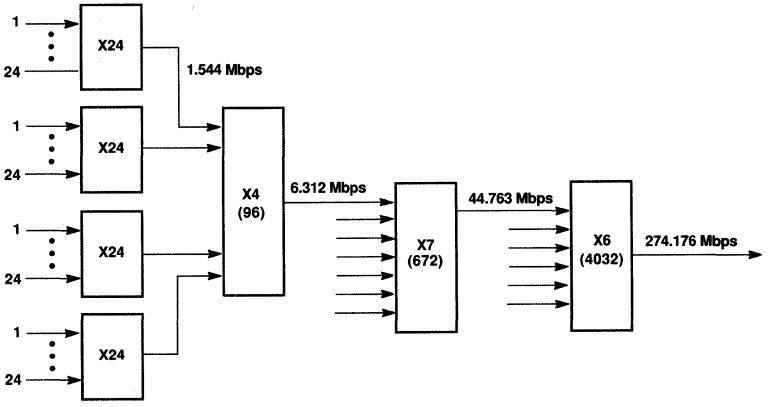


Figure 1. TDM Hierarchy at 1.544 Mbps Primary Rate

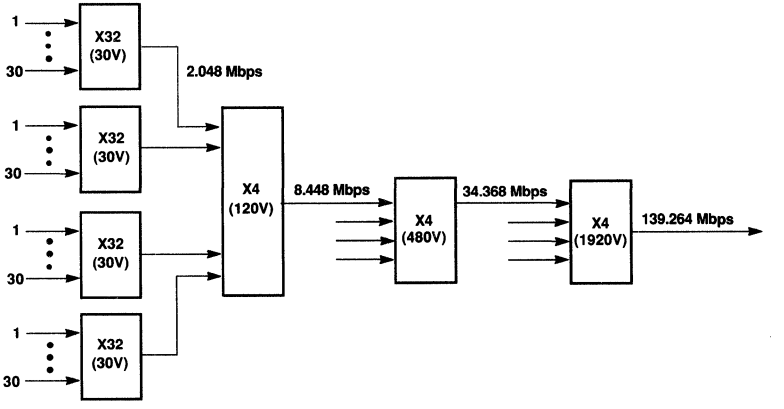


Figure 2. TDM Hierarchy at 2.048 Mbps Primary Rate

ORDER	CHANNEL BANK	NUMBER OF VOICE CHANNELS	BIT RATE (Mbps)	LINE CODING
1	DS-1	24	1.544	AMI or B8ZS
2	DS-2	96	6.312	B8ZS-Coaxial Pair B6ZS-Symmetric Pair
3	DS-3	672	44.736	B3ZS
4	DS-4	4032	274.176	Polar Binary

Table 1. Line Coding for TDM Hierarchy at 1.544 Mbps Primary Rate

ORDER	NUMBER OF VOICE CHANNELS	BIT RATE (Mbps)	LINE CODING
1	30	2.048	HDB3
2	120	8.448	HDB3
3	480	34.368	HDB3
4	1920	139.264	CMI (2-level NRZ)

Table 2. Line Coding for TDM Hierarchy at 2.048 Mbps Primary Rate

Binary Codes

Unipolar binary codes form the information exchange in computer systems from one device to another over short distances. Unipolar binary codes work well at short distances but DC wander and a lack of timing information make them unsuitable for long transmission lines. DC wander is caused from unipolar binary signals being transmitted via a certain media over long-distance transmission lines. The signals will be attenuated after a few kilometers.

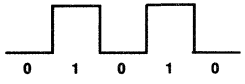


Figure 3a. Unipolar Binary Codes

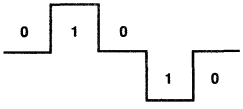


Figure 3b. Bipolar Codes

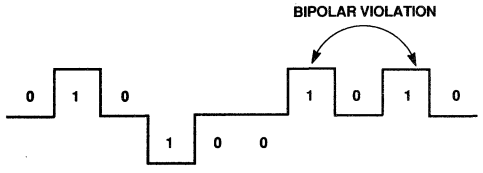


Figure 4. Bipolar Violation

Bipolar Codes

Unipolar binary codes require two voltages to represent a binary state (see Figure 3a). On the other hand, bipolar binary codes, also known as AMI codes, need three different voltages (+5 V, 0 V, and -5 V) to represent the same binary condition: logic zero or logic one (see Figure 3b). By making logic one signals alternate between +5 V and -5 V, DC wander is eliminated on a long transmission line because the mean DC level is integrated to zero volts. If, however, two continuous ones have the same polarity a bipolar violation occurs (see Figure 4). A long string of ones of bipolar codes can provide timing information but a long string of zeros cannot. To counteract this problem, several modified bipolar codes provide timing information for a long string of logic zeros.

Modified Bipolar Codes

The B8ZS, B6ZS, and HDB3 codes are all modified bipolar codes. They provide timing information for a long string of zeros by forcing bipolar violations.

B8ZS line coding encodes logic ones as alternating positive (+5 V) and negative (-5 V) signals. It substitutes any continuous sequence of eight zeros with a special pattern (see Table 3). Depending on the preceding polarity, one of two B8ZS substitution codes is generated: 000+-0-+ or 000-+0+-. The "+" indicates positive voltage, "-" indicates negative voltage, and "0" indicates zero voltage. If the preceding polarity is positive, 000+-0-+ is generated; otherwise, 000-+0+- is generated. The B8ZS substitution code forces bipolar violations in the fourth and the seventh bit positions.

B6ZS is similar to B8ZS. Rather than detecting eight continuous zeros, B6ZS substitutes for six continuous zeros with a special pattern. The B6ZS substitution codes are 0+-0-+ and 0-+0+-, depending on the preceding polarity. B6ZS substitution code forces the second and fifth zeros to bipolar violation (see Table 4).

PRECEDING POLARITY	B8ZS SUBSTITUTION
+	000+-0-+
-	000-+0+-

Table 3. B8ZS Substitution Table

PRECEDING POLARITY	B6ZS SUBSTITUTION
+	0+-0-+
-	0-+0+-

Table 4. B6ZS Substitution Table

The European HDB3 line coding also uses alternating positive and negative signals to represent logic ones. It also substitutes any four continuous zeros with a special HDB3 substitution code. This HDB3 substitution code forces the fourth bit to be a bipolar violation. The four possible HDB3 substitution codes are: +00+, 000+, -00-, and 000-. The choice of substitution code depends on the preceding polarity and the number of logic ones after the preceding bipolar violation. When the number of logic ones is odd, 000V is generated. Otherwise, B00V is generated. "V" indicates the bipolar violation while "B" indicates the bipolar signal. The polarity of V in 000V is the same as that of the preceding logic one signal. If the HDB3 code is B00V, the polarity of B is opposite that of the preceding logic one signal, and the polarity of V is the same as B (see Table 5).

PRECEDING POLARITY	NUMBER OF LOGIC ONES AFTER THE PREVIOUS BIPOLAR VIOLATION	
	ODD	EVEN
Positive +	000+	-00-
Negative -	000-	+00+

Table 5. HDB3 Substitution Table

Logic Cell Array Implementation

The Logic Cell Array (LCA) device is a high-density CMOS integrated circuit. Its user-programmable static RAM array architecture consists of Input/Output Blocks (IOBs), Configurable Logic Blocks (CLBs), and Interconnect. All of them are fully user-programmable and user-reconfigurable. High-density, programmability and reconfigurability of LCA devices allow customer-defined LSI functions to be incorporated and modified at low cost and with fast turnaround.

Currently, two LCA devices are available, their complexity is based on the number of CLBs within the device. All CLBs are arranged in a matrix in the center of the device. The M2064 has sixty-four CLBs which are arranged in an 8-row by 8-column matrix. The M2018 has one hundred CLBs arranged as a 10 by 10 matrix.

A CLB can be configured to be a storage element (registered or latched), a combinational logic block or a mixed combinational logic block with storage element. Each CLB has four general-purpose inputs (A, B, C, and D); and a special clock input (K). Also, each CLB can perform any function of up to four variables or any two functions of up to three variables. These variables can be selected from external inputs or from the internal register feedback path.

Initially, an attempt was made to design the line transcoder with the M2064 device. However, all three line transcoder designs required more than sixty-four CLBs each. Fortunately, it is easy to convert a design from M2064 to M2018 because the change is made just by selecting the M2018 part type in the convert command when using the XACT Design Editor. Any of the preceding line coding schemes can be implemented in a single M2018 because of its higher density (100 CLBs). Since the LCA device is reconfigurable, a line transcoder implemented in a single LCA device (M2018),can support the North American T1 carrier (B8ZS line coding) or T2 carrier (B6ZS line coding), or even the European standard (HDB3 line coding).

All three line transcoders can be configured and have the same pinout assignments. This offers a simple solution for supporting the different standards without component replacement. Once a design has been established, "on-the-fly" modification is possible through the reconfigurability of the component. For example, if the communication medium is changed in the T2

PIN NUMBER	DESCRIPTION	B8ZS	B6ZS	HDB3
Pin 01	GND	GND	GND	GND
Pin 11	Clock	CLK	CLK	CLK
Pin 12	Master Reset	MR	MR	MR
Pin 13	Positive input for decoder	IPB	IPB	IPH
Pin 14	Positive output from encoder	OPB	OPB	OPH
Pin 15	NRZ input	INR	INR	INR
Pin 17	Negative input for decoder	INB	INB	INH
Pin 18	V_{CC}	V_{CC}	V_{CC}	V_{CC}
Pin 19	Negative output from encoder	ONB	ONB	ONH
Pin 34	NRZ output	ONR	ONR	ONR
Pin 35	GND	GND	GND	GND
Pin 52	V_{CC}	V_{CC}	V_{CC}	V_{CC}
Pin 56	Error*	ERR	ERR	ERR
Pin 57	Bipolar violation error	BVP	BVP	BVP
Pin 59	Special pattern detected	B8Z	B6Z	HDB

* Means that the ERROR is asserted when both positive and negative signals are high simultaneously.

Table 6. Cross Reference of Pin Names for Three Line Transcoders

standard, the transcoder may be reprogrammed from B8ZS to B6ZS, or vice versa, without any other alterations to the hardware. In total, fifteen pins are used in these designs; four power pins, two central control pins, three pins for the encoder, and six pins for the decoder. Table 6 cross references the pin names for the three different line transcoders.

B8ZS Line Coding

A B8ZS encoder converts NRZ data to B8ZS code. The B8ZS encoder consists of a 3-CLB, 3-bit counter, a 5-CLB, 5-bit shift register, a 6-CLB sequence state machine, and a 13-CLB encoding state machine (see Figure 5). A 3-bit counter detects eight continuous zeros. A 5-bit shift register delays the NRZ input data to synchronize with the 3-bit counter's outputs. The sequence state machine provides the sequence state (Q2, Q1, and Q0) for encoding state machine. The encoding state machine generates the B8ZS code. It is a pair of signals which interface to line driver to provide three level signals: positive (PB8ZS), negative (NB8ZS), and zero signals.

A B8ZS decoder detects B8ZS code on a pair of positive and negative B8ZS signals (see Figure 6). When either of the following sequences: 000+-0-+ or 000-+0+- is detected, the decoder generates a string of eight zeros. Otherwise, the encoder converts bipolar signals to binary NRZ data.

Figure 7 is a block diagram of the B8ZS decoder. It includes a 19-CLB decoding state machine, an 8-CLB, 8-bit shift register, a 5-CLB RESET generator, and a 7-CLB bipolar violation and error flag generator. Four state variables (HQ3, HQ2, HQ1, and HQ0) are generated from the decoding state machine. The 8-bit shift register buffers DSOUT data in order to provide eight continuous zeros when the B8ZS signal is asserted. The B8ZS signal is generated when the B8ZS substitution code is detected. The bipolar violation error (BVP) occurs when any two sequential signals of the same polarity are detected except the B8ZS substitution code. If both positive B8ZS and negative B8ZS are detected simultaneously, the error signal (ERROR) is generated.

The implementation details of the B8ZS line transcoder within a single M2018 are shown in Figure-8. This design requires sixty-six out of the one hundred available CLBs.

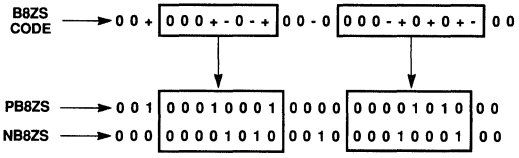


Figure 6. B8ZS Codes in Positive-B8ZS and Negative-B8ZS Signals

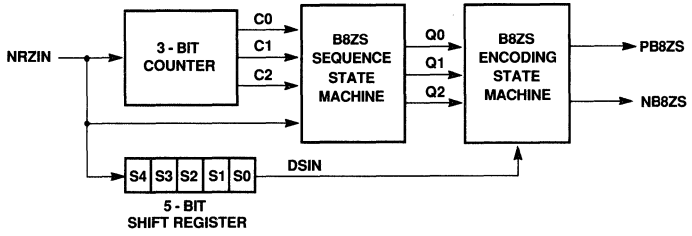


Figure 5. Block Diagram of the B8ZS Encoder

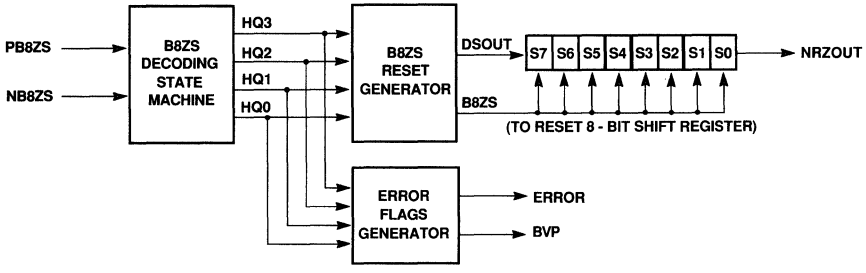


Figure 7. Block Diagram of the B8ZS Decoder

Print World: B8ZS.LCA (2018PC68-70), XACT 1.30, 10:10:45 MAY 12, 1987

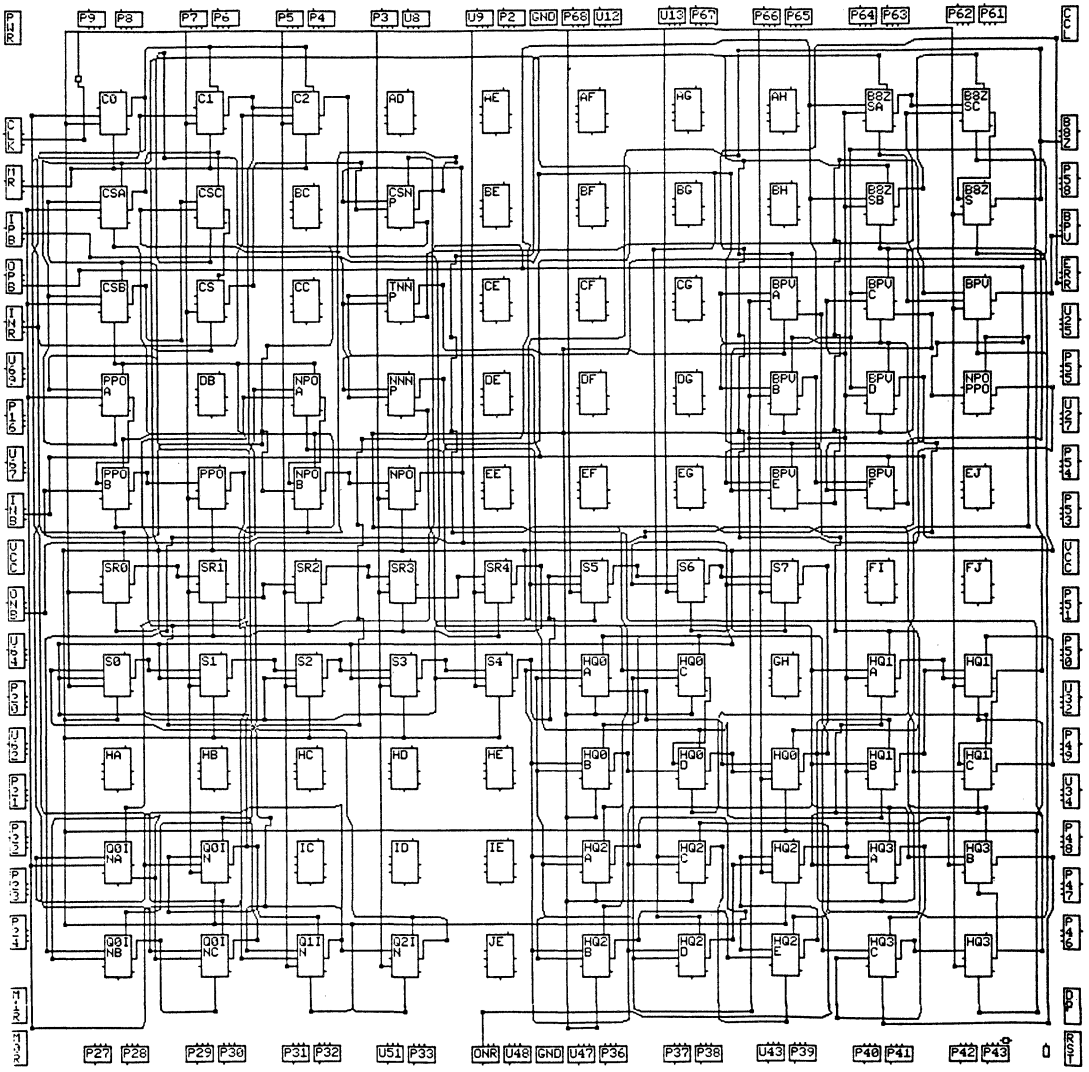


Figure 8. B8ZS Line Transcoder LCA Design

B6ZS Line Coding

The M2018 implementation of the B6ZS line transcoder is shown in Figure 9. In total, sixty-seven CLBs are used to create the design. The single difference between a B8ZS encoder and a B6ZS encoder is the count for continuous zeros. While the 3-bit counter counts eight continuous zeros for the B8ZS encoder, it counts six for the B6ZS encoder. The B8ZS encoder consists of a 5-CLB, six-zeros detector, a 5-CLB, 5-bit shift register, a 6-CLB sequence state machine, and a 13-CLB encoding state machine.

A B6ZS decoder detects two B6ZS substitution codes: 0+-0-+ or 0-+0+-. When one of those two B6ZS substitution codes is detected, the decoder generates six continuous zeros. Otherwise, the decoder converts bipolar signals to binary NRZ data. The B6ZS decoder includes a 20-CLB decoding state machine, a 6-CLB, 6-bit shift register, a 5-CLB RESET generator, and a 7-CLB bipolar violation and error flag generator. The 6-bit shift register is needed to buffer DSOUT data in order to provide six continuous zeros.

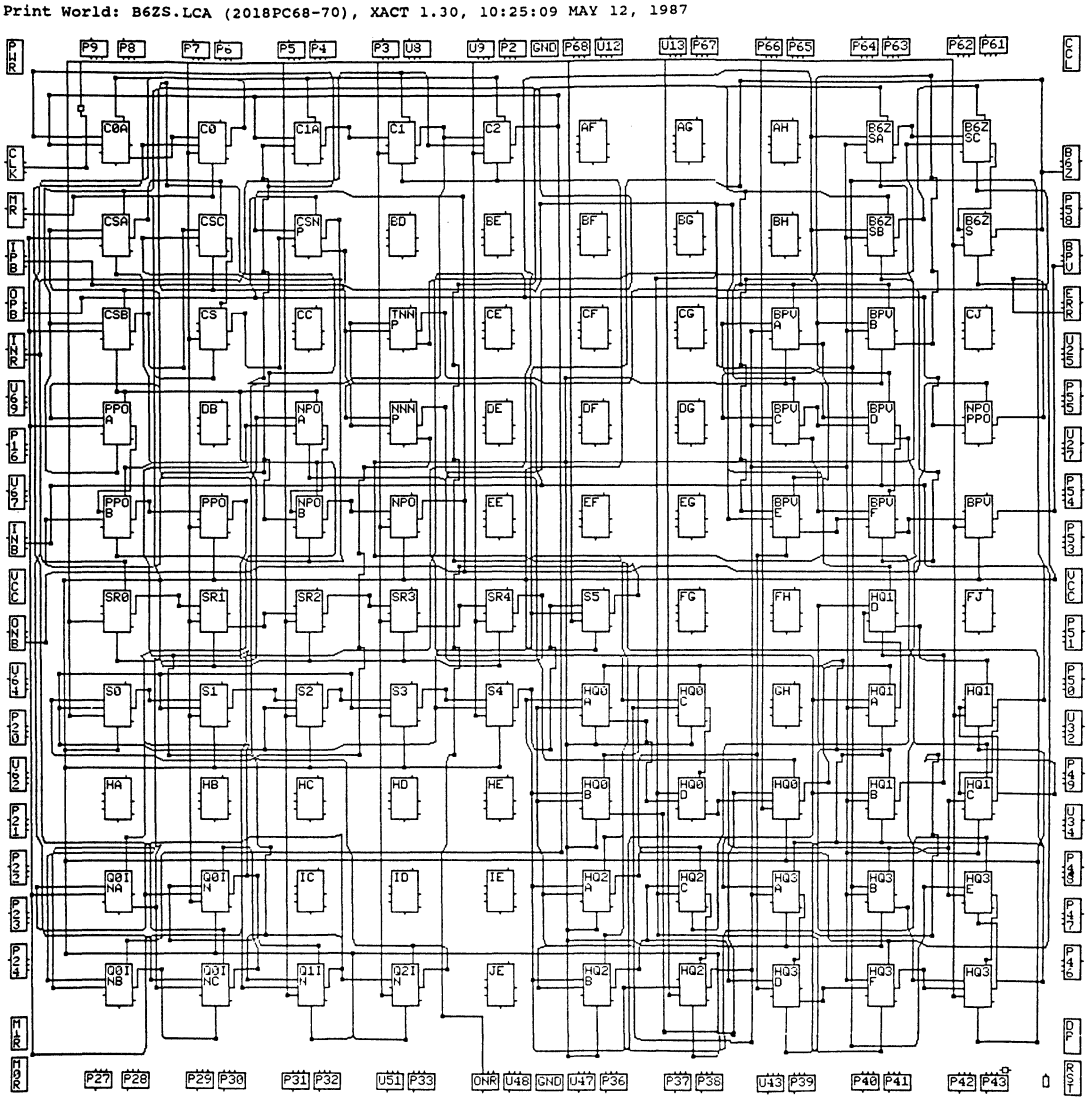


Figure 9. B6ZS Line Transcoder LCA Design

HDB3 Line Coding

In Figure 10, the HDB3 encoder includes a 4-CLB, 4-bit shift register, a 5-CLB HDB3 signal generator, and three-state machines. These state machines are, a 3-CLB ODD_EVEN state machine, a 5-CLB DETECT_4_ZEROS state machine, and 23-CLB encoding state machine. The ODD_EVEN state machine detects the polarity of the preceding pulse. The DETECT_4_ZEROS state machine detects four continuous zeros. The 4-bit shift register delays NRZIN data to synchronize with the output of DETECT_4_ZEROS state machine. The encoding state machine provides four encoding states variables (Q3, Q2, Q1, and Q0) to generate two HDB3 signals: the positive HDB3 signal (PHDB) and negative HDB3 signal (NHDB).

A HDB3 decoder detects four HDB3 substitution codes: +00+, 000+, -00-, and 000-. When any of these four HDB3 substitution codes is detected, the decoder generates four continuous zeros. Otherwise, the decoder converts bipolar signals to binary NRZ data. Figure 11 is a block diagram of the HDB3 decoder. It includes a 23-CLB decoding state machine, a 4-CLB, 4-bit shift register, a 3-CLB RESET generator, and a 2-CLB bipolar violation and error flags generator. Four state variables (HQ3, HQ2, HQ1, and HQ0) are generated from the decoding state machine. The 4-bit shift register buffers DSOUT data in order to provide four continuous zeros when the HDB3 signal is asserted. The HDB3 signal is generated when the HDB3 substitution code is detected. The bipolar violation error (BVP) occurs when any two sequential signals of the same polarity are detected except the HDB3 substitution code. If both positive HDB3 and negative HDB3 are detected simultaneously, the error signal (ERROR) is generated. The M2018's layout of the HDB3 line transcoder is shown in Figure 12. This design requires seventy-two out of the one hundred available CLBs.

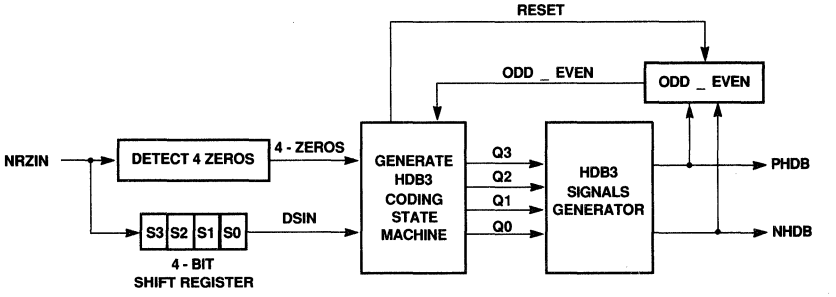


Figure 10. Block Diagram of the HDB3 Encoder

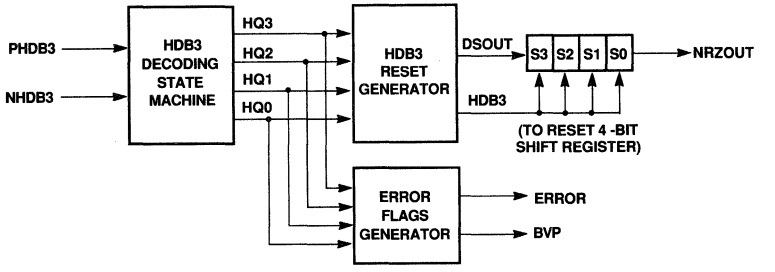


Figure 11. Block Diagram of the HDB3 Decoder

Print World: HDB3.LCA (2018PC68-70), XACT 1.30, 09:36:22 MAY 12, 1987

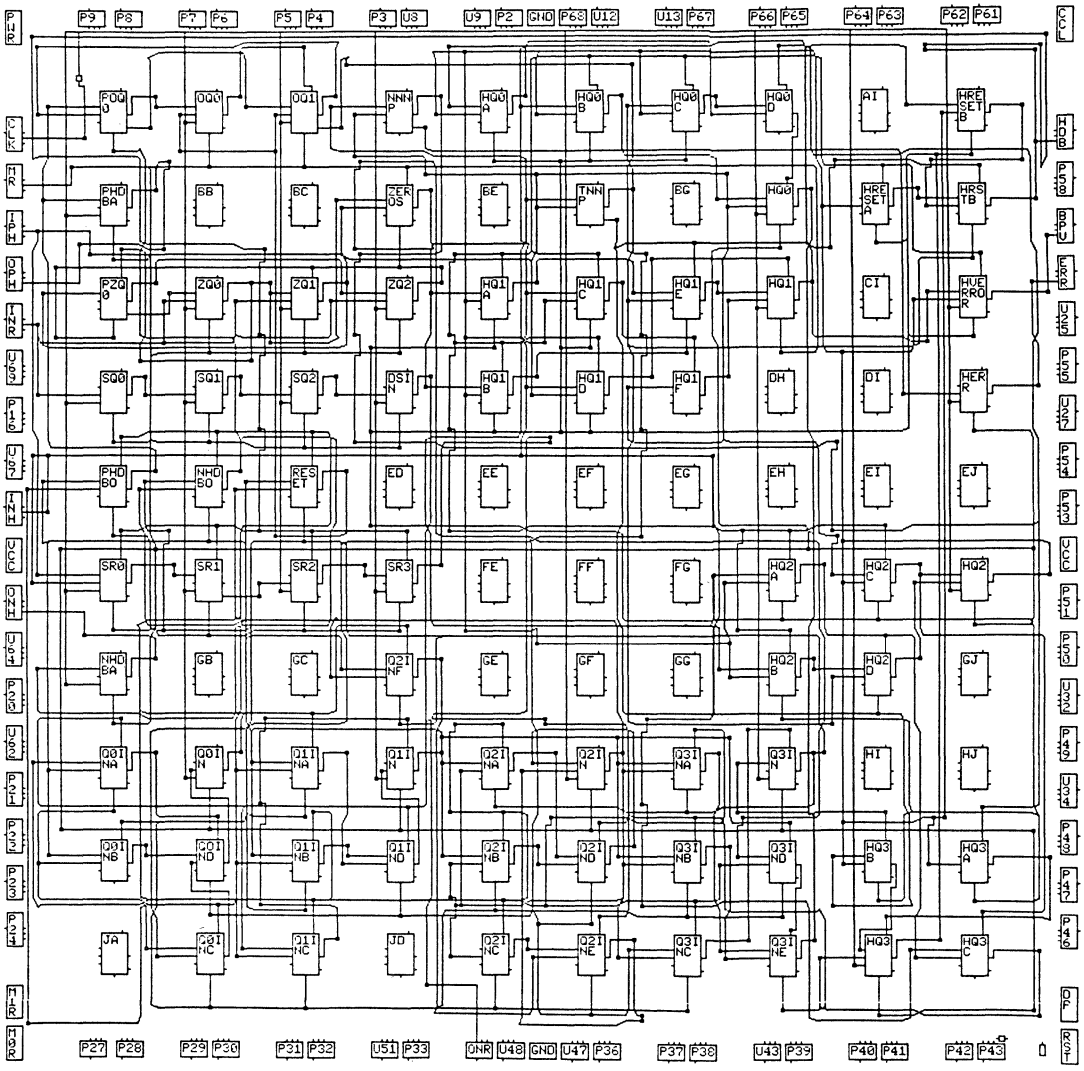


Figure 12. HDB3 Line Transcoder LCA Design

Logic Cell Array Design Methodology

All three designs are composed of similar building blocks, they all use shift registers, combinational logic and state machine logic.

Each of the three line transcoders require the use of two sets of shift registers. One set delays the input NRZ data, and the other set buffers the output NRZ data. To design a shift register using a CLB is simple. Two inputs (input variable, clock) are used to generate a single output (Figure 13); this output is then cascaded with additional registers to form a shift register. In the B8ZS design, five CLBs are used to implement a 5-bit shift register.

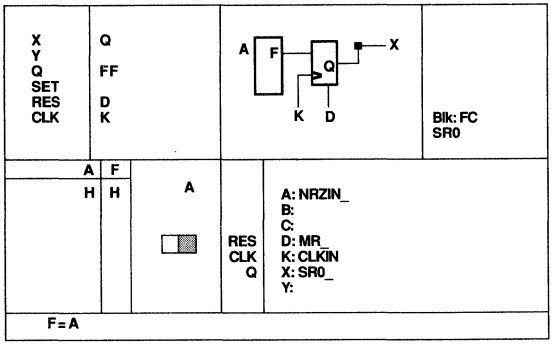


Figure13. Basic Element of N-bit Shift Register Configured in a Single CLB

For state machines and combinational logic, both the state machine equations and combinational logic equations must be carefully partitioned to fit them into CLBs and IOBs. Two forms of optimization can be done. One can either optimize for implementation efficiency i.e., minimize the usage of CLBs and IOBs, or, one can optimize for speed. To optimize for CLB usage, one must carefully analyze the equations and group signals which can potentially share the same CLB outputs. On the other hand, to optimize for speed, one must pay attention to the routing paths used as well as fanout from each CLB output and the way in which each CLB is configured.

For example, the HDB3 encoding state machine uses four state variables (Q3, Q2, Q1, and Q0) in order to generate fifteen states. First, the state equations are derived from the state diagram. Then, the state equation is partitioned to fit into CLBs. The following equation is the most significant state variable (Q3) of this HDB3 encoding state machine.

Six CLBs are used to implement this Q3 state equation. Five of the CLBs (Q3INA, Q3INB, Q3INC, Q3IND, and Q3INE) are configured as 4-input variable CLBs which generate a single output. Because these CLBs perform the preliminary logic for input into the block Q3IN, no storage element is necessary in these CLBs. Instead, the storage function is implemented in the block Q3IN. The design details for each CLB is shown in Figure 14.

The B8ZS line transcoder can be implemented using three PAL® devices: two 16R8 devices and one 16R6 device. All equations for those PAL devices are assertive low; e.g., $\overline{C2}$:= C2 * C1 * C0 + $\overline{C2}$ * $\overline{C1}$ + $\overline{C2}$ * $\overline{C0}$ + RST + NRZIN. The LCA device can easily incorporate this assertive low equation simply by adding a tilde ("~") in front of the output equation (see Figure 15).

Q3 :=

$\overline{Q0}$ * $\overline{Q1}$ * Q2 ; Q3INA
+ $\overline{Q1}$ * Q2 * Q3
+ Q0 * Q1 * Q2 * $\overline{Q3}$

+ DSIN * Q2 * Q3 ; Q3INB
+ DSIN * $\overline{Q1}$ * $\overline{Q2}$ * $\overline{Q3}$

+ $\overline{Q0}$ * $\overline{Q2}$ * Q3 * $\overline{ZEROS}$; Q3INC
+ Q0 * Q2 * Q3 * $\overline{ZEROS}$

+ DSIN * $\overline{Q0}$ * ZEROS
+ DSIN * $\overline{Q1}$ * ZEROS ; Q3IND
+ DSIN * $\overline{Q3}$ * ZEROS

; Q3INE

; Q3IN

The HDB3 encoder has an ODD_EVEN state machine to check the number of preceding logic ones being odd number or even number. Its ODD_EVEN signal feeds back into the HDB3 encoding state machine. If these two state machines reference the same rising clock edge. It is possible to obtain the wrong result because the HDB3 encoding state machine always gets the ODD_EVEN signal one clock early. Using the falling clock edge can compensate for this timing problem. The positive and negative clock edges are available in each CLB. In this ODD_EVEN state machine, the negative clock is selected (see Figure 16).

Other common functions of communication equipment can be easily implemented using the reconfigurability of the LCA device. For example, two duplex schemes: half duplex and full duplex, can be implemented easily because of the reconfigurablilty of the IOB. Each IOB can be configured to an input, an output, or a bidirectional input/output with three-state control. Two IOBs are used to implement the full duplex communication, this scheme allows the device to transmit and receive at the same time (see Figure 17a). It can be easily modified to the half-duplex communication simply by using one bidirectional IOB, the information must be transmitted or received exclusively (see Figure 17b).

In addition, loop-back feature can be implemented by reconfiguring LCA's interconnect. Remote loop-back is implemented by disconnecting the line transcoder with the input and output pads, but connecting the input pad to the output pad directly within the transcoder (see Figure 18a). Local loop-back can also be implemented. Connecting the output of the transmitter to the input of the receiver directly, but disconnecting with input and output pads (see Figure 18b).

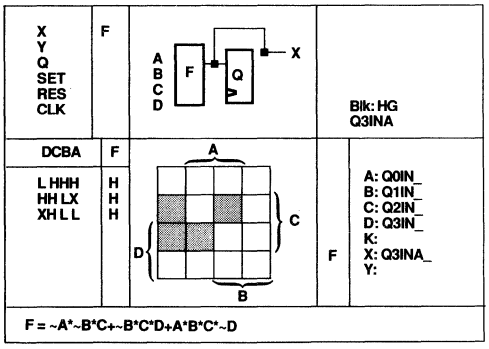


Figure 14a. Q3INA's CLB Design

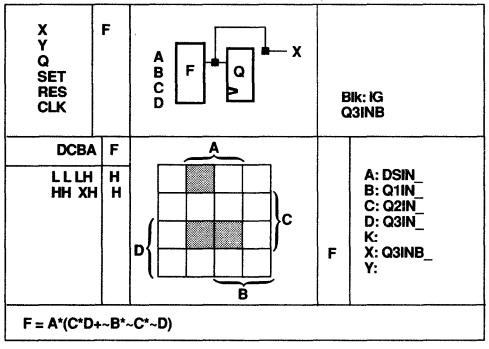


Figure 14b. Q3INB's CLB Design

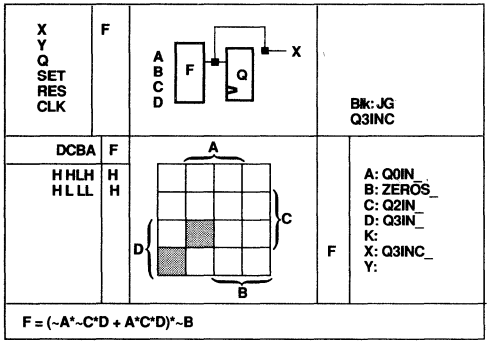


Figure 14c. Q3INC's CLB Design

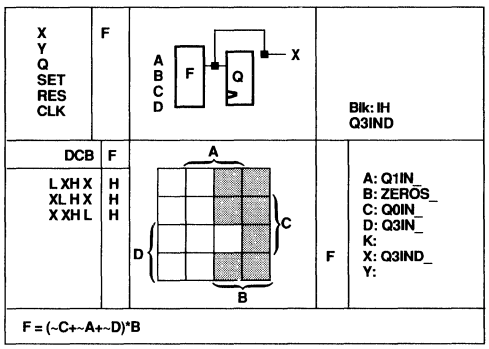


Figure 14d. Q3IND's CLB Design

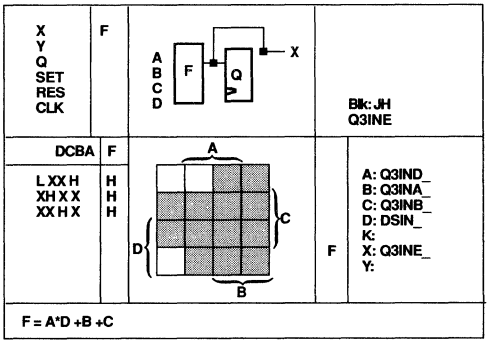


Figure 14e. Q3INE's CLB Design

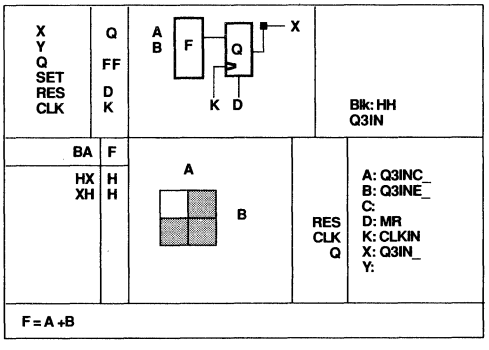


Figure 14f. Q3IN's CLB Design

Figure 14. The CLB's Design for the Q3 Equation

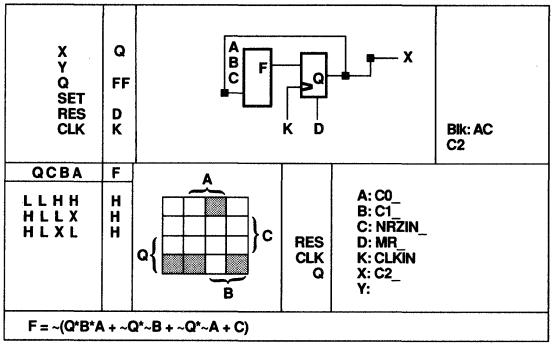


Figure-15. C2 is Represented by ~(C2)

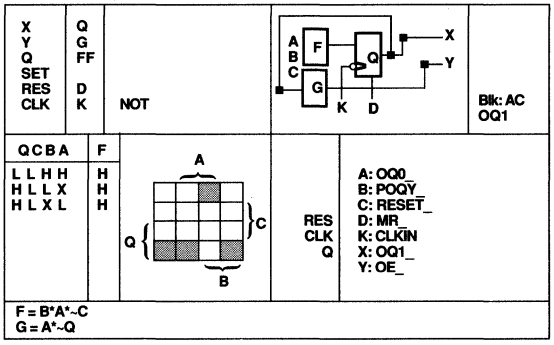
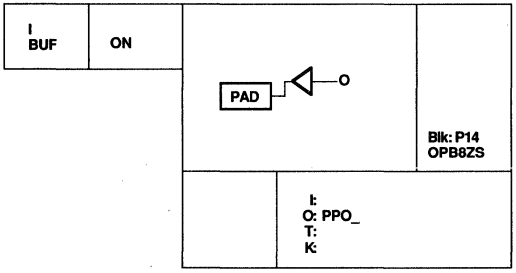


Figure 16. Negative Clock Edge in the CLB



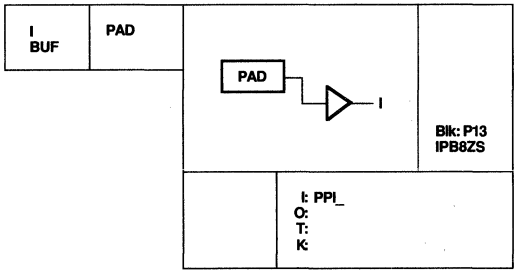


Figure 17a. Full-Duplex Communication

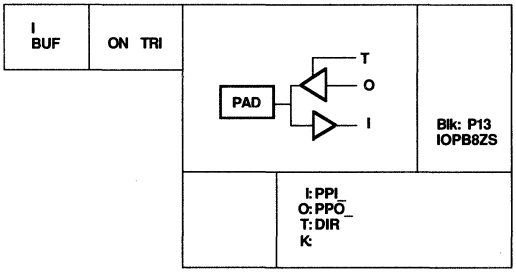


Figure 17b. Half-Duplex Communication

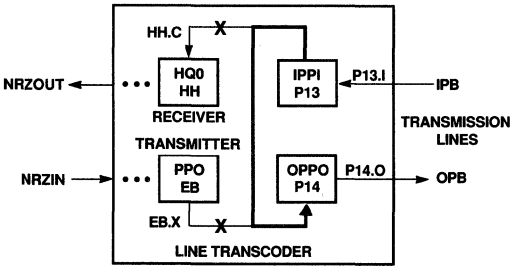


Figure 18a. Remote Loop Back

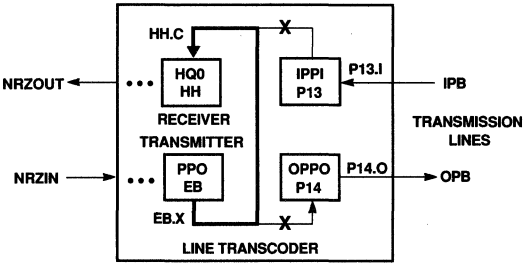


Figure 18b. Local Loop Back

Conclusion

This design demonstrates the power versatility and benefit of using the LCA device in line coding by exploiting the reconfigurable feature of the device. A single M2018 LCA device can support three different line codes, such as B8ZS, B6ZS, and HDB3 codes.

The B8ZS is used at either 1.544 Mbps T1 carrier or 6.312 Mbps T2 carrier on coaxial pairs. The B8ZS encoder uses twenty-seven CLBs to convert NRZ data into two B8ZS signals, and the B8ZS decoder uses thirty-nine CLBs to convert two B8ZS signals to NRZ data. The B6ZS is used at 6.312 Mbps T2 carrier on symmetrical pairs. The B6ZS encoder uses twenty-nine CLBs to convert NRZ data into two B6ZS signals, and the B6ZS decoder uses thirty-eight CLBs to convert two B6ZS signals to NRZ data. The HDB3 is used at 2.048 Mbps or 8.448 Mbps transmission lines. The HDB3 encoder uses forty CLBs to convert NRZ data into two HDB3 signals and the HDB3 decoder uses thirty-two CLBs to convert two HDB3 signals to NRZ data.

The line transcoder is required when the signal is transmitted over transmission lines. It converts unipolar binary codes to bipolar codes or modified bipolar codes in order to eliminate DC wander and provide the timing information.

References

1. Theresa Shafer and Steve Patterson, *"B8ZS Coding,"* Monolithic Memories Application Note AN-169.
2. Cindy Lee, *"HDB3 Line Coding using Three PAL® Devices,"* Monolithic Memories Application Note AN-176.
3. John C. Bellamy, *"Digital Telephony,"* John Wiley & Son, 1982.
4. Recommendation G.703, CCITT Red Book, Volume III.

The detailed LCA design files are available from Monolithic Memories, Inc.

The design file of B8ZS refers to XDES12.LCA.

The design file of B6ZS refers to XDES13.LCA.

The design file of HDB3 refers to XDES14.LCA.

Advanced
Micro
Devices
AN-179

Building an ESDI Translator Using the M2064 Logic Cell™ Array

Ken Won and Ken Tseng

Abstract

The ESDI (Enhanced Small Device Interface) Standard is a low-cost, high-performance Winchester disk drive interface. The ESDI Translator is an interface controller that is implemented in an ESDI-compatible disk drive.

The ESDI Translator is implemented in one M2064 LCA device. The LCA device is a RAM-based high-density CMOS integrated circuit which is reprogrammable. This application note describes the design implementation and design considerations of the ESDI Translator on the LCA device.

Building an ESDI Translator Using the M2064 Logic Cell Array

AN-179

The ESDI Translator

ESDI is a low-cost, high-performance interface standard suitable for the smaller, high-performance Winchester disk drives currently being produced. The ESDI interface consists of a control cable and a data cable. The control cable allows for a daisy chain connection of up to seven devices (disk or optical drives) with only the last device being terminated. In our design, we assumed that the device is a disk drive. The data cable is attached in a radial configuration (See Figure 1).

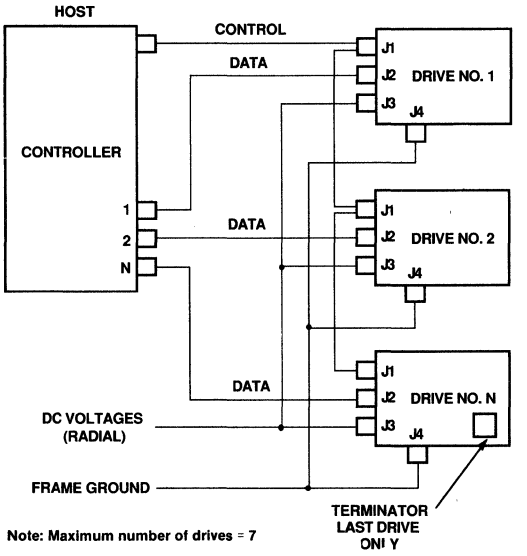


Figure 1. Connection Between the Controller and Multiple Drives

The ESDI Translator handshakes serial commands from a disk controller, deserializes the commands and passes the commands to a microcontroller. The command data word structure is shown in Figure 2.

The Command Function bits define functions to be executed by the disk drive. These functions are seek, recalibrate, request status, request configuration, select head group, control, data strobe offset, track offset, initiate diagnostics, set bytes per sector and set configuration. Some of these functions, such as the control function, have modifiers for more detailed functional description. Other commands have parameters that contain numbers. For the seek command, the parameter specifies the cylinder number that the drive will seek to. The request status and request configuration commands require data from the disk drive to be transferred back to the disk controller. In our current design, internal registers A and B in the LCA device represent the upper and lower bytes of the command respectively.

Figure 3 illustrates the relationships between the disk controller, the ESDI Translator, and the microcontroller. The PROM is used to store the configuration data for the LCA device. The Done/$\overline{\text{Program}}$ (D/$\overline{\text{P}}$) output is driven LOW when the device is being configured. Configuration data is read from the PROM device during configuration. After configuration is complete, the LCA device drives the D/P pin HIGH to deselect the PROM. Drive selection, write protection, command completion and fault detection are also handled by the ESDI Translator.

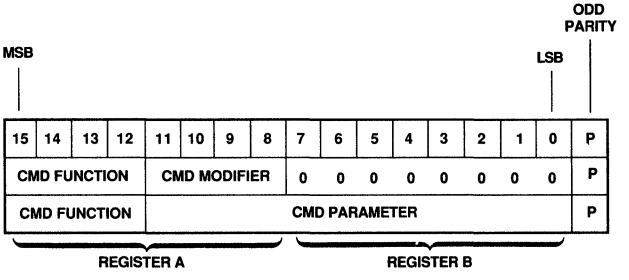


Figure 2. Command Data Word Structure

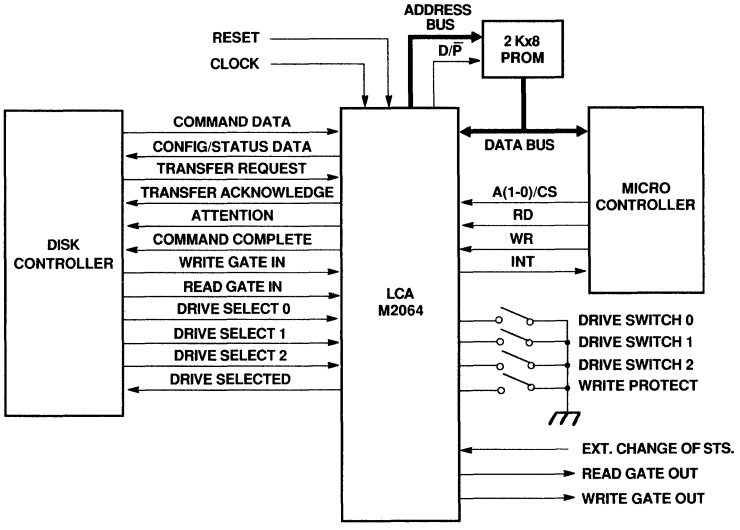


Figure 3. An ESDI Translator Implemented on the LCA Device

Why Use an LCA Device in an ESDI Translator

The ESDI interface standard requires more logic functions to be built into the disk drive than some other interface standards, such as the ST506 standard. However, the external dimensions of a disk drive usually have to conform to an industrial standard form factor. Thus, the use of high-density semicustom chips is the logical solution to increase functionality without increasing the external dimensions.

The LCA device is a high-density CMOS integrated circuit available from Monolithic Memories. Its high gate density allows the implementation of an ESDI Translator in a single chip. Fifteen standard SSI and MSI chips would be necessary for the same application. If PLDs are used to implement the ESDI Translator, more than one would be necessary because a large amount of logic is required, thus occupying more board space.

A major advantage in using the LCA device is that it can speed up the design cycle, enabling the manufacturer to have a shorter time-to-market. Also, many peripheral products are produced in relatively small quantities aiming at very specialized markets. The LCA device, which has no NRE cost, makes the production of small quantities more economical than the gate array. Another advantage of the LCA device over other semicustom chips, such as the gate array, is its reprogrammability feature. The LCA device is RAM-based which can easily be reprogrammed by the user in the final system. This feature is especially important in the peripheral products market, where many products have short life spans.

Design Implementation

The ESDI Translator is responsible for all control interfaces between the disk controller and the disk drive. An internal block diagram of the ESDI Translator is shown in Figure 4. It consists of five major logic building blocks:

- Drive selection
- Read gate/write gate
- Counter/controller
- Shift register and parity generator/checker
- Internal register address decoder

Drive selection on ESDI-compatible drives involves three signals from the disk controller, Drive Select 0-2. These three drive select lines are encoded so that up to seven drives may be connected to the same ESDI port, as shown in Table 1.

On the LCA device, the drive number is selected by connecting the drive switch pins to either VCC or GND. When the code on the drive select lines, DSX, equals the code on the drive switch pins, DSWX, where X may be 0, 1, or 2, the drive is selected and the Drive Selected signal, DSELD, is asserted. Once the drive has been selected, serial commands output by the disk controller will be read by the LCA device. The actual implementation is shown in Figure 5, using two CLBs and seven IOBs. BDS0 and BDS1 are the names of the CLBs in the current design. In our design, names that begin with the letter B or P are used to designate a CLB or an IOB respectively. SCLK is the system clock.

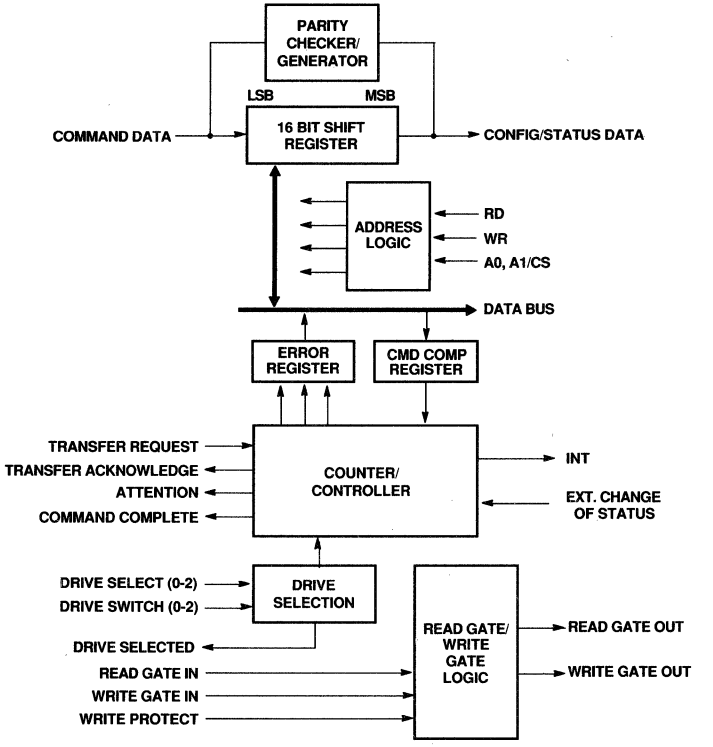


Figure 4. ESDI Translator Internal Block Diagram

DS2/ DSW2	DS1/ DSW1	DS0/ DSW0	DRIVE
0	0	0	None
0	0	1	Select Drive 1
0	1	0	Select Drive 2
0	1	1	Select Drive 3
1	0	0	Select Drive 4
1	0	1	Select Drive 5
1	1	0	Select Drive 6
1	1	1	Select Drive 7

Table 1. Drive Selection

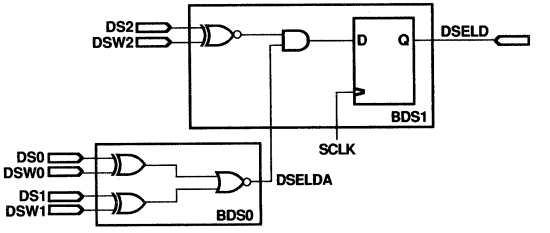


Figure 5. Configuration of the BDS0 and BDS1 CLBs to Provide the Drive Selected Signal

Figure 6 shows the configurations of the two CLBs as displayed on the computer screen by the XACT development software. In Figure 6a, the CLB is configured as one function of four variables. The D flip-flop is not used. The logic representation, truth table, Karnaugh map, signal names, block name and Boolean equation are shown.

In Figure 6b, the CLB is configured as one function of three variables, the output of which is connected to the D flip-flop. The Q output of the D flip-flop becomes the output of this CLB.

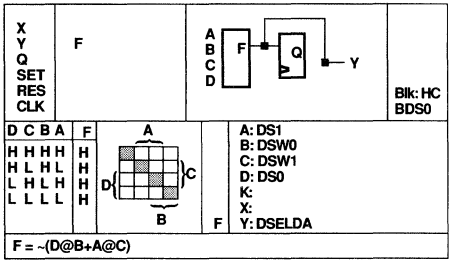


Figure 6a. Drive Selection Logic Implemented in a CLB (BDS0)

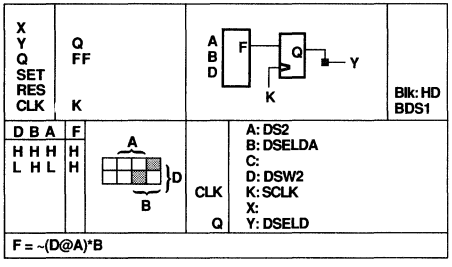


Figure 6b. Drive Selection Logic Implemented in a CLB (BDS1)

The LCA device also performs logic functions for the Read Gate and Write Gate logic blocks. The Read Gate signal allows data to be read from the disk, and the Write Gate signal allows data to be written on the disk. These signals from the disk controller are input to the LCA device as Read Gate In and Write Gate In. Under normal operating conditions, Read Gate Out is asserted when Read Gate In is asserted and Write Gate Out is asserted when Write Gate In is asserted. When both Read Gate In and Write Gate In are asserted, a write error condition results and the Attention line is asserted, signalling the disk controller that an error has occurred. Also, the LCA device may be used to provide write protection to a disk drive. When the Write Protect signal is asserted, the Write Gate Out signal will not be asserted when the Write Gate In signal is asserted, preventing the write circuitry from being activated. These logic functions are implemented in two CLBs, BRWG0 and BRWG1, as shown in Figure 7.

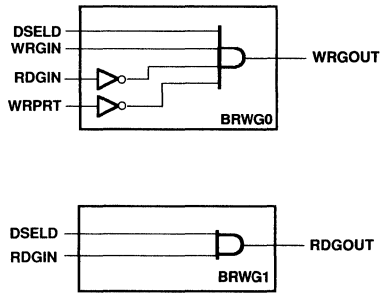


Figure 7. Read Gate/Write Gate Logic Implementation

The Counter/Controller handshakes commands from the disk controller. It also handshakes status/configuration data to the disk controller. It is also responsible for generating the Interrupt, Attention and Command Complete signals. Seventeen bit commands (one bit is parity) are transferred from the disk controller to the LCA device via the Command Data line. The serial bit transfer is performed using a pair of handshaking signals, Transfer Request (TREQ) and Transfer Acknowledge (TACK). TREQ is asserted by the disk controller when a bit is valid on the Command Data line, and TACK is asserted by the LCA device when the command bit has been read. The handshaking action is shown in Figure 8.

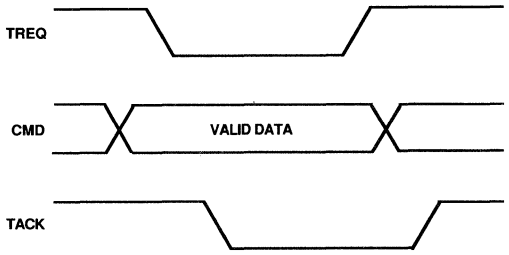


Figure 8. Command Data Transfer

A 17-state counter counts the number of command bits shifted in or shifted out of the data registers. Its implementation is shown in Figure 9. The SHIFT signal is asserted, thus incrementing the counter, whenever there is a transfer request and the LCA device is selected. CNT_Q0 is the lowest bit of the shift register counter. This bit is inverted whenever SHIFT is asserted unless sixteen bits have already been shifted into the LCA device (CNT16 asserted, or CKCMD asserted when all seventeen bits have been shifted in). The DATAOUT signal is asserted after a Request Status command or a Request Configuration command has been transferred and the internal data registers have been loaded with data to be serially shifted out. Hence, when DATAOUT is asserted, CKCMD is negated and the counter is enabled.

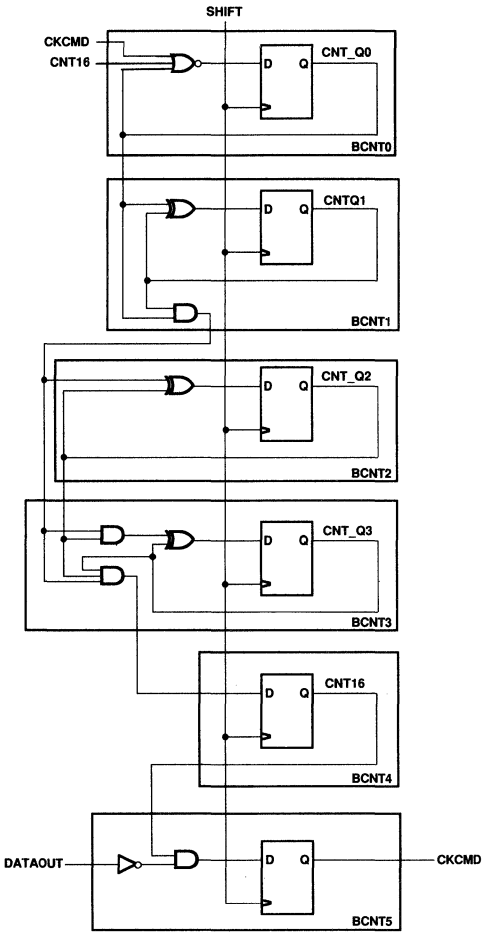


Figure 9. 17-State Counter Implemented in six CLBs

The logic generation of the Attention signal, ATTEN, is shown in Figure 10. Whenever there is a write fault, WRFLT, a parity error, PARERR, an interface fault, INTFLT, or an external error, CSTS(Change of Status), the ATTEN signal is asserted. When both WRGIN and RDGIN signals are asserted and the LCA device is selected, the WRFLT signal is asserted. The WRFLT signal is negated when the command transferred to the LCA device Register A is the Reset command defined by the ESDI standard, which has a Command Function of 0101 (Control) and a Command Modifier of 0000 (Reset Attention and Standard Status). The Command Function and Command Modifier formats are shown in Figure 2.

Three CLBs, BPG0, BPG1, and BPG2, are responsible for generating the PARERR signal. BPG0 is a multiplexer that selects the source of the input to the parity generator/checker (BPG1). If data is being shifted into the LCA device (parity checker mode), the CMDBITA input is chosen. If data is being shifted out of the LCA device (parity generator mode), the RA_Q7 output is chosen. These two signals are also shown in Figure 11. BPG1 is an odd parity generator/checker. In the parity checker mode, whenever an odd number of ones are passed through this CLB, the output is one. This output signal is connected to BPG2, which inverts the signal and asserts or negates the PARERR signal accordingly. If parity is correct, an interrupt is asserted to the microcontroller informing the microcontroller that a command has been received and is ready to be read. In the parity generator mode, the output of BPG1 is the parity bit. BPG1 is clocked by the SHIFT input, which is asserted whenever there is a transfer request and the LCA device is selected. BPG1 is reset by the Reset Parity Generator input, RSTPGEN. This signal is asserted when either INT is asserted or an interface fault is detected.

The Interface Fault signal is asserted when the LCA device is selected and CNTR is negated before seventeen bits have been transferred. CNTR is asserted after the first command bit is shifted in and is negated after the seventeenth bit is shifted out. BINTFLT0 is clocked by the system clock, SCLK, and reset by the RSTCOS signal. The IOB PCSTS is configured as a buffered input to signal external error conditions.

The microcontroller is able to address four register locations in the LCA: two data registers, BRGA and BRGB, one error register, BDMX, and one command complete register, BRC7. Only three bits in the error register are used. They are bit 0 for parity error, bit 4 for interface fault and bit 7 for write fault. Only one bit in the command complete register is used. This is bit 0, which has a value of zero when the command is completed. The addresses and contents of the registers are shown in Table 2.

A1	A0	REGISTER	BIT NUMBER 7 6 5 4 3 2 1 0
0	0	REGISTER A	
0	1	REGISTER B	
1	0	ERROR REGISTER	
1	1	COMMAND COMPLETE REGISTER	

Table 2. Register Addresses and Contents

The LCA device implementation of the registers and multiplexer is shown in Figure 11. When the interrupt signal is asserted, the microcontroller reads the two data registers by setting the A1 and A0 address lines appropriately and asserting the RD signal to the LCA device. These data registers contain the command that is transferred from the disk controller through the CMDBITA input. If the command is a request data command, configuration or status data is written to these two data registers by the microcontroller. These two bytes, plus a parity bit that is generated by the parity generator in the LCA device, are serially transferred to the disk controller over the Config/Status Data line through RA_Q7. After all seventeen bits have been transferred, the Command Complete signal is asserted. If the command is not a request data command, the microcontroller executes the command and upon completion, writes a byte of zeros to the command complete register of the LCA. When the command complete register is written with all zeros, the Command Complete signal is asserted by the LCA device.The command complete register may only be written, not read. The status register contains error bits that are set when errors are detected. This register may only be read and not written.

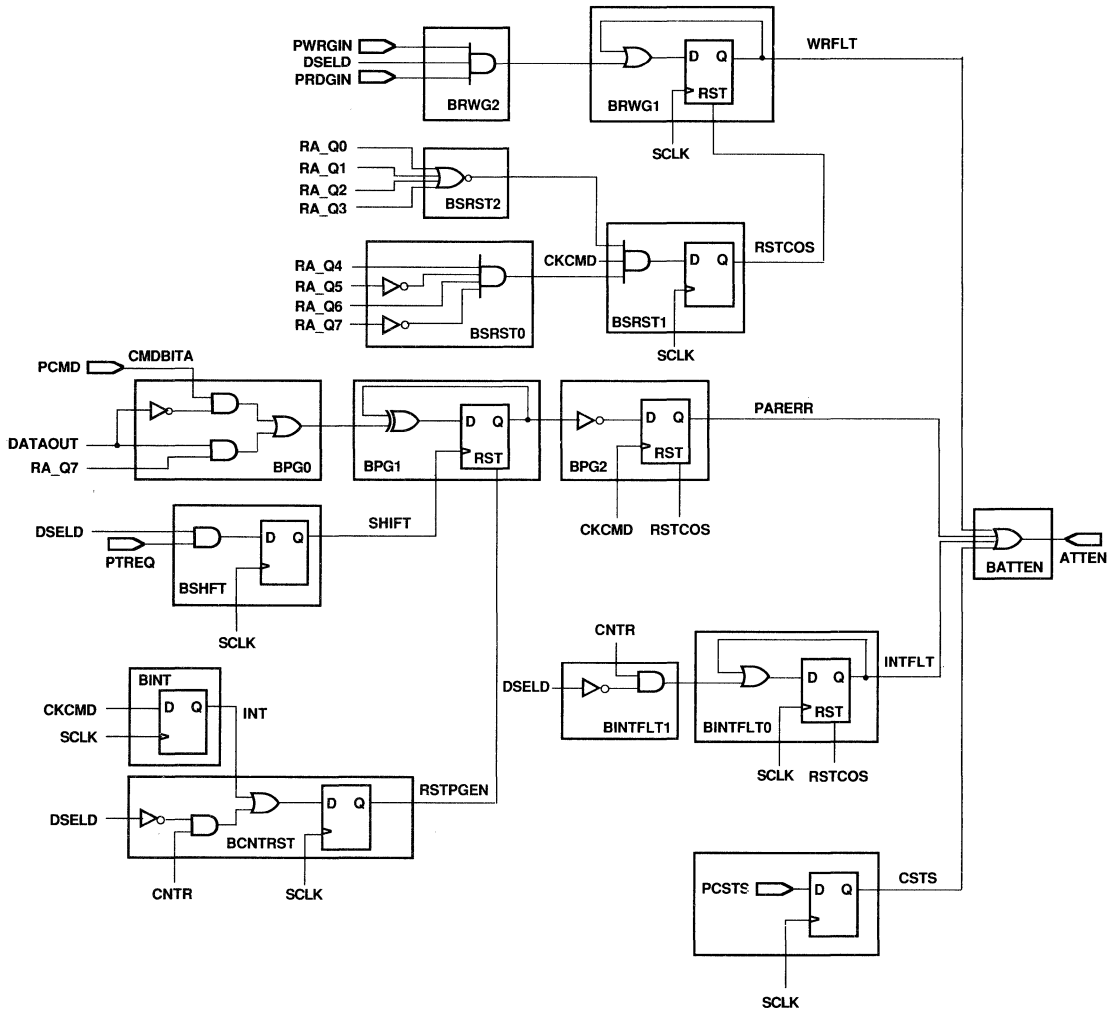


Figure 10. Generation of the Attention Signal by the Four Possible Error Conditions

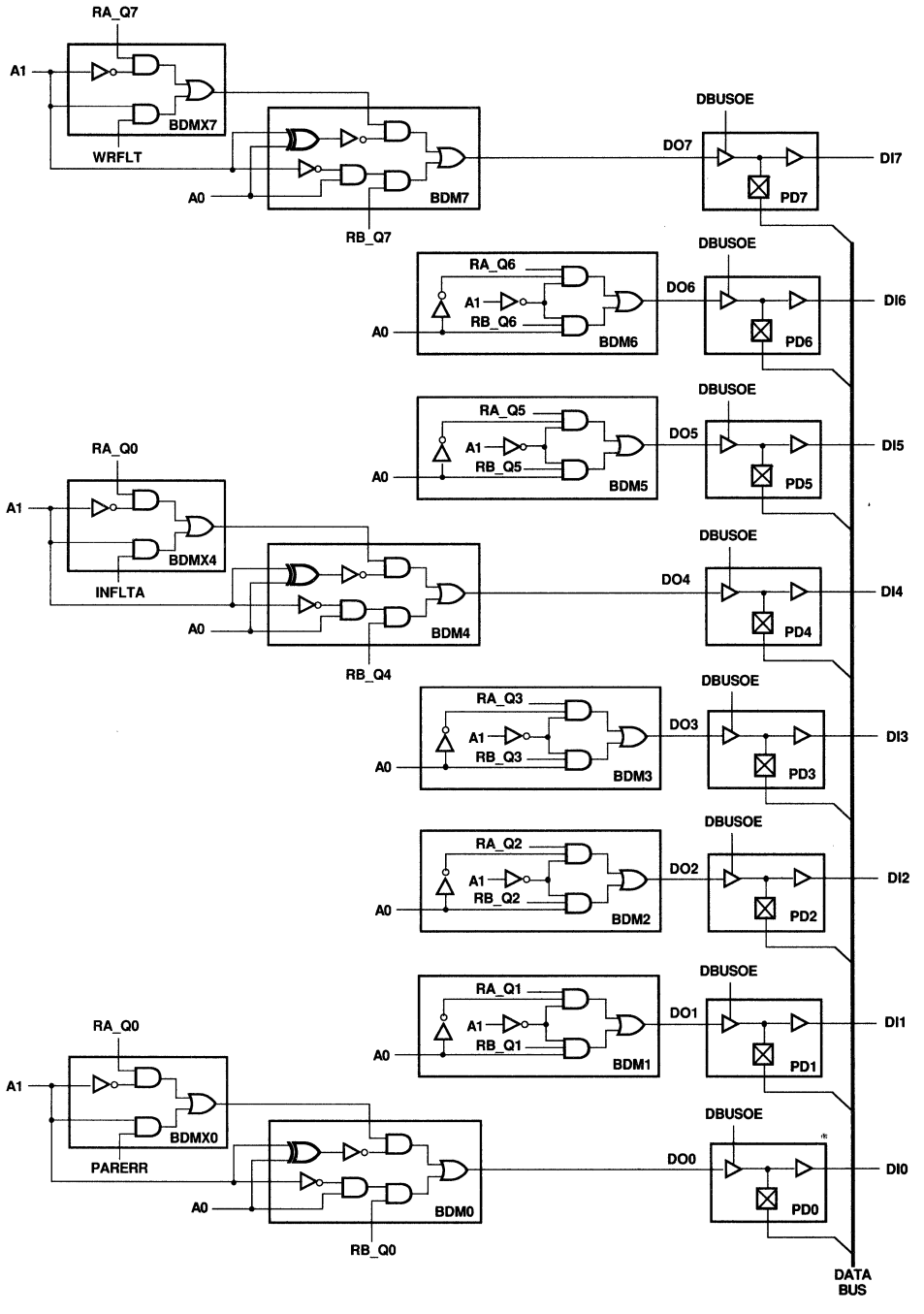


Figure 11. Registers A, B, Error Register, and Multiplexer

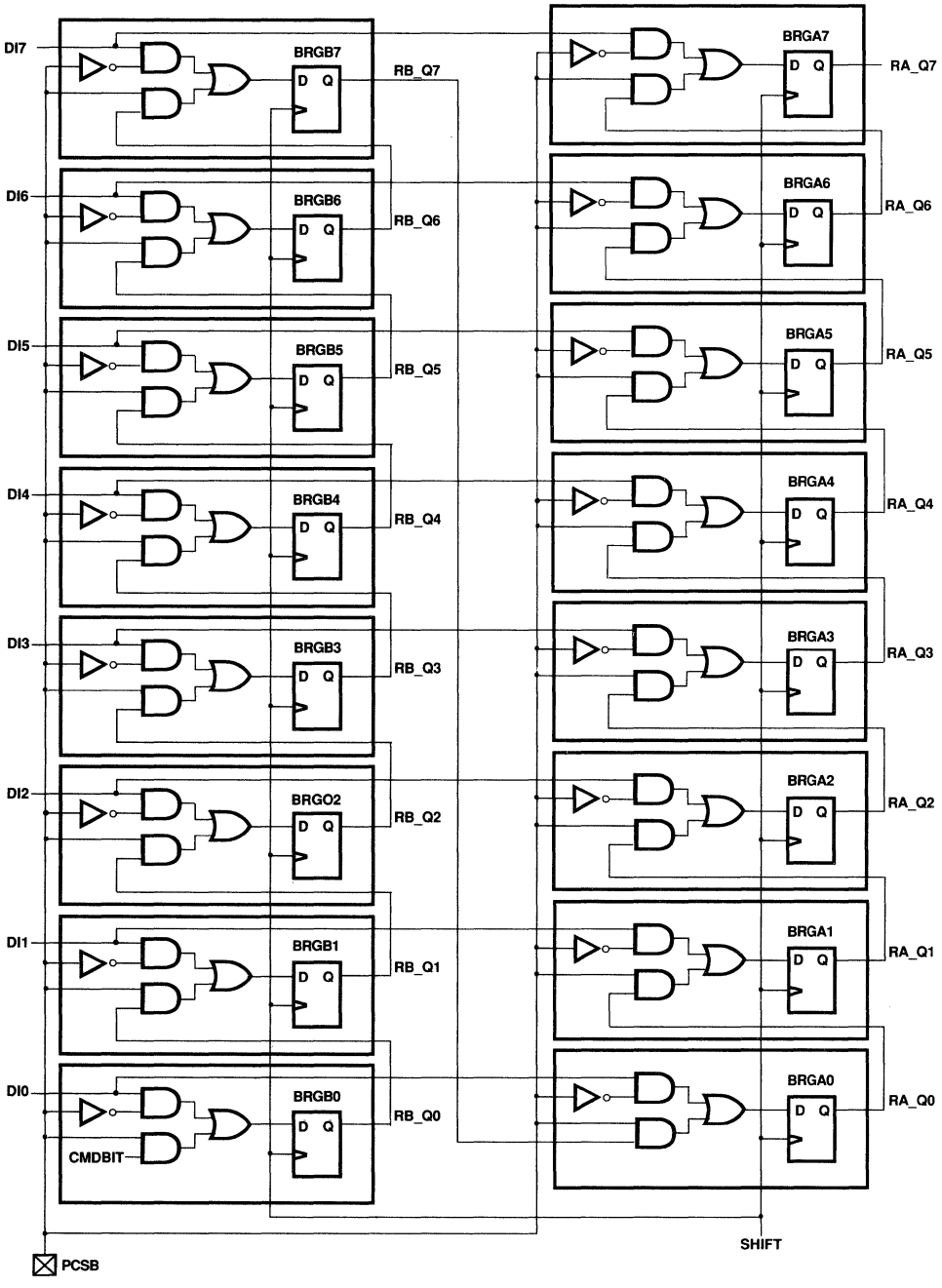


Figure 11. Registers A, B, Error Register, and Multiplexer (Continued)

Design Considerations

The circuit diagram of the programmed LCA device is shown in Figure 12. This design implementation uses sixty-three of the sixty-four CLBs within the LCA. This translates into a 98% usage. The rightmost column of CLBs contains the multiplexer, which selects between registers A, B, or the error register to be placed on the data bus. Registers A and B are placed on the third and second column from the right end, respectively. The registers and multiplexer are placed physically close to each other to simplify routing. The CLBs which execute a particular function are placed physically next to each other. These functions include the parity generator/checker, 17-state counter, read gate/write gate logic, drive selection logic and error detection logic. The read gate/write gate logic and the drive selection logic, which require much I/O activity, are placed in CLBs near the edges of the device.

Another consideration in implementing an LCA design is routing. Long lines are available for signals that have to travel a long distance within the LCA device. These lines can also be used for signals that must have minimal skew between different destinations. An automatic routing program is available in the XACT software package.

Routing requires careful consideration when a design uses a high percentage of the available CLBs (above 95%). Although automatic routing is available, designs with high CLB usage may require point-to-point routing (EDITNET command in XACT). Some of the techniques that can be used are swapping input pins, swapping CLBs, and implementing buffers in unused sections of the CLBs. When swapping input pins, the designer must be careful because certain functions, such as the clock input to the flip-flop, may only be input on certain pins. Swapping CLBs is very simple because the XACT software provides a command for swapping CLBs, but sometimes not all of the signals can be successfully rerouted. Passing a signal through a CLB presents a routing channel that is not otherwise available. However, a delay is added to the signal. Usually, a combination of these techniques can be used to successfully route a high-density design.

The high number of IOBs in the LCA device gives the designer much flexibility in designing the pinout of the device. In this design, thirty-five of the IOBs are used. The IOBs that are not being used for the actual design are not left unused, but are configured as outputs and are connected to various signals within the LCA device to provide test points for the LCA device. This greatly increases the testability of the design once it is placed on the board.

XACT is used by the designer to define the CLBs and IOBs, and to perform EDITNET. Alternately, the design may be input using the schematic capture software offered by Daisy and Futurenet. In these methods, an automatic place and route software package divides the design into blocks that may be implemented in CLBs and IOBs, and routes the CLBs and IOBs automatically. Once the design has been completed, it may be simulated using the software package P-SILOS.

Conclusion

The LCA device provides many advantages to the user. Its high gate count and I/O capability could potentially replace several PLDs in many applications, hence reducing board space. The LCA device is preferred by many customers over gate arrays because it is reprogrammable 'on-the-fly' and there are no long design cycles and initial NRE cost of a gate array. The current design file, XDES15.LCA, is available upon request. The bit pattern and the .LCA file will be provided for programming the LCA device in an EPROM.

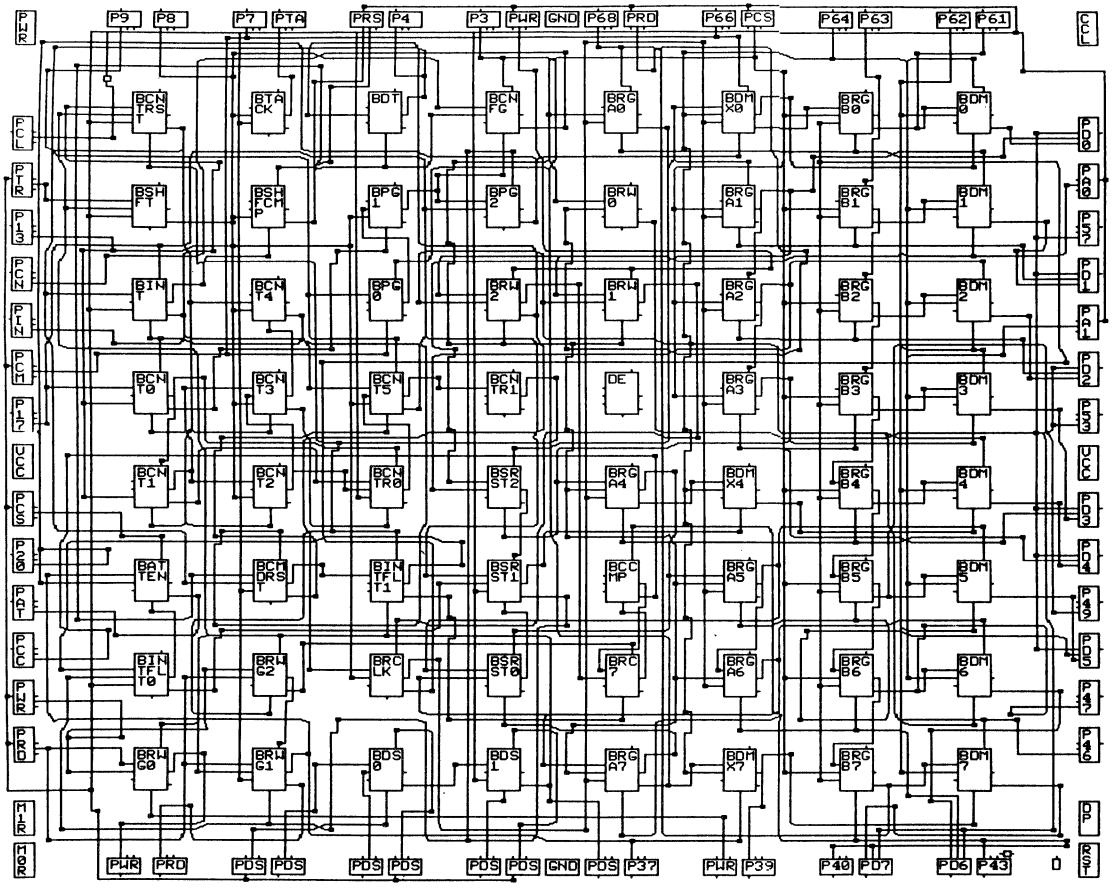


Figure 12. ESDI Translator LCA Design

Advanced
Micro
Devices

AN-184

Using the Logic Cell™ Array to Build a Pseudo-Random-Number Generator

Mohammed Wasfi

Abstract

The Logic Cell Array is a programmable integrated CMOS circuit that can easily be configured to perform many LSI functions. This Application Note will discuss how the LCA device can be used in the design of an 8-bit Pseudo-Random-Number Generator. The design consists of three major components: the registers that generate the random numbers, the 1-Hz Clock, and the decoding circuitry. The decoding circuitry can be used with two seven-segment displays to show the value of the binary random number in hexadecimal (00-FF) format.

10288A
MARCH 1988

Using the Logic Cell Array to Build a Pseudo-Random-Number Generator

Mohammed Wasfi

Introduction

The LCA device is a high-density programmable integrated circuit. Due to its flexible architecture and programmability, it can be configured to perform many functions. In this case, the LCA device was configured as an 8-bit Pseudo-Random-Number (PRN) Generator. The high density of this device made it possible to implement additional needed circuits such as the registers, clock, and the decoders for the seven-segment displays. Additional information on the LCA device can be found in the *LCA Applications Handbook.* Information on configuring the LCA device is discussed in Application Note 182.

Random Number Applications

A Pseudo-Random-Number Generator can be used in a wide range of applications. A common application for the PRN Generator is electronic games where a random number is used to create a random event. Another sophisticated application is signal scrambling, for securing communications, where a random number is added to a transmitted signal for scrambling, and subtracted for descrambling.

Pseudo-Random-Number (PRN) Generation

There are several different methods of generating a random number. In this LCA device application, however, the PRN is generated using a shift register with feedback. To achieve a random output, the feedback comes from the prime number bits (flip-flops).

Although the output of the shift register is a random number, the cycle of random numbers generated is repeated after a certain period. This is indicated by the term "pseudo." The length of the random number generation period depends on the number of flip-flops used. The period length is calculated as follows:

Period = 2^n-1, where n is the number of flip-flops.

The reason for the subtraction of one state is that the state of all flip-flops set to zero is illegal (causes a lockout).

The Logic Cell Array

The LCA device is a configurable large scale integrated CMOS circuit. It is RAM based, so its configuration can be changed by simply loading it with a new configuration bit stream. The device architecture consists of Configurable Logic Blocks (CLBs) and configurable I/O Blocks (IOBs). Each CLB has a register and can implement any one function of up to four variables or any two functions of up to three variables. The IOB can be configured as an input, registered input, output, or an output with a three-state function.

Two types of LCA devices are currently available from Monolithic Memories, the M2064 and the M2018. The M2064 has 64 CLBs arranged as an 8 by 8 matrix. The M2018 has 100 CLBs arranged as a 10 by 10 matrix. For this design, an M2064 was used because only 52 CLBs were needed to implement the design fully (Figure 1).

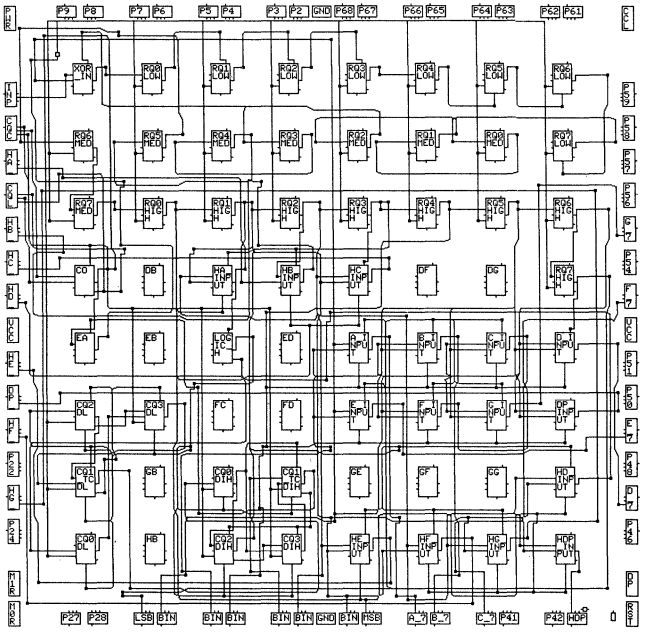


Figure 1. LCA World View

Why Use an LCA Device?

There are several reasons for choosing the LCA device to design a random number generator. The LCA device not only offers a large number of flip-flops in the CLBs, but also offers additional IOB flip-flops (one flip-flop per CLB/IOB, one hundred CLBs and seventy-four IOBs in the M2018, sixty-four CLBs and fifty-eight IOBs in the M2064). This enables the user to generate a very large period during which the PRN sequence will not repeat itself. In addition, the large number of configurable IOBs available makes it easy to generate a large random number.

One of the many features of the LCA device is its high density. There are approximately 1,200 gates in the M2064 and 1,800 gates in the M2018. The high gate count allows the implementation of additional circuits, such as the clock for the registers that normally are added outside of the chip.

Another feature of the LCA device is its configurability. For example, the ability to output a PRN of a certain size, then change the size to make it smaller or larger, or change other logic without having to use a new part is a big advantage.

With an LCA device, one has access to a macro library in the XACT™ Development System, which is the software used to configure the LCA device (available from Monolithic Memories). These macros simplify the designs considerably. Approximately 75% of this design used Monolithic Memories-supplied macros.

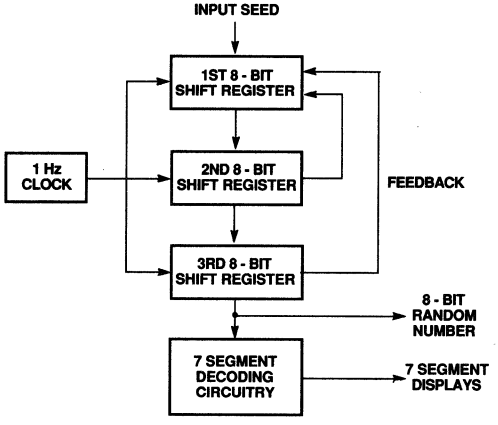


Figure 2. Block Diagram

Design Breakdown

The design of the PRN generator consists of three major components (Figure 2). The first component is the shift register. Three 8-bit shift registers are used to generate the PRNs using D-type flip flops. Since there are 24 flip-flops, the length of the PRN sequence period is:

$2^{24}-1 = 16,777,215$ states,

after which the PRN sequence repeats itself.

The second component is the clock for the flip-flops. A low frequency (100-HZ) R-C Oscillator was used. This frequency is then divided by 100 (using two modulo-10 counters) which results in a 1-Hz clock that is used for the flip-flops.

The third component is the decoding circuitry for the seven-segment displays. Two seven-segment display decoding circuits are implemented in the LCA device to show the hexadecimal equivalent of the binary output. This makes it easier to study the nature of the PRNs generated.

Design Implementation in the LCA Device

The main component in this design is the shift register. The design of a shift register is relatively simple. A register (flip-flop) in a CLB uses one input, a clock, and outputs the value that was the state of the input delayed by one clock period (Figure 5). Then all twenty-four registers in the CLBs are cascaded together forming a 24-bit shift register. The macro, RS8, available with the Monolithic Memories XACT Macrocell Library, is an 8-bit shift register (Figure 3). Therefore, it is possible to invoke that macro and cascade three 8-bit shift registers (RQL, RQM and RQH) to get the required shift register size.

The feedback from the shift register, in order to produce a random outcome, has to come from a prime number bit (register). Therefore, the feedback from bits 11 and 23 were chosen. Since the PRN sequence cannot start without a logic high input seed from an IOB configured as an input (Figure 7b), a certain function (XOR_F) implementing the input seed and the two feedbacks was used. This function guarantees lockout will not occur (such as in the case of the input staying at a logic HIGH), and also adds to the randomness of registered outputs (Figure 6).

XOR_F = (B + C) @ A

B : input seed
C : 2nd feedback (bit 23)
A : 1st feedback (bit 11)
@ : exclusive or function

The 8-bit output of the third shift register (RQH) is connected to eight (8) IOBs configured as outputs (Figure 7a). The output of RQH is also connected to the two seven-segment decoders.

The clock for the registers was chosen as 1 Hz. This frequency was generated using a low-frequency (100-Hz) resistor-capacitor oscillator. This oscillator is also available as a macro, GOSC, with the XACT Development System (Figure 8a). The values for the resistors and capacitors (R1, R2, C1, C2) used with CQ and CQL are 100 Kohms and 0.1 microfarads. The Boolean equations for the Clock are:

Q = ~R * (S+R)
QL = ~Q

A modulo-100 counter was used as a "divide by" to get the 1 HZ clock needed. The modulo-100 is implemented using two cascaded 4-bit BCD (modulo-10) counters. The macro for the counters is also available with the XACT Development System (Figure 8b). The output of the modulo-100 is routed to the Global Clock Buffer (GCB). The output of the GCB is routed to the registers via long-line interconnects.

Finally, two decoders for the seven-segment displays were designed. The least and the most significant four bits of the PRN used one decoder each to decode binary 0000-1111 as hexadecimal 0-F (Table I). The decoder outputs were routed to IOBs configured as output buffers, where they were wired to a bus driver (74LS244), that could be used to drive the seven-segment display. Eight CLBs per decoder were used (Figure 4). Each CLB was configured to control one segment in the display, including the Decimal Point (DP). See Table II.

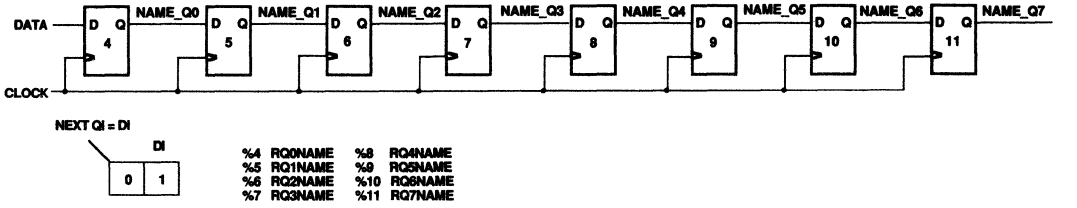


Figure 3. Shift Register Macro

4-BIT INPUT (FROM PRN)				HEXADECIMAL OUTPUT	SEGMENTS							
D	C	B	A		A	B	C	D	E	F	G	DP
0	0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	1	0	1	1	0	0	0	0	0
0	0	1	0	2	1	1	0	1	1	0	1	0
0	0	1	1	3	1	1	1	1	0	0	1	0
0	1	0	0	4	0	1	1	0	0	1	1	0
0	1	0	1	5	1	0	1	1	0	1	1	0
0	1	1	0	6	1	0	1	1	1	1	1	0
0	1	1	1	7	1	1	1	0	0	0	0	0
1	0	0	0	8	1	1	1	1	1	1	1	0
1	0	0	1	9	1	1	1	1	0	1	1	0
1	0	1	0	A	1	1	1	0	1	1	1	1
1	0	1	1	B	1	1	1	1	1	1	1	1
1	1	0	0	C	1	0	0	1	1	1	0	1
1	1	0	1	D	1	1	1	1	1	1	0	1
1	1	1	0	E	1	0	0	1	1	1	1	1
1	1	1	1	F	1	0	0	0	1	1	1	1

Table I. Decoding Table

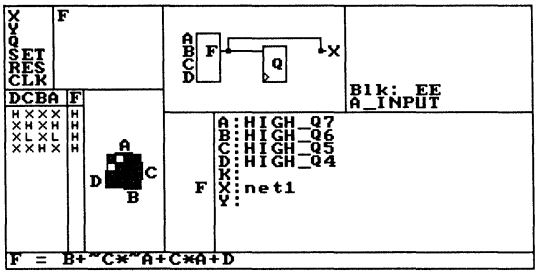


Figure 4a. A-Segment CLB

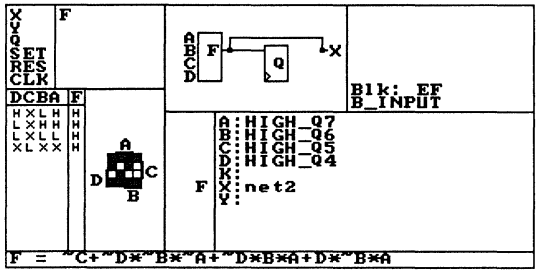


Figure 4b. B-Segment CLB

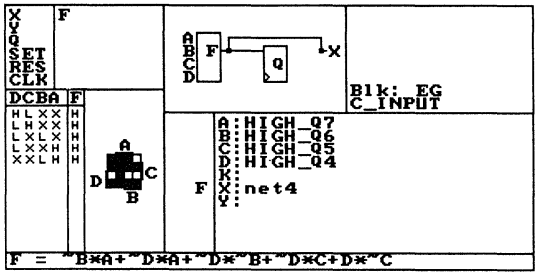


Figure 4c. C-Segment CLB

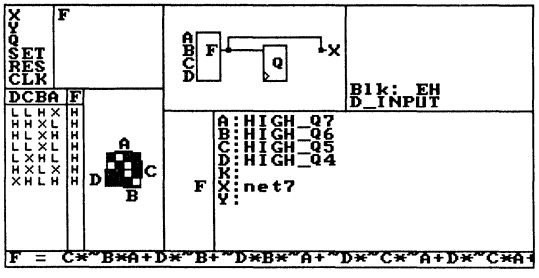


Figure 4d. D-Segment CLB

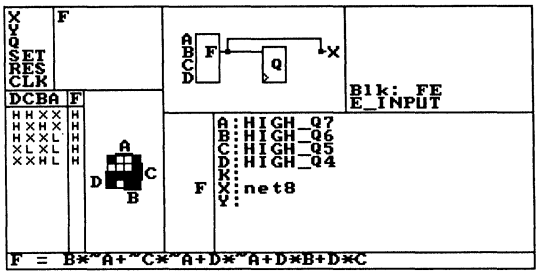


Figure 4e. E-Segment CLB

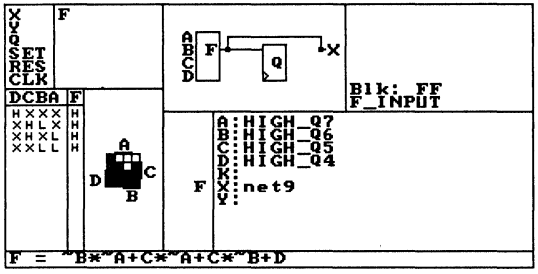


Figure 4f. F-Segment CLB

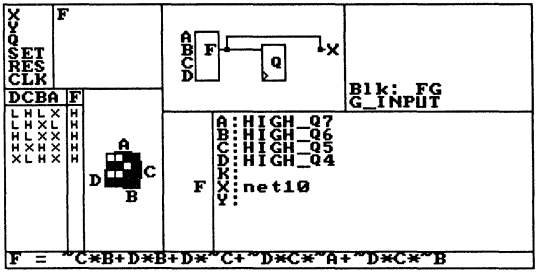


Figure 4g. G-Segment CLB

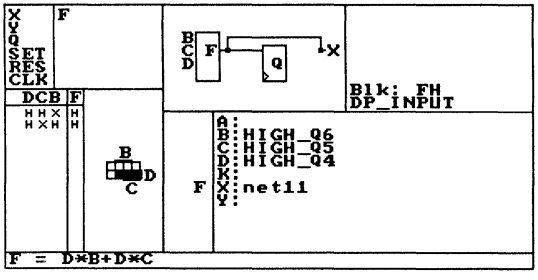
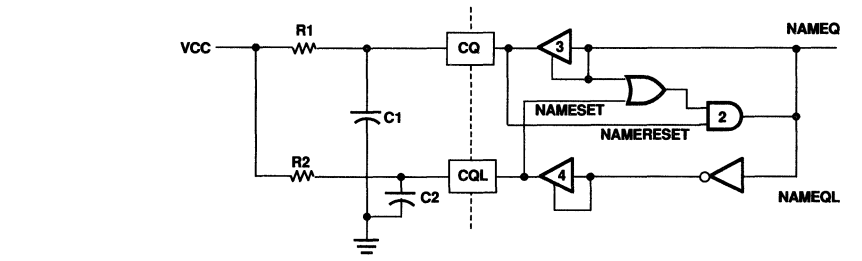
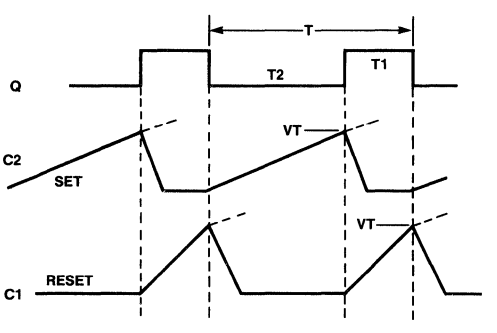
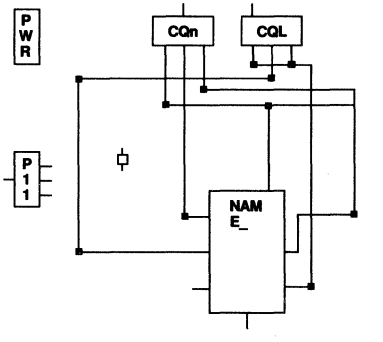


Figure 4h. DP-Segment CLB

Figure 4. Seven Segment Decoder







T = T1 + T2 = N ((R1 C1) + (R2 C2))
where N = approx. 0.35 for TTL threshold
= approx. 0.75 for CMOS threshold
when each capacitor is allowed to be discharged by the LCA during opposite timing phase
Capacitor might partially charge due to a delayed three-state routing.

Q = ~R * (S + Q)
QL = ~Q

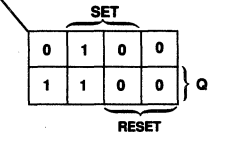


Figure 8a. Clock Macro

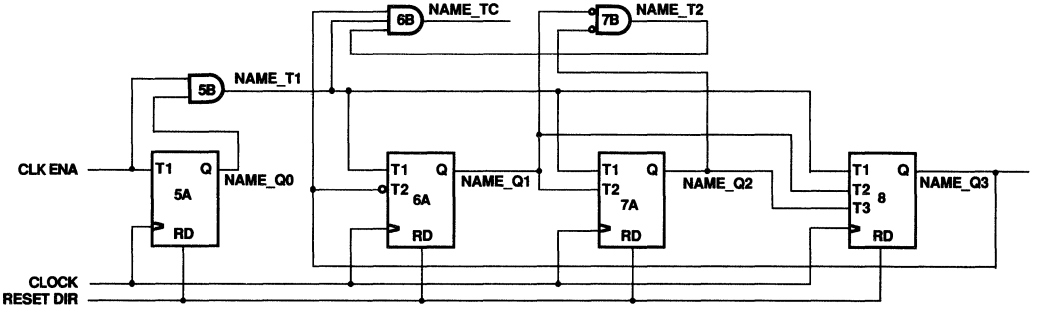


Figure 8b. BCD Counter Macro

Figure 8. Register Clock Components

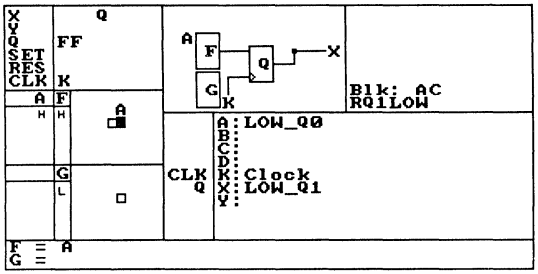


Figure 5. Shift Register CLB

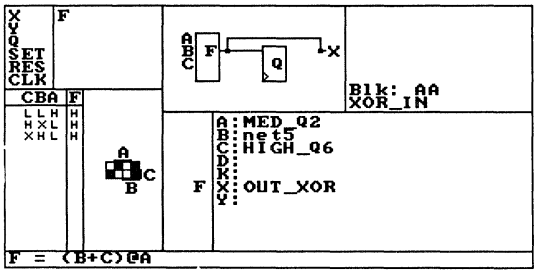


Figure 6. XOR_F CLB

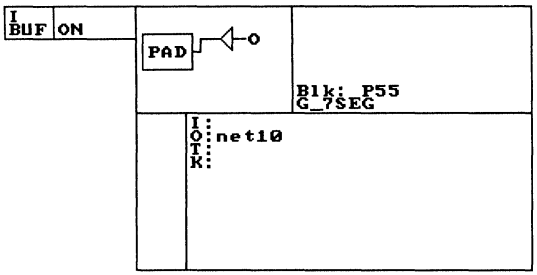


Figure 7a. Output CLB

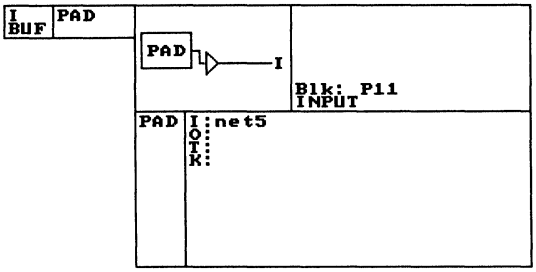


Figure 7b. Input CLB

Figure 7. I/O CLB

3

PIN	NAME	DESCRIPTION
1,35	GND	Ground
52,18	VCC	Power
11	INPUT	Input seed
12,14	CQ,CQL	rc oscillator inputs
13	HA_7seg	Decoder outputs for segments in high order display (a)
15	HB_7seg	b segment
16	HC_7seg	c segment
17	HD_7seg	d segment
19	HE_7seg	e segment
21	HF_7seg	f segment
23	HG_7seg	g segment
43	HDP_7seg	dp segment
38	A_7seg	Decoder outputs for segments in low order display (a)
39	B_7seg	b segment
40	C_7seg	c segment
47	D_7seg	d segment
49	E_7seg	e segment
53	F_7seg	f segment
55	G_7seg	g segment
20	DP_7seg	dp segment
29	LSBIT_out	Least significant PRN bit
30	BIN6_out	PRN bit
31	BIN5_out	PRN bit
32	BIN4_out	PRN bit
33	BIN3_out	PRN bit
34	BIN2_out	PRN bit
36	BIN1_out	PRN bit
37	MSBIT_out	Most significant PRN bit

Table II. LCA Device Pins Used and Their Description

Conclusion

From the PRN Generator design, the advantages of using the LCA device in such an application were shown. The configurability of the LCA device makes it very versatile. The M2064 was capable of replacing several parts that normally would be needed to implement this design. The availability of the macros also helped considerably in the design implementation. It is always possible to modify certain features in this design if needed, such as the clock frequency or the size of the PRN generated, by simply re-routing certain lines in the LCA device. These features make the use of the LCA device very desirable. This design file is available from Monolithic Memories upon request. Please ask for XDES16.LCA.

References

1. Nadia Sachs, *"Pseudo-Random-Number Generator (a Disguised PAL),"* Monolithic Memories Application Note AN-118.
2. LCA device Macrocell Library Book, Monolithic Memories Inc.
3. Edward Valleau, *"Configuring the LCA Device,"* Monolithic Memories Application Note AN-182.

Advanced
Micro
Devices

AN-185

64K Deep FIFO - Dynamic RAM Controller is Implemented in the M2018 LCA Device

Karen Spesard and Chris Jay

Abstract

First-in, first-out (FIFO) buffers are used extensively in interface and communication systems where it is necessary to provide temporary storage between two asynchronously operating devices. For relatively shallow FIFOs, up to 128 locations deep, dedicated register-based FIFOs are the best solution. For medium and deep organizations, up to 64K deep, the register architecture would be unsuitable because of the high package count required for implementation. Also, with a large number of registered FIFO devices, power dissipation could be very high and lead to system reliability problems. Using the high-storage density of RAM and controlling it as a FIFO (in other words, a FIFO-RAM Controller) solves the problem of a high package count in medium to large FIFO arrays.

The two types of RAM to be considered are static RAM (SRAM) and dynamic RAM (DRAM). SRAMs usually require more board space to produce large memory arrays because their packages are larger than DRAM packages. As a result, SRAMs may be used for medium-sized FIFO organizations, from 128 bytes to 8K bytes. For large FIFO arrays, SRAM devices occupy too much board space. DRAMs, used as stand-alone memory cells require the support of more components to handle multiplexing and refresh, whereas SRAMs can be used without that support. To address this issue, a low-chip-count, low-cost DRAM solution (FIFO-DRAM Controller) has been developed for large FIFO buffers in the M2018 Logic Cell™ Array (LCA device). The LCA device addresses the DRAM memory chips as a FIFO array and provides the interface and refresh control signals to the DRAMs.

The LCA device, a high-density programmable CMOS device, has been programmed to perform all of the necessary logic functions for a large FIFO - DRAM Controller. It can control 64K X 4, X 8, X 16, X 32 (and so on) DRAMs. The M2018 design handles the refresh, read and write functions as well as handshake activity to external circuits. Status flags, such as FIFO full and empty, are also provided to the two asynchronous transmitter and receiver circuits. In addition, since the LCA circuit is reconfigurable, the design can be modified as needed to meet specific design requirements.

10255A
FEBRUARY 1988

64K Deep FIFO - Dynamic RAM Controller is Implemented in the M2018 LCA Device

Karen Spesard and Chris Jay

Introduction

Data communication often requires buffering of large amounts of information between asynchronously operating devices along a data channel. For instance, in a video teleconferencing system, one Mbyte of data per video image (1K x 1K pixels) is generated. After compression, 32 Kbytes of each image must be passed to a buffer to wait for transmission along the data channel. The next image is then free to be processed by the system. For optimal performance, first-in, first-out (FIFO) buffers can be effectively used to hold the data in temporary storage sites between the transmitter and receiver nodes. The data can be stored either in a stack of registers organized as a FIFO, or a RAM controlled as a FIFO. For very large FIFO organizations, however, the use of inexpensive RAM devices is the only practical alternative. It keeps the overall system size to a minimum which will, in effect, reduce the overall system cost.

Dynamic RAM (DRAM) devices, when controlled, can be used as a temporary buffer. Though relatively inexpensive and small in size, compared with static RAM (SRAM) devices, stand-alone DRAMs require more complex interface and refresh control circuitries. As a result, more components could be required in DRAM systems. To reduce the overall chip count, a Logic Cell Array (LCA device) has been programmed to perform as a 64K x 4, x 8, x 16, x 32, (and so on) FIFO-DRAM Controller. The Controller allows the DRAM to function as a FIFO by providing the control circuitry necessary for operation.

With the design implemented in the M2018 LCA device, minor changes in the design can be implemented quickly and easily due to the device's re-programmability feature. For example, if the type of DRAM or any of its parameters had to be changed for any reason, the original LCA device can accommodate the necessary logic modification.

The LCA device provides an intelligent approach to designing circuits in an efficient and timely manner. It can speed up the design process significantly (to just a few days) with low-cost user-friendly software tools that do not require a non-recurring expense (NRE), as gate arrays often do. As a result, many products can be introduced quickly and economically. The LCA device can automatically load itself and is field programmable and reconfigurable. Therefore, it can be used for system development, for existing design modifications and for volume production. This application note describes the design methodology used for the development of the Controller in the LCA device.

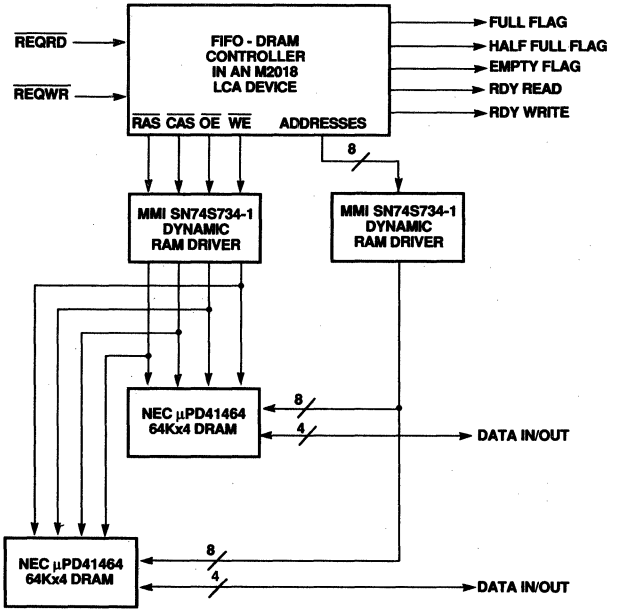


Figure 1. A Typical DRAM Interface Using the FIFO-DRAM Controller in the M2018 LCA Device

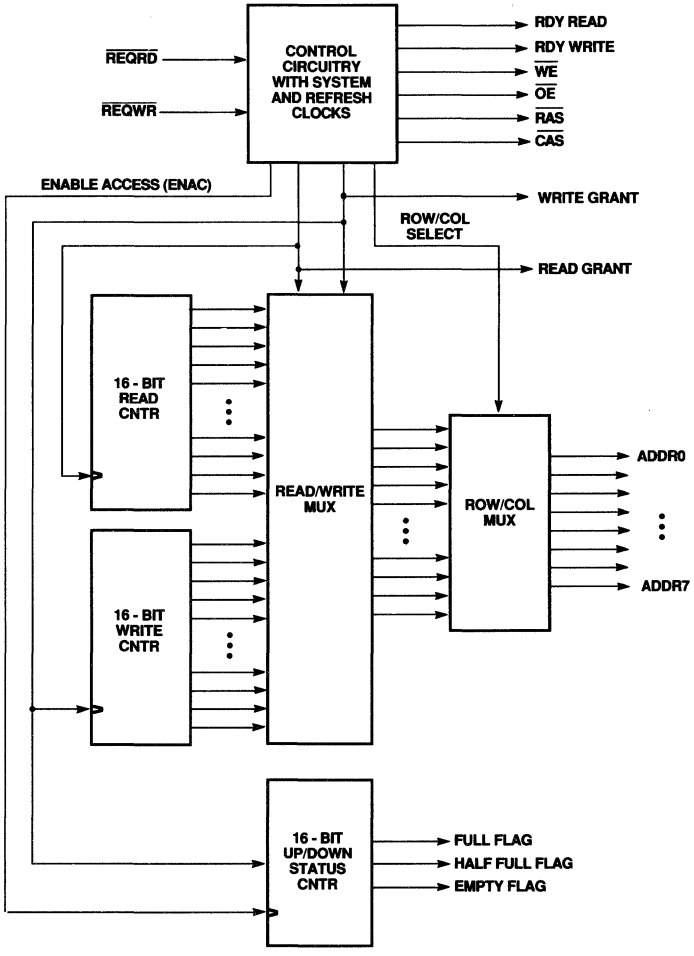


Figure 2. Block Diagram of the FIFO-DRAM Controller

The FIFO-DRAM Controller

The FIFO - DRAM Controller in the LCA device, together with an array of two DRAMs and two DRAM drivers, comprises a FIFO memory. (See Figure 1.) The Controller handles the row and column address multiplexing by applying a $\overline{RAS}$ before $\overline{CAS}$ cycle for memory read and write operations. Refresh timing, however, is applied as a $\overline{CAS}$ before $\overline{RAS}$ cycle, which takes precedence over read/write activity. The DRAMs used for this application must have on-chip refresh counters. The μPD41464 64K x 4 DRAMs manufactured by NEC were used in this application because they have a $\overline{CAS}$ before $\overline{RAS}$ internal address refresh mode. Thus, no refresh counter was needed in the design of the FIFO-DRAM Controller. The Controller in the LCA device also provides for access and refresh timing via the control signals, $\overline{RAS}$, $\overline{CAS}$, $\overline{OE}$, and $\overline{WE}$, and provides for read/write status flags.

The block diagram of the controller is shown in Figure 2. The only controller inputs are the request-to-read ($\overline{REQRD}$) and request-to-write ($\overline{REQWR}$) signals. These signals are active LOW and come from some external logic or a microprocessor. They drive the control circuitry section of the device, which generates the $\overline{CAS}$ and $\overline{RAS}$ signals, among others. The FIFO-DRAM Controller in the LCA device also consists of an address generation section and a buffer status section. Two internal 16-bit counters and 24 2:1 multiplexers generate the addresses for the DRAM array. A 16-it up/down counter generates the status information for the three status flags: full, empty, and half-full.

The Control Circuitry

The control circuitry provides the ready-to-read (RDY READ) and ready-to-write (RDY WRITE) signals for the system. Other signals include the row/column select (ROW/COL SELECT) signal for multiplexing the addressed data, the enable access (ENAC) signal for clocking the status counter, and the read and write grant signals for the three 16-bit counters. It also provides the four active low signals which directly interface with the DRAM: the row address strobe ($\overline{RAS}$), the column address strobe ($\overline{CAS}$), the write enable ($\overline{WE}$), and the output enable ($\overline{OE}$) signals.

The logic used for generating each of the control signals is shown in Figure 3. Much of this logic was derived from the published input signal specifications of the DRAM. For example, the $\overline{RAS}$ (A), ROW/COL SELECT (B), and $\overline{CAS}$ (C) logic was created from the write cycle, read cycle, and CAS before RAS refresh cycle timing waveforms published in the NEC μPD41464 datasheet. Samples of these timing cycles are given in Figures 4, 5, and 6. The state diagrams were then produced from these waveforms as in Figure 7A. Next, the state excitation maps or Karnaugh maps for each signal were constructed. Reducing the logic for each map and "OR"ing the enable access and enable refresh cycles together, produced the appropriate registered equations as in Figures 7B, 7C, and 7D. The logic for the $\overline{OE}$ and $\overline{WE}$ signals were generated in a similar manner.

To resolve simultaneous read, write, and refresh requests, arbitration logic was added. This was done while creating logic for the other control outputs. The logic designed in the controller gives the refresh request highest priority and the read request the lowest priority. (Refresh requests should have the highest priority because data integrity must be maintained. Write requests have the next highest priority because data is typically held on the data bus for a specific period of time and must be written before being lost.) Therefore, the read grant signal will not go HIGH until refresh and/or write cycle requests already pending have been completed. Likewise, the write grant signal will not go HIGH until a pending refresh cycle request has been completed. Also, when a refresh cycle is requested, both the RDY READ and RDY WRITE signals will be held LOW until the refresh cycle has been finished. This insures that any read and/or write requests already in queue will be completed first. The timing diagram illustrating the refresh and write priority is shown in Figure 8.

The request-to-read and request-to-write registers are reset after every read and write operation, respectively. Accordingly, the refresh register is also reset after every refresh operation. When no requests are pending, a register "sets" or holds the $\overline{RAS}$, ROW/COL SELECT, and $\overline{CAS}$ signals HIGH. The signals are now ready for the next request.

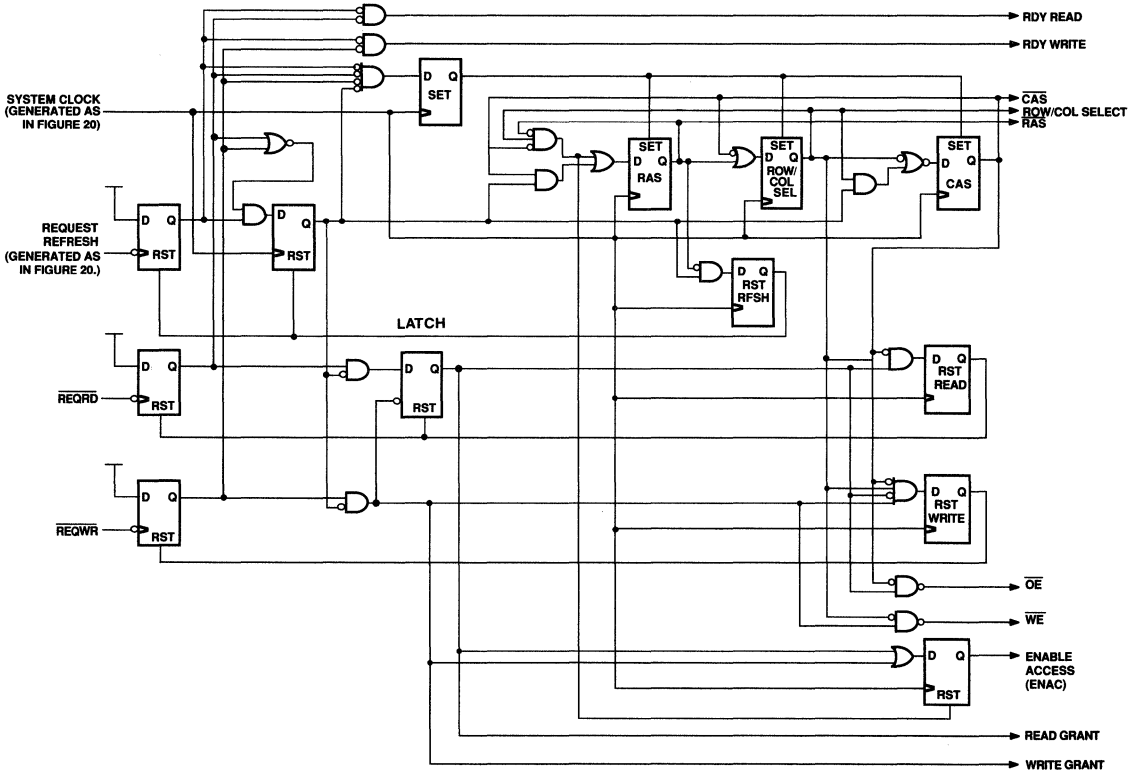


Figure 3. Control Circuitry for the FIFO-DRAM Controller

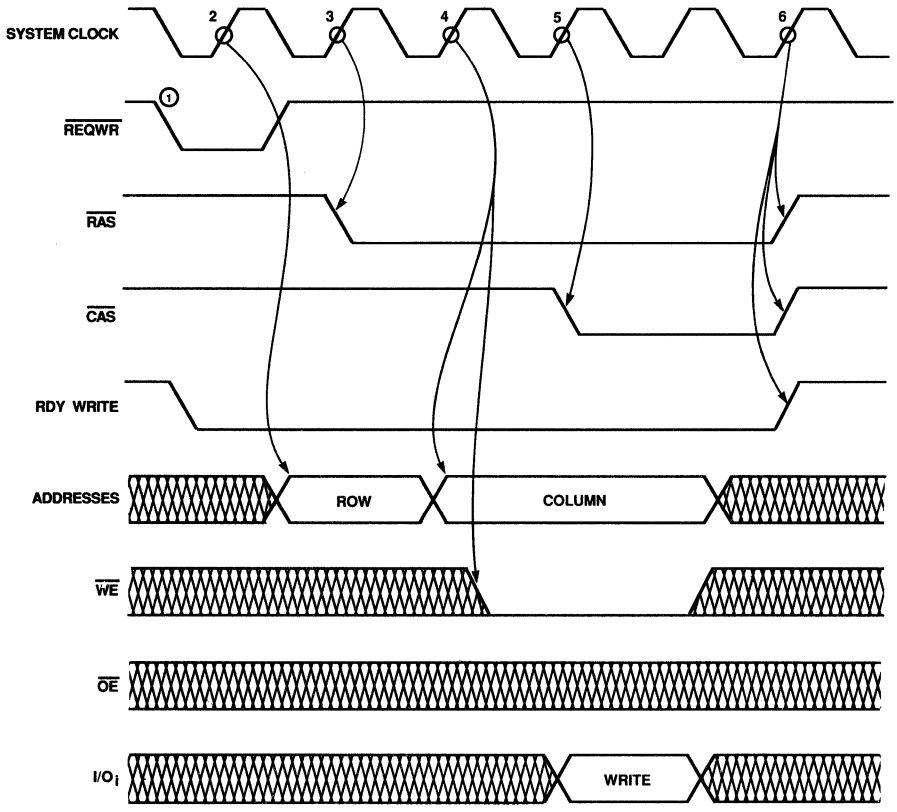


Figure 4. Write Cycle Timing

Figure 4.

1. The request to write REQWR line transitions LOW to enable an active write cycle. An active LOW transition starts the write process. The ready RDY WRITE output goes LOW to indicate that a current write cycle is in progress and no further writes should be attempted until it goes HIGH again.
2. The first active HIGH transition of the system clock sets the ROW address outputs from the write counter to the address output pins.
3. The second clock rising edge enables the active LOW transition of the RAS output to the DRAM array.
4. The third clock edge switches an internal address multiplexer which sets up a valid column address output on the input to the memory address lines. This signal is internally gated with the request to write flag to produce an early LOW active write output signal to the DRAM devices.
5. The fourth clock edge strobes the CAS output with the WE input LOW for a valid write. The OE input can be either a HIGH or LOW during this write cycle.
6. The internal state machine controlling RAS and CAS activity holds the WE LOW for one and a half clock cycles and the RAS and CAS line LOW for two clock cycles. The CAS and RAS lines are taken inactive simultaneously. The ready write signal, RDY WRITE goes HIGH to indicate that another write cycle may commence.

Note:

The frequency of the system clock is determined by the selection of external components which are configured to an internal oscillator. Details of this are shown in Figure 20.

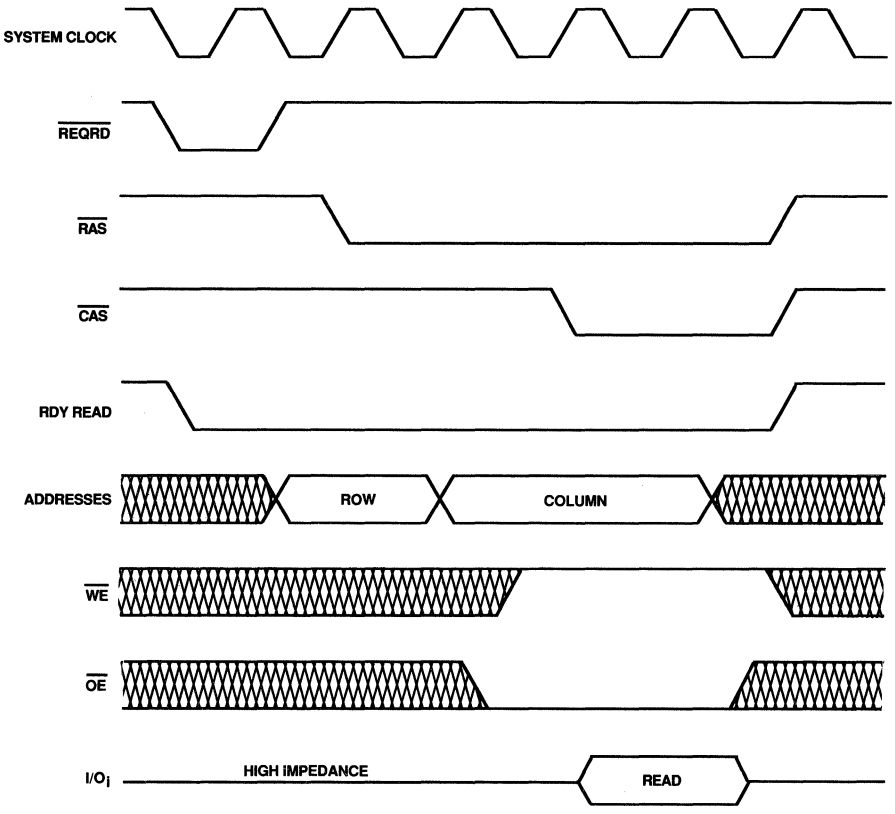


Figure 5. Read Cycle Timing

Figure 5.

1. The request to read REQRD input transitions LOW to enable an active read cycle the ready to read output, RDY READ, from the FIFO DRAM Controller goes LOW to indicate that a read cycle is in progress and no further read cycles should be requested until the read cycle is completed.

The timing of row address valid, active RAS, column address valid and active CAS is identical to that shown in Figure 4. The only difference in the timing is that the WE line stays inactive HIGH and the OE signal is gated active LOW one clock period after the LOW transition of RAS.

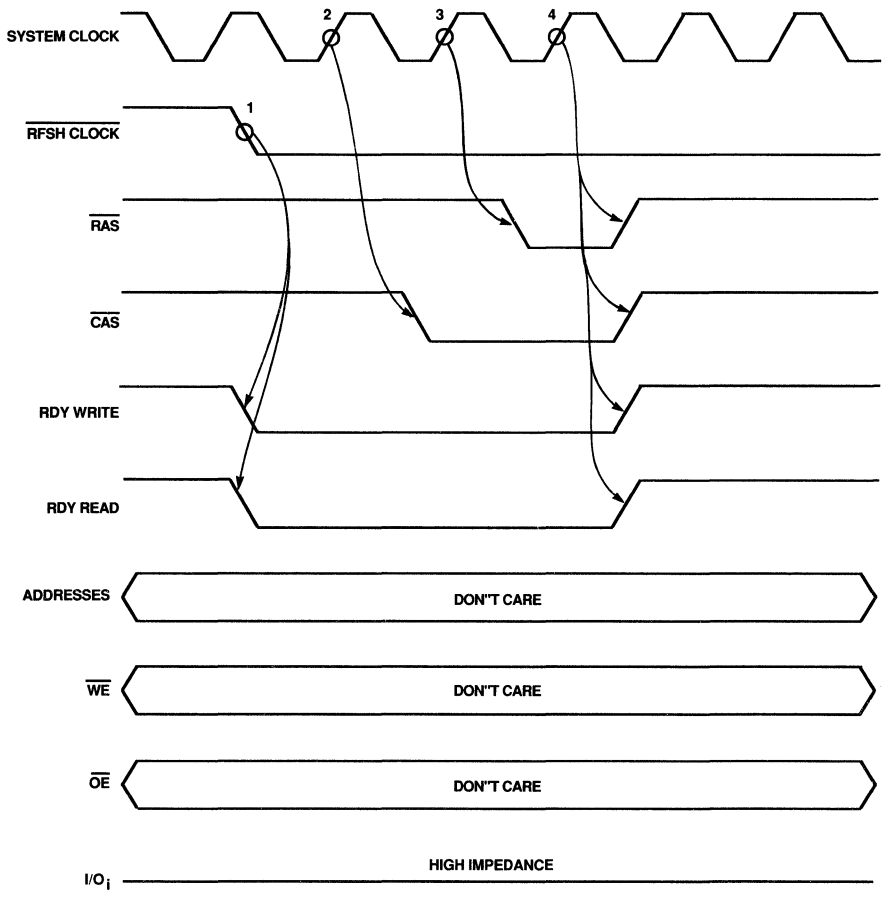


Figure 6. $\overline{CAS}$ Before $\overline{RAS}$ Refresh Cycle Timing

Figure 6.

1. The RFSH CLOCK is driven active LOW to initiate a $\overline{CAS}$ before $\overline{RAS}$ refresh cycle. Ready to Read and Ready to Write Controller outputs are driven LOW to allow an uninterrupted DRAM refresh.
2. The $\overline{CAS}$ output goes LOW after the first clock edge.
3. The $\overline{RAS}$ output goes LOW following the second clock edge.
4. The $\overline{RAS}$ and $\overline{CAS}$ outputs go inactive after the third clock edge and RDY WRITE and RDY READ go HIGH to indicate that read and write operations may recommence.

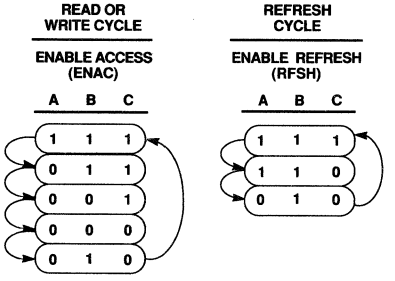


Figure 7a. State Diagrams for Read and Write Access Enable (ENAC) and Refresh Enable (RFSH) where A = RAS Signal, B = ROW/COLUMN ADDRESS SELECT Signal, and C = CAS Signal

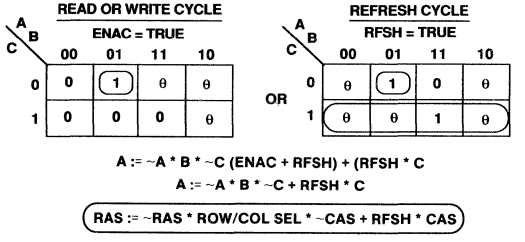


Figure 7b. State Excitation Map for RAS (A) Signal and Corresponding Equation

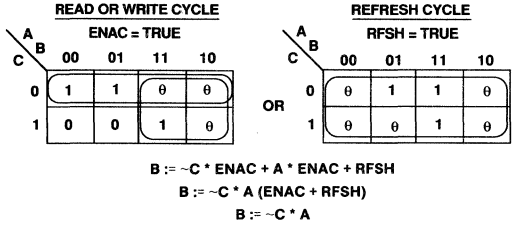


Figure 7c. State Excitation Map for ROW/COLUMN ADDRESS SELECT (B) Signal and Corresponding Equation

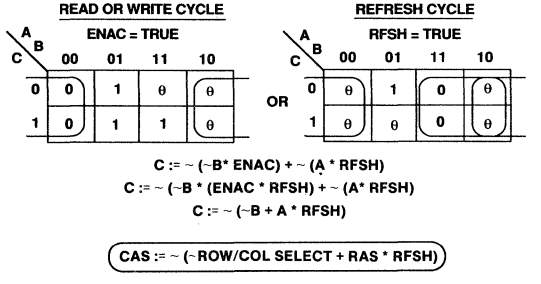


Figure 7d. State Excitation Map for CAS (C) Signal and Corresponding Equation

Figure 7.

1. Figure 7a shows the truth tables for the controlling state machine within the FIFO DRAM controller. Two tables should be considered, one for access, read and write activity, and one for DRAM refreshing. The Enable access ENAC and enable refresh RFSH are mutually exclusive events such that ~RFSH may be considered as ENAC and ~ENAC as RFSH. When ENAC is TRUE (1) an access cycle is taking place, and RFSH will be FALSE (0). Refreshing takes place when RFSH is TRUE and ENAC is FALSE. The Karnaugh map shown in Figure 7b represents the state excitation map for the RAS output during access and refresh cycles. A logic one entry represents an active condition and a logic zero a passive state. The Greek letter theta represents a "don't care" state and can be included in minimization considerations. For the map shown in Figure 7b, minimization yields;

ENAC = 1, A := ~A*B*~C 1

RFSH = 1, A := ~A*B*~C + C 2

now, combining 1 and 2;

A := ~A*B*~C*ENAC + ~A*B*~C*RFSH + RFSH*C 3

reduces to

A := ~A*B*~C + RFSH*C 4

because

ENAC + RFSH = 1.

substituting the values of RAS, ROW/COL and CAS for A, B and C respectively we get;

RAS := ~RAS*(ROW/COL SEL)*~CAS + RFSH*CAS

The equations for ROW/COL SEL and CAS are derived in a similar way.

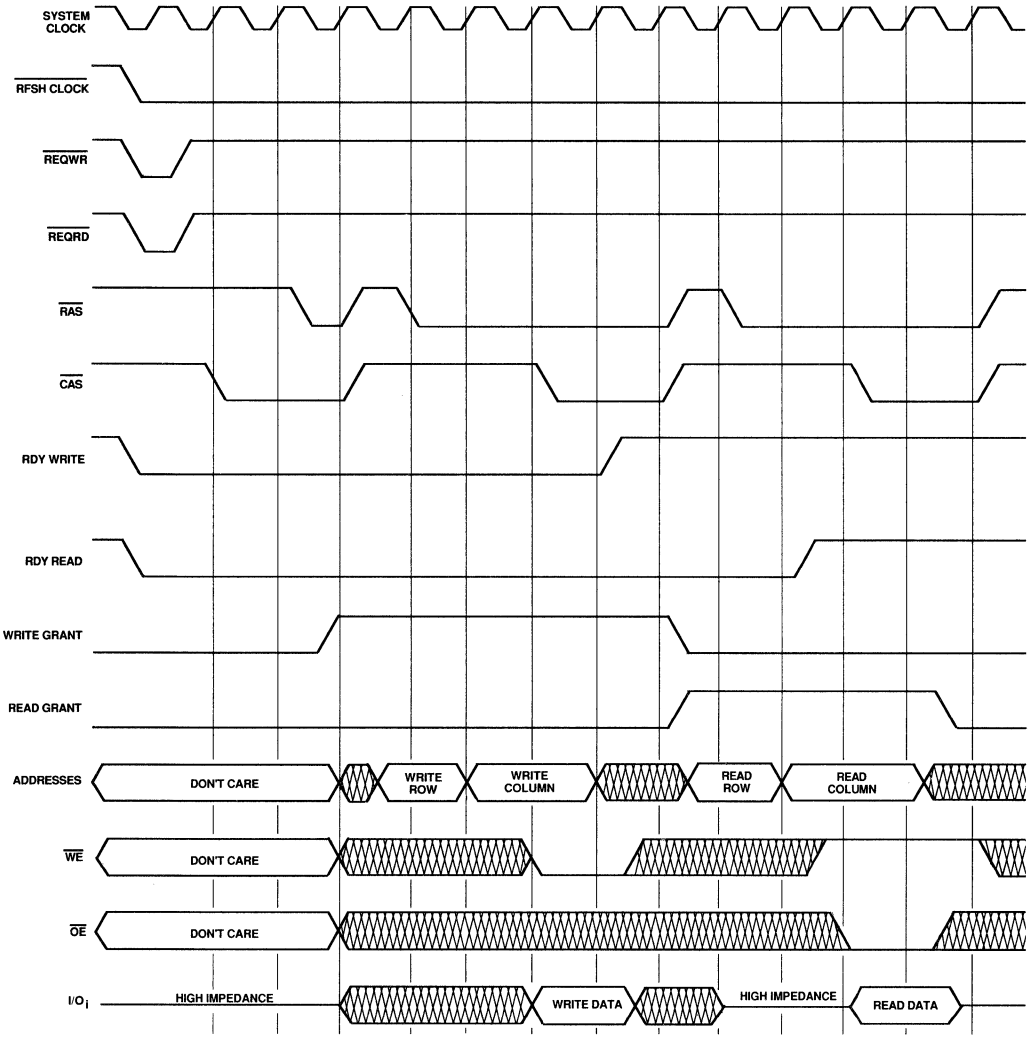


Figure 8. Timing with Refresh and Write Priority

The system and refresh clocks are generated internally with two resistor-capacitor oscillator networks each. The system clock can be designed to run as fast as 5 MHz. The refresh clock, which is the request to refresh signal, is designed to operate at 15.6 μs or 64 KHz. This is because the DRAM must be refreshed with 256 refresh cycles in a period of 4 ms. (The implementation of the oscillator network will be described in the System Clock and Refresh Request Clock Configurations section.)

READ/WRITE Address Circuitry

The FIFO-DRAM Controller in the LCA device is based on a FIFO architecture and consists of a read and write pointer. The addresses of these pointers are generated by two 16-bit counters and 24 2:1 multiplexers, as shown in Figure 9. When the write grant signal is activated, the first 16-bit counter is incremented. The write address is then multiplexed through to the row/column select multiplexer. As the ROW/COL SELECT signal goes HIGH, the first eight bits of the counter (the row address of the DRAM array) are transferred onto the address bus, and as the ROW/COL SELECT signal goes LOW, the second eight bits of the counter (the column address of the DRAM array) are transferred onto the address bus. An equivalent sequence also occurs when a read grant signal is activated.

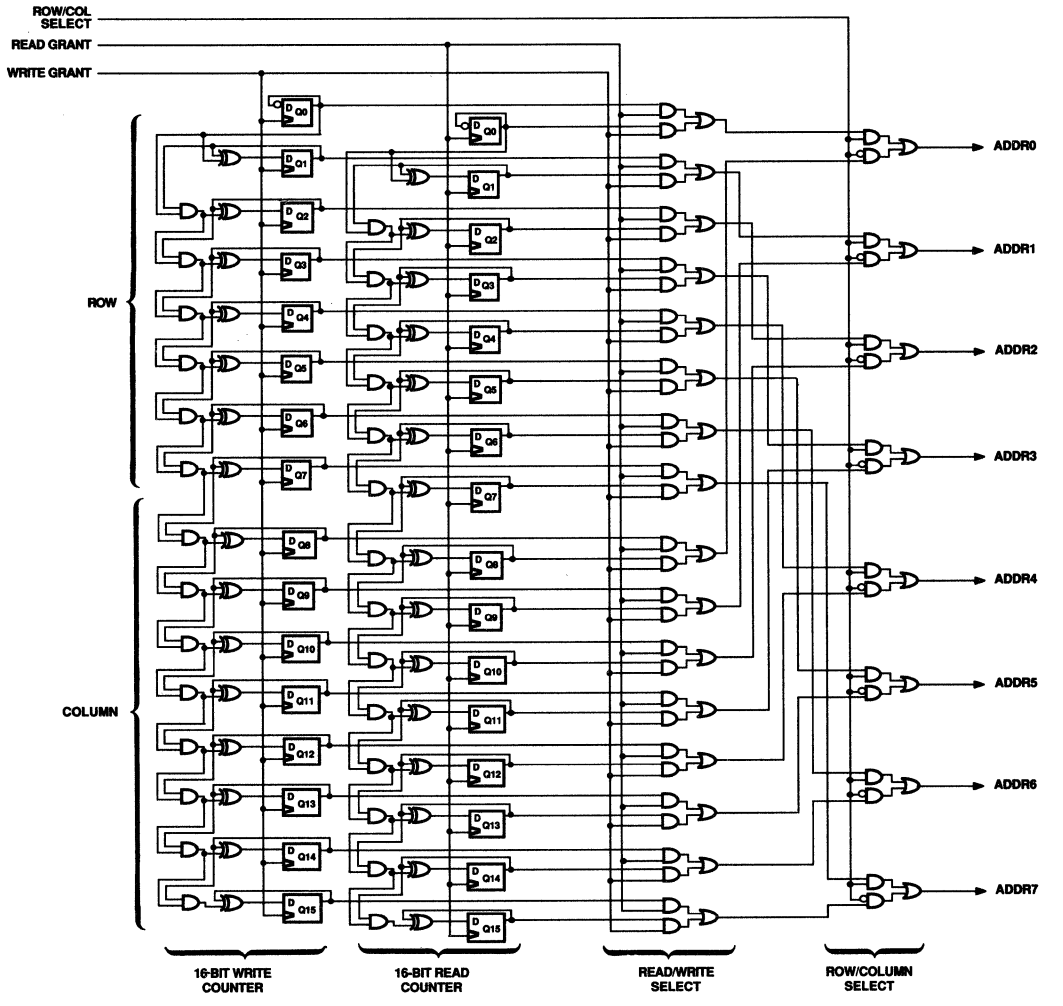


Figure 9. Read/Write-Row/Column Address Generation

The Status Counter

Information about the FIFO buffer is provided by the three status flags: full, half-full, and empty, which are all active HIGH. The full and empty flags indicate overflow and underflow conditions, whereas the half-full flag gives an indication to a processor of the speed at which the buffer is being filled. It acts as a monitor rather than as a prevention flag.

The flags are constructed from the status counter, a 16-bit up/down counter shown in Figure 10. The clock input to the counter is the enable access signal that "OR"s the write grant and read grant signals. When the write grant signal is HIGH, the counter is incremented, and when the write grant signal is LOW, the counter is decremented. For example, if the count reaches 0000 Hex during a read operation, the FIFO buffer is empty, and if the count reaches FFFF Hex during a write operation, the buffer is full. If the count reaches 7FFF Hex, the buffer is half-full.

Logic Cell Array Implementation

The FIFO-DRAM Controller design was implemented in one chip: the M2018 Logic Cell Array, a high-density, programmable CMOS device. This device contains 100 Configurable Logic Blocks (CLBs) containing one register element each. The register element can be configured as a D-type flip-flop or latch, or it can be bypassed totally, supplying a strictly combinational output. The two available CLB outputs can be configured as a function of four Boolean input variables, or two functions of three input variables each.

The M2018 also contains 74 configurable input/output blocks (IOBs). Sixty-four of these blocks can be configured to perform a variety of logic functions. Each has the capability to drive an output, receive an input, clock the input into a flip-flop, or provide both input and output capability under three-state control.

Together, the CLBs and the IOBs provide 1800 usable logic gates. These gates are configured when data is loaded into the LCA device for programming — usually from an EPROM or microprocessor. For example, the configuration data file, <filename>.prm, to be programmed into an EPROM, is generated using the program, MAKEPROM (part of Monolithic Memories' XACT™ Development System), which loads the bit stream into PROM memory locations. The input to the MAKEPROM program, <filename>.bit, is generated from the MAKEBITS program which outputs the bit stream of the current design. The design is entered by either using one of the schematic capture-based systems from Daisy, Mentor Graphics®, OrCAD™, and Futurenet® or by partitioning the required logic and inserting it in a graphics environment on the IBM® PC-XT/AT® using Monolithic Memories' XACT™ LCA Editor. The LCA Editor was used to create the LCA design of the FIFO-DRAM Controller. The design implementation using CLBs and IOBs is described below.

CLB Implementation

The $\overline{\text{RAS}}$, ROW/COL SELECT, and $\overline{\text{CAS}}$ outputs are implemented in four CLBs. The RAS output utilizes two CLBs but also includes the enable access signal (ENAC2), which is reset with part of a RAS function. Figure 11 shows the CLBs that comprise the RAS output. The COMRASB signal, in Figure 11A, is configured from a three-input combinational function (the Y output-G base) and is an input to the RAS block, in Figure 11B. This signal is "OR"ed with two more signals. The RAS block, configured as a D-type flip-flop with a "set" function, produces the $\overline{\text{RAS}}$ output. The CLB configurations for the ROW/COL SELECT and $\overline{\text{CAS}}$ outputs are shown in Figures 12 and 13. The CLB equations correspond to the equations for each signal given in Figure 7.

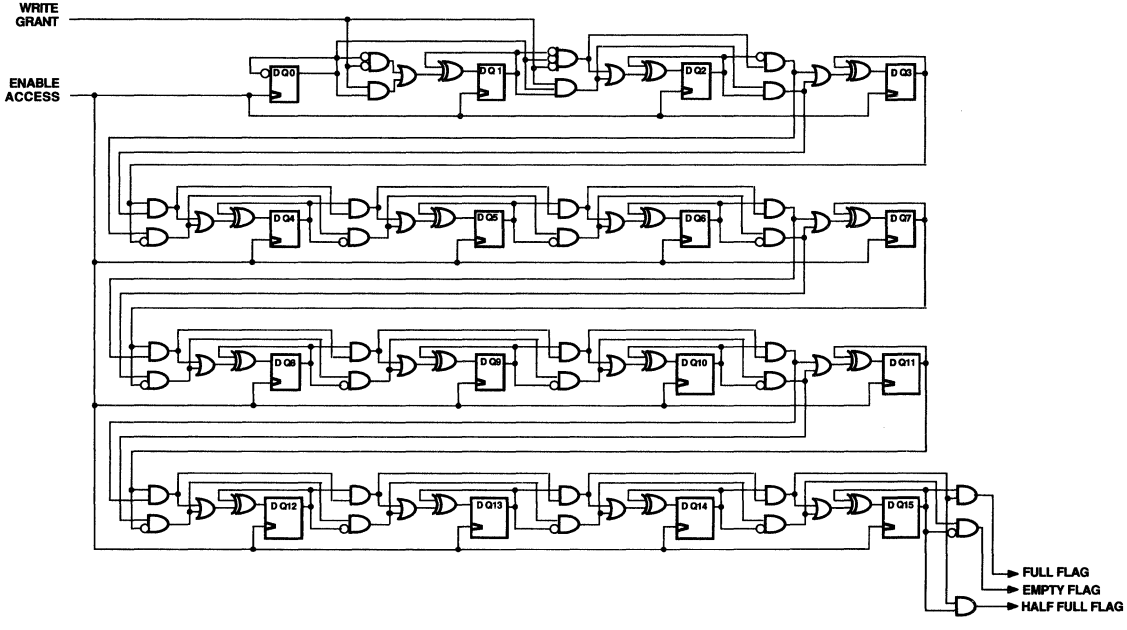


Figure 10. 16-Bit Up/Down Status Counter Circuitry

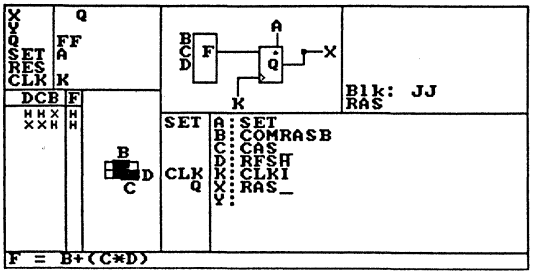


Figure 11a.

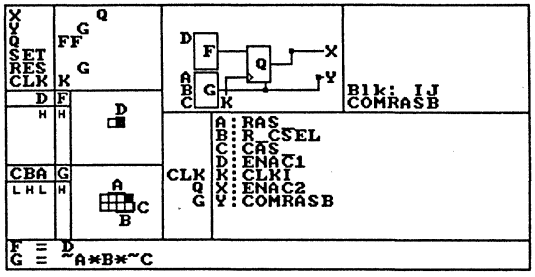


Figure 11b.

Figure 11. CLBs which Comprise the $\overline{RAS}$ Signal. The Y Output of Figure 11a, COMRASB, is the B Input to Figure 11b.

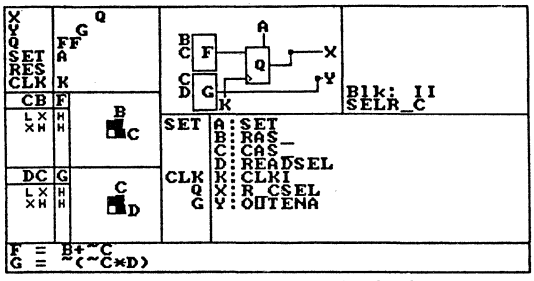


Figure 12. CLB Configuration for the Row/Column Select Signal

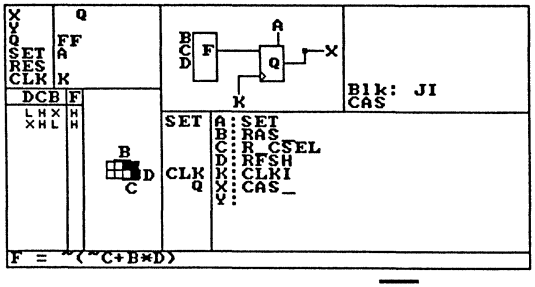


Figure 13. CLB Configuration for the $\overline{CAS}$ Signal

The CLB configuration for the 16-bit read counter is shown in Figure 14. The first (row - read) bit, RRQ0 on output Y, is illustrated in Figure 14A. It is generated through a flip-flop and is clocked with the read select signal. The terminal count for the next bit is generated from RRBIT0, the X output. Figure 14B shows the configuration for the second bit of the read counter.

The read/write address multiplexer is designed with four Boolean input variables as in Figure 15. The RRQ0 signal (row-read) is selected if the read select signal is active HIGH, and the RWQ0 signal (row-write) is selected if the write select signal is active HIGH. This implementation is combinational and has only one output.

The row/column address multiplexer is designed with a CLB internal multiplexer base, FGM, as in Figure 16. When the ROW/-COL SELECT signal is HIGH, the ROW address, MA0, is selected, and when the ROW/COL SELECT signal is LOW, the COLUMN address, MB0, is selected. This last multiplexer is registered and outputs the address that is output to the DRAM.

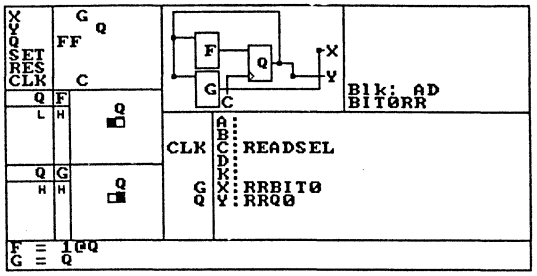


Figure 14a.

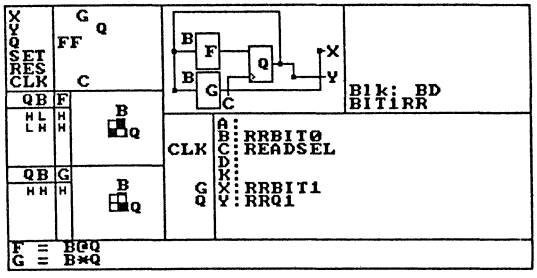


Figure 14b.

Figure 14. Two CLBs which Produce Bit0 and Bit1 of the Row/Read Address Bit Respectively

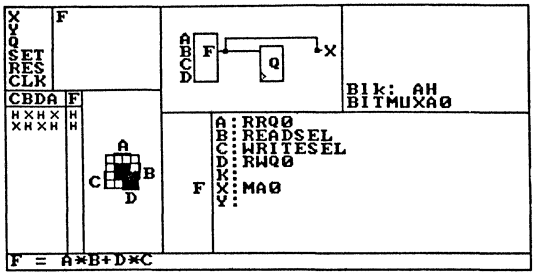


Figure 15. CLB Configuration of the Read/Write Select Multiplexer

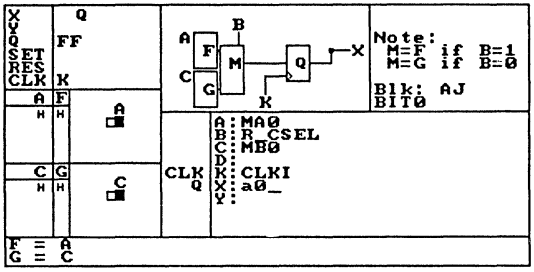


Figure 16. CLB Configuration of the Row/Column Select Multiplexer

The up/down status counter implementation is shown in Figure 17A. The second status counter bit is produced from the Y output, a registered function of input C2X, C2Y and the feedback Q. The flip-flop clock is ENAC, the DRAM access signal. The X output is a combinational function of C2X and Q. The third bit is produced from a registered equation as shown in Figure 17B.

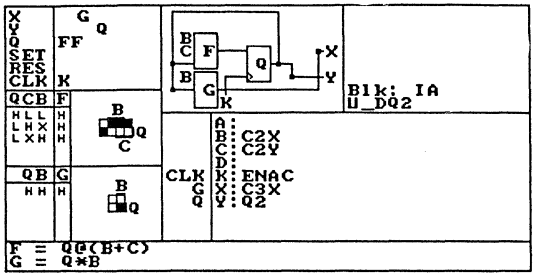


Figure 17a. The Third Bit of the Status Counter

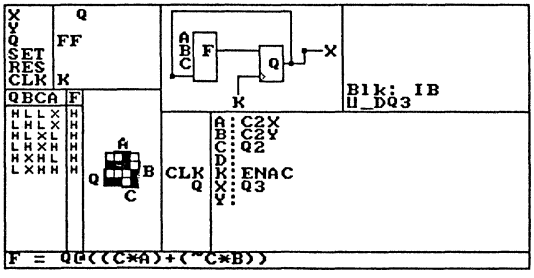


Figure 17b. The Fourth Bit of the Status Counter

Figure 17. Two CLBs which Produce the Third and Fourth Bits of the Status Counter

IOB Implementation

An example of an IOB configured to receive data and drive inputs is shown in Figure 18A for the write request (WRR) block. An IOB configured as an output buffer for the RAS signal is shown in Figure 18B. The placement of the IOBs, like the CLBs, was optimized for the best circuit layout.

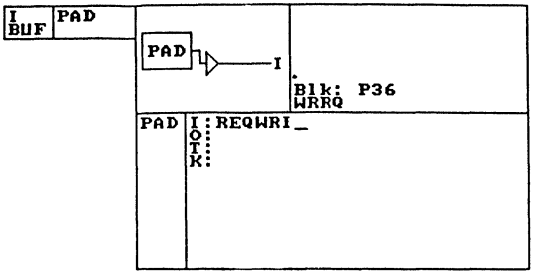


Figure 18a. The Input IOB Configuration Used for the Write Request Signal

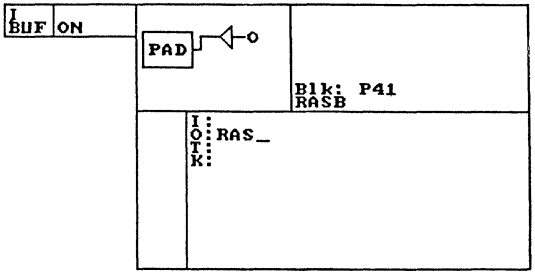


Figure 18b. The Output IOB Configuration Used for the RAS Signal

Figure 18. Two IOB Configurations Used in the FIFO-RAM Controller Design

System Clock and Refresh Request Clock Configurations

The system and refresh request clocks were designed with two general purpose oscillator networks. Each oscillator was built using two IOBs, one CLB, and two external resistor/capacitor networks — R1, C1, and R2, C2. The IOBs and CLBs were configured by calling up the dedicated macro, GOSC, from Monolithic Memories' Macrocell Library, available as part of the schematic capture entry package or the XACT EditLCA entry package. The LCA implementation of the relaxation oscillator is shown in Figure 19. Here, the programmable routing connections of CLB block RFSCLK and IOB blocks CQLrfs and CQrfs are shown for the refresh clock, Figure 20 illustrates the logic schematic for both the system and refresh clocks and the correct connections to the resistor/capacitor network.

The CLB and IOB configurations for the generation of the refresh clock are shown in Figure 21. Notice that in both the IOB configurations, the bidirectional three-state buffer is used. Notice also that in the CLB configuration, the output X, RFSQ, is the same as the input D. This is possible even though the function is not registered.

The calculation of the resistor/capacitor values for the clocks is given below. For an even mark/space ratio, R = R1 = R2 and C = C1 = C2. Each timing phase (1/2 of a cycle period) is given by the following formula:

T1 = 0.35 (C1 * R1 * 2) for TTL voltage thresholds, and

T2 = 0.75 (C2 * R2 * 2) for CMOS voltage thresholds.

The general expression for the calculation becomes:

T = N * ((R1 * C1) + (R2 * C2))

where N is .35 for TTL and 0.75 for CMOS.

The resistor and capacitor values chosen for the system clock are R1 = R2 = 30 ohms and C1 = C2 = .01 microfarads. These components are connected to pins 11 and 13 and result in a frequency of 5 Mhz. The values chosen for the refresh request clock are R1 = R2 = 2.2 Kohms and C1 = C2 = .01 microfarad. They are connected to pins 24 and 28 and result in a frequency of approximately 64 KHz. The GOSC macro IOBs can be connected to any pins as long as they are not connected to pins dedicated to configuring the LCA device. Dedicated configuration pins cannot be loaded with components such as capacitors because they could prevent the LCA device from reading valid data by corrupting hold and setup times.

Design Considerations

The circuit diagram of the FIFO-DRAM Controller in the LCA device is shown in Figure 22. This implementation used all 100 CLBs available in the M2018 device. The three columns of CLBs, on the left, contain the status counter. The next seven columns of CLBs contain the address counters and the multiplexers, and the two bottom rows of CLBs contain the control circuitry.

The placement of the blocks are positioned so that routing delays between the blocks can be minimized. Long line interconnects are used frequently for signals which would have to travel the length or breadth of the chip. These lines were also used for signals that must have minimal skew between destination points. For more information on routing, please refer to Chapter 9 of the LCA Design and Applications Handbook.

Summary

For DRAM designs, a FIFO-DRAM Controller was efficiently implemented in a single CMOS M2018 LCA device. The controller, which enables the DRAM to function as a FIFO, allows large amounts of data to be held in temporary storage. The design can be modified quickly and easily with the LCA device because it is configurable and reprogrammable. Configuration of the LCA device is discussed in the LCA Applications Note 182.

The FIFO-DRAM Controller design is available upon request. The design file and bit file will be provided for configuring the LCA device. Please ask for design XDES10.LCA.

References

NEC Electronics Inc. μPD41464 65,536 x 4-Bit Dynamic NMOS Ram Datasheet, January 1987

Monolithic Memories, Inc., FIFO-RAM Controller 57/674219 Datasheet, September 1986

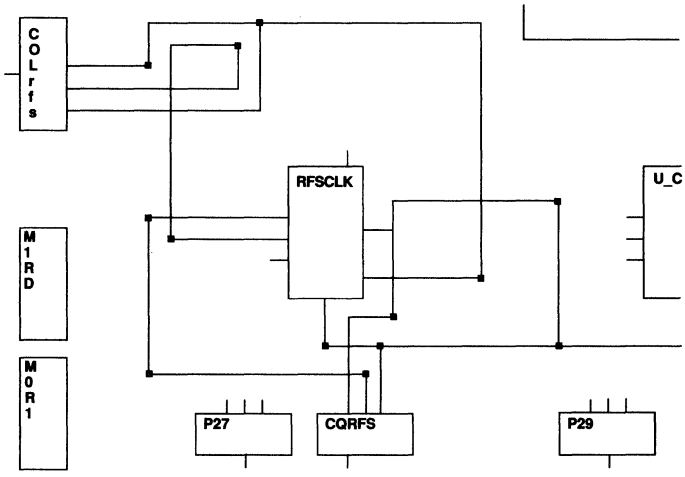


Figure 19. LCA Implementation of the Relaxation Oscilaltor with the CLB/IOB View of the External Routing Connections

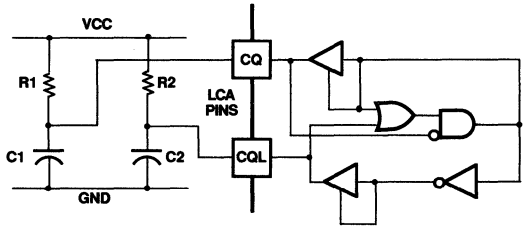


Figure 20. GOSC Macro Logic Schematic for System and Refresh Clocks

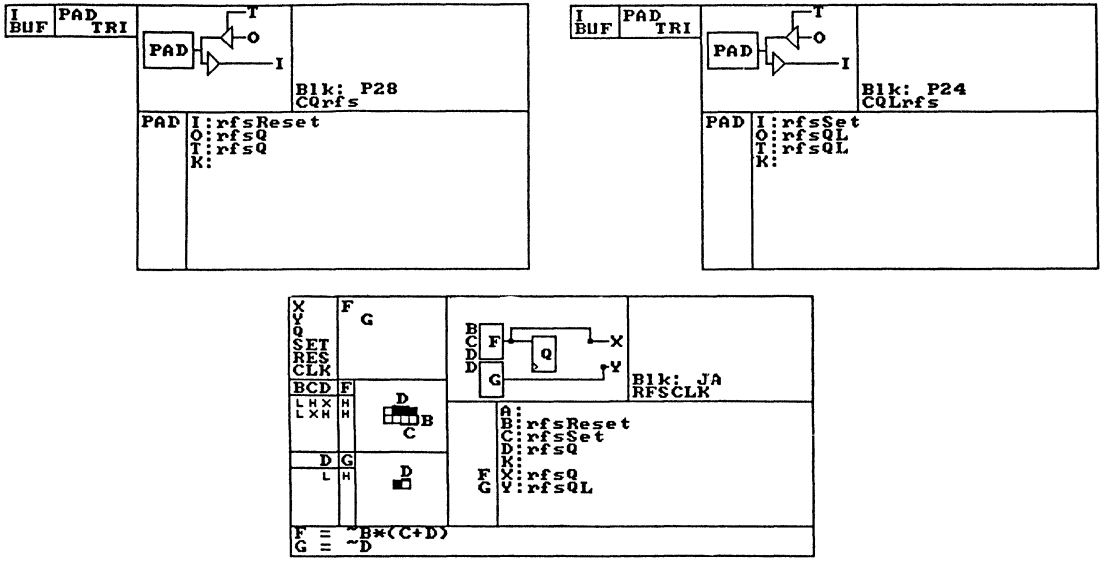


Figure 21. CLB and IOB Configurations for the Refresh Clock

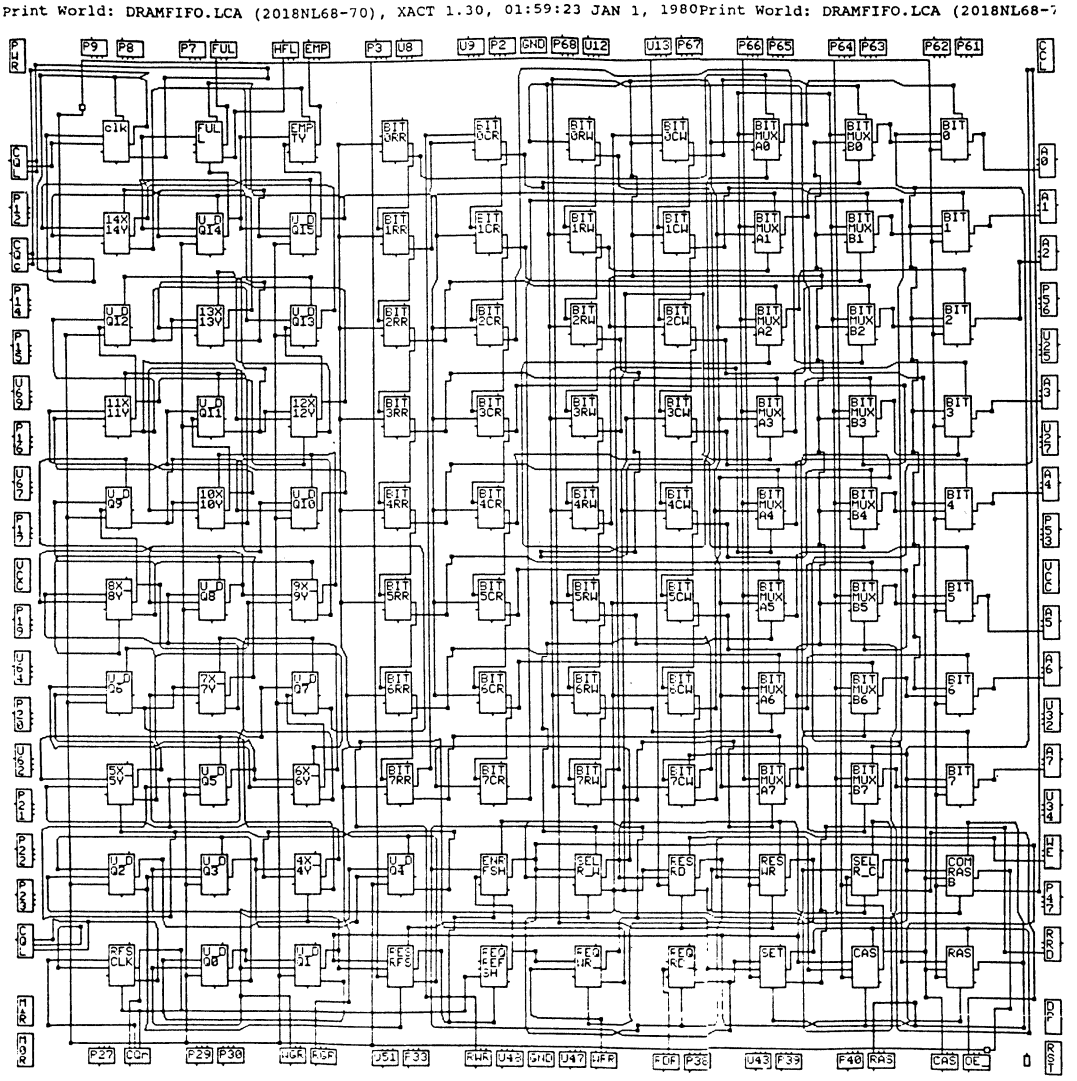


Figure 22. FIFO-DRAM Controller Circuit Implementation in the LCA Device

Configuring the LCA™ from the PC Bus

By Robert Botchek of Trantor Systems Ltd. and Chris Jay of AMD

ABSTRACT

The Logic Cell™ Array (LCA) is a high density programmable device based on a complex arrangement of static RAM cells. The device must go through an initialization and configuration cycle each time power is applied. This cycle must occur before the LCA can function as a logic unit or subsystem. The configuration time is small, ranging from 17 to 35 milliseconds for the M2000 series of devices, and when the device is fully functional no re-configuration is necessary provided power to the device is maintained.

There are a number of modes of configuration, each suited to a particular device application. This application note gives detailed information on configuring the LCA device from the IBM PC® bus, choosing one of the five configuration modes available. Configuring the LCA device directly from the PC bus presents numerous advantages to the PC adapter card designer. The direct link between the PC bus and the LCA enhances debugging and testing, and reduces the product's time to market. The concept of soft programmability (based on the LCA's SRAM design) significantly reduces the development cycle of a PC product using the LCA. Once complete, the configuration software can exist in ROM and be called each time the system boots, so PC cards containing LCA devices can be configured in a manner transparent to the user. The interface requires both software and hardware considerations. A high level language program is written to perform data code conversion, and assembly language programs supervise the actual LCA downloading and programming cycle. The hardware interface to the LCA is achieved with a low-cost PAL16L8 device. A description of hardware and software design considerations is given in the following application note.

Publication #	Rev.	Amendment
10719	A	/0

Issue Date: **May 1988**

Configuring the LCA from the PC Bus

By Robert Botchek of Trantor Systems LTD. and Chris Jay of AMD

INTRODUCTION

The Logic Cell Array (LCA) is a very high density programmable device. The 2000 series consists of two devices, the M2064 and the M2018. The M2064 consists of an eight-by-eight matrix of programmable Configurable Logic Blocks (CLBs) with fifty-eight programmmable Input/Output Blocks (IOBs) to handle a variety of input/output functions. The M2018 has a ten-by-ten matrix of CLBs with seventy-four IOBs. Both devices have a complex array of programmable interconnect which is used to connect the logic blocks and input/output functions. The M3000 series was developed from the 2000 series and is suitable for even more dense logic architechtures. The 3020 device is a development of the 2064 having an eight-by-eight array of more complex logic blocks. The IOBs and interconnect are also more complex in this new generation of programmable gate array circuits. This application note applies equally well to the configuration of 3000 series and 2000 series devices.

These high density devices require good software support tools to assist the designer in achieving a successful design with rapid turn-around, a benefit well known to the engineer familiar with Programmable Logic Devices; PLDs. Most manufacturers of PLDs support their products with at least one software package, and as complexity of programmable logic increases there is a need for more sophisticated computer-based tools. The LCA device is probably the most complex PLD on the market today, but fortunately there exists a wide repertoire of software support packages to provide the logic systems designer with rapid design entry and logic verification. In addition, conversion software provides a bridge from the finished design into raw configuration data which in turn is suitable for downloading and configuring the device. LCA device design entry is well supported in the PC environment with a wide range of CAE tools. The XACT™ software is designed to run on a PC XT/AT, for the 2000 series of LCA products only, and PC/AT for both 2000 and 3000 family of devices. Many packages for schematic entry and simulation also exist as software support for the LCA product, and will run in the PC environment.

The route from design entry to a configuration bit stream is a relatively fast process. Thus, even if a number of reprogramming cycles are needed to prove a design, the overall process remains comparatively short.

CONFIGURATION MODES

There are five LCA configuration modes. These are listed in Table 1 along with corresponding LCA mode select inputs M0-M2. For the specific application of configuring the LCA from the PC bus the slave mode is used. In this mode, for a single LCA device, only three pins take part in the configuration process. This can be the deciding factor in using the slave mode because some of the IOBs have a dual function of address and data assignment in the master low and high modes. During configuration these IOBs might need isolation from external logic. IOB intensive applications would favor the slave mode to avoid the additional time and space penalty of designing in additional logic buffers to perform this isolation.

Of the three pins used, two are required for handshake and one for data transmission. The data is set up at the DIN input to the LCA and clocked into the device by applying a rising clock edge to the CCLK input. Each configuration bit is synchronously loaded in this manner at a rate determined by the PC interface. The remaining handshake pin is the Done/~Program pin, when LOW the LCA is in configuration mode and when HIGH the configuration process has been completed and the LCA device is functionally operational.

The slave mode of configuration can also be used effectively for chaining multiple LCA designs. The DOUT pin from a preceding LCA device can be fed directly to the DIN pin of a succeeding LCA so two or more devices can be configured. Waveforms in Figure 3 show the timing requirements for DIN, DOUT and CCLK.

M2018 CONFIGURATION FILE

1111	FOUR DUMMY BITS MINIMUM
0010	PREAMBLE CODE
<24 BIT LENGTH COUNT>	TOTAL NUMBER OF BITS
1111	FOUR DUMMY BITS MINIMUM
0<DATA FRAME NUMBER 001>1111	196 CONFIGURATION FRAMES
0<DATA FRAME NUMBER 002>1111	EACH FIELD CONSISTS OF A
0<DATA FRAME NUMBER 003>1111	START ZERO FOLLOWED BY A
	71 DATA FIELD ENDING IN TWO
	OR MORE DUMMY BITS.
0<DATA FRAME NUMBER 195>1111	
0<DATA FRAME NUMBER 196>1111	
1111	MIMIMUM OF FOUR POST-
	AMBLE CODE BITS.

Figure 1. Data Stream Format

Mode Pin			Mode Selected
M0	M1	M2	
0	0	0	Master serial
0	0	1	Master low
0	1	1	Master high
1	0	1	Peripheral
1	1	1	Slave

Table 1. Five Configuration Modes Truth Table

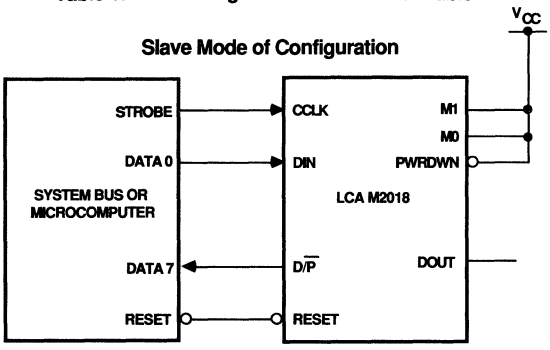


Figure 2. Configuration Diagram of Slave Mode

In the slave mode configuration, the data transfer occurs over a single data line, data 0 in this case. When data is valid the clock rising edge synchronizes the loading of data into the LCA device. The done/program pin may be monitored for end of configuration. This circuit may be used for microcomputers, microprocessors and interface adapters, but in this case the PC system bus is used.

PC CONFIGURATION HARDWARE

The LCA Applications Handbook (10098A) describes a number of hardware arrangements whereby configuration data may be downloaded serially to the LCA device. It is also supported with a download cable which is included with the XACT development system. The cable permits communication from a PC to the LCA through a parallel port. However, this method of downloading configuration data is not practical for an LCA device mounted on a card that is plugged into the IBM PC bus. This system uses the bus as a medium for configuration data transmission. Although the option for using the download cable has not been precluded from the interface described here. In the early stages of development it might be easier to use the download cable, until bus configuration is established. Figure 2 shows a block diagram of serial configuration mode while Figures 3 and 10 show the waveforms associated with this mode. In slave mode configuration the time to configure the LCA is determined by the loading source, in this case the controlling/loading program running in the PC.

Slave Mode Configuration Timing Considerations

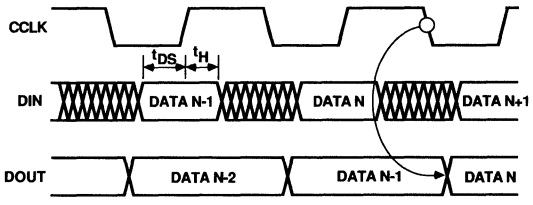


Figure 3. Slave Mode Timing

In the slave mode the CCLK pin is enabled as an input. Data set up at the DIN line is clocked into the LCA on the rising edge of the clock pulse. For the purposes of chaining LCA devices, data can be passed to the DOUT line but delayed by one clock cycle. Note that this data is synchronized to the falling edge of the clock. The configuration process is not complete until three additional clocks are appended to the pulse train. That is three more than the total number of configuration bits.

For all speed grades of the device the data setup and hold times are identical. A minimum figure of zero nanoseconds for setup and 40 nanoseconds for hold. The worst case figure for valid data at the DOUT pin is 65 nanoseconds after the falling edge of the clock.

Figure 7 is a schematic of the actual hardware used to drive the LCA device during configuration. The schematic shows PC bus signals involved in the process and the hardware used to decode them. A 74ALS520 decodes the PC address (set by SW1-SW7) for an I/O read/write cycle and a PAL16L8 decodes commands sent by download software (described later). Alternatively, a 74LS688 or equivalent could be used in place of the 74ALS520 but external pull-up resistors would be required on the Q inputs. Wired as shown, eight consecutive addresses will match as far as the 74ALS520 is concerned (e.g., 320H to 327H). The first of these eight addresses is termed the Base Port. Figure 8 shows the port addressing assignment. The port interface was required to work in a specific application that required a large decode range so the interface had a dual function. The port interface configuration was a biproduct of a real design.

Although the PAL16L8 (for which complete PAL design specification equations may be found in Appendix 7) is not a registered device, its outputs may be fed back such that latching can be achieved. This technique is used to store control information sent by configuration software. The C_0 and C_1 outputs from the PAL16L8 are latched condition codes where C_1 and C_0 store the data on D_7 and D_6, respectively, during an I/O write cycle to Base Port + 2. DPEN enables the D/~P PAL output and is latched from D5 on the same I/O write cycle.

A summary of the four condition codes is given in Table 2. For condition code 0, <C1, C0> = <0, 0>, the PAL device is in an idle mode. This is the mode preceding and following configuration. For condition code 1 the PAL asserts the the D/~P output as the trigger placing the LCA device in configuration mode. Condition code 2 directs the PAL to toggle -RESET for each subsequent write to Base Port + 3. This permits clearing of the LCA register contents. Finally, condition code 3 directs the PAL to toggle CCLK LOW, then HIGH, for each subsequent write to Base Port + 3. It is in this mode that configuration data bits are clocked into the LCA device, one bit at a time.

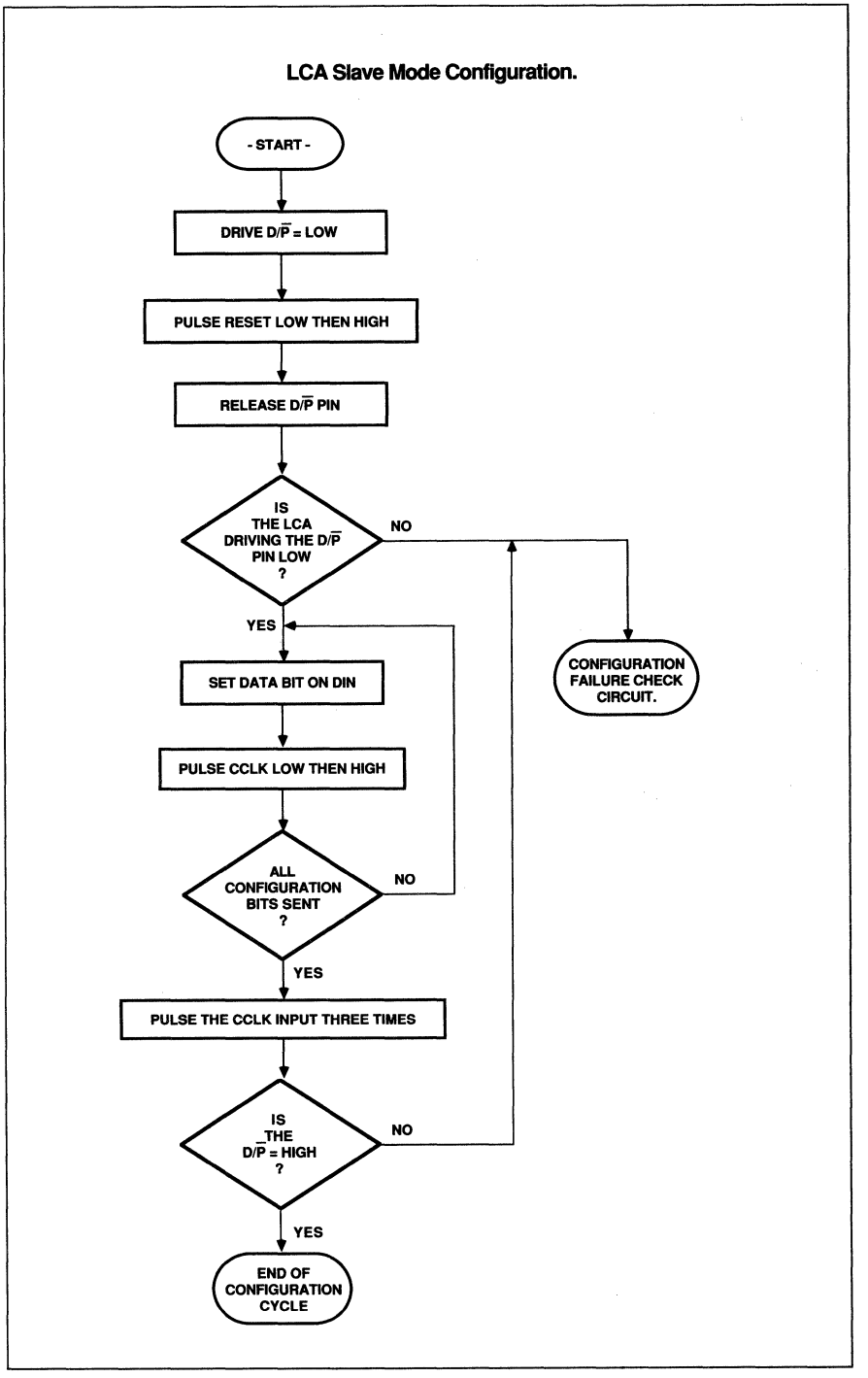


Figure 4. Flow Diagram of Configuration

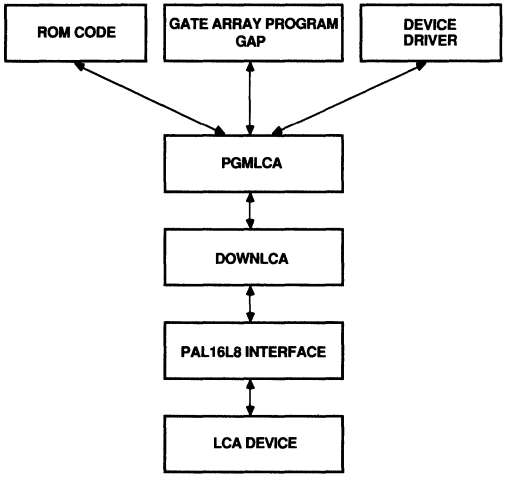


Figure 5. Flow Diagram of Software Packages, GAP

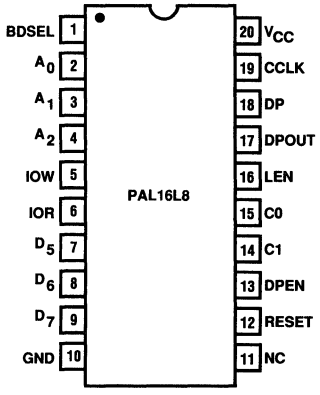


Figure 6. PAL16L8 Pinout Diagram

Condition Codes

C0	C1	Code	Function
0	0	0	Releases $\overline{D}$/P for for normal operation
1	0	1	Assert $\overline{D}$/P Low
0	1	2	Toggle RESET mode
1	1	3	Toggle CCLK mode

Table 2. PAL Condition Codes

After asserting D/~P (LOW) and toggling -RESET (condition codes 1 and 2) the software should read from Base Port + 2. During this read the PAL will drive the state of the D/~P onto bit 0 of the PC data bus. This is done through the DPOUT pin which is enabled only during reads from the I/O address Base Port + 3. An additional output, active LOW-LEN is also asserted at this time, allowing another on-board latch to drive data, such as switch settings, onto the bus. If the state of the D/~P pin is not 0 (LOW) then the LCA has failed to enter the configuration mode. Likewise, after sending the entire configuration bit stream and an additional three dummy bits (as required by the LCA) the software reads from Base Port + 2 to check that D/~P has been allowed to return HIGH, this indicates that the LCA has terminated the configuration process.

Timing considerations for this procedure are shown in Figures 9 and 10. Figure 9 shows a single I/O write cycle. The valid I/O address is put onto the bus, and decoded by the address decoder circuitry shown in Figure 7. The trailing (positive) edge of -IOW is used to latch condition bits. Also, in condition 2, CCLK follows -IOW during writes to Base Port + 3. Thus, the rising edge of CCLK corresponds to the rising edge of -IOW. This ensures that data will be latched by the LCA only when it is valid (write data is not guaranteed to be valid at the leading edge of -IOW).

The C_0 and C_1 outputs from the PAL16L8 are latching outputs. Both C_0 and C_1 are addressed into the address baseport +2. The data lines D_7 and D_6 are routed to C_1 and C_0 respectively during an I/O write operation. The one of four condition codes shown in Table 2 reflects the current status of the PAL during its interface activity with the LCA device.

As shown, SD0 on the PC bus is connected directly to DIN on the LCA through an in-line SPST switch. This switch should normally be closed, and need not be present at all in a production layout. It was added to allow the download cable, included with XACT, to function properly if required. If the switch is opened and the PAL removed, the download cable may be connected directly to the D/~P, -RESET, CCLK, and DIN pins and will interface directly through the cable.

BIT STREAM FORMAT AND CONVERSION

The format of the configuration bit stream used to program the LCA M2018 is shown in Figure 1. The bit stream starts with four dummy bits (1111), followed by a four-bit preamble code (0010), a 24-bit length count and another four dummy bits. The length count indicates the total number of bits in the bit stream, including the header just described. This number will be loaded into an internal counter in the LCA device allowing it to synchronize the loading of data during the configuration process. The actual configuration process is always serial, even during parallel configuration modes, internal serialization takes place in the LCA device.

The usual design entry technique uses the XACT design editor to create the logic design. The XACT conversion software then generates a bit pattern of the design that can be loaded directly into the LCA device. XACT can produce the bit stream in several different binary and ASCII formats. In this application the need to produce the bit stream in a format amenable to inclusion in an assembly language program dictated that the RAWBITS format be used. The RAWBITS output, placed in a file with the extension ".RBT," is ASCII. (Appendix 1 shows a portion of an ".RBT" file.) In general, one ASCII '0' or '1' is used to represent each bit of the configuration data. This data is not packed so it is not an efficient way of generating data for storage but it is suitable for loading into the LCA in real time.

Assembly language modules tend to define data in terms of bytes or words. For this reason, a Pascal program, BITCON.PAS, was written (see Appendix 2 for a complete listing). BITCON filters the ".RBT" file and produces an assembly language include file (given the extension ".BLB") which defines the configuration bit stream in terms of define byte directives acceptable to the Microsoft MASM™ assembler. This has the effect of packing the bit stream. For the sake of discussion the packed bit stream will be referred to as a byte stream. Additionally, BITCON counts the bits in the stream and the bytes used to hold the stream and defines two words at the beginning of the ".BLB" file for these quantities. A portion of a typical ".BLB" file may be found in Appendix 3. The net result is a structure which, once assembled into a main program, may be manipulated easily by machine language modules. Further, the assembled byte stream requires only 1.5K to 2K depending on the device being configured. Thus, the byte stream and associated software may easily be placed in a 4Kx8 or 8Kx8 ROM.

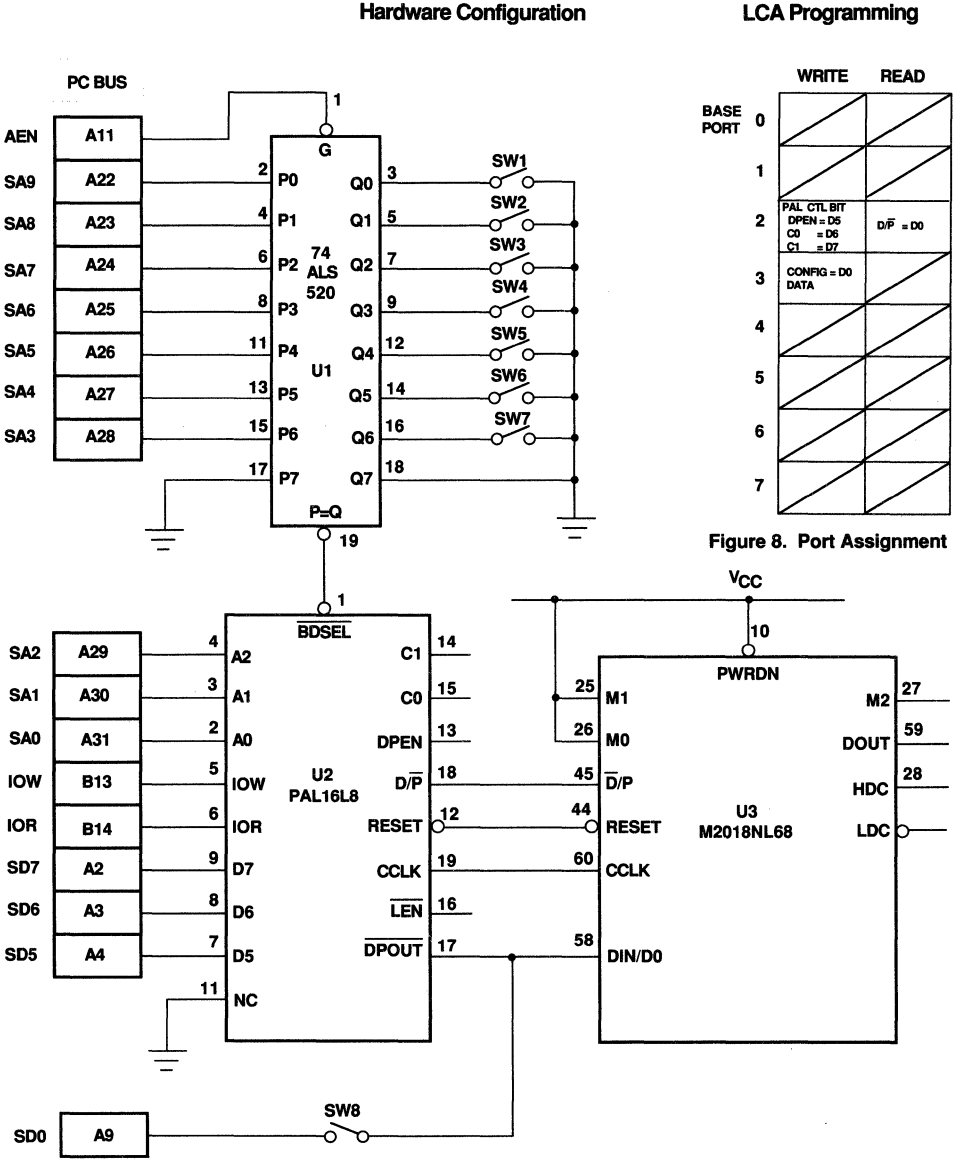
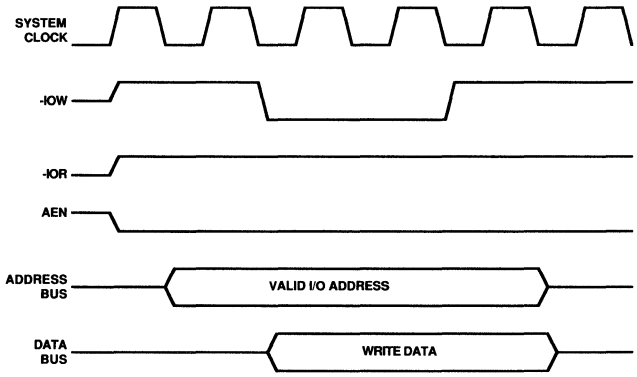


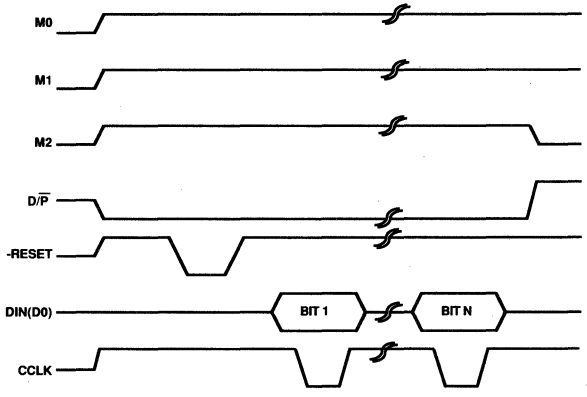
Figure 8. Port Assignment

Figure 7. Circuit Implementation of PC BUS to LCA



Note: During bus write cycles, write data becomes valid after -IOW has been asserted. Data can be registered on the rising edge of -IOW.

Figure 9. Single WRITE Cycle. Waveform



Notes:
1. The CCLK must not remain low longer than 5 microseconds or LCA timing will be violated.
2. The D/$\overline{P}$ pin is sampled after all the configuration bits + three additional bits have been sent. If the D/P pin is high then configuration was successful.

Figure 10. Configuration Cycle. Waveform

CONFIGURATION SOFTWARE

To perform the writing of configuration data to the LCA, the driver software was written in assembly language. High level languages such as 'C' or Pascal could be used, but assembly language was chosen because it is more efficient when considering the space constraints of a ROM. Three programs were written to interface to the LCA device;

GAP.ASM	Gate Array Program. See Appendix 4.
PGMLCA.ASM	Program LCA. See Appendix 5.
DOWNLCA.ASM	Download LCA. See Appendix 6.

Figure 4 shows a flow chart of a standard configuration cycle and Figure 5 shows the route taken from GAP to the programmed LCA device. The configuration driver may be partitioned into two halves: one generic and one hardware specific. The first half is implemented in a module called PGMLCA.ASM, which sequences through the configuration steps and decomposes the byte stream into the original configuration bit stream, while the second module DOWNLCA.ASM controls the bit manipulation required by the configuration hardware. Neither module carries the byte stream, it is assembled as part of a parent module whose job it is to invoke the actual configuration process. This is the function of a demonstration program GAP.ASM which displays some messages and calls the PGMLCA module, passing a pointer to the byte stream.

The byte stream is concatenated to the end of the GAP.ASM file as an include file. Any file that has been converted to byte stream ".BLB" can be downloaded to the LCA device by reassembling and linking GAP. The programs GAP.ASM, PGMLCA.ASM and DOWNLCA.ASM are written for Microsoft MASM version 5.0. A few utility routines in the library GAPLIB.LIB must also be linked to GAP but are not required by PGMLCA and DOWNLCA.

PGMLCA and DOWNLCA may just as easily be linked with a different parent and placed in ROM. This would be the production method for configuration used in the PC adapter card. The program in the ROM might perform a functional test at boot time to determine if the LCA is already programmed. If it is not, the PGMLCA module may be invoked by passing it a pointer to the byte stream. The entire process need only occur during the initial power on or cold boot, and takes less than a second to execute.

Looking at the modules in more detail, DOWNLCA interfaces to the PAL and provides three functions to PGMLCA. These include placing the LCA in the program mode, sending the LCA configuration bits and placing the LCA in the normal operation mode. As shown in the module listing, only a few port I/O instructions are used. The DOWNLCA program also checks to make sure that the LCA device enters and leaves the program mode at the correct times and reports this status to PGMLCA.

PGMLCA, sequences configuration by requesting program mode, sending configuration bits, and leaving the program mode. As discussed earlier, PGMLCA has the additional task of unpacking the byte stream that was originally packed by BITCON.

This partitioning allows the replacement of the DOWNLCA module with a different module suitable for supporting a different download hardware interface, without changing PGMLCA or BITCON. For example, the software for a download module to support the download cable of the development system could be generated. If the download cable is used, SW8 in Figure 7 must be switched to open.

CONCLUSION

Programming the LCA directly from the PC bus is a straightforward matter requiring a minimum of extra hardware. Once the hardware is breadboarded, the designer uses XACT to produce a RAWBITS file, which is converted by BITCON to an assembly language include file. This include file is then assembled as part of a parent program which is linked to the support modules PGMLCA and DOWNLCA. PGMLCA directs the configuration process through DOWNLCA which manages the specifics of the PAL-based interface.

The hardware interface may be easily adjusted to suit multiple LCA designs and the specific needs of other PC adapter cards or may be modified with a minimum of effort to support other microcomputer architectures. Likewise, BITCON is easily modified to produce include files for different assemblers. PGMLCA and DOWNLCA are also small modules which may be modified or ported as needed.

There are other advantages to this hardware and software scheme. Since the LCA configuration data is carried in software, which might be placed in ROM, an MS-DOS device driver, or even an application program, the realm of software updates is extended into the realm of hardware. At one end of the spectrum a vendor might supply hardware timing fixes or optimizations on diskette. At the other, the LCA could be designed as a generic logic block, whose function is dictated by the application software. In fact, at Trantor Systems a number of these capabilities are being exploited now in products incorporating such elements as RAM controllers and SCSI interfaces. Future LCA devices available from AMD, with even higher densities and faster speed grades will increase the scope of designs for which the part is appropriate.

SOFTWARE AVAILABILITY

The programs and PAL equations described in this application note, along with the binary library required for relinking GAP, are available on an MS-DOS 360K 5.25-inch diskette from:

Trantor Systems, Ltd.
33447 Western Avenue
Union City, CA 94587
(415) 489-3731

The charge per diskette is $25. Additional shipping charges may be necessary.

- APPENDIX 1 -

RAW BITS CONFIGURATION FILE.

```
XACT  LCA DESIGN.LCA 2018NL68
File design.rbt
22:15:17 DEC 10, 1987
22:15:17 DEC 10, 1987
Source
Version
Produced by XACT version 1.30
11110010000000000100010111010101101111
01111111111110100111111101111111101111111111111111101111011001111111001111110011111111111111111
01101101111111111011111110111111110011111111111111111011100111011101110111111111011111111111111
```

•
•
•

192 LINES OF RAW BIT CONFIGURATION DATA

•
•
•

```
00011111011110110111101101111100101111101011111111001111111001111010011110100111101011111111
01110111111110111011101111111111111111111111111111111111111111111111111111111111111111111011101111111
1111
```

Appendix 1. Raw Bit Listing

```
{
                        APPENDIX 2

  BitCon.pas

  BitCon is a bit conversion program which takes as its input the
  "*.rbt" file produced by the makebits command in the gate array
  development system and produces an assembly language file defining
  the bit stream.

  The "*.rbt" file contains a text prolog indicating the design name
  and the time/date of creation. The file then contains a number
  of lines containing ascii '1's and '0's which represent the bit
  stream. The first line of this stream contains the bit stream
  prolog, '1111', which BitCon uses to identify the bit stream.

  BitCon packs the bit stream into bytes where the highest order
  bit in a byte corresponds to the first encountered bit in the
  bit stream and the first byte corresponds to the first eight
  bits of the bit stream, and so forth.

  The output file has the form:

    ; text prolog created by makebits

      dw  length_of_bit_stream_in_bits
      dw  length_of_bit_stream_in_bytes

      db  byte_0,byte_1,...,byte_7
      db  byte_8,byte_9,...,byte_15
      ...
      ...
      db  ...,byte_n

  If the number of bits is not an even multiple of eight then then
  only the highest order bits of the last byte will be valid.

  The output file has the filetype ".blb" (for Bit LiBrary) and may
  be included in an assembly language program for assembly.

  history:

    12-09-87  RCB  first cut
    12-11-87  RCB  emit ';' instead of ':' in third line of
                   our portion of the prolog (1.0b)
    12-12-87  RCB  place the number of bits we counted into the
                   24-bit length field of the stream to ensure that
                   the two are the same (1.0c)
    01-04-88  RCB  make release version for ap. note (1.1a)
    01-26-88  RCB  more of mod. on 01-04-88 (1.1b)
}


const
  title = 'BitCon: .rbt -> .blb Filter, Version 1.1b';
```

```
  copyright = 'Copyright (c) 1987, Trantor Systems, Ltd.';

  max_byte = 8191;          { max bytes in byte stream - 1 }
  stream_prolog = '1111';   { ascii form of bit stream prolog }
  max_bytes_per_line = 8;   { max bytes defined per line in output file }

  intype = '.rbt';          { input filetype }
  outtype = '.blb';         { output filetype }


type
  st = string [132];


var
  byte_stream: array [0..max_byte] of byte;
  bit_length: integer;
  byte_length: integer;

  rbt_file: text;
  blb_file: text;
  current_line: st;


{ convert the byte argument to a three-digit hex number (includes leadin
  zero). also add a trailing 'h'. }

function hex (value: byte): st;

function nibble (value: byte): st;
begin
  nibble := copy ('0123456789abcdef', value + 1, 1);
end;

begin
  hex := '0' + nibble (value div 16) + nibble (value mod 16) + 'h';
end;


{ if the filename doesn't have a filetype, append the specified one. if
  there is a filetype, replace it with the specified one. }

function fnm (filename: st; filetype: st): st;
begin
  if pos ('.', filename) = 0 then
    fnm := concat (filename, filetype)
  else
    fnm := concat (copy (filename, 1, pos ('.', filename) - 1), filetype
end;


{ copy lines from the prolog of the .rbt file to the prolog of the .blb
  file until the bit stream prolog is found. leave the first line of the
  bit stream in current_line. }
```

```
procedure read_prolog;
begin
  readln (rbt_file, current_line);
  while (not eof (rbt_file)) and
        (copy (current_line, 1, length (stream_prolog)) <> stream_prolog
    writeln (blb_file, ';', ^I, current_line);
    readln (rbt_file, current_line);
  end;
  writeln (blb_file);
  writeln (blb_file, ';', ^I, 'Translated by:');
  writeln (blb_file, ';', ^I, title);
  writeln (blb_file, ';', ^I, copyright);
  writeln (blb_file);
end;


{ read the bit stream and construct a byte stream in memory. keep a count
  of the number of bits and the number of bytes. we assume that the first
  line of the bit stream is already in the variable current_line.}

procedure read_bit_stream;
var
  cur_bit: byte;
  their_length: integer;

procedure pack_bits;
var
  char_index: integer;

begin
  for char_index := 1 to length (current_line) do begin
    bit_length := bit_length + 1;
    if cur_bit = 7 then
      byte_stream [byte_length] := 0;
    byte_stream [byte_length] := byte_stream [byte_length] or
      ((ord (current_line [char_index]) - ord ('0')) shl cur_bit);
    if cur_bit > 0 then
      cur_bit := cur_bit - 1
    else begin
      cur_bit := 7;
      byte_length := byte_length + 1;
    end;
  end;
end;

begin
  bit_length := 0;
  byte_length := 0;
  cur_bit := 7;
  pack_bits;
  while not eof (rbt_file) do begin
    readln (rbt_file, current_line);
    pack_bits;
  end;
  if cur_bit <> 7 then
```

```
    byte_length := byte_length + 1;
  their_length := byte_stream [3] + (byte_stream [2] shl 8);
  if their_length <> bit_length then begin
    writeln ('Fixing up bit stream length...');
    writeln ('Old length = ',their_length,' bits');
    byte_stream [3] := bit_length and $ff;
    byte_stream [2] := (bit_length shr 8) and $ff;
  end;
end;


{ write_byte_stream formats the byte_stream just created by read_bit_str
  a bit length count and byte length count preceed the byte stream. }

procedure write_byte_stream;
var
  count: integer;

begin
  writeln (blb_file, ^I, 'dw', ^I, bit_length, ^I, ';# of bits in stream
  writeln (blb_file, ^I, 'dw', ^I, byte_length, ^I, ';# of bytes to hold
  for count := 0 to byte_length - 1 do begin
    if count mod max_bytes_per_line = 0 then begin
      writeln (blb_file);
      write (blb_file, ^I, 'db', ^I);
    end;
    if count mod max_bytes_per_line <> 0 then
      write (blb_file, ', ');
    write (blb_file, hex (byte_stream [count]));
  end;
  writeln (blb_file);
  writeln (blb_file);
end;


begin { main }
  writeln (title);
  writeln (copyright);
  if paramstr (1) = '' then begin
    writeln;
    writeln ('Usage: bitcon bitfile');
    writeln;
    writeln ('Where "bitfile" is a ".rbt" file produced by');
    writeln ('makebits. BitCon will produce an output file,');
    writeln ('"bitfile.blb," which contains an assembly');
    writeln ('language representation of the bit stream.');
    writeln;
  end
  else begin
    writeln;
    assign (rbt_file, fnm (paramstr (1), intype));
    {$i-}
    reset (rbt_file);
    if ioresult <> 0 then begin
      writeln ('Unable to open file "',fnm (paramstr (1), intype), '".')
```

```
      halt;
    end;
    writeln ('Reading "', fnm (paramstr (1), intype), '".');
    assign (blb_file, fnm (paramstr (1), outtype));
    rewrite (blb_file);
    if ioresult <> 0 then begin
      writeln ('Unable to create file "',fnm (paramstr (1), outtype), '"
      halt;
    end;
    {$i+}
    writeln ('Copying prolog...');
    read_prolog;
    if not eof (rbt_file) then begin
      writeln ('Reading bit stream...');
      read_bit_stream;
      writeln ('Bit stream = ',bit_length, ' bits');
      writeln ('           = ',byte_length, ' bytes');
      writeln ('Writing bit stream...');
      write_byte_stream;
    end
    else
      writeln ('Input file ended prematurely.');
    close (blb_file);
    close (rbt_file);
    writeln ('File "',fnm (paramstr (1), outtype),'" produced.');
  end;
end.
```

- APPENDIX 3 -

LISTING OF FILE AFTER CONVERSION BY BITCON

```
;   XACT  LCA  DESIGN.LCA 2018NL68
;   File design.rbt
;   22:15:17 DEC 10,  1987
;   22:15:17 DEC 10,  1987
;   Source
;   Version
;   Produced by XACT version 1.30

;   Translated by:
;   bitcon: .rbt -> .blb filter, version 1.0b
;   Copyright (c) 1987, Trantor Systems, Ltd.

    dw      17876   ;# of bits in stream
    dw      2235    ;# of bytes to hold stream

    db      0f2h, 000h, 045h, 0d4h, 0f7h, 0ffh, 04fh 0dfh
    db      0efh, 0ffh,  0efh,  067h, 0f3h, 0f3h,0ffh, 0feh

                 •                    •
                 •                    •
                 •                    •

          193 LINES OF COMPRESSED CODE

                 •                    •
                 •                    •
                 •                    •

    db      0bbh,  0bfh,  0ffh,  0ffh,  0ffh,  0ffh,  0ffh,  0ffh
    db      0eeh,  0ffh,  0f0h
```

Appendix 3. Listing of <file>.blb

```
        page    62,132
        title   GAP, Gate Array Programmer
;-------------------------------------------------------------------------
;
;       APPENDIX 4
;
;       GAP.asm
;
;       GAP takes the byte stream generated by BitCon (which, in turn,
;       took its input from MAKEBITS in the LCA development system) and
;       downloads it to the gate array. GAP itself only displays signon
;       messages and statistics. It then passes a pointer to a "byte
;       stream" (a packed configuration bit stream) which the PGMLCA
;       module decodes.
;
;       GAP is only intended for debugging and demonstration. The gate
;       array programming modules, on the other hand, are intended for
;       production use. Invoke GAP with:
;
;               A>gap
;
;       Invoking GAP without the hardware described in the ap. note
;       shouldn't have any ill effect. However, be careful that no
;       other card occupies port address 328h-32fh for which this demo.
;       has been "hard-wired."
;
;       MASM 5.0 may be used to assemble this module. It must then be
;       linked to the other modules.
;
;       History:
;
;               12-11-87  RCB   first (1.0a)
;               01-26-88  RCB   make mods. for ap. note release (1.1a)
;
;       Copyright (C) 1988, Trantor Systems, Ltd.
;       All rights reserved.
;
;-------------------------------------------------------------------------

        page
;-------------------------------------------------------------------------
;       Externals and Defs.
;-------------------------------------------------------------------------

;       pgm_gate_array is an entry in the pgmlca module. this routine
;       sequences the programming phases.

        extrn   pgm_gate_array:near

;       dsp_str and dsp_num are utility routines to display an ASCIIZ
;       string and a decimal number, respectively.

        extrn   dsp_str:near
        extrn   dsp_num:near
```

```
prog_title              equ     'GAP: Gate Array Programmer, version 1.1
prog_copyright          equ     'Copyright (c) 1987, Trantor Systems, Lt

false                   equ     0
true                    equ     not false
cr                      equ     0dh
lf                      equ     0ah
eof                     equ     1ah
null                    equ     0
dos                     equ     21h
dos_program_terminate   equ     4ch
combase                 equ     100h

ok_ret_code             equ     0               ;program return codes
bad_ret_code            equ     1

        page
;-----------------------------------------------------------------------
;       COM file header and program entry
;-----------------------------------------------------------------------

code    segment para public
        assume  cs:code,ds:code,es:code,ss:code

                org     combase
start:          jmp     main
signon:         db      cr,prog_title,cr,lf
                db      prog_copyright,cr,lf,lf,null,eof

                dw      256 dup (?)
stack:


main    proc

        mov     ax,cs
        mov     ds,ax
        mov     es,ax
        cli
        mov     ss,ax
        lea     sp,stack
        sti

        lea     dx,signon               ;signon
        call    dsp_str
        lea     dx,pgm_msg              ;now programming...
        call    dsp_str

        lea     di,gate_array_code      ;di -> bit stream structure

;       the bit stream structure consists of 3 fields: a word count
;       of the bits in the bit stream, a word count of the bytes needed
;       to store the bit stream, and an array of bytes of the length
;       indicated by the previous (second) field.
```

```
        mov     ax,[di]                 ;show bit length
        call    dsp_num
        lea     dx,bit_msg
        call    dsp_str
        mov     ax,[di + 2]             ;show byte length
        call    dsp_num
        lea     dx,byte_msg
        call    dsp_str

        call    pgm_gate_array          ;es:di -> bit stream structure
                                        ;cy indicates error upon return
        lea     dx,ok_pgm_msg
        mov     al,ok_ret_code
        jnc     done
        lea     dx,bad_pgm_msg          ;show error
        mov     al,bad_ret_code
done:
        call    dsp_str

        mov     ah,dos_program_terminate        ;al is still the return
        int     dos

main    endp

pgm_msg:        db      'Programming gate array...',cr,lf,null
bit_msg:        db      ' bits in stream.',cr,lf,null
byte_msg:       db      ' bytes in stream.',cr,lf,null
ok_pgm_msg:     db      'Gate array programmed.',cr,lf,null
bad_pgm_msg:    db      'Programming failure.',cr,lf,null

        page
;-------------------------------------------------------------------------
;       "Byte stream"
;
;       The "*.BLB" file containing the "byte stream" (packed
;       configuration bit stream) which was produced from the "*.RBT"
;       file bit BitCon is included here.
;-------------------------------------------------------------------------

gate_array_code:        include dummy.blb

code    ends
        end     start

;       end of file
```

```
        page    62,132
        title   DOWNLCA, LCA download routines (using PAL)
;-------------------------------------------------------------------------
;
;       APPENDIX 5
;
;       DOWNLCA.asm
;
;       DOWNLCA contains the hardware specific code for downloading
;       a configuration bit stream to the LCA using the Trantor
;       configuration PAL interface.
;
;       The PAL interface gives us direct control over the D/-P,
;       -RESET, and CCLK pins on the LCA.
;
;       To place the LCA in program mode we set the D/-P low and
;       toggle reset. We then release the D/-P pin and check to see
;       that the LCA is still driving it low, signifying that it
;       entered program mode.
;
;       The CCLK pin is normally held high and is pulsed low as
;       each configuration bit is sent. Each configuration bit is
;       sent on SD0 (bit 0 of the PC's data bus). Thus, the rising
;       edge of CCLK occurs at the end of the write cycle. This is
;       necessary as the PC doesn't drive write data until after the
;       leading edge of the write signal.
;
;       After the configuration bit stream has been sent we check
;       to see if the LCA allowed D/-P to go high, signifying the
;       end of configuration.
;
;       This module may be placed in rom if so desired.
;
;       History:
;
;               12-11-87  RCB   first
;               01-26-88  RCB   make mods. for ap. note release
;               04-11-88  RCB   release dp pin after toggling
;                               reset
;
;       Copyright (C) 1988, Trantor Systems, Ltd.
;       All rights reserved.
;
;-------------------------------------------------------------------------

        page
;-------------------------------------------------------------------------
;       Publics and Defs.
;-------------------------------------------------------------------------

        public  program_lca             ;program/normal entry
        public  send_lca_bit            ;send bit entry

;       LCA programming codes passed between caller and program_lca
;       entry point.
```

```
lca_func_normal         equ     0
lca_func_program        equ     1

sr1_rsc_dta             equ     3       ;rsc data port (bi-directional)
sr1_dta_port            equ     3       ;board data port (out)

;       This PAL configuration interface was designed into an existing
;       port mapping scheme. For demo purposes, the base port to which
;       the card containing the LCA responds is defined as an equate.
;       The ports which control configuration are defined as offsets
;       from this base port.

base_port               equ     328h
config_ctl_port         equ     base_port + 2
config_dta_port         equ     base_port + 3

;       Control port bit assignments. The configuration PAL latches
;       c1, c0, and dpen during writes to the ctl port. The config. PAL
;       drives data onto SD0 during reads from this port.

ctlbit  record  c1:1,c0:1,dpen:1,clt_unused:5
statbit record  stat_unused:7,dp:1

;       Control bits c1 and c0 set PAL modes. The function implied
;       by a given mode doesn't take effect until a subsequent write
;       to the config_dta_port (this scheme eliminates spiking on
;       the control lines).

mode_idle       equ     0                       ;configuration PAL idle
mode_program    equ     mask c0 + mask dpen     ;set pgm mode
mode_reset      equ     mask c1 + mask dpen     ;toggle reset
mode_reset_only equ     mask c1                 ;   "     "   without dp
mode_cclk       equ     mask c1 + mask c0       ;send configuration bits

        page
;-------------------------------------------------------------------------
;       module entry
;-------------------------------------------------------------------------

code    segment byte public
        assume  cs:code

;-------------------------------------------------------------------------
;       program_lca
;
;       input:  al = program/normal function code
;
;       output: nc = function ok
;               cy = function failed
;
;       we support two functions: set program mode, set normal mode.
;       when we go to program mode we also pulse the lca's reset.
;       when going to normal mode we check to make sure that the lca
;       has allowed d/-p to go high, indicating programming finished.
```

```
;
;       saves all registers
;-------------------------------------------------------------------------

program_lca     proc

        push    ax
        push    dx

        mov     dx,config_ctl_port      ;dx -> sr1 board control port

        cmp     al,lca_func_normal      ;go to normal operation?
        jz      go_normal

;       Program mode
;
;       1. set program mode
;       2. toggle reset
;       3. check status of d/-p
;       4. go to cclk mode

        mov     al,mode_program         ;set program mode and assert d/-
        out     dx,al
        inc     dx                      ;dx -> config data port
        out     dx,al                   ;any write latches program mode
        dec     dx
        mov     al,mode_reset           ;assert d/-p and enable -reset
        out     dx,al
        inc     dx
        out     dx,al                   ;any write here toggles -reset
        dec     dx

        mov     al,mode_reset_only      ;stay in reset mode but stop
                                        ;driving d/-p
        out     dx,al

        in      al,dx                   ;get status of d/-p pin
        test    al,mask dp              ;bit should be low if in pgm mod
        stc
        jnz     program_done

        mov     al,mode_cclk            ;enable cclk for configuration b
        out     dx,al
        clc
        jmp     short program_done

;       Normal mode
;
;       1. set PAL to idle mode
;       2. check d/-p to make sure lca took it high
;       3. also output a zero to base port to disable all funcs.

go_normal:
        mov     al,mode_idle            ;shut-down programming PAL
        out     dx,al
```

```
        inc     dx                              ;dx -> board dta port
        out     dx,al                           ;any write shuts down PA

;       We write a 0 to the base_port to disable all board functions.
;       This is necessitated by the particular board and not by the
;       demands of the configuration architecture.

        mov     dx,base_port
        xor     al,al
        out     dx,al

;       check the d/-p pin

        mov     dx,config_ctl_port
        in      al,dx                   ;get board status
        test    al,mask dp              ;check d/-p, should be high
        jnz     program_done
        stc

program_done:
        pop     dx
        pop     ax
        ret

program_lca     endp

;-------------------------------------------------------------------------
;       send_lca_bit
;
;       input:  al, bit 0 = bit of configuration data
;
;       output: nc = bit sent ok
;               cy = error
;
;       send a bit of configuration data to the lca.
;       this routine must not be called unless the lca has been placed
;       in program mode by the program_lca routine.
;
;       saves all regs.
;-------------------------------------------------------------------------

send_lca_bit    proc

        push    dx

        mov     dx,config_dta_port
        out     dx,al                   ;send the bit, the pal
                                        ;will toggle cclk

        clc                             ;for now we have no handshake
                                        ;on each bit so we can't determi
                                        ;whether or not an error occurre
        pop     dx
        ret

send_lca_bit    endp

code    ends
        end

;       end of file
```

```
        APPENDIX 6

        page    62,132
        title   PGMLCA: Gate array programming module
;-------------------------------------------------------------------------
;
;       PGMLCA.asm
;
;       PGMLCA sequences the phases of LCA programming. In general
;       these phases include:
;
;               o       place LCA in program mode, toggle reset
;               o       check that the LCA entered program mode
;               o       send the configuration bit stream, one
;                       bit at a time
;               o       place the LCA in normal mode
;               o       check that the LCA is in normal mode
;
;       PGMLCA may be placed in ROM, a DOS device driver, or other
;       program. The caller passes a far pointer to a "byte stream",
;       a packed configuration bit stream. The byte stream has the
;       structure:
;
;               word            # of bits in stream (N)
;               word            # of bytes used to store stream (M)
;               byte [0..M-1]   M bytes holding packed bit stream
;
;       PGMLCA uses another module, DOWNLCA, to perform hardware-
;       dependent functions such as sending bits and setting program/
;       normal mode.
;
;       History:
;
;               12-11-87  RCB   first
;               01-26-88  RCB   make mods. for ap. note release
;
;       Copyright (C) 1988, Trantor Systems, Ltd.
;       All rights reserved.
;
;-------------------------------------------------------------------------

        page
;-------------------------------------------------------------------------
;       Publics, Externals, and Defs.
;-------------------------------------------------------------------------

        public  pgm_gate_array          ;our entry

;       program_lca places the LCA in program/normal mode and
;       reports the success or failure of the function. send_lca_bit
;       sends individual bits to the LCA.

        extrn   program_lca:near
        extrn   send_lca_bit:near
```

```
;       lca programming codes passed between us and the program_lca
;       entry point.

lca_func_normal         equ     0
lca_func_program        equ     1

        page
;-------------------------------------------------------------------------
;       module entry (pgm_gate_array)
;
;       input:  es:di -> bit stream structure
;                        es:[di] = word, bit stream length in bits
;                        es:[di+2] = word, bytes used to hold stream
;                        es:[di+4] -> array of bytes holding stream
;
;       output: nc = programming ok
;               cy = unable to program lca
;
;       saves all regs
;-------------------------------------------------------------------------

code    segment byte public
        assume  cs:code

pgm_gate_array  proc

        push    ax
        push    bx
        push    cx
        push    di

        mov     al,lca_func_program     ;set lca to programming mode
        call    program_lca
        jc      pgm_error

        mov     cx,word ptr es:[di]     ;cx = number of bits to send
        add     di,4                    ;es:di -> byte array
pgm_byte:
        mov     bl,8                    ;8 bits/byte
        mov     al,byte ptr es:[di]     ;al = packed array of bits
                                        ;bit 7 is first to be sent
        inc     di
pgm_bit:
        jcxz    pgm_done
        dec     cx
        rol     al,1                    ;al, bit 0, is bit to send
        push    ax
        and     al,1                    ;mask bit
        call    send_lca_bit            ;and send
        pop     ax
        jc      pgm_error               ;couldn't send bit
        dec     bl
        jnz     pgm_bit
        jmp     pgm_byte
```

```
pgm_done:

;       the LCA requires that three additional bits be sent after
;       the configuration bit stream in order to complete configuration.

        call    send_lca_bit
        call    send_lca_bit
        call    send_lca_bit

        mov     al,lca_func_normal      ;set lca to normal operation
        call    program_lca
        jnc     pgm_finished

pgm_error:
        stc                             ;set error return code

pgm_finished:
        pop     di
        pop     cx
        pop     bx
        pop     ax
        ret

pgm_gate_array  endp

code    ends
        end

;       end of file
```

```
;               APPENDIX 7
;
title           LCA Programming PAL
pattern         config
revision        B
author          Robert Botchek
company         Trantor Systems, Ltd
date            January 26, 1988
chip            config pal16l8

bdsel a0 a1 a2 iow ior d5 d6 d7 gnd
nc /reset /dpen /c1 /c0 /len /dpout /dp /cclk vcc


equations

; c1 and c0 are latching outputs which are used as function selects.
; they are latched during a write to baseport+2. the meaning of the
; function selects are defined:
;
;       c1 c0   code    function
;       -- --   ----    --------
;       0  0    0       releases d/-p for normal lca operation
;       0  1    1       assert lca program (d/-p low)
;       1  0    2       toggle -reset mode
;       1  1    3       toggle cclk mode
;
; c1 and c0 will glitch during latching and so should not be used
; as direct inputs to a combinatorial output.

c1              =       /bdsel * /a2 * a1 * /a0 * /iow * d7
                +       bdsel * c1
                +       a2 * c1
                +       /a1 * c1
                +       a0 * c1
                +       iow * c1
c1.trst         =       vcc

c0              =       /bdsel * /a2 * a1 * /a0 * /iow * d6
                +       bdsel * c0
                +       a2 * c0
                +       /a1 * c0
                +       a0 * c0
                +       iow * c0
c0.trst         =       vcc

; dpen latches similarly to c1 and c0. when asserted it enables the
; dp pin.

dpen            =       /bdsel * /a2 * a1 * /a0 * /iow * d5
                +       bdsel * dpen
                +       a2 * dpen
                +       /a1 * dpen
                +       a0 * dpen
                +       iow * dpen
```

```
dpen.trst       =       vcc

; dp drives the d/-p input on the lca. when at vcc, the lca operates
; normally (assuming it has been programmed). when at gnd, the lca
; is in a program mode.
;
; d/-p is latched during functions 1 and 2 of (<c1,c0> = <0,1> or
; <c1,c0> = <1,0>) and any write to baseport+3 is given. an assertion
; of d/-p also enables the d/-p output.

dp              =       /bdsel * /a2 * a1 * a0 * /iow * /c1 * c0
                +       /bdsel * /a2 * a1 * a0 * /iow * c1 * /c0
                +       bdsel * dp
                +       a2 * dp
                +       /a1 * dp
                +       /a0 * dp
                +       iow * dp
dp.trst         =       dpen

; reset drives the -reset line on the lca. it is asserted only during
; a write to baseport+3 when function code 2 is selected.

reset           =       /bdsel * /a2 * a1 * a0 * /iow * c1 * /c0
reset.trst      =       vcc

; cclk is the configuration clock used by the lca to latch incoming
; configuration data. the lca latches data on the rising edge of cclk.
; we keep cclk high all the time and only lower it during a write to
; baseport+3 when function code 3 is selected. thus, at the end of
; the write cycle the data will be latched.
;
; note: data is written to baseport+3, not baseport+2. this simplifies
; the timing of the cclk enable and the data output.

cclk            =       /bdsel * /a2 * a1 * a0 * /iow * c1 * c0
cclk.trst       =       vcc

; len enables the board status latch onto the pc's data bus, it also
; enables the dpout output (below).
;
; board status includes the condition of the lca's done/-program
; (d/-p) pin, etc.

len             =       /bdsel * /a2 * a1 * /a0 * /ior
len.trst        =       vcc

; dpout drives data onto the pc's data bus during a read from base
; port+2 as enabled by len. dpout indicates program mode when high.

dpout           =       /dp
dpout.trst      =       len
```

Advanced
Micro
Devices

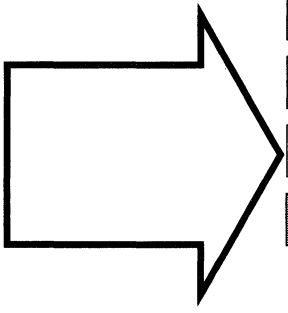


Table of Contents Section 4
Product Information

Logic Cell™ Array M2064/M2018

Features/Benefits

- CMOS programmable Logic Cell Array (LCA) for replacement of standard logic
- Completely reconfigurable by the user in the final system
- High performance equivalent to TTL SSI/MSI
  - 33 MHz flip-flop toggle rate (-33 speed grade)
  - 50 MHz flip-flop toggle rate (-50 speed grade)
  - 70 MHz flip-flop toggle rate (-70 speed grade)
- User configurable logic functions, interconnect and I/O for maximum flexibility
- 64/100 user-Configurable Logic Blocks (CLBs) providing usable gate equivalency of up to 1200/1800 gates
- 58/74 individually-configurable I/O pins allowing any mix of inputs, outputs or bidirectional signals (68/84-pin package)
- User -selectable TTL or HCMOS input threshold levels
- Multiple configuration modes for greatest flexibility and ease-of-use
- Verification feature allows user to check configuration data
- User-selectable security feature prevents read-back of configuration data
- Read-back of internal register states for system debug
- On-chip clock oscillator and clock buffer circuits provide flexible internal and external clocking functions
- Master reset of all internal register elements in addition to user-configurable Reset and/or Set control of individual CLB storage elements
- Complete development system support

General Description

The Logic Cell Array (LCA) is a high-density CMOS-integrated circuit available from Monolithic Memories. Its user-programmable array architecture is made up of three types of configurable elements: Input/Output Blocks, Logic Blocks and Interconnect. The designer can define individual I/O blocks for interface to external circuitry, define logic blocks to implement logic functions and define interconnection network to compose larger scale logic functions. The XACT™ Development system provides interactive graphic design capture and automatic routing. Both logic simulation and in-circuit emulation are available for design verification.

The Logic Cell Array is available in a variety of logic capacities, package styles, temperature ranges and speed grades.

PART NUMBER	LOGIC CAPACITY (USABLE GATES)	CONFIG-URABLE LOGIC BLOCKS	USER I/Os	CONFIG-URATION PROGRAM (BITS)
M2064	1200	64	58	12040
M2018	1800	100	74	17880

The Logic Cell Array's logic functions and interconnections are determined by data stored in internal static memory cells. On-chip logic provides for automatic loading of configuration data at power-up. The program data can reside in an EEPROM, EPROM or ROM on the circuit board or on a floppy disk or hard disk. The program can be loaded in a number of modes to accommodate various system requirements.

Package Availability

PART NUMBER	48-PIN PLASTIC DIP N48	68-PIN PLCC NL68	68-PIN PGA P68	84-PIN PLCC NL84	84-PIN PGA P84
M2064	X	X	X		
M2018		X	X	X	X

Ordering Information

M2018-70 C NL 84

- PART NUMBER
  - 2064 (1200 Gates, 58 IOB)
  - 2018 (1800 Gates, 74 IOB)
- SPEED GRADE
  - -33 (33 MHz Toggle Rate)
  - -50 (50 MHz Toggle Rate)
  - -70 (70 MHz Toggle Rate)
- NUMBER OF PINS
  - 48 (48 Pins)
  - 68 (68 Pins)
  - 84 (84 Pins)
- PACKAGE TYPE
  - NL = Pin Molded Chip Carrier
  - P = Pin Grid Array
  - N = Pin Molded DIP
- TEMPERATURE RANGE
  - C = Commercial
  - M = Military

10352A
JANUARY 1988

Pin Description

$\overline{\text{PWRDWN}}$
An active low power-down input stops all internal activity to minimize VCC power and puts all output buffers in a high-impedance state. Configuration is retained, however, internal storage elements are Reset. When the $\overline{\text{PWRDWN}}$ pin returns HIGH, the device returns to operation with the same sequence of reset, buffer enable and DONE, $\overline{\text{PROGRAM}}$ as at the completion of configuration.

M0, RTRIG
As Mode 0, this input and M1, M2 are sampled before the start of configuration to establish the configuration mode to be used.

As a read trigger, an input transition to a HIGH, after configuration is complete, will initiate a readback of configuration and storage element data. This operation may be limited to a single request, or be inhibited altogether, by selecting the appropriate readback option when generating the bit stream.

M1, $\overline{\text{RDATA}}$
As Mode 1, this input and M0, M2 are sampled before the start of configuration to establish the configuration mode to be used.

As an active-low read data; after configuration is complete, this pin is the output of the readback data.

M2
As Mode 2, this input and M0, M1 are sampled before the start of configuration to establish the configuration, mode to be used. After configuration, this pin becomes a user-programmable I/O.

HDC
High during configuration is held at a HIGH level by the LCA until after configuration. It is intended to be available as a control indication that configuration is not complete. After configuration, this pin is a user I/O.

$\overline{\text{LDC}}$
Low during configuration is held at a LOW level by the LCA until after configuration. It is intended to be available as a control indication that configuration is not completed. It is particularly useful in master mode as a LOW enable for an EPROM. After configuration, this pin is a user I/O. If used as a LOW EPROM enable, it should be programmed as a HIGH after configuration.

$\overline{\text{RESET}}$
This is an active-low input which has three functions. Prior to the start of configuration, a LOW input will delay the start of the configuration process. An internal circuit senses the application of power and begins a minimal time-out cycle on the order of 100 ms. When the time-out and $\overline{\text{RESET}}$ are complete, the levels of the "M" mode lines are sampled and configuration begins. If $\overline{\text{RESET}}$ is asserted during a configuration, the LCA is reinitialized and will restart the configuration at the termination of RESET. If $\overline{\text{RESET}}$ is asserted after configuration is complete, it will provide an asynchronous reset of all IOB and CLB storage elements of the LCA.

DONE, $\overline{\text{PROG}}$
The DONE open drain output is configurable with or without a pull-up resistor of about 3 KΩ. At the completion of configuration, the circuitry of the LCA becomes active in a synchronous order and one configuration clock cycle later DONE is asserted. Once configuration is done, a HIGH-to-LOW transition of this program pin will cause an initialization of the LCA and start a reconfiguration if that mode is selected in the current configuration.

XTL1
This user I/O pin may be configured to operate as the output of an amplifier usable with an external crystal and bias circuitry to form an oscillator.

XTL2
This user I/O pin may be configured to operate as the input of an amplifier usable with an external crystal and bias circuitry to form an oscillator.

CCLK
During configuration, configuration clock is an output of an LCA in either master or peripheral mode. LCAs in slave mode use it as a clock input. During a readback operation, it is an input clock for the configuration data being output.

DOUT
This user I/O pin is used during configuration to output serial configuration data out for daisy-chained slaves' data in.

DIN
This user I/O pin is used as serial data in during slave or peripheral configuration. This pin is D0 in master configuration mode.

$\overline{\text{CS0}}$, $\overline{\text{CS1}}$, CS2, $\overline{\text{WRT}}$
These four inputs represent a set of signals, three active low and one active high, which are used in the peripheral mode to control configuration data entry. The assertion of all four generates a LOW CCLK and shifts DOUT data. The removal of any assertion clocks in the DIN data present and causes a HIGH CCLK. In master mode, these pins become part of the parallel configuration byte (D4, D3, D2, D1). After configuration is complete, they are user-programmed I/O.

$\overline{\text{RCLK}}$
During master mode configuration, this pin represents a read clock of an external memory device. After configuration is complete, this pin becomes a user-programmed I/O.

D0-D7
This set of eight pins represents the parallel configuration data byte for the master mode. After configuration is complete, they are user-programmed I/O.

A0-A15
This set of sixteen pins presents an address output for an external configuration memory during master mode. After configuration is complete, they are user-programmed I/O. A12 through A15 are not available in packages with less than sixty-eight pins.

I/O
A pin which may be programmed by the user to be input and/or output following configuration. Some of these pins present a high-impedance pull-up or perform other functions before configuration is complete.

Functional Description

The general structure of a Logic Cell Array is shown in Figure 1. The elements of the array include three categories of user-programmable elements: I/O Blocks, Configurable Logic Blocks and Programmable Interconnections. The I/O Blocks provide an interface between the logic array and the device package pins. The Configurable Logic Blocks perform user-specified logic functions, and the interconnect resources are programmed to form networks that carry logic signals among blocks.

Configuration of the Logic Cell Array is established through a distributed array of memory cells. The XACT development system generates the program used to configure the Logic Cell Array. The Logic Cell Array includes logic to implement automatic configuration.

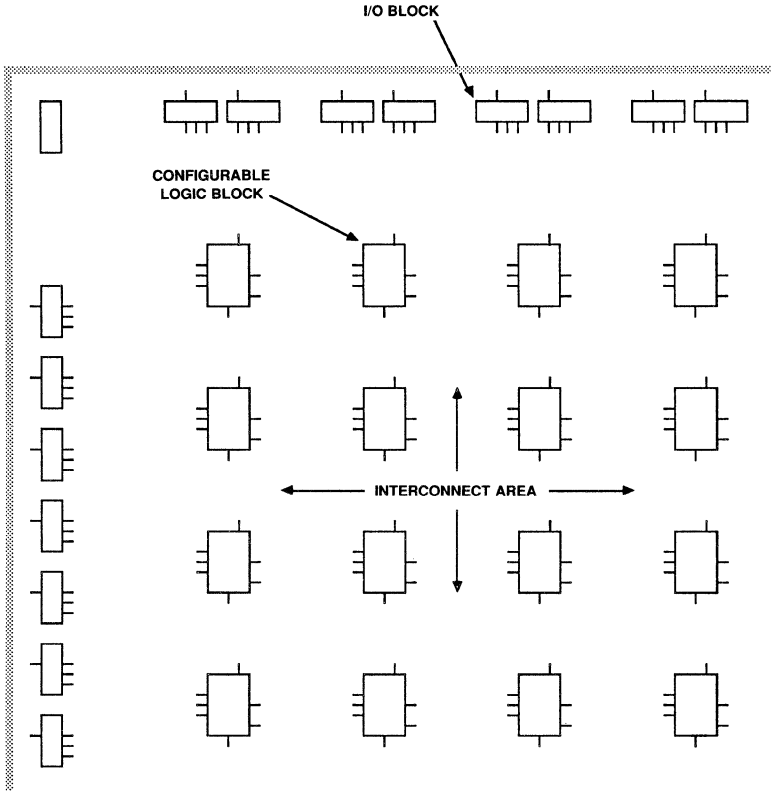


Figure 1. Logic Cell Array Structure

Configuration Memory

The configuration of the Monolithic Memories' Logic Cell Array is established by programming memory cells which determine the logic functions and interconnections. The memory loading process is independent of the user logic functions.

The static memory cell used for the configuration memory in the Logic Cell Array has been designed specifically for high reliability and noise immunity. Based on this design, integrity of the LCA configuration memory is assured even under adverse conditions. Compared with other programming alternatives, static memory provides the best combination of high density, high performance, high reliability and comprehensive testability. As shown in Figure 2, the basic memory cell consists of two CMOS inverters plus a pass transistor used for writing data to the cell. The cell is only written during configuration and only read during readback. During normal operation the pass transistor is "off" and does not affect the stability of the cell. This is quite different from the normal operation of conventional memory devices, in which the cells are continuously read and written.

The outputs Q and $\overline{Q}$ control pass-transistor gates directly. The absence of sense amplifiers and the output capacitive load provide additional stability to the cell. Due to the structure of the configuration memory cells, they are not affected by extreme power supply excursions or very high levels of alpha particle radiation. In reliability testing no soft errors have been observed, even in the presence of very high doses of alpha radiation.

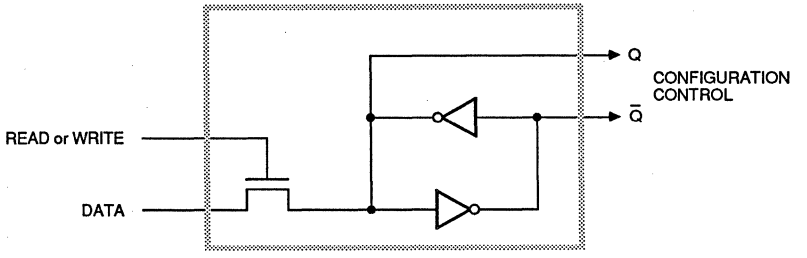


Figure 2. Configuration Memory Cell

Input/Output Block

Each user-configurable I/O block (IOB) provides an interface between the external package pin of the device and the internal logic. Each I/O block includes a programmable input path and a programmable output buffer. It also provides input clamping diodes to provide protection from electrostatic damage, and circuits to protect the LCA from latch-up due to input currents. Figure 3 shows the general structure of the I/O block.

The input buffer portion of each I/O block provides threshold detection to translate external signals applied to the package pin to internal logic levels. The input buffer threshold of the I/O blocks can be programmed to be compatible with either TTL (1.4 V) or CMOS (2.2 V) levels. The buffered input signal drives both the data input of an edge-triggered D-type flip-flop and one input of a two-input multiplexer. The output of the flip-flop provides the other input to the multiplexer. The user can select either the direct input path or the registered input, based on the content of the memory cell controlling the multiplexer. The I/O blocks along each edge of the die share common clocks. The flip-flops are reset during configuration as well as by the active-low chip $\overline{\text{RESET}}$ input.

Output buffers in the I/O blocks provide 4-mA drive for high fan-out CMOS or TTL-compatible signal levels. The output data (driving I/O block pin O) is the data source for the I/O block output buffer. Each I/O block output buffer is controlled by the contents of two configuration memory cells which turn the buffer ON or OFF or select logical three-state buffer control. The user may also select the output buffer three-state control (I/O block pin TS). When this I/O block output control signal is HIGH (a logic "1") the buffer is disabled and the package pin is high-impedance.

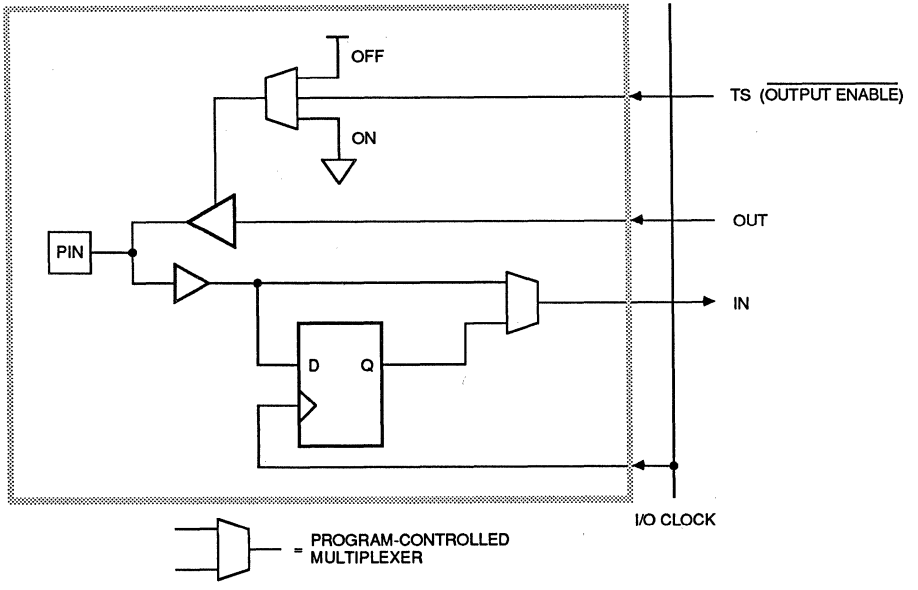


Figure 3. I/O Block

Configurable Logic Block

An array of Configurable Logic Blocks (CLBs) provides the functional elements from which the user's logic is constructed. The logic blocks are arranged in a matrix in the center of the device. The M2064 has sixty-four such blocks arranged in an eight-row by eight-column matrix. The M2018 has one hundred logic blocks arranged in a ten by ten matrix. Each logic block has a combinatorial logic section, a storage element, and an internal routing and control section. Each CLB has four general-purpose inputs; A, B, C and D; and a special clock input (K), which may be driven from the interconnect adjacent to the block. Each CLB also has two outputs, X and Y, which may drive interconnect networks. Figure 4 shows the resources of a Configurable Logic Block.

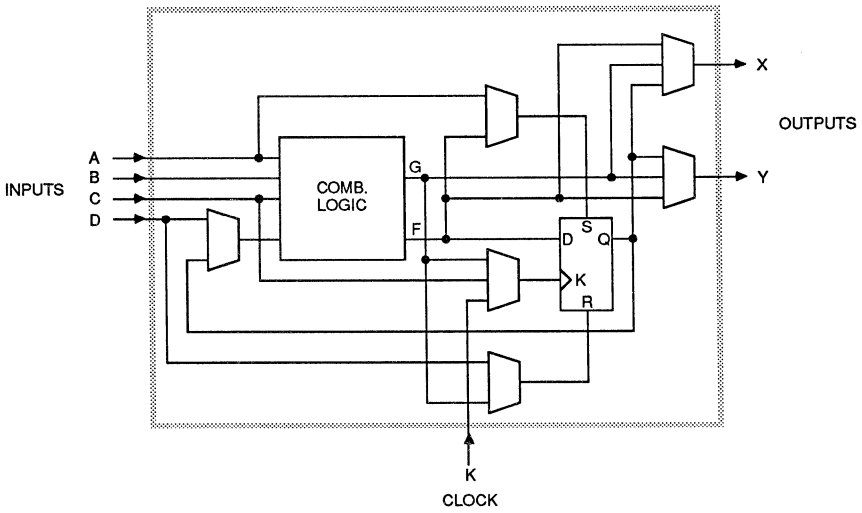


Figure 4. Configurable Logic Block

The logic block combinatorial logic uses a table look-up memory to implement Boolean functions. This technique can generate any logic function of up to four variables with a high-speed sixteen-bit memory. The propagation delay through the combinatorial network is independent of the function generated. Each block can perform any function of four variables or any two functions of three variables each. The variables may be selected from among the four inputs and the block's storage element output "Q". Figure 5 shows various options which may be specified for the combinatorial logic.

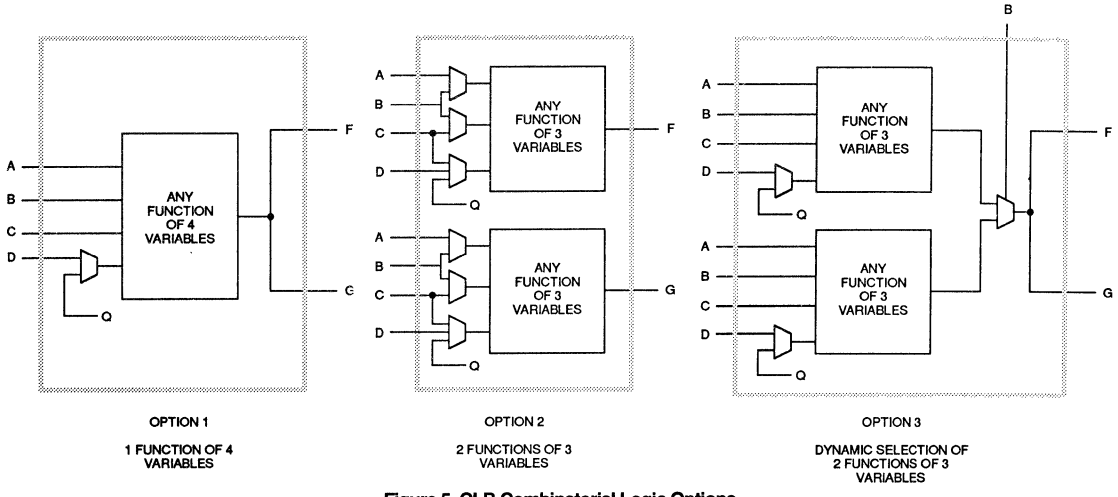


Figure 5. CLB Combinatorial Logic Options

If the single four-variable configuration is selected (Option 1), the F and G outputs are identical. If the two-function alternative is selected (Option 2), logic functions F and G may be independent functions of three variables each. The three variables can be selected from among the four logic block inputs and its storage element output Q. A third form of the combinatorial logic (Option 3) is a special case of the two-function form in which the B input dynamically selects between the two function tables providing a single merged logic function output. This dynamic selection allows some five-variable functions to be generated from the four block inputs and storage element Q. Combinatorial functions are restricted in that one may not use both its storage element output Q and the input variable of the logic block pin D in the same function.

If used, the storage element in each Configurable Logic Block (Figure 6) can be programmed to be either an edge-sensitive D-type flip-flop or a level-sensitive D latch. The clock or enable for each storage element can be selected from:

- The special-purpose clock input K
- The general-purpose input C
- The combinatorial function G

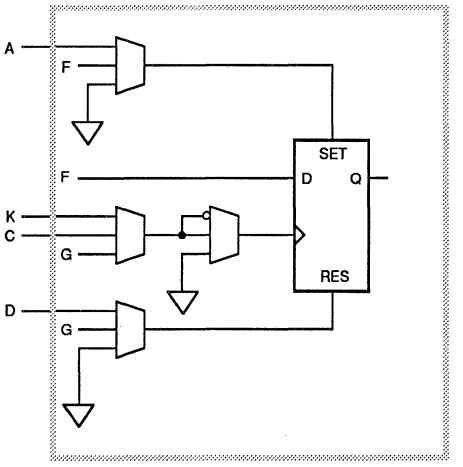


Figure 6. CLB Storage Element

The user may also select the clock active sense within each logic block. This programmable inversion eliminates the need to route both phases of a clock signal throughout the device.

The storage element data input is supplied from the function F output of the combinatorial logic. Asynchronous SET and RESET controls are provided for each storage element. The user may enable these controls independently and select their source. They are active-high inputs and the asynchronous reset is dominant. The storage elements are reset by the active-low chip $\overline{\text{RESET}}$ pin as well as by the initialization phase preceding configuration. If the storage element is not used, it is disabled.

The two block outputs, X and Y, can be driven by either the combinatorial functions, F or G, or the storage element output Q (Figure 4). Selection of the outputs is completely interchangeable and may be made to optimize routing efficiencies of the networks interconnecting the logic blocks and I/O blocks.

Programmable Interconnect

Programmable interconection resources in the Logic Cell Array provide routing paths to connect inputs and outputs of the I/O and logic blocks into desired networks. All interconnections are composed of metal segments, with programmable switching points provided to implement the necessary routing. Three types of resources accommodate different types of networks:

- General purpose interconnect
- Long lines
- Direct connection

General-Purpose Interconnect

General-purpose interconnect, as shown in Figure 7a, is composed of four horizontal metal segments between the rows and five vertical metal segments between the columns of logic and I/O blocks. Each segment is only the "height" or "width" of a logic block. Where these segments would cross at the intersections of rows and columns, switching matrices are provided to allow interconnections of metal segments from the adjoining rows and columns. Switches in the switch matrices and on block outputs are specially designed transistors, each controlled by a configuration bit.

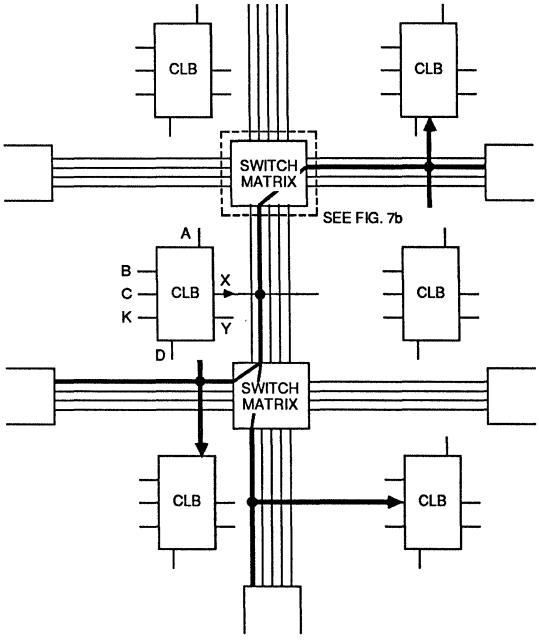
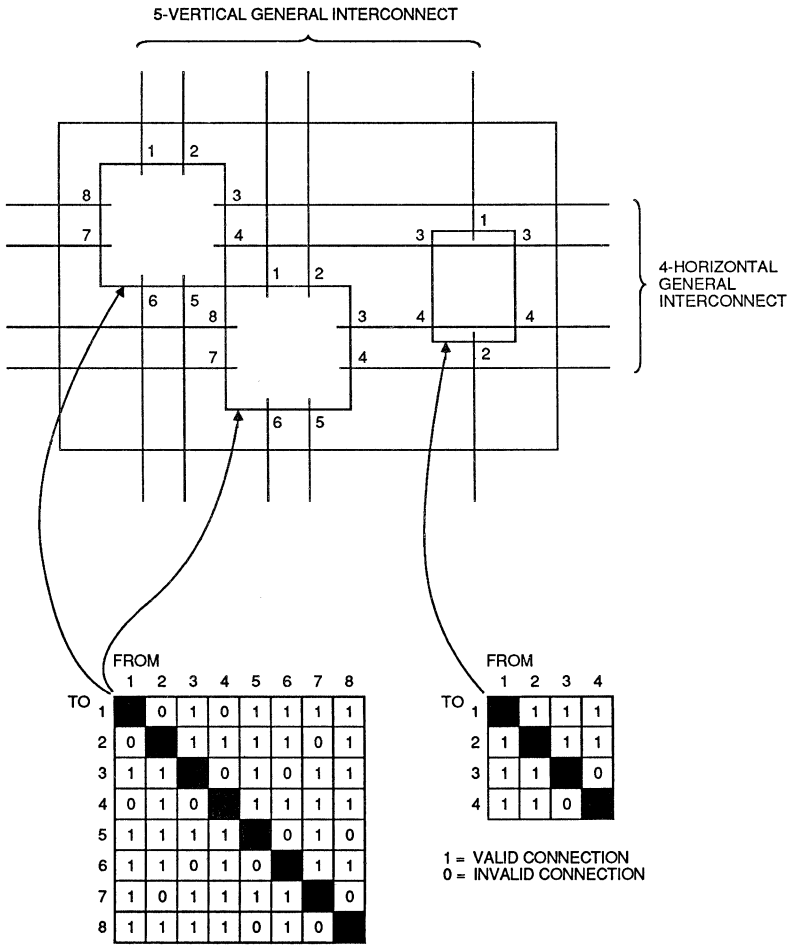


Figure 7a. General-Purpose Interconnect

Logic block output switches provide contacts to adjacent general interconnect segments and therefore to the switching matrix at each end of those segments. A switch matrix can connect an interconnect segment to other segments to form a network. Figure 7a shows the general interconnect used to route a signal from one logic block to three other logic blocks. As shown, combinations of closed switches in a switch matrix allow multiple branches for each network. The inputs of the logic or I/O blocks are multiplexers that can be programmed with configuration bits to select an input network from the adjacent interconnect segments. Since the switch connections to block inputs are undirectional (as are block outputs) they are usable *only* for input connections. The development system software provides automatic routing of these interconnections. Interactive routing is also available for design optimization. This is accomplished by selecting a network and then toggling the states of the interconnect points by selecting them with the "mouse". In this mode, the connections through the switch matrix may be established by selecting pairs of matrix pins. The switching matrix combinations are indicated in Figure 7b.

Special buffers within the interconnect area provide periodic signal isolation and restoration for higher general interconnect fan-out and better performance. The repowering buffers are bidirectional, since signals must be able to propagate in either direction on a general interconnect segment. Direction controls are automatically established by the Logic Cell Array development system software. Repowering buffers are provided only for the general-purpose interconnect since the direct and long-line resources do not exhibit the same R-C delay accumulation. The Logic Cell Array is divided into nine sections with buffers automatically provided for general interconnect at the boundaries of these sections. These boundaries can be viewed with the development system. For routing within a section, no buffers are used. The delay calculator of the XACT development system automatically calculates and displays the block, interconnect and buffer delays for any selected paths.



TO \ FROM	1	2	3	4	5	6	7	8
1	■	0	1	0	1	1	1	1
2	0	■	1	1	1	1	0	1
3	1	1	■	0	1	0	1	1
4	0	1	0	■	1	1	1	1
5	1	1	1	1	■	0	1	0
6	1	1	0	1	0	■	1	1
7	1	0	1	1	1	1	■	0
8	1	1	1	1	0	1	0	■

TO \ FROM	1	2	3	4
1	■	1	1	1
2	1	■	1	1
3	1	1	■	0
4	1	1	0	■

Figure 7b. Interconnection Switching Matrix

Long Lines

Long lines, shown in Figure 8a, run both vertically and horizontally the height or width of the interconnect area. Each vertical interconnection column has two long lines; each horizontal row has one, with an additional long line adjacent to each set of I/O blocks. The long lines bypass the switch matrices and are intended primarily for signals that must travel a long distance or must have minimum skew among multiple destinations.

A global buffer in the Logic Cell Array is available to drive a single signal to all B and K inputs of logic blocks. Using the global buffer for a clock provides a very low skew, high fan-out synchronized clock for use at any or all of the logic blocks. At each block, a configuration bit for the K input to the block can select this global line as the storage element clock signal. Alternatively, other clock sources can be used.

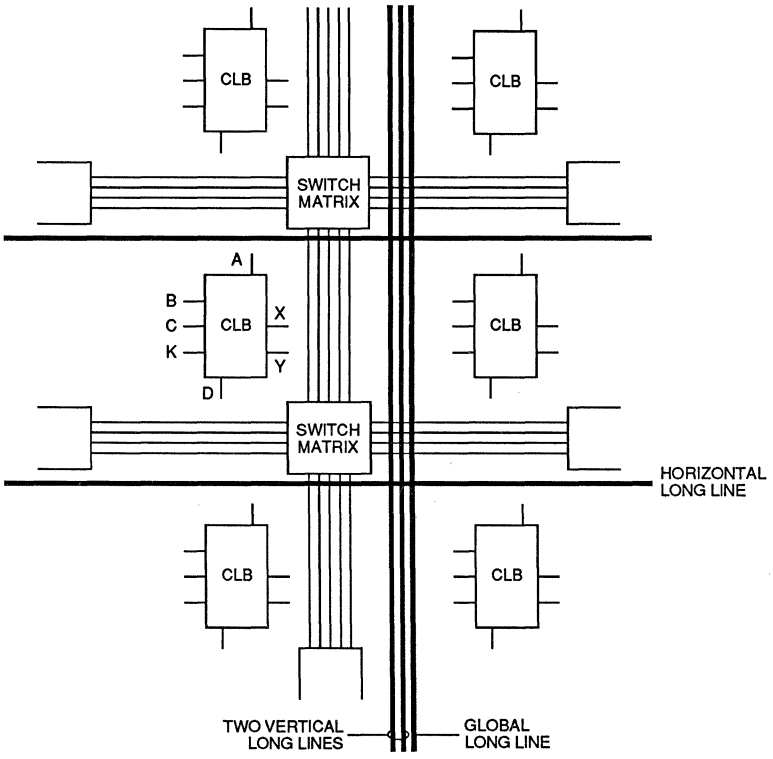


Figure 8a. Long Line Interconnect

A second buffer below the bottom row of the array drives a horizontal long line which, in turn, can drive a vertical long line in each interconnection column. This alternate buffer also has low skew and high fan-out capability. The network formed by this alternate buffer's long lines can be selected to drive the B, C or K inputs of the logic blocks. Alternatively, these long lines can be driven by a logic or I/O block on a column-by-column basis. This capability provides a common, low-skew clock or control line within each column of logic blocks. Interconnections of these long lines are shown in Figure 8b.

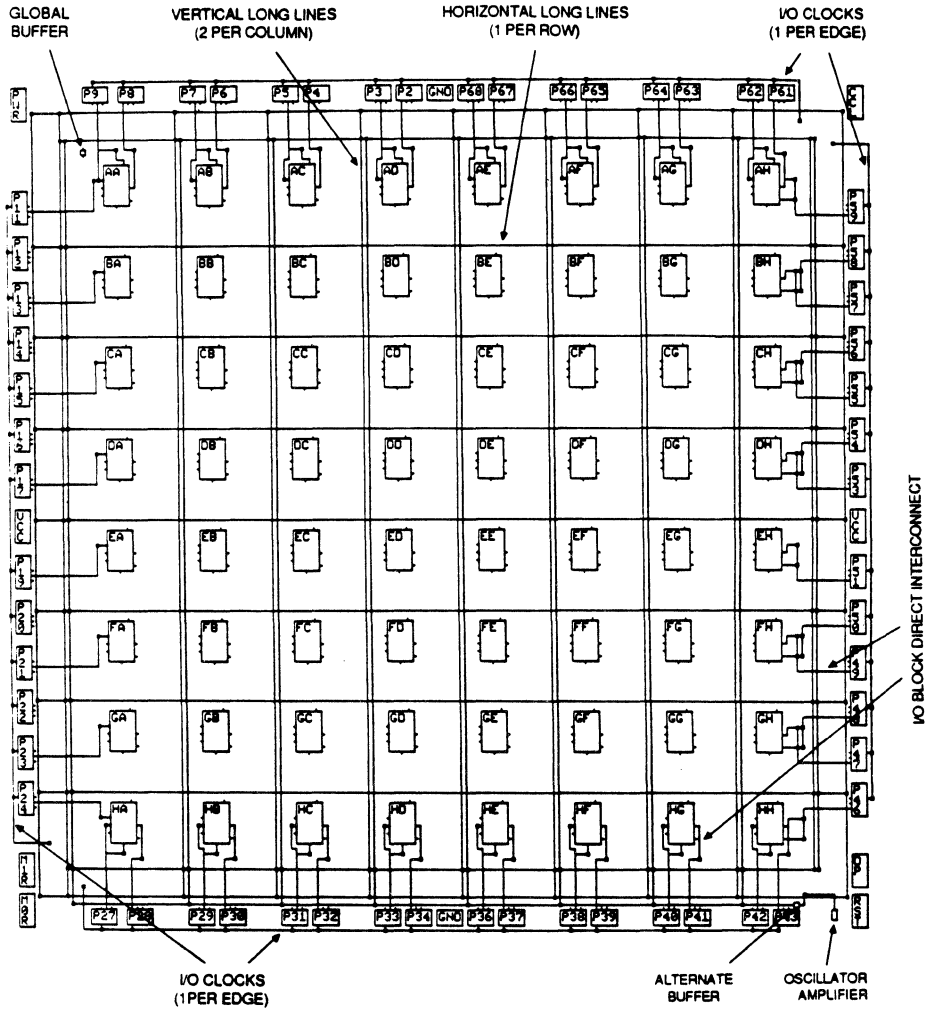


Figure 8b. M2064 Long Lines, I/O Clocks, I/O Direct Interconnect

Direct Interconnect

Direct interconnect, shown in Figure 9, provides the most efficient implementation of networks between adjacent logic or I/O blocks. Signals routed from block to block by means of direct interconnect exhibit minimum interconnect propagation and use minimum interconnect resources. For each CLB, the X output may be connected directly to the C or D inputs of the CLB above and to the A or B inputs of the CLB below it. The Y output can use direct interconnect to drive the B input of the block immediately to its right. Where logic blocks are adjacent to I/O blocks, direct connect is provided to the I/O block input (I) on the left edge of the die, the output (O) on the right edge, or both on I/O blocks at the top and bottom of the die. Direct interconnections of I/O blocks with CLBs are shown in Figure 8b.

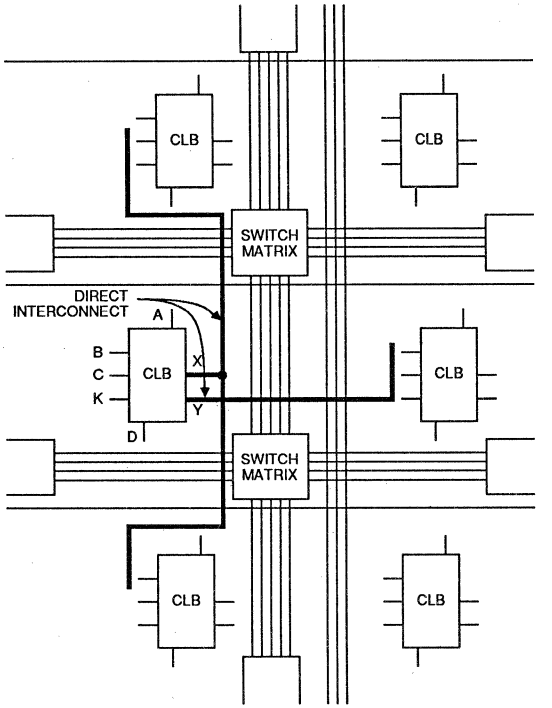
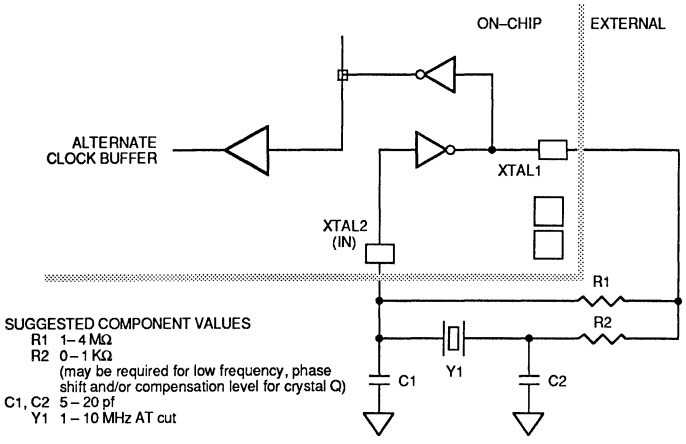


Figure 9. Direct Interconnect

Crystal Oscillator

An internal high-speed inverting amplifier is available to implement an on-chip crystal oscillator. It is associated with the auxiliary clock buffer in the lower right corner of the die. When configured to drive the auxiliary clock buffer, two special adjacent user I/O blocks are also configured to connect the oscillator amplifier with external crystal oscillator components, as shown in Figure 10. This circuit becomes active before configuration is complete in order to allow the oscillator to stabilize. Actual internal connection is delayed until completion of configuration. The feedback resistor R1 between output and input, biases the amplifier at threshold. It should be as large a value as practical to minimize loading of the crystal. The inversion of the amplifier, together with the R-C networks and crystal, produces the 360-degree phase shift of the Pierce oscillator.

A series resistor R2 may be included to add to the amplifier output impedance when needed for phase-shift control or crystal resistance matching or to limit the amplifier input swing to control clipping at large amplitudes. Excess feedback voltage may be adjusted by the ratio of C2/C1. The amplifier is designed to be used over the range from 1 MHz up to one-half the specified CLB toggle frequency. Use at frequencies below 1 MHz may require individual characterization with respect to a series resistance. Operation at frequencies above 20 MHz generally requires a crystal to operate in a third overtone mode, in which the fundamental frequency must be suppressed by the R-C networks. When the amplifier does not drive the auxiliary buffer, these I/O blocks and their package pins are available for general user I/O.



	XTAL1	XTAL2
48 DIP	33	30
68 PLCC	46	43
68 PGA	J10	L10
84 PLCC	56	53
84 PGA	K11	L11

Figure 10. Crystal Oscillator

Power

Power Distribution

Power for the LCA is distributed through a grid to achieve high noise immunity and isolation between logic and I/O. For packages having more than fourty-eight pins, two VCC pins and two ground pins are provided (see Figure 11). Inside the LCA, a dedicated VCC and ground ring surrounding the logic array provides power to the I/O drivers. An independent matrix of VCC and ground lines supplies the interior logic of the device. This power distribution grid provides a stable supply and ground for all internal logic, providing the external package power pins are appropriately decoupled. Typically a 0.1-μF capacitor connected between the VCC and ground pins near the package will provide adequate decoupling.

Output buffers capable of driving the specified 4-mA loads under worst-case conditions may be capable of driving 25 to 30 times that current in a best case. Noise can be reduced by minimizing external load capacitance and reducing simultaneous output transitions in the same direction. It may also be beneficial to locate heavily-loaded output buffers near the ground pads. Multiple VCC and ground pin connections are required for package types which provide them.

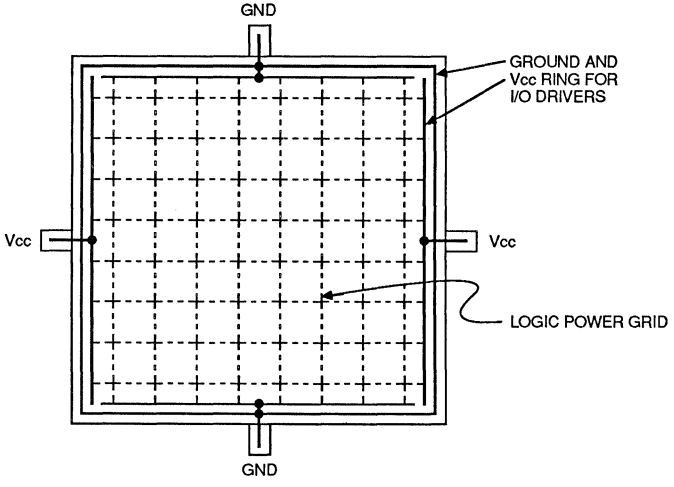


Figure 11. LCA Power Distribution

Power Dissipation

The Logic Cell Array exhibits the low power consumption characteristic of CMOS ICs. Only quiescent power is required for the LCA configured for CMOS input levels. The TTL input level configuration option requires additional power for level shifting. The power required by the static memory cells which hold the configuration data is very low and may be maintained in a power-down mode.

Typically most of power dissipation is produced by capacitive loads on the output buffers, since the power per output is 25 µW/pF/MHz. Another component of I/O power is the DC loading on each output pin. For any given system, the user can calculate the power requirement based on the resistive loading of the devices driven by the Logic Cell Array.

Internal power supply dissipation is a function of clock frequency and the number of nodes changing on each clock. In an LCA the fraction of nodes changing on a given clock is typically low (10-20%). For example, in a 16-bit binary counter, the average clock produces a change in slightly less than two of the sixteen bits. In a 4-input AND gate there will be two transitions in sixteen states. Typical global clock buffer power is about 3 mW/MHz for the M2064 and 4 mW/MHz for the M2018. With a "typical" load of three general interconnect segments, each CLB output requires about 0.4 mW/MHz of its output frequency. Graphs of power versus operating frequency are shown in Table 1.

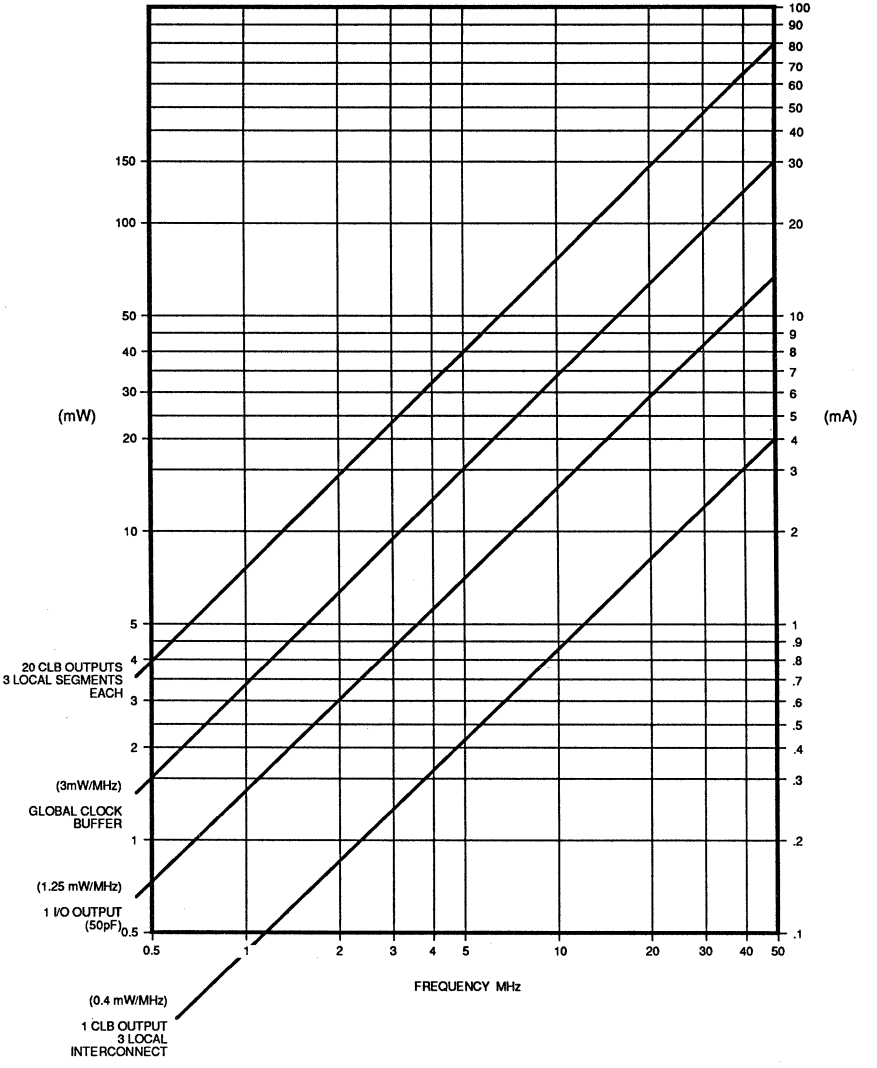


Table 1. Typical LCA Power Consumption by Element

Programming

Configuration data to define the function and interconnection within a Logic Cell Array are loaded automatically at power-up or upon command. Several methods of automatically loading the required data are designed into the Logic Cell Array and are determined by logic levels applied to mode selection pins at configuration time. The form of the data may be either serial or parallel, depending on the configuration mode. The programming data are independent of the configuration mode selected. The state diagram of Figure 12 illustrates the configuration process.

Input thresholds for user I/O pins can be selected to be either TTL-compatible or CMOS-compatible and remain in that state until the LCA begins operation. If the user has selected CMOS compatibility, the input thresholds are changed to CMOS levels during configuration.

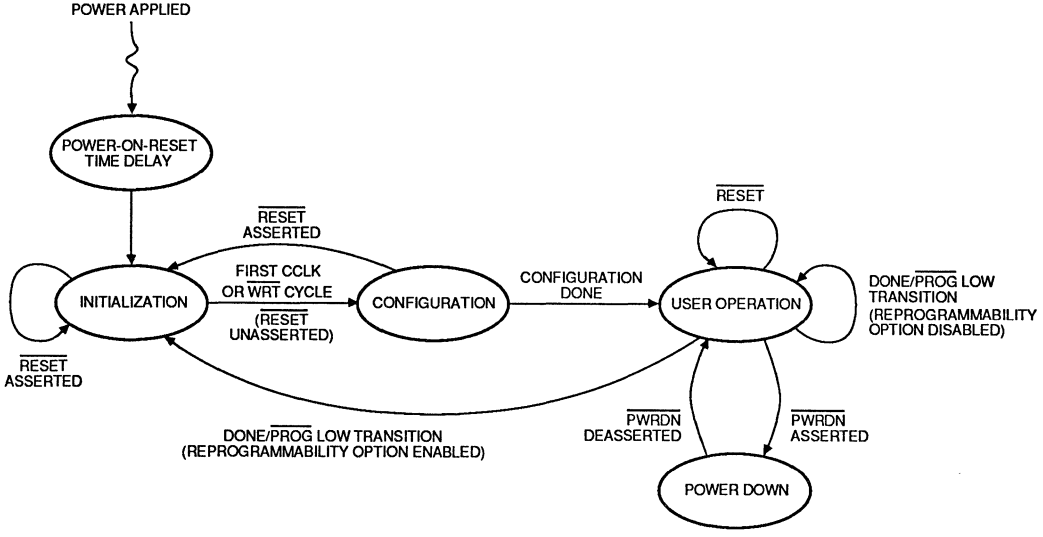


Figure 12. Configuration State Diagram

Figure 13 shows the specific data arrangement for the M2064 device. Future products will use the same data format to maintain compatibility between different devices of the Monolithic Memories' product line, but they will have different sizes and numbers of data frames. For the M2064 configuration requires 12,038 bits for each device. For the M2018, the configuration of each device requires 17,878 bits. The M2064 uses 160 configuration data frames and the M2018 uses 197.

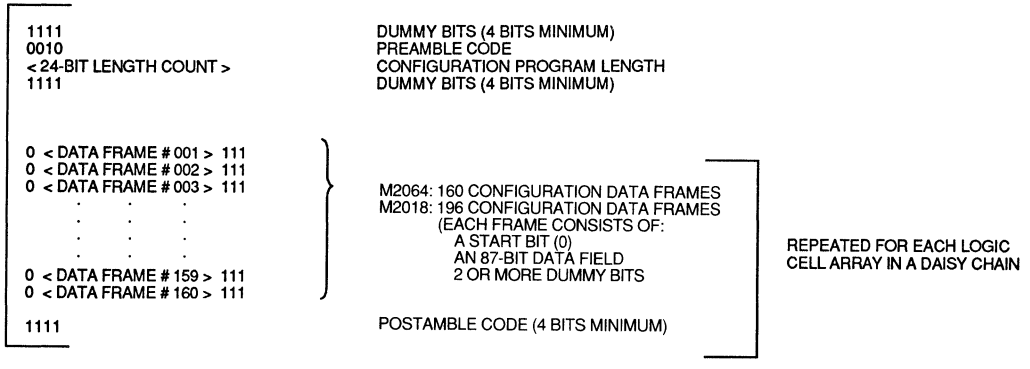


Figure 13. M2064 Configuration Data Arrangement

The configuration bit stream begins with preamble bits, a preamble code and a length count. The length count is loaded into the control logic of the Logic Cell Array and is used to determine the completion of the configuration process. When configuration is initiated, a 24-bit length counter is set to 0 and begins to count the total number of configuration clock cycles applied to the device. When the current length count equals the loaded length count, the configuration process is complete. Two clocks before completion, the internal logic becomes active and is reset. On the next clock, the inputs and outputs become active as configured and consideration should be given to avoid configuration signal contention. (Attention must be paid to avoid contention on pins which are used as inputs during configuration and become outputs in operation.) On the last configuration clock, the completion of configuration is signalled by the release of the DONE, $\overline{PROG}$ pin of the device as the device begins operation. This open-drain output can be AND-tied with multiple Logic Cell Arrays and used as an active-high READY or active-low, $\overline{RESET}$, to other portions of the system. High during configuration (HDC) and low during configuration ($\overline{LDC}$), are released one CCLK cycle before DONE is asserted. In master mode configurations, it is convenient to use $\overline{LDC}$ as an active-low EPROM chip enable.

As each data bit is supplied to the LCA, it is internally assembled into a data word. As each data word is completely assembled, it is loaded in parallel into one word of the internal configuration memory array. The last word must be loaded before the current length count compare is true. If the configuration data are in error, e.g., PROM address lines swapped, the LCA will not be ready at the length count and the counter will cycle through an additonal complete count prior to configuration being "done".

Figure 14 shows the selection of the configuration mode based on the state of the mode pins M0 and M1. These package pins are sampled prior to the start of the configuration process to determine the mode to be used. Once configuration is DONE and subsequent operation has begun, the mode pins may be used to perform data readback, as discussed later. An additional mode pin, M2, must be defined at the start of configuration. This package pin is a user-configurable I/O after configuration is complete.

MODE PIN			MODE SELECTED
M0	M1	M2	
0	0	0	Master serial
0	0	1	Master LOW mode
0	1	1	Master HIGH mode
1	0	1	Peripheral mode
1	1	1	Slave mode

Master LOW addresses begin at 0000 and increment.
Master HIGH addresses begin at FFFF and decrement.

Figure 14. Configuration Mode Selection

Initialization Phase

When power is applied, an internal power-on-reset circuit is triggered. When VCC reaches the voltage at which the LCA begins to operate (2.5 to 3 Volts), the chip is initialized, outputs are made high-impedance and a time-out is initiated to allow time for power to stabilize. This time-out (15 to 35 ms) is determined by a counter driven by a self-generated, internal sampling clock that drives the configuration clock (CCLK) in master configuration mode. This internal sampling clock will vary with process, temperature and power supply over the range of 0.5 to 1.5 MHz. LCAs with mode lines set for master mode will time-out of their initialization using a longer counter (60 to 140 ms) to assure that all devices, which it may be driving in a daisy chain, will be ready. Configuration using peripheral or slave modes must be delayed long enough for this initialization to be completed.

The initialization phase may be extended by asserting the active-low external $\overline{RESET}$. If a configuration has begun, an assertion of $\overline{RESET}$ will initiate an abort, including an orderly clearing of partially loaded configuration memory bits. After about three clock cycles for synchronization, initialization will require about 160 additional cycles of the internal sampling clock (197 for the M2018) to clear the internal memory before another configuration may begin. The same is true of a configured part in which the reconfigurable control bit is set. When a HIGH-to-LOW transition on the DONE, $\overline{PROG}$ package pin is detected, thereby initiating a reprogram, the configuration memory is cleared. This insures an orderly configuration in which no internal signal conflicts are generated during the loading process.

Master Mode

In master mode, the Logic Cell Array automatically loads the configuration program from an external memory device. Figure 15a shows an example of the master mode connections required. The Logic Cell Array provides sixteen address outputs and the control signals $\overline{RCLK}$ (read clock), HDC (high during configuration) and $\overline{LDC}$ (low during configuration) to execute read cycles from the external memory. Parallel eight-bit data words are read and internally serialized. As each data word is read, the least significant bit of each byte, normally D0, is the next bit in the serial stream.

Addresses supplied by the Logic Cell Array can be selected by the mode lines to begin at address 0 and incremented to read the memory (master low mode), or they can begin at address FFFF Hex and be decremented (master high mode). This capability is provided to allow the Logic Cell Array to share external memory with another device, such as a microprocessor. For example, if the processor begins its execution from low memory, the Logic Cell Array can load itself from high memory and enable the processor to begin execution once configuration is completed. The DONE, $\overline{PROG}$ output pin can be used to hold the processor in a Reset state until the Logic Cell Array has completed the configuration process.

The master serial mode uses serial configuration data, synchronized by the rising edge of RCLK, as in Figure 15b.

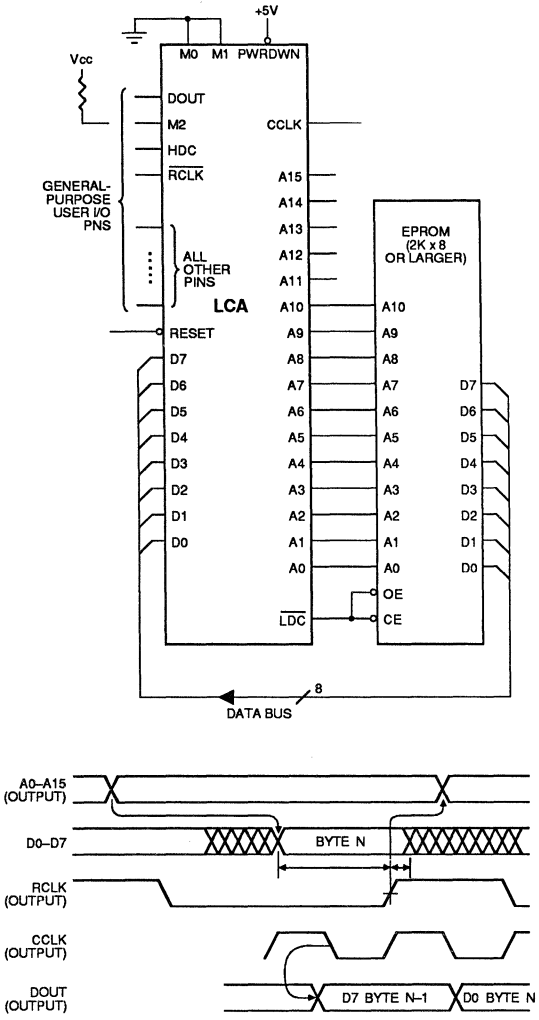


Figure 15a. Master Low Address Configuration

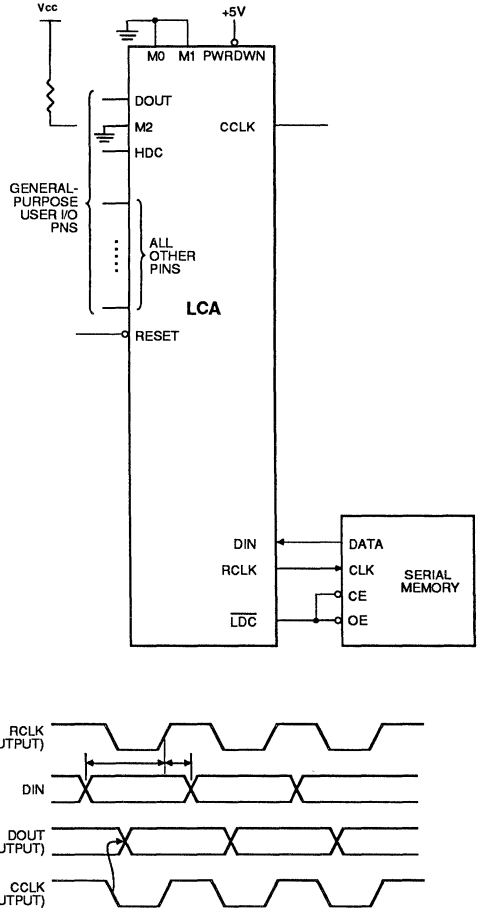


Figure 15b. Master Serial Mode Configuration

4

Peripheral Mode

Peripheral mode provides a simplified interface through which the device may be loaded as a processor peripheral. Figure 16 shows the peripheral mode connections. Processor write cycles are decoded from the common assertion of the active-low write strobe ($\overline{WRT}$), and two active-low and one active-high chip selects ($\overline{CS0}$, $\overline{CS1}$, CS2). If all these signals are not available, the unused inputs should be driven to their respective active levels. The Logic Cell Array will accept one bit of the configuration program on the data input (DIN) pin for each processor write cycle. Data is supplied in the serial sequence described earlier.

Since only a single bit from the processor data bus is loaded per cycle, the loading process involves the processor reading a byte or word of data, writing a bit of the data to the Logic Cell Array, shifting the word and writing a bit until all bits of the word are written, then continuing in the same fashion with the next word, etc. After the configuration program has been loaded, an additional three clocks (a total of three more than the length count) must be supplied in order to complete the configuration process. When more than one device is being used in the system, each device can be assigned a different bit in the processor data bus, and multiple devices can be loaded on each processor write cycle. This "broadside" loading method provides a very easy and time-efficient method of loading several devices.

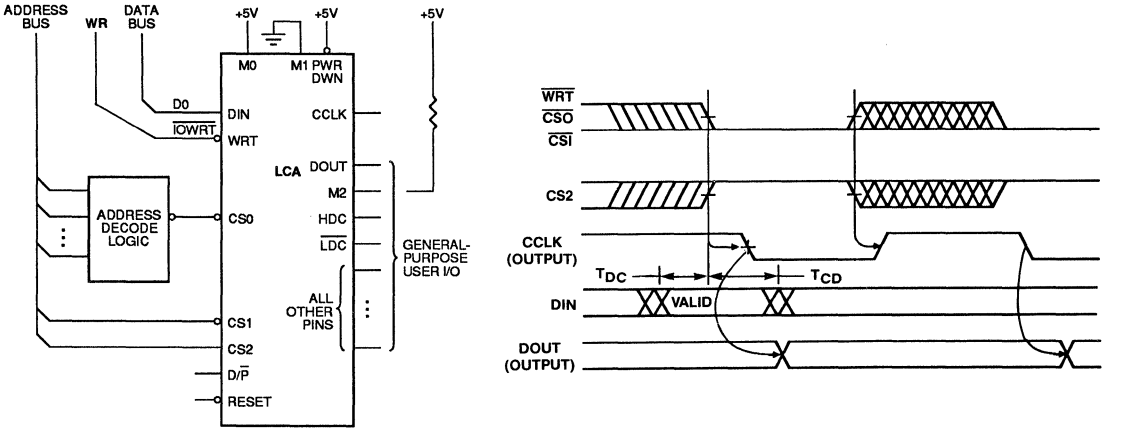


Figure 16. Peripheral Mode Configuration

Slave Mode

Slave mode, Figure 17, provides the simplest interface for loading the Logic Cell Array configuration. Data is supplied in conjunction with a synchronizing clock. For each LOW-to-HIGH input transition of configuration clock (CCLK), the data present on the data input (DIN) pin is loaded into the internal shift register. Data may be supplied by a processor or by other special circuits. Slave mode is used for downstream devices in a daisy-chain configuration. The data for each slave LCA are supplied by the preceding LCA in the chain, and the clock is supplied by the lead device, which is configured in master or peripheral mode. After the configuration program has been loaded, an additional three clocks (a total of three more than the length count) must be supplied in order to complete the configuration process.

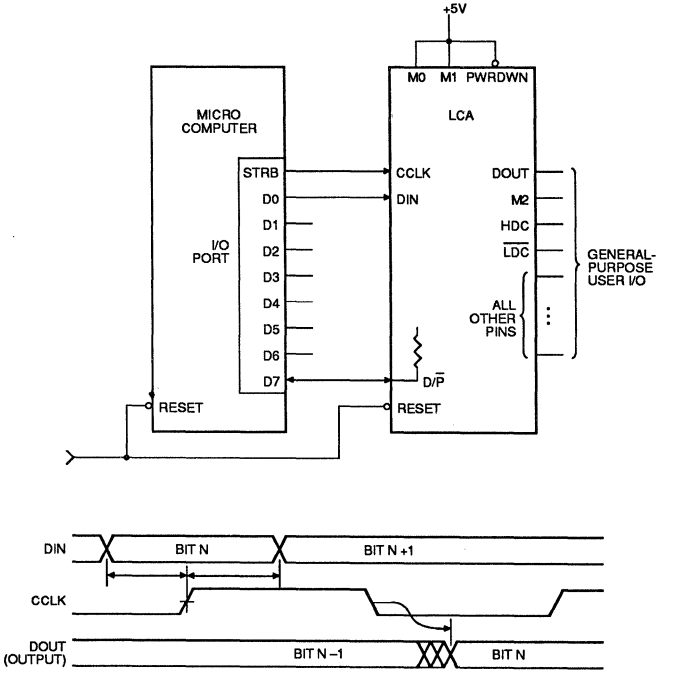


Figure 17. Slave Mode Configuration

Daisy Chain

The daisy-chain programming mode is supported by Logic Cell Array in all programming modes. In master mode and peripheral mode, the LCA can act as a source of data and control for slave devices. For example, Figure 18 shows a single device in master mode, with two devices in slave mode. The master mode device reads the external memory and begins the configuration loading process for all of the devices.

The data begin with a preamble and a length count which is supplied to all devices at the beginning of the configuration. The length count represents the total number of cycles required to load all of the devices in the daisy chain. After loading the length count, the lead device will load its configuration data while providing a HIGH DOUT to downstream devices. When the lead device has been loaded and the current length count has not reached the full value, memory access continues. Data bytes are read and serialized by the lead device. The data are passed through the lead device and appear on the data out (DOUT) pin in serial form. The lead device also generates the configuration clock (CCLK) to synchronize the serial output data. A master mode device generates an internal CCLK of eight times the EPROM address rate, while a peripheral mode device produces CCLK from the chip select and write strobe timing.

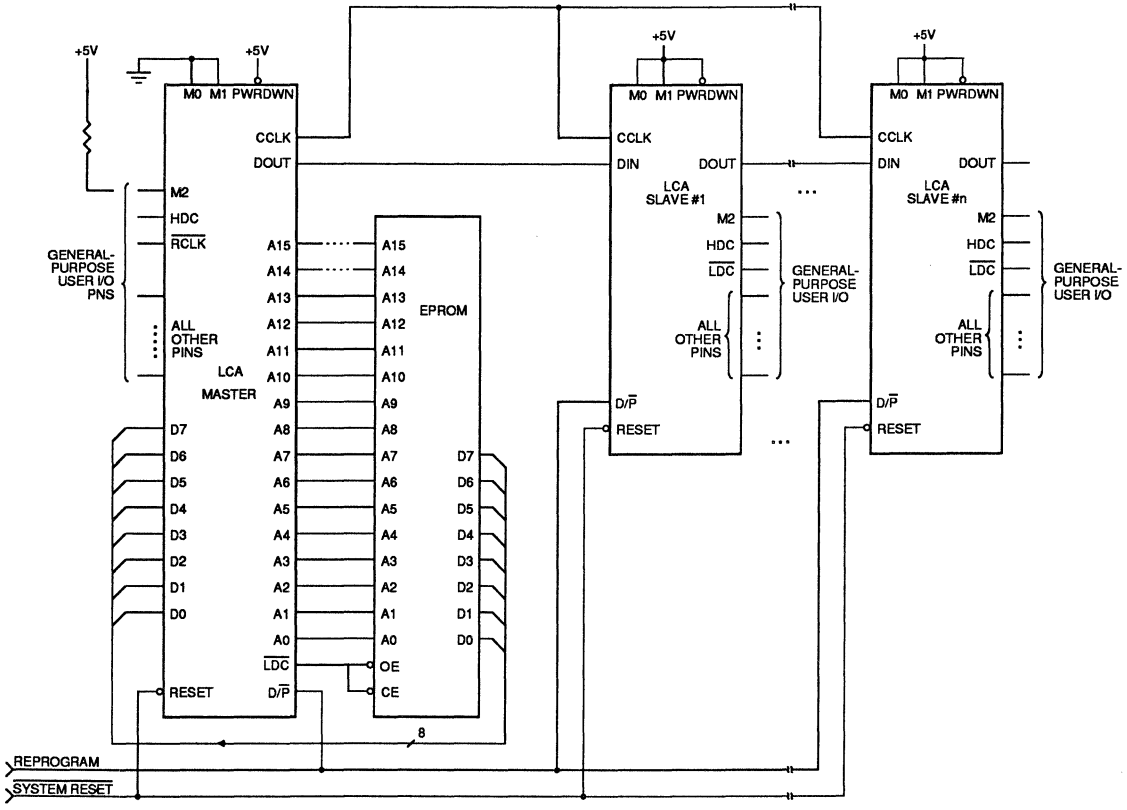


Figure 18. Master Mode with Daisy Chain

Operation

When all of the devices have been loaded and the length count is complete, a synchronous start-up of operation is performed. On the clock cycle following the end of loading, the internal logic begins functioning in the reset state. On the next CCLK, the configured output buffers become active to allow signals to stabilize. The next CCLK cycle produces the DONE condition. The length count control of operation allows a system of multiple Logic Cell Arrays to begin operation in a synchronized fashion. If the crystal oscillator is used, it will begin operation before configuration is complete to allow time for stabilization before it is connected to the internal circuitry.

Special Features

In addition to the normal user logic functions and interconnect, the configuration data include control for several special functions:

- Input thresholds
- Readback enable
- Reprogram enable
- DONE pull-up resistor

Each of these functions is controlled by a portion of the configuration program generated by the XACT Development System.

Input Thresholds

During configuration, all input thresholds are TTL level. During configuration input thresholds are established as specified, either TTL or CMOS. The $\overline{\text{PWRDN}}$ input threshold is an exception; it is always a CMOS level input. The TTL threshold option requires additional power for threshold shifting.

Readback

After a Logic Cell Array has been programmed, the configuration program may be read back from the device. Readback may be used for verification of configuration and as a method of determining the state of internal logic nodes during debugging. In applications in which the verification is not used, it may be desirable to limit access to the configuration data. Three readback options are provided: 'on command', 'only once', and 'never'. If 'on-command readback' is selected, the device will respond to all readback requests. If 'readback once' is selected, the device will respond only to the first readback request after programming is complete. Subsequent readback requests will be ignored. If 'readback never' is selected, the device will not respond to a readback command.

Readback is accomplished without the use of any of the user I/O pins; only M0, M1, and CCLK pins are used. An initiation of readback is produced by a LOW-to-HIGH transition of the M0, RTRIG (read trigger) pin. Once the readback command has been given, CCLK is cycled to read back each data bit in a format similar to loading. After two dummy bits, the first data frame is shifted out, in inverted sense, on the M1, $\overline{\text{RDATA}}$ (read data) pin. All data frames must be read back to complete the process and return the mode select and CCLK pins to their normal functions.

In addition to the configuration program, the readback includes the current state of each of the internal logic block storage elements, and the state of the input (I) connection pin on each I/O block. This state information is used by the Logic Cell Array development system In-Circuit Emulator to provide visibility into the internal operation of the logic while the system is operating. To readback a uniform time sample of all storage elements it may be necessary to inhibit the system clock.

Reprogram

The configuration memory of the Logic Cell Array may be rewritten while the device is in the user's system, if that option is selected when the LCA is configured. If another programming cycle is to be initiated, the dual function package pin DONE, $\overline{\text{PROG}}$ must be given a HIGH-to-LOW transition. Sensitivity to noise is reduced, by confirming the HIGH-to-LOW transition over two to three cycles using the LCA's internal sampling oscillator. When a reprogram command is recognized, all internal logic and connectivity definitions are erased and the I/O package pins are forced to a high impedance condition. The device returns to the initialization state. Reprogram control is often implemented with an external open collector driver which pulls DONE, $\overline{\text{PROG}}$ LOW. Once it recognizes a stable request, the Logic Cell Array will hold a LOW until the new configuration has been completed. Whether or not the reprogram request is maintained, the Logic Cell Array will begin operation upon completion of configuration.

DONE Pull-up

The DONE, $\overline{\text{PROG}}$ pin is an open drain I/O that indicates programming status. As an input, it initiates a reprogram operation. An optional internal pull-up resistor may be enabled.

Battery Backup

Because the control store of the Logic Cell Array is a CMOS static memory, its cells require only a very low standby current for data retention. In some systems, this low data retention current characteristic facilitates preserving configurations in the event of a primary power loss. The Logic Cell Array has built in power-down logic which, when activated, will disable normal operation of the device and retain only the configuration data. All internal operation is suspended and output buffers are placed in their high-impedance state.

Power-down data retention is possible with a simple battery-backup circuit because the power requirement is extremely low. For retention at 2.0 V, the required current is typically on the order of 0.5 mA. Screening of this parameter is available. To force the Logic Cell Array into the power-down state, the user must pull the $\overline{\text{PWRDWN}}$ pin low and continue to supply a retention voltage to the VCC pins of the package. When normal power is restored, VCC is elevated to its normal operating voltage and $\overline{\text{PWRDWN}}$ is returned to a HIGH. The Logic Cell Array resumes operation with the same internal sequence that occurs at the conclusion of configuration. Internal I/O and logic block storage elements will be reset, the outputs will become enabled and then the DONE, $\overline{\text{PROG}}$ pin will be released. No configuration programming is involved.

Performance

The high performance of the Logic Cell Array results from its patented architectural features and from the use of an advanced high-speed CMOS manufacturing process. Performance may be measured in terms of minimum propagation times for logic elements.

Flip-flop loop delays for the I/O block and logic block flip-flops are about 3 ns. This short delay provides very good performance under asynchronous clock and data conditions. Short loop delays minimize the probability of a metastable condition which can result from assertion of the clock during data transitions. Because of the short loop delay characteristic in the Logic Cell Array, the I/O block flip-flops can be used very effectively to synchronize external signals applied to the device. Once synchronized in the I/O block, the signals can be used internally without further consideration of their clock relative timing, except as it applies to the internal logic and routing path delays.

Device Performance

The single parameter which most accurately describes the overall performance of the Logic Cell Array is the maximum toggle rate for a logic block storage element configured as a toggle flip-flop. The configuration for determining the toggle performance of the Logic Cell Array is shown in Figure 19. The clock for the storage element is provided by the global clock buffer and the flip-flop output Q is fed back through the combinatorial logic to form the data input for the next clock edge. Using this arrangement, flip-flops in the Logic Cell Array can be toggled at clock rates from 33-70 MHz, depending on the speed grade used.

Actual Logic Cell Array performance is determined by the critical path speed, including both the speed of the logic and storage elements in that path, and the speed of the particular network routing. Figure 20 shows a typical system logic configuration of two flip-flops with an extra combinatorial level between them. Depending on speed grade, system clock rates to 35 MHz are practical for this logic. To allow the user to make the best use of the capabilities of the device, the delay calculator in the XACT Development System determines worst-case path delays using actual impedance and loading information.

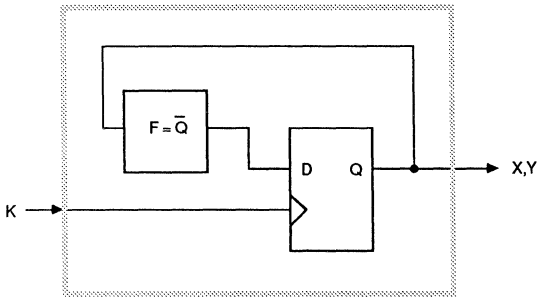


Figure 19. Logic Block Configuration for Toggle Rate Measurement

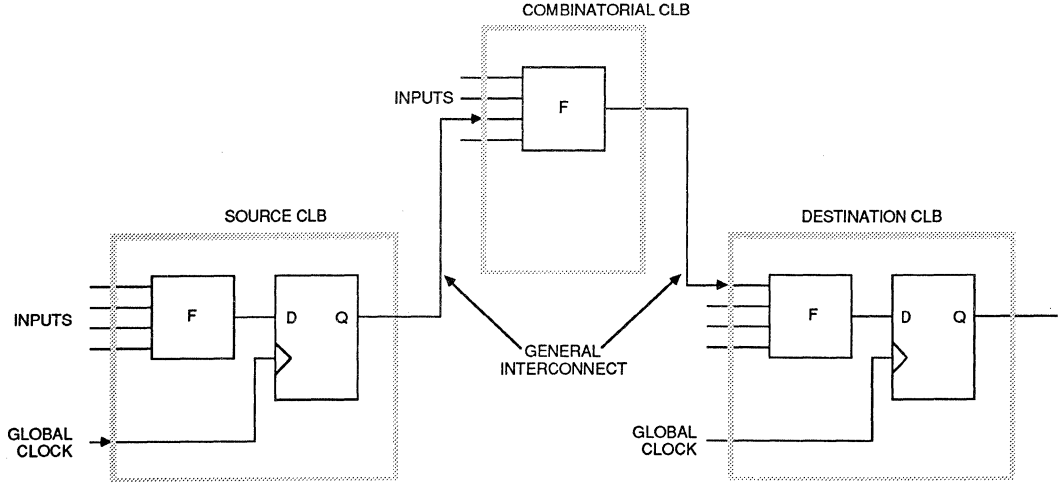
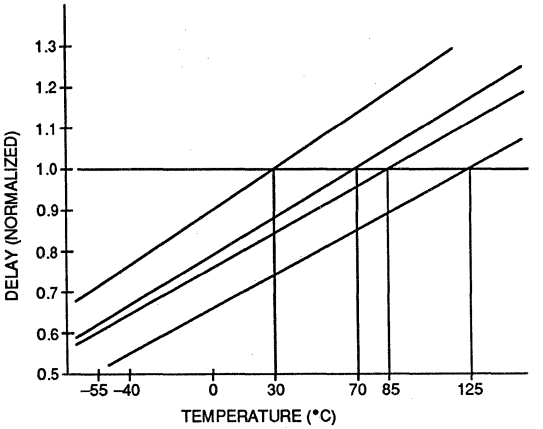


Figure 20. Typical Logic Path

Logic Block Performance

Logic Block propagation times are measured from the interconnect point at the input of the combinatorial logic to the output of the block in the interconnect area. Combinatorial performance is independent of logic function because of the table look-up based implementation. Timing is different when the combinatorial logic is used in conjunction with the storage element. For the combinatorial logic function driving the data input of the storage element, the critical timing is data set-up relative to the clock edge provided to the storage element. The delay from the clock source to the output of the logic block is critical in the timing of signals produced by storage elements. The loading on a logic block output is limited only by the additional propagation delay of the interconnect network. Performance of the logic block is a function of supply voltage and temperature, as shown in Figures 21 and 22.



NOTE: NORMALIZED FOR FOUR TEMPERATURES

Figure 21. Delay vs. Temperature

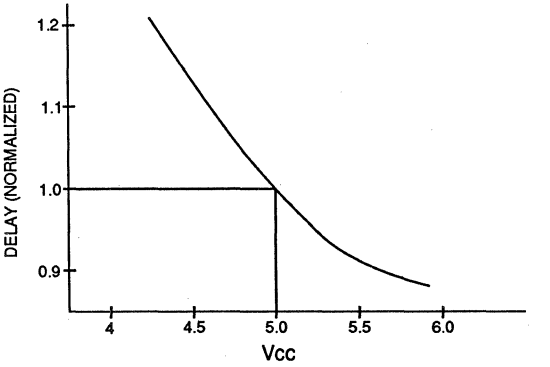


Figure 22. Delay vs. Power Supply

Interconnect Performance

Interconnect performance depends on the routing resource used to implement the signal path. As discussed earlier, direct interconnect from block to block provides a minimum delay path for a signal.

The single metal segment used for long lines exhibits low resistance from end to end, but relatively high capacitance. Signals driven through a programmable switch will have the additional impedance of the switch added to their normal drive impedance.

General-purpose interconnect performance depends on the number of switches and segments used, the presence of the bidirectional repowering buffers and the overall loading on the signal path at all points along the path. In calculating the worst-case delay for a general interconnect path, the delay calculator portion of the XACT development system accounts for all of these elements. As an approximation, interconnect delay is proportional to the summation of totals of local metal segments beyond each programmable switch. In effect, the delay is a sum of R-C delays each approximated by an R times the total C it drives. The R of the switch and the C of the interconnect are functions of the particular device performance grade. For a string of three local interconnects, the approximate delay at the first segment, after the first switch resistance, would be three units; an additonal two delay units after the next switch plus an additional delay after the last switch in the chain. The interconnect R-C chain terminates at each repowering buffer. Nearly all of the capacitance is in the interconnect metal and switches; the capacitance of the block inputs is not significant. Figure 23 shows an estimation of this delay.

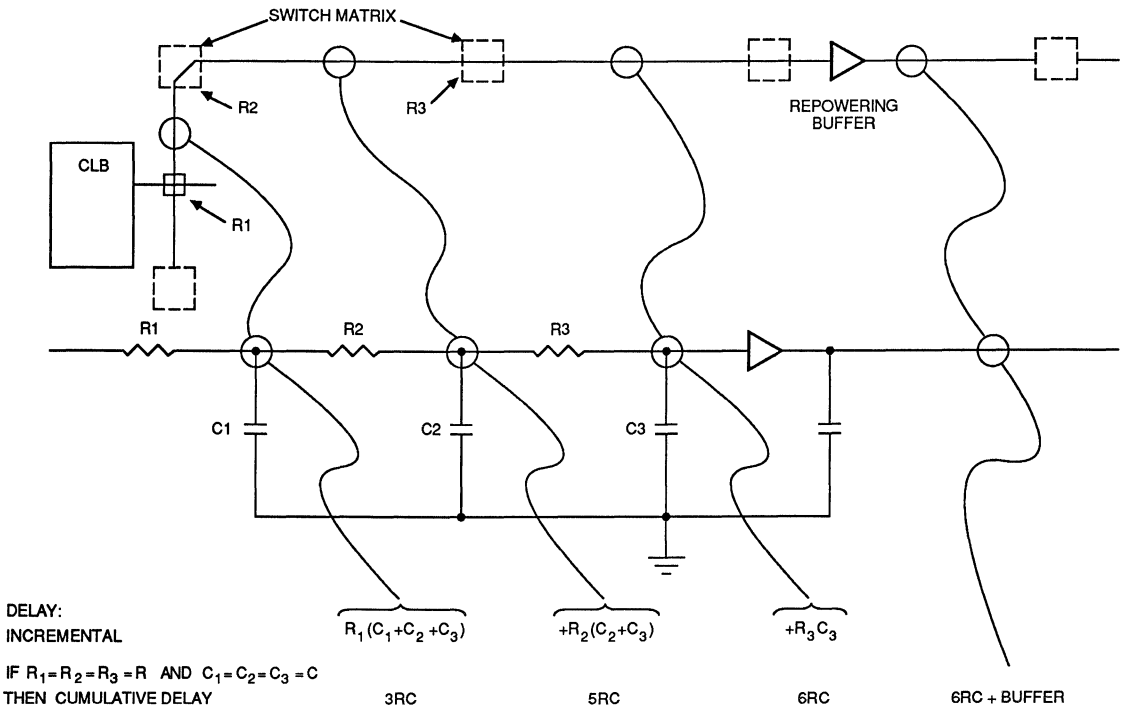


Figure 23. Interconnection Delay Example

Development System

To support designers using the Logic Cell Array , Monolithic Memories provides a basic development system with several options for additional productivity. The XACT system provides the following:

- Graphic-driven design entry
- Schematic entry
- Interactive timing delay calculations
- Macrocell library support, both for standard Monolithic Memories supplied functions and user-defined functions
- Design entry checking for consistency and completeness
- Automatic design documentation generation
- Automatic placement and routing
- Simulation interface support, including automatic netlist (circuit description) and timing extraction
- In-circuit emulation for multiple devices

The host system on which the XACT system operates is an IBM™ PC-XT™ or PC-AT™ or compatible system with MS-DOS™ 2.1 or higher. Color graphics is required as well as 640 K bytes of internal RAM (an Expanded Memory Specification (EMS) card with 256 K bytes of memory is required for the M2018). A complete system requires one parallel I/O port and two serial ports and a mouse.

For more detailed information of the XACT Development System, please refer to Logic Cell Array Development System Datasheet.

48-PIN DIP	68-PIN PLCC	68-PIN PGA	CONFIGURATION MODE: ≪M2: M1: M0≫				USER OPERATION
			SLAVE <1:1:1>	PERIPHERAL <1:0:1>	MASTER-HIGH <1:1:0>	MASTER-LOW <1:0:0>	
	1	B6	GND				
	2	A6	≪HIGH≫		A13 (O)		I/O
1	3	B5			A6 (O)		
	4	A5			A12 (O)		
2	5	B4			A7 (O)		
3	6	A4			A11 (O)		
4	7	B3			A8 (O)		
5	8	A3			A10 (O)		
6	9	A2			A9 (O)		
7	10	B2	$\overline{\text{PWRDWN}}$ (I)				
8	11	B1	≪HIGH≫				I/O
	12	C2					
9	13	C1					
	14	D2					
10	15	D1					
	16	E2					
11	17	E1					
12	18	F2	VCC				
13	19	F1	≪HIGH≫				I/O
	20	G2					
14	21	G1					
	22	H2					
15	23	H1					
16	24	J2					
17	25	J1	M1 (HIGH)	M1 (LOW)	M1 (HIGH)	M1 (LOW)	$\overline{\text{RDATA}}$ (O)
18	26	K1	M0 (HIGH)	M0 (HIGH)	M0 (LOW)	M0 (LOW)	RTRIG (I)
19	27	K2	M2 (HIGH)				I/O
20	28	L2	HDC (HIGH)				
	29	K3	≪HIGH≫				
21	30	L3	LDC (LOW)				
	31	K4	≪HIGH≫				
22	32	L4					
	33	K5					
23	34	L5					

≪HIGH≫ is high impedance with a 20 to 50-KΩ internal pull-up resistor during configuration

Table 2a. M2064 Pin Assignments

(continued on next page)

48-PIN DIP	68-PIN PLCC	68-PIN PGA	CONFIGURATION MODE: ≪M2: M1: M0≫				USER OPERATION
			SLAVE <1:1:1>	PERIPHERAL <1:0:1>	MASTER-HIGH <1:1:0>	MASTER-LOW <1:0:0>	
24	35	K6	GND				
	36	L6	≪HIGH≫				I/O
25	37	K7					
	38	L7					
26	39	K8					
27	40	L8					
28	41	K9			D7 (I)		
29	42	L9			D6 (I)		
30	43	L10					XTL2 or I/O
31	44	K10	$\overline{\text{RESET}}$ (I)				
32	45	K11	DONE (O)				$\overline{\text{PROG}}$ (I)
33	46	J10	≪HIGH≫				XTL1 or I/O
	47	J11					I/O
34	48	H10			D5 (I)		
	49	H11					
35	50	G10		$\overline{\text{CS0}}$ (I)	D4 (I)		
36	51	G11		$\overline{\text{CS1}}$ (I)	D3 (I)		
	52	F10	VCC				
	53	F11	≪HIGH≫				I/O
37	54	E10		CS2 (I)	D2 (I)		
	55	E11					
38	56	D10		$\overline{\text{WRT}}$ (I)	D1 (I)		
39	57	D11			$\overline{\text{RCLK}}$		
40	58	C10	DIN (I)		D0 (I)		
41	59	C11	DOUT (O)				
42	60	B11	CCLK (I)	CCLK (O)			CCLK (I)
43	61	B10	≪HIGH≫		A0 (O)		I/O
44	62	A10			A1 (O)		
45	63	B9			A2 (O)		
46	64	A9			A3 (O)		
	65	B8			A15 (O)		
47	66	A8			A4 (O)		
	67	B7			A14 (O)		
48	68	A7			A5 (O)		

≪HIGH≫ is high impedance with a 20 to 50-KΩ internal pull-up resistor during configuration

Table 2a. M2064 Pin Assignments (continued)

68-PIN PLCC	68-PIN PGA	84-PIN PLCC	84-PIN PGA	CONFIGURATION MODE: ≪M2: M1: M0≫				USER OPERATION
				SLAVE <1:1:1>	PERIPHERAL <1:0:1>	MASTER-HIGH <1:1:0>	MASTER-LOW <1:0:0>	
1	B6	1	C6	GND				
2	A6	2	A6	≪HIGH≫		A13 (O)		I/O
		3	A5					
		4	B5					
3	B5	5	C5			A6 (O)		
4	A5	6	A4			A12 (O)		
5	B4	7	B4			A7 (O)		
6	A4	8	A3			A11 (O)		
7	B3	9	A2			A8 (O)		
8	A3	10	B3			A10 (O)		
9	A2	11	A1			A9 (O)		
10	B2	12	B2	$\overline{\text{PWRDWN}}$ (I)				
11	B1	13	C2	≪HIGH≫				I/O
12	C2	14	B1					
13	C1	15	C1					
14	D2	16	D2					
15	D1	17	D1					
		18	E3					
16	E2	19	E2					
		20	E1					
17	E1	21	F2					
18	F2	22	F3	VCC				
19	F1	23	G3	≪HIGH≫				I/O
		24	G1					
20	G2	25	G2					
		26	F1					
21	G1	27	H1					
22	H2	28	H2					
23	H1	29	J1					
24	J2	30	K1					
25	J1	31	J2	M1 (HIGH)	M1 (LOW)	M1 (HIGH)	M1 (LOW)	$\overline{\text{RDATA}}$ (O)
26	K1	32	L1	M0 (HIGH)	M0 (HIGH)	M0 (LOW)	M0 (LOW)	RTRIG (I)
27	K2	33	K2	M2 (HIGH)				I/O
28	L2	34	K3	HDC (HIGH)				
29	K3	35	L2	≪HIGH≫				
30	L3	36	L3	LDC (LOW)				
31	K4	37	K4	≪HIGH≫				
32	L4	38	L4					
		39	J5					
33	K5	40	K5					
34	L5	41	L5					
		42	K6					

≪HIGH≫ is high impedance with a 20 to 50-KΩ internal pull-up resistor during configuration

Table 2b. M2018 Pin Assignments (continued on next page)

68-PIN PLCC	68-PIN PGA	84-PIN PLCC	84-PIN PGA	CONFIGURATION MODE: ≪M2: M1: M0≫ SLAVE <1:1:1>	PERIPHERAL <1:0:1>	MASTER-HIGH <1:1:0>	MASTER-LOW <1:0:0>	USER OPERATION
35	K6	43	J6	GND				
		44	J7					
36	L6	45	L7					
37	K7	46	K7					
38	L7	47	L6	≪HIGH≫				
		48	L8					I/O
39	K8	49	K8					
40	L8	50	L9					
41	K9	51	L10			D7 (I)		
42	L9	52	K9			D6 (I)		
43	L10	53	L11					XT2 or I/O
44	K10	54	K10	$\overline{\text{RESET}}$ (I)				
45	K11	55	J10	DONE (0)				$\overline{\text{PROG}}$ (1)
46	J10	56	K11					XTL1 or I/O
47	J11	57	J11					
48	H10	58	H10	≪HIGH≫		D5 (I)		
		59	H11					
49	H11	60	F10					I/O
		61	G10					
50	G10	62	G11		$\overline{\text{CS0}}$ (I)	D4 (I)		
51	G11	63	G9		$\overline{\text{CS1}}$ (I)	D3 (I)		
52	F10	64	F9	VCC				
53	F11	65	F11					
54	E10	66	E11		CS2 (I)	D2 (I)		
		67	E10					
55	E11	68	E9	≪HIGH≫				
		69	D11					I/O
56	D10	70	D10		$\overline{\text{WRT}}$ (I)	D1 (I)		
57	D11	71	C11			$\overline{\text{RCLK}}$		
58	C10	72	B11	DIN (I)		D0 (I)		
59	C11	73	C10	DOUT (O)				
60	B11	74	A11	CCLK (I)	CCLK (O)			CCLK (I)
61	B10	75	B10			A0 (O)		
62	A10	76	B9			A1 (O)		
63	B9	77	A10			A2 (O)		
64	A9	78	A9			A3 (O)		
65	B8	79	B8			A15 (O)		
66	A8	80	A8	≪HIGH≫		A4 (O)		I/O
67	B7	81	B6			A14 (O)		
		82	B7					
		83	A7					
68	A7	84	C7			A5 (O)		

≪HIGH≫ is high impedance with a 20 to 50-KΩ internal pull-up resistor during configuration

Table 2b. M2018 Pin Assignments (continued)

Absolute Maximum Ratings*

Supply voltage V_{CC} ..... -0.5 V to 7 V
Power down V_{CC} ..... 2 V to 7 V
Input voltage ..... -0.5 V to V_{CC} 0.5 V
Voltage applied to three-state output ..... -0.5 V to V_{CC} 0.5 V
Storage temperature range ..... -65°C to +150°C
Lead temperature (soldering, 10 seconds) ..... 260°C

* Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those listed under "Recommended Operating Conditions" is not implied. Exposure to "Absolute Maximum Ratings" conditions for extended periods of time may affect device reliability.

Operating Conditions

SYMBOL	PARAMETER	MIN	TYP	MAX	UNIT
V_{CC}	Supply voltage relative to GND	4.75		5.25	V
V_{IHT}	High level input voltage—TTL configuration	2.0		V_{CC}	V
V_{IHC}	High level input voltage—CMOS configuration	0.7 V_{CC}		V_{CC}	V
V_{ILT}	Low level input voltage—TTL configuration	0		0.8	V
V_{ILC}	Low level input voltage—CMOS configuration	0		0.2 V_{CC}	V
I_{IT}	Input leakage current—TTL configuration	±10			μA
I_{IC}	Input leakage current—CMOS configuration	±10			μA
I_{OZ}	Three-state output off current (V_{CC} = 5.5 V)	±10			μA
t_{OP}	Operating free-air temperature	0		70	°C

Electrical Characteristics Over Operating Conditions

SYMBOL	PARAMETER		TEST CONDITION		MIN	TYP	MAX	UNIT
V_{OH}	High level output voltage		V_{CC} = 4.75 V	I_{OH} = -4.0 mA	3.86			V
V_{OL}	Low level output voltage		V_{CC} = 4.75 V	I_{OL} = 4.0 mA			0.32	V
I_{CCO}	Quiescent operating power supply current	CMOS inputs	V_{CC} = 5.0 V				5	mA
		TTL inputs	V_{CC} = 5.0 V				10	mA
I_{CCPD}	Power down supply current		V_{CC} = 5.0 V			0.5		mA

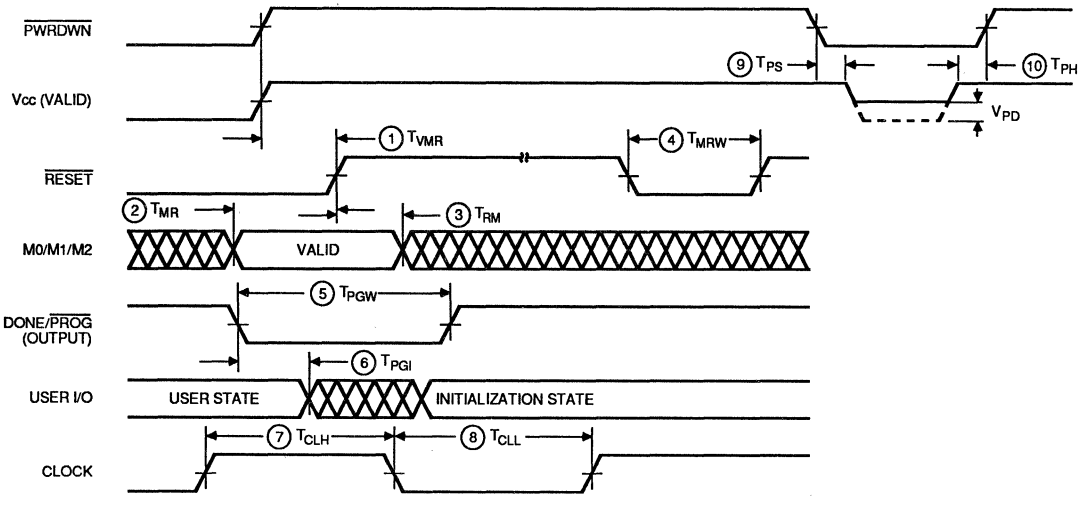
Power On Timing

The LCAs contain on-chip reset timing logic for power-up operation. To insure proper master mode system operation, VCC must rise from 2.0 V to minimum specification level in 10 ms or less. For other modes, initiation of configuration must be delayed for 60 ms after VCC reaches the minimum specified level.

Switching Characteristics — General

SYMBOL	DESCRIPTION		-33		-50		-70		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	
t_{VMR} ①	$\overline{RESET}$ (2)	V_{CC} setup (2.0 V)	150		150		150		ns
t_{MR} ②		M2, M1, M0 setup	60		60		60		ns
t_{RM} ③		M2, M1, M0 hold	60		60		60		ns
t_{MRW} ④		Width (LOW)	150		150		150		ns
t_{PGW} ⑤	DONE/ $\overline{PROG}$	Program width (LOW)	6		6		6		µs
t_{PGI} ⑥		Initialization		7		7		7	µs
t_{CLH} ⑦	CLOCK	Clock (HIGH)	12		8		7		ns
t_{CLL} ⑧		Clock (LOW)	12		8		7		ns
t_{PS} ⑨	$\overline{PWR}$ $\overline{DWN}$	Setup to V_{CC}	0		0		0		ns
t_{PH} ⑩		Hold from V_{CC}	0		0		0		ns
V_{PD}		Power Down	2.0		2.0		2.0		V

Notes: 1. V_{CC} must rise from 2.0 Volts to V_{CC} minimum in lest than 10 ms for master mode.
2. $\overline{RESET}$ timing relative to power-on and valid mode lines (M0, M1, M2) is relevant only when $\overline{RESET}$ is used to delay configuration.
3. Minimum CLOCK widths for the auxilliary buffer are 1.25 times the t_{CLH}, t_{CLL}.



Switching Characteristics — CLB

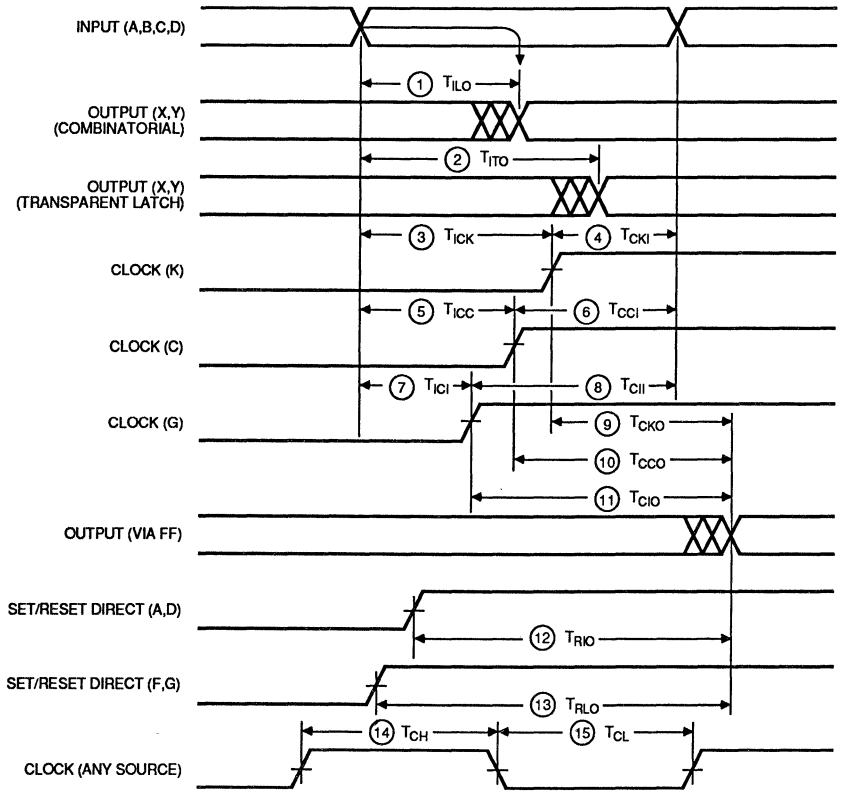
SYMBOL	DESCRIPTION		-33		-50		-70		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	
t_{ILO} ①	Logic input to output	Combinatorial		20		15		10	ns
t_{ITO} ②		Transparent latch		25		20		14	ns
t_{QLO}		Additional for Q through F or G to out		13		8		6	ns
t_{CKO} ⑨	K Clock	To output		20		15		10	ns
t_{ICK} ③		Logic-input setup	12		8		7		ns
t_{CKI} ④		Logic-input hold	0		0		0		ns
t_{CCO} ⑩	C Clock	To output		25		19		13	ns
t_{ICC} ⑤		Logic-input setup	12		9		6		ns
t_{CCI} ⑥		Logic-input hold	6		0		0		ns
t_{CIO} ⑪	Logic input to G Clock	To output		37		27		20	ns
t_{ICI} ⑦		Logic-input setup	6		4		3		ns
t_{CII} ⑧		Logic-input hold	9		5		4		ns
t_{RIO} ⑫	Set/reset direct	Input A or D to out		25		22		16	ns
t_{RLO} ⑬		Through F or G to out		37		28		21	ns
t_{MRQ}		Master Reset pin to out		35		25		20	ns
t_{RS}		Separation of set/reset	17		9		7		ns
t_{RPW}		Set/reset pulse-width	12		9		7		ns
F_{CLK}	Flip-flop toggle rate	Q through F to flip-flop	33		50		70		MHz
t_{CH} ⑭	Clock	Clock HIGH	12		8		7		ns
t_{CL} ⑮		Clock LOW	12		8		7		ns

Note: All switching characteristics apply to all valid combinations of process, temperature and supply.

Cross Reference Guide

XILINX	MMI	V_{CC}		F_{MAX}
		MIN	MAX	MIN
XC2064-1		4.5 V	5.5 V	20 MHz
	M2064-20	4.75 V	5.25 V	20 MHz
XC2064-2		4.5 V	5.5 V	33 MHz
XC2064-33	M2064-33	4.75 V	5.25 V	33 MHz
XC2064-50	M2064-50	4.75 V	5.25 V	50 MHz
XC2064-70	M2064-70	4.75 V	5.25 V	70 MHz
XC2018-33	M2018-33	4.75 V	5.25 V	33 MHz
XC2018-50	M2018-50	4.75 V	5.25 V	50 MHz
XC2018-70	M2018-70	4.75 V	5.25 V	70 MHz

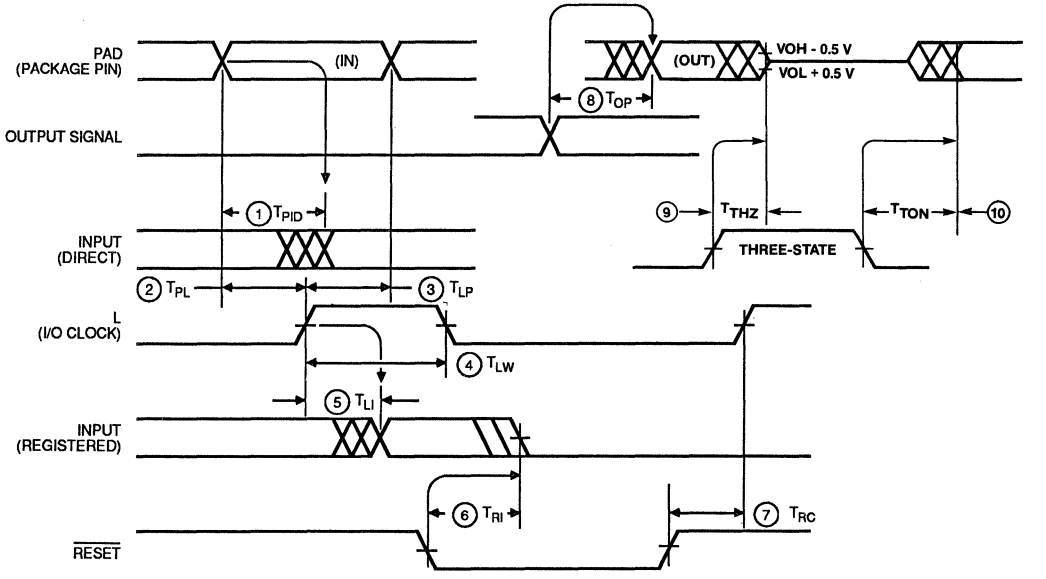
Switching Characteristics CLB



Switching Characteristics — IOB

SYMBOL	DESCRIPTION	-33		-50		-70		UNIT
		MIN	MAX	MIN	MAX	MIN	MAX	
t_{PID} ①	Pad (package pin) to input (direct)		12		8		6	ns
t_{LI} ⑤	I/O Clock to input (storage)		20		15		11	ns
t_{PL} ②	I/O Clock to pad-input setup	12		8		6		ns
t_{LP} ③	I/O Clock to pad-input hold	0		0		0		ns
t_{LW} ④	I/O Clock pulse width	12		9		7		ns
	I/O Clock frequency	33		50		70		MHz
t_{OP} ⑧	Output to pad (output enabled)		15		12		9	ns
t_{THZ} ⑨	Three-state to pad begin hi-Z		25		20		15	ns
t_{TON} ⑩	Three-state to pad end hi-Z		25		20		15	ns
t_{RI} ⑥	$\overline{\text{RESET}}$ to input (storage)		40		30		25	ns
t_{RC} ⑦	$\overline{\text{RESET}}$ to input clock		35		25		20	ns

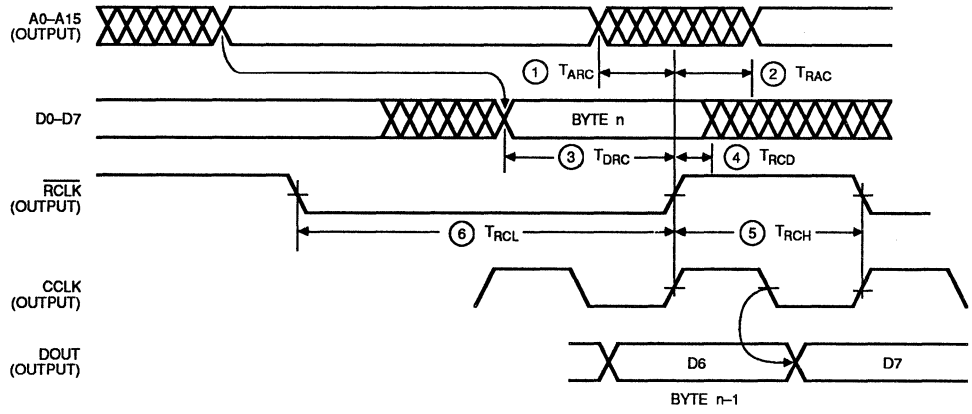
Note: Timing is measured at 0.5 V_{CC} levels with 50 pF output load.



Switching Characteristics — Programming — Master Mode

SYMBOL	DESCRIPTION		-33		-50		-70		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	
t_{ARC} ①	RCLK	From address invalid		0		0		0	ns
t_{RAC} ②		To address valid		200		200		200	ns
t_{DRC} ③		To data setup	60		60		60		ns
t_{RCD} ④		To data hold	0		0		0		ns
t_{RCH} ⑤		$\overline{RCLK}$ HIGH	600		600		600		ns
t_{RCL} ⑥		$\overline{RCLK}$ LOW	4.0		4.0		4.0		μs

Notes: 1. CCLK and DOUT timing are the same as for slave mode.
2. At power up, V_{CC} must rise from 2.0 V to V_{CC} minimum in less than 10 ms.

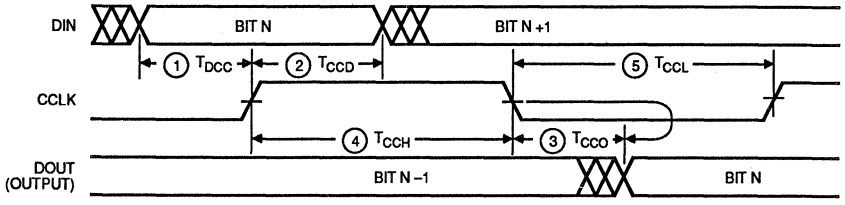


4

Switching Characteristics — Programming — Slave Mode

SYMBOL	DESCRIPTION	-33		-50		-70		UNIT
		MIN	MAX	MIN	MAX	MIN	MAX	
t_{CCO} ③	CCLK to DOUT		65		65		65	ns
t_{DCC} ①	CCLK DIN setup	0		0		0		ns
t_{CCD} ②	CCLK DIN hold	40			40		40	ns
t_{CCH} ④	CCLK HIGH time	0.25		0.25		0.25		μs
t_{CCL} ⑤	CCLK LOW time	0.25	5.0	0.25	5.0	0.25	5.0	μs
F_{CC}	CCLK frequency		2		2		2	MHz

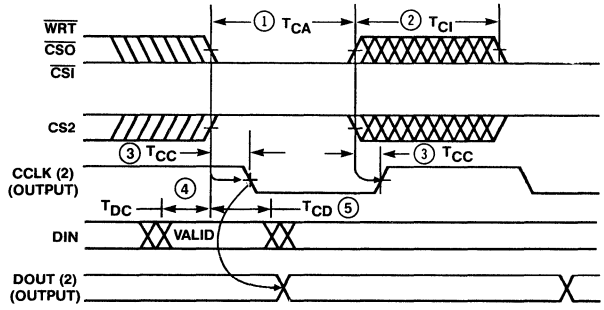
Note: Configuration must be delayed at least 40 ms after V_{CC} minimum.



Switching Characteristics — Programming — Peripheral Mode

SYMBOL	DESCRIPTION		-33		-50		-70		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	
t_{CA} ①	Controls[1] ($\overline{CS0}$, $\overline{CS1}$, CS2, $\overline{WRT}$)	Active (last active input to first inactive)	0.25	5.0	0.25	5.0	0.25	5.0	μs
t_{CI} ②		Inactive (first inactive input to last active)	0.25		0.25		0.25		μs
t_{CCC} ③		CCLK[2]		75		75		75	ns
t_{DC} ④		DIN setup	35		35		35		ns
t_{CD} ⑤		DIN hold	5		5		5		ns

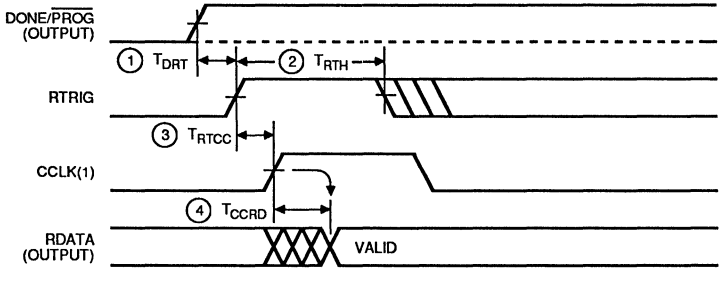
Notes: 1. Peripheral mode timing determined from last control signal of the logical AND of ($\overline{CS0}$, $\overline{CS1}$, CS2, $\overline{WRT}$) to transition to active or inactive state.
2. CCLK and DOUT timing are the same as for slave mode.
3. Configuration must be delayed at least 40 ms after V_{CC} minimum.



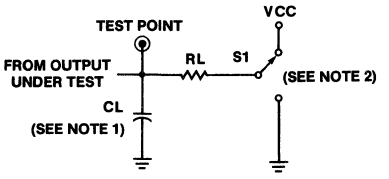
Switching Characteristics — Program Readback

SYMBOL	DESCRIPTION		-33		-50		-70		UNIT
			MIN	MAX	MIN	MAX	MIN	MAX	
t_{DRT} ①	RTRIG	PROG setup	300		300		300		ns
t_{RTH} ②		RTRIG HIGH	250		250		250		ns
t_{RTCC} ③	CCLK	RTRIG setup	100		100		100		ns
t_{CCRD} ④		RDATA delay		100		100		100	ns

Notes: 1. CCLK and DOUT timing are the same as for slave mode.
2. DONE/PROG output/input must be HIGH (device programmed) prior to a positive transition of RTRIG (M0).



Switching Test Load



Note: CL includes probe and jig capacitance.

CL = 50 pF
RL = 1K

Design Aids

Designing with the Logic Cell Array is similar to using conventional MSI elements or gate array macrocells. The first step is to partition the desired logic design into Logic Blocks and I/O blocks, usually based on shared input variables or efficient use of flip-flop and combinatorial logic. Following a plan for placement of the blocks, the design information may be entered using the interactive Graphic Design Editor. The design information includes both the functional specifications for each block and a definition of the interconnection networks. A macrocell library provides a simplified entry of commonly-used logic functions. As an alternative to interactive block placement and configuration, a schematic may be created using elements from the macrocell library. Automatic placement and routing is available for either method of design entry. After routing the interconnections, various checking stages and processing of that data are performed to ensure that the design is correct. Design changes may be implemented in minutes. The design file is used to generate the programming data which can be downloaded directly into an LCA in the target system and operated. The program information may be used to program PROM, EPROM or ROM devices, or stored in some other media as needed by the final system.

Design verification may be accomplished by using the XLINX XACTOR™ In-Circuit Emulation System directly in the target system and/or the P-Silos™ logic simulator.

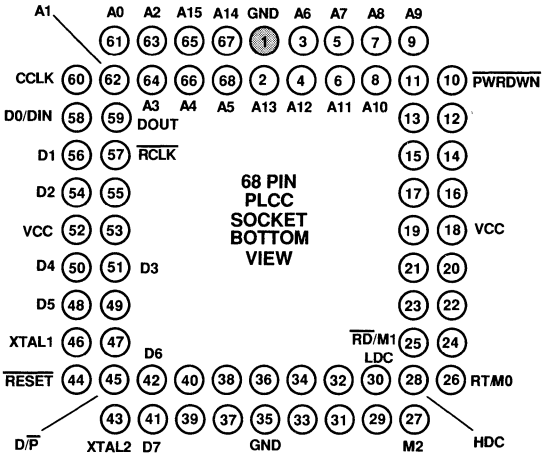
Recommended Sockets

The following sockets, with matching hole patterns, are available for PLCC devices.

	DESCRIPTION	VENDOR	PART NUMBER
68-pin	PCB solder tail, tin plate	AMP	821574-1
	Surface mount, tin plate	AMP	821542-1
	PCB solder tail, tin plate	Burndy*	QILE68P-410T
	PCB solder tail, tin plate	Midland-Ross*	709-2000-068-4-1-1
	PCB solder tail, tin plate	Methode*	213-068-001
	Surface mount, tin plate	Methode*	213-068-002
84-pin	PCB solder tail, tin plate	AMP	821573-1
	Surface mount, tin plate	AMP	821546-1
	PCB solder tail, tin plate	Burndy*	QILE84P-410T
	PCB solder tail, tin plate	Midland-Ross*	709-2000-084-4-1-1
	PCB solder tail, tin plate	Methode*	213-084-001
	Surface mount, tin plate	Methode*	213-084-002

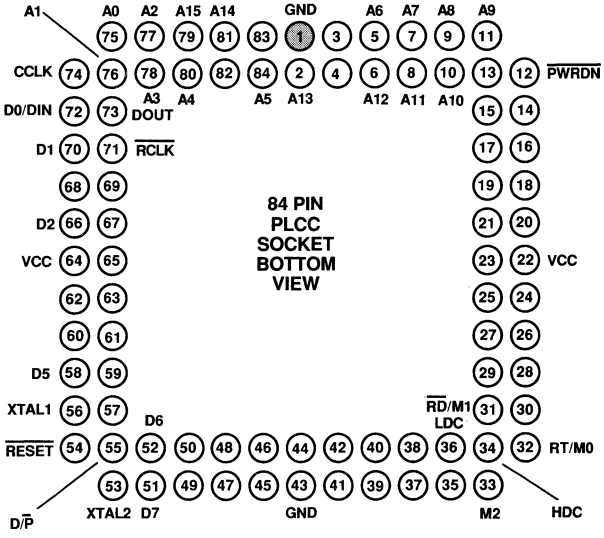
* Sockets will plug into pin-grid array (PGA) wire-wrap sockets for breadboard use.

M2064/18 PLCC SOCKET PIN ASSIGNMENT
WIRING REFERENCE. BOTTOM VIEW.



FOR EASE OF WIRING, AND PIN IDENTIFICATION, THE BOTTOM VIEW OF THE PLCC IS SHOWN ALONG WITH KEY PIN ASSIGNMENTS, SUCH AS ADDRESS, DATA, MODE, POWER AND CRYSTAL OSCILLATIOR INPUTS.

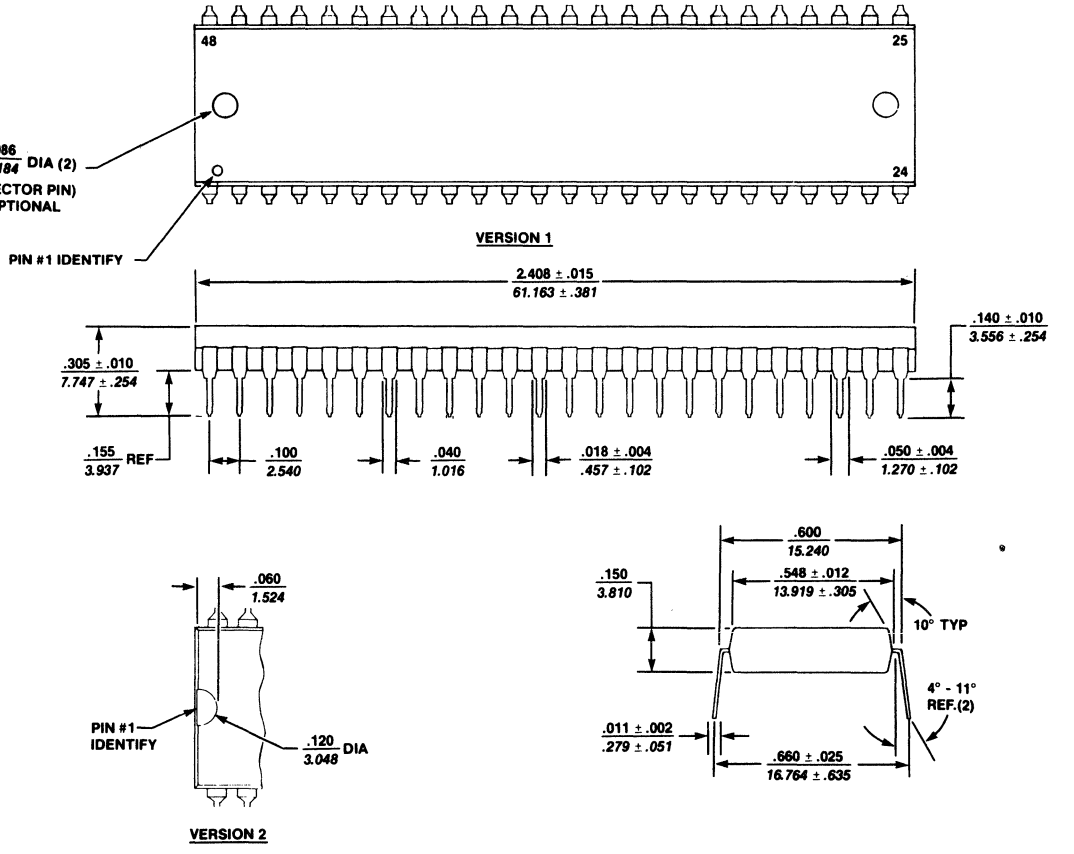
M2018 PLCC SOCKET PIN ASSIGNMENT
WIRING REFERENCE. BOTTOM VIEW.



FOR EASE OF WIRING, AND PIN IDENTIFICATION, THE BOTTOM VIEW OF THE PLCC IS SHOWN ALONG WITH KEY PIN ASSIGNMENTS, SUCH AS ADDRESS, DATA, MODE, POWER AND CRYSTAL OSCILLATIOR INPUTS.

Package Drawing

48N Molded DIP
(9/16"x2 13/32")



UNLESS OTHERWISE SPECIFIED:
ALL DIMENSIONS MIN.-MAX. IN INCHES
ALL DIMENSIONS MIN.-MAX. IN MILLIMETERS
ALL TOLERANCES ARE ± .007 INCHES

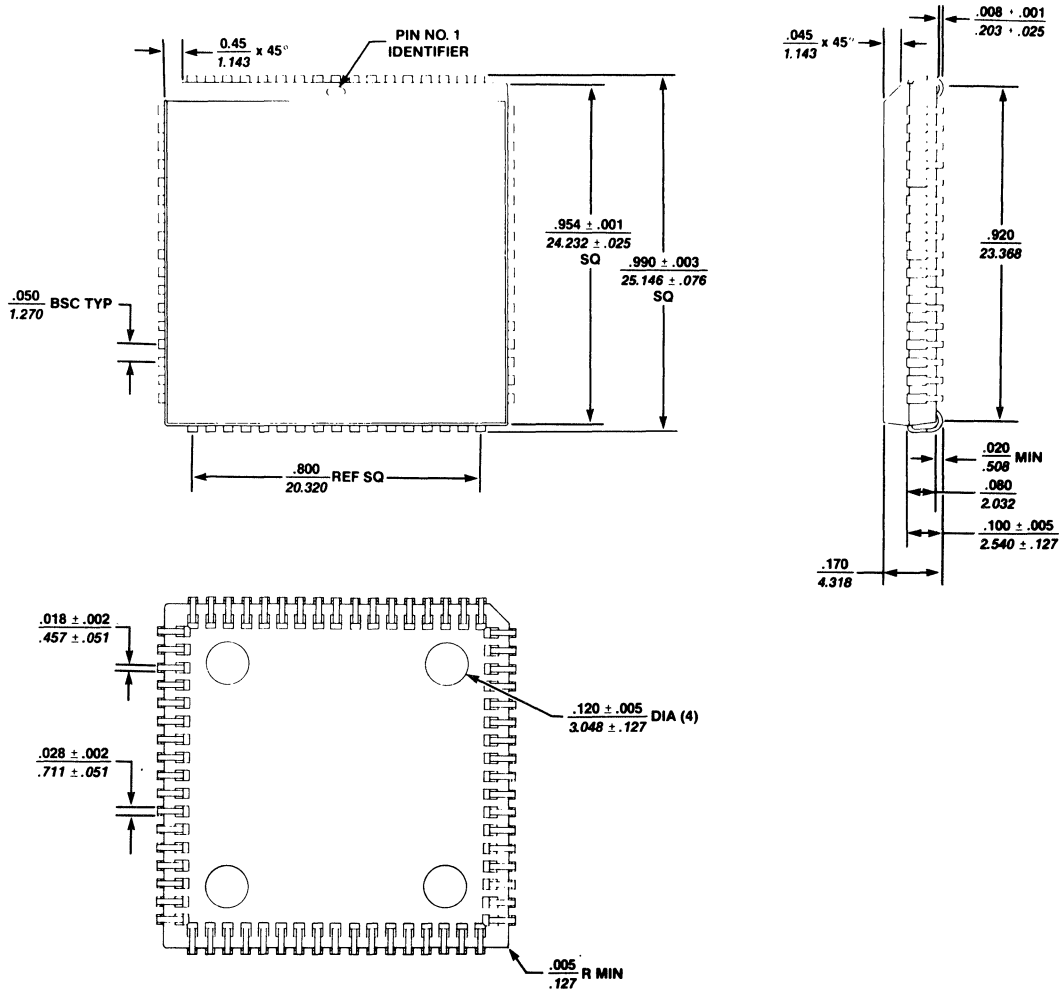
Notes:

1. Lead material tolerances are for tin plate finish only. Solder dip finish adds 2-10 mils thickness to all lead tip dimensions.
2. Both version 1 and version 2 configurations are manufactured interchangeably.
3. Ejector pin marks on version 1 are optional.

10749A

Package Drawing

68NL Molded Chip Carrier
(.950"x.950")

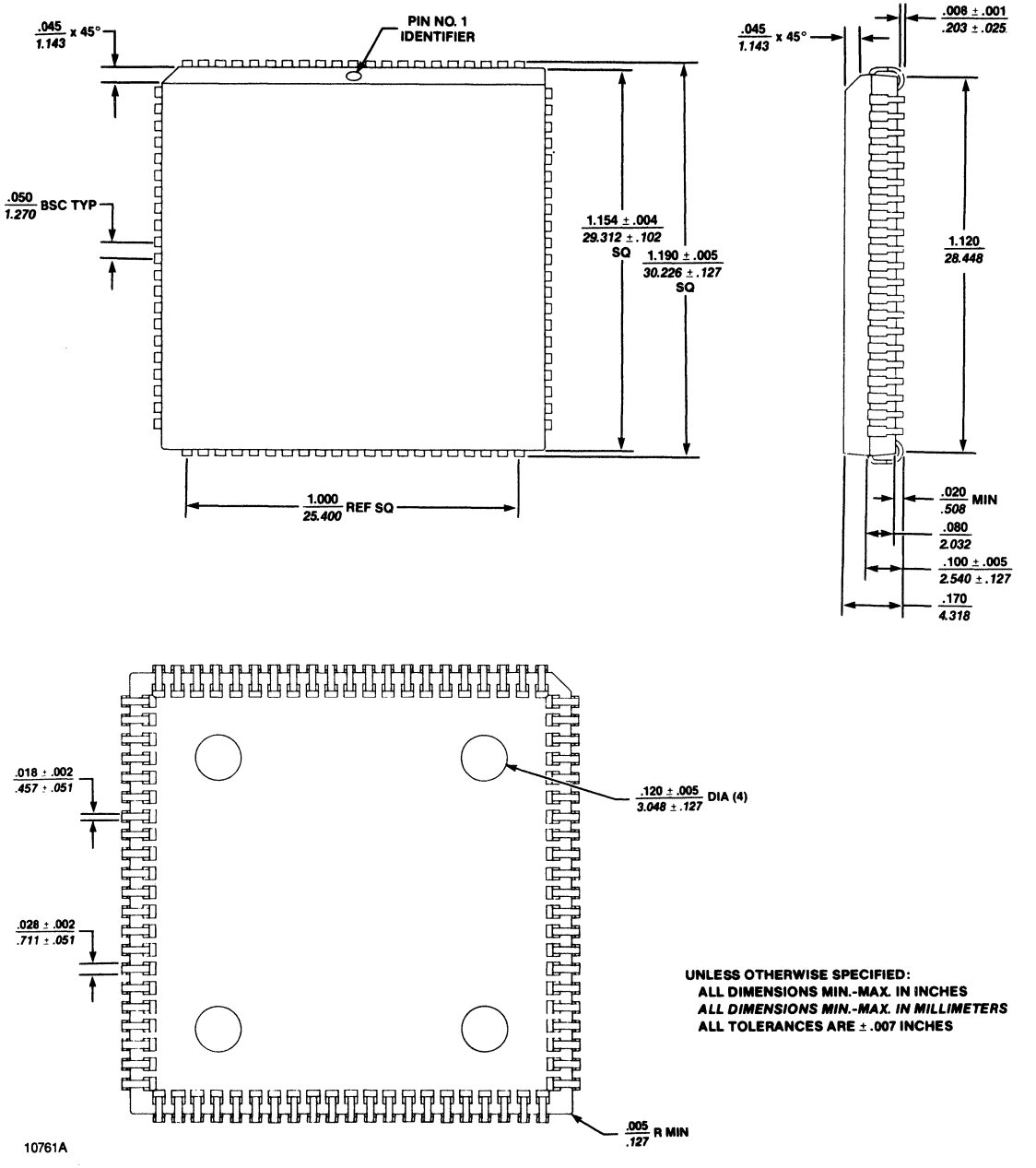


UNLESS OTHERWISE SPECIFIED:
ALL DIMENSIONS MIN.-MAX. IN INCHES
ALL DIMENSIONS MIN.-MAX. IN MILLIMETERS
ALL TOLERANCES ARE ± .007 INCHES

10760A

Package Drawing

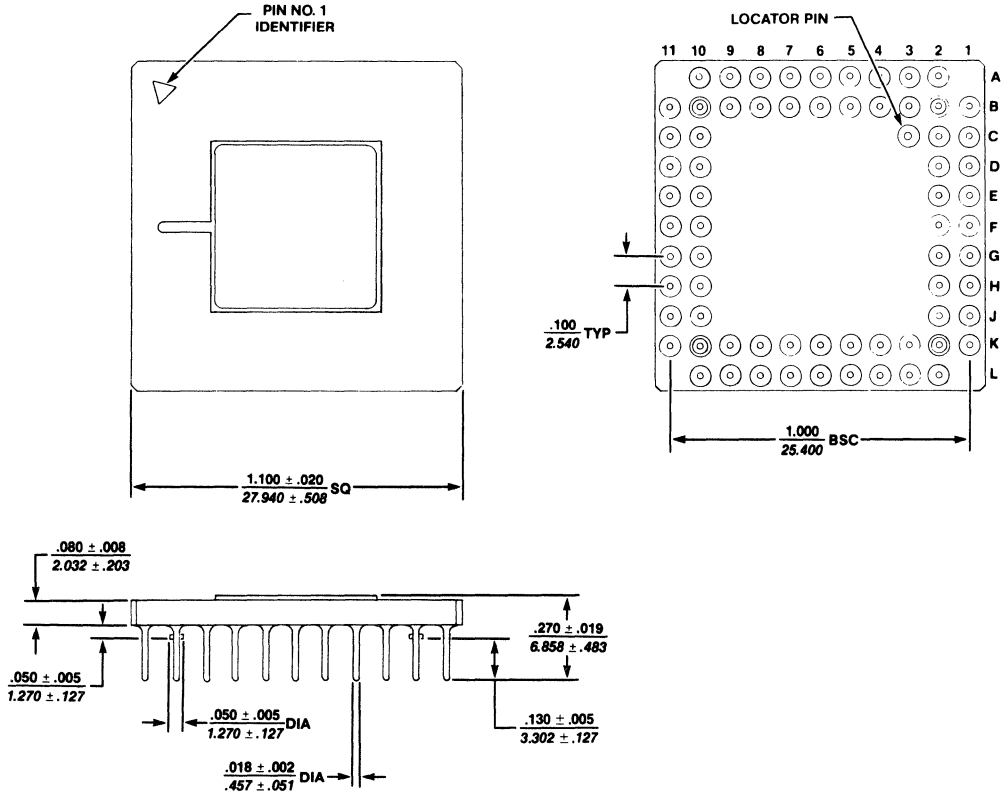
84NL Plastic Leaded Chip Carrier
(1.154" x 1.154")



UNLESS OTHERWISE SPECIFIED:
ALL DIMENSIONS MIN.-MAX. IN INCHES
ALL DIMENSIONS MIN.-MAX. IN MILLIMETERS
ALL TOLERANCES ARE ± .007 INCHES

Package Drawing

68P Ceramic Pin Grid Array

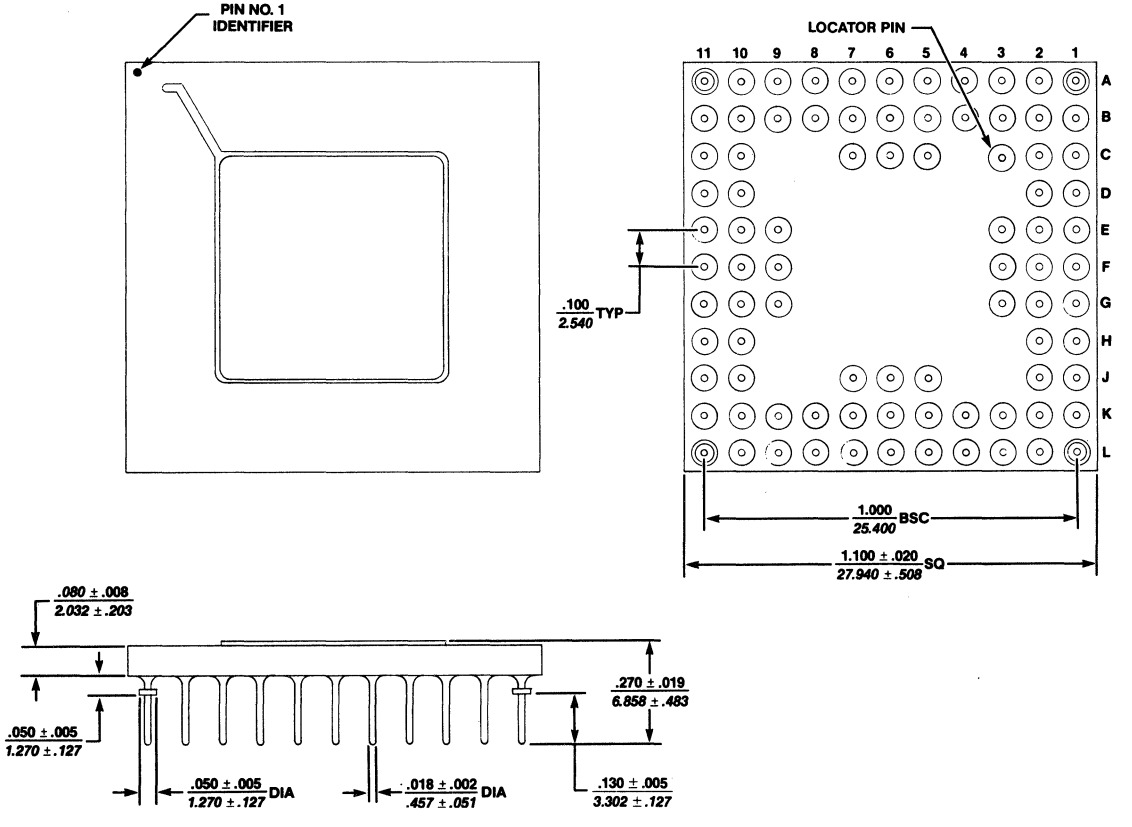


UNLESS OTHERWISE SPECIFIED:
ALL DIMENSIONS MIN.-MAX. IN INCHES
ALL DIMENSIONS MIN.-MAX. IN MILLIMETERS
ALL TOLERANCES ARE ± .007 INCHES

10765A

Package Drawings

84P Ceramic Pin Grid Array



UNLESS OTHERWISE SPECIFIED:
ALL DIMENSIONS MIN.-MAX. IN INCHES
ALL DIMENSIONS MIN.-MAX. IN MILLIMETERS
ALL TOLERANCES ARE ± .007 INCHES

10766A

3000 SERIES FAMILY OF PROGRAMMABLE GATE ARRAYS

PRELIMINARY

DISTINCTIVE CHARACTERISTICS

- Second generation user-programmable gate array
- Flexible array architecture
- High performance (50, 70 MHz)
- Improved interconnection resources
- Density of up to 9000 gates
- 100% factory pre-tested
- Selectable configuration modes
- 100% compatibility with AMD PGA development tools
- Standard PROM file interface
- Off the shelf availability

GENERAL DESCRIPTION

The AM 3000 Series Logic Cell™ Array (LCA) is a high-performance, second generation user-programmable gate array. The array contains three types of configurable elements that are customized in accordance with the user-defined system design; a perimeter of Input/Output Blocks (IOBs), a core array of Configurable Logic Blocks (CLBs), and interconnection resources.

The final configuration of the three main programmable elements is determined by the user and easily implemented by AMD user-programmable gate array design tools.

AMD's development tools let users produce a complete design, from schematic capture through device customization, on an IBM PC-AT or compatible computer. LCA macro libraries and interface software are also available to support schematic capture and simulation on popular CAE workstations.

4

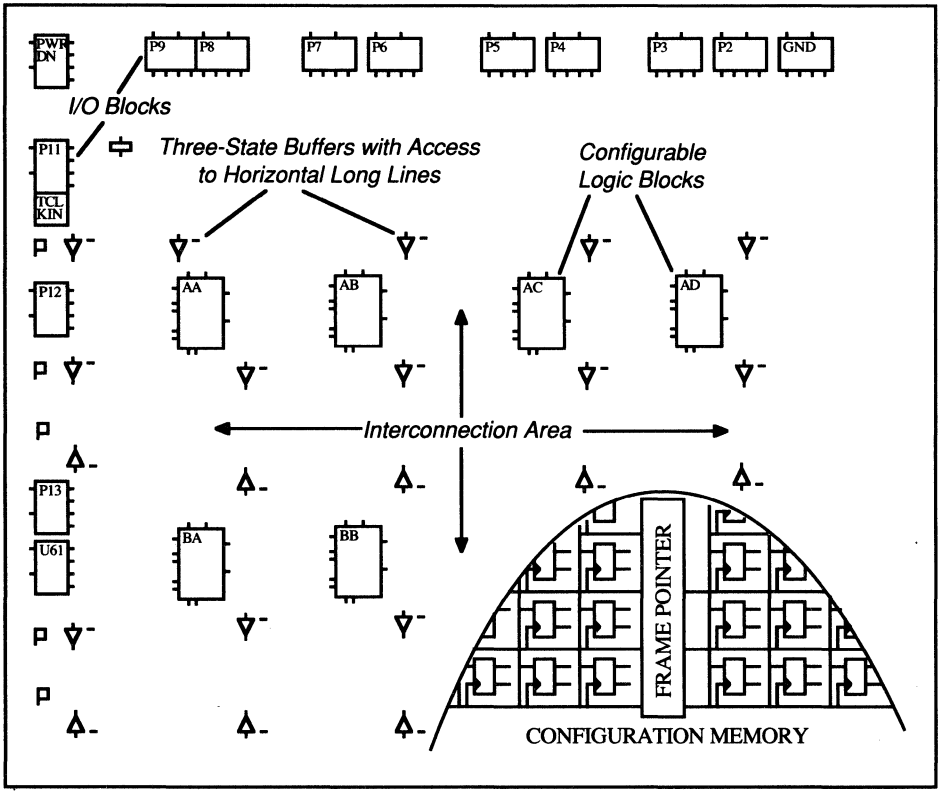
PRODUCT SELECTOR GUIDE

Basic Array	Logic Capacity (usable gates)	Configurable Logic Blocks	User I/Os	Program Data (bits)
M3020	2000	64	64	14779
M3090	9000	320	144	64160

Publication #	Rev.	Amendment
10642	A	/0

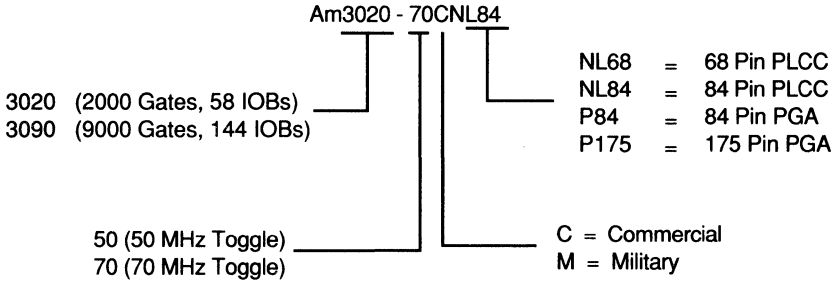
Issue Date: June 1988

BLOCK DIAGRAM



ORDERING INFORMATION

Further information is available from AMD franchised distributors or from the nearest AMD sales representative. Part numbers are composed as follows:



ARCHITECTURE

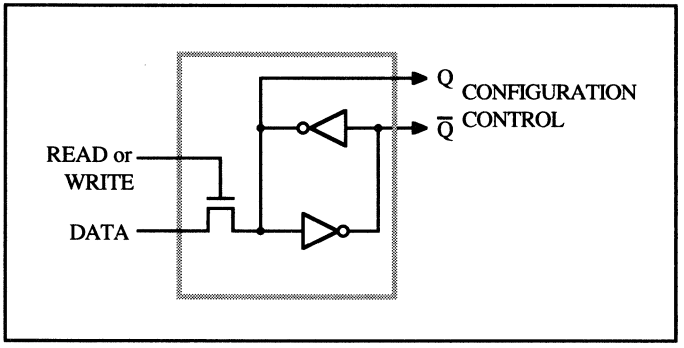
Functional Description

The perimeter of configurable IOBs provides a programmable interface between the internal logic array and the device package pins. The array of CLBs performs user-specified logic functions. The interconnection are programmed to form networks, carrying logic signals among blocks. This is analogous to printed circuit board traces connecting MSI/SSI packages.

The logic functions of these blocks are determined by programmed look-up tables. Functional options are performed by program-controlled multiplexers. Interconnecting networks between blocks are composed of metal segments joined by program-controlled pass transistors. These LCA functions are activated by a configuration bit stream that is loaded into an internal, distributed array of configuration memory cells. The configuration bit stream is loaded into the LCA device at power-up and can be reloaded on command. The LCA device includes logic and control signals for automatic or passive configuration. Configuration data can be either bit serial or byte parallel. The XACT LCA Development System generates the configuration bit stream used to configure the LCA device. The memory loading process is independent of the user logic functions.

Configuration Memory

The static memory cell used for the LCA's configuration memory has been designed specifically for high reliability and noise immunity, ensuring integrity even under adverse conditions. Static memory provides the best combination of high density, high performance, high reliability, and comprehensive testability. As shown below, the basic memory cell consists of two CMOS inverters and a pass transistor, which is used for writing and reading cell data. The cell is only written during configuration and only read during readback. During normal operation, the pass transistor is off and does not affect the stability of the cell. This is quite different from the operation of conventional memory devices, in which the cells are frequently read and re-written.



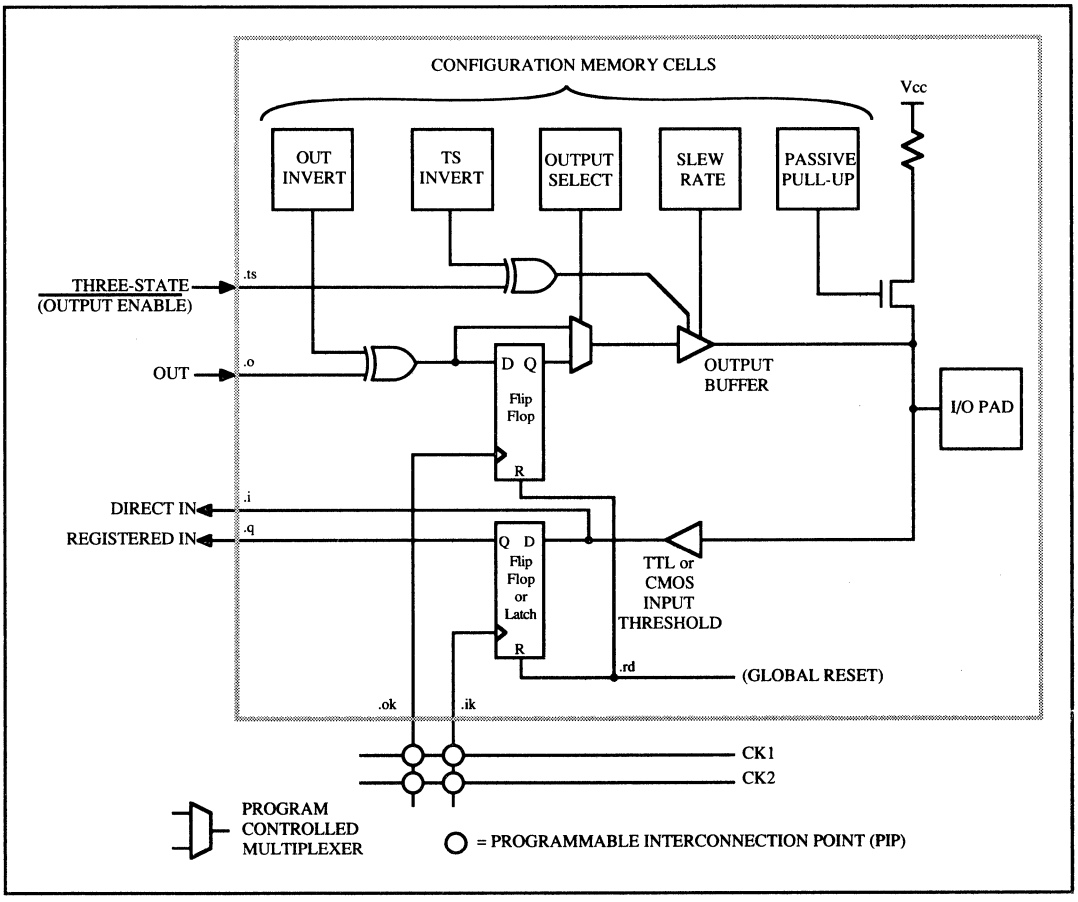
A static configuration memory cell is loaded with one bit of the configuration bit stream and controls one data selection in the LCA device. The memory cell outputs Q and ~Q use full Ground and Vcc levels and provide continuous, direct control. The additional capacitive load, together with the absence of address decoding and sense amplifiers, gives the cell high stability.

Due to the structure of the configuration memory cells, they are not affected by extreme power supply excursions or very high levels of alpha particle radiation. No soft errors have been observed in reliability testing, even in the presence of very high doses of alpha radiation.

The method of loading the configuration data is selectable. Two methods use serial data, while three use byte-wide data. The internal configuration logic uses framing information, which is embedded in the configuration data by the XACT LCA Development System, to direct memory cell loading. The serial data framing and length count preamble provide synchronous, serial, or daisy-chained compatibility with various AMD programmable gate arrays.

Input/Output BlockS

Each user-configurable IOB, shown below, provides an interface between the device's external package pin and the internal user logic. Each IOB includes both registered and direct input paths and each provides a programmable three-state output buffer that can be driven by a registered or direct output signal. Configuration options allow a choice of polarity on the output and three-state control signals, a controlled slew rate, and a high impedance pull-up. Each input circuit provides input clamping diodes for electrostatic protection, and circuits to inhibit latch-up produced by input currents.



The input buffer portion of each IOB provides threshold detection to translate external signals applied to the package pin to internal logic levels. The global input-buffer threshold of the IOBs can be programmed for TTL or CMOS voltage levels. The buffered input signal drives the data input of a storage element that can be configured as a positive edge-triggered D flip-flop, or a level-transparent latch. Clock/load signals, IOB pin .ik, can be chosen from either of two available metal lines along each die edge. I/O storage elements are reset during configuration or by the active low chip ~RESET input. Both direct input signals, from IOB pin.q, are available.

For reliable operation, inputs should have transition times less than 100 ns and should not be left undriven, or floating. Unused CMOS input-pin circuits can be at threshold and produce oscillations. This produces additional power dissipation and system noise. A typical hysteresis of about 50 mV reduces input noise sensitivity . Each user IOB includes a programmable high impedance pull-up resistor that

can be selected by the bit stream and which provides a constant HIGH for undriven pins. Although the LCA device provides circuitry for input protection against electrostatic discharge, normal CMOS handling precautions should be observed.

Loop delays for the IOB and logic block flip-flops are about 3 ns. This increases reliability, especially for asynchronous clock and data conditions. Short loop delays minimize the probability of a metastable condition, which can result from assertion of the clock during data transitions. Because of the short loop delay in LCA devices, the flip-flops can be used to synchronize external signals applied to the device. Once synchronized in the IOB, the signals can be used internally without regard to their clock-relative timing, except as it applies to the internal logic and routing path delays.

Output buffers of the IOBs provide CMOS-compatible 4 mA source-or-sink drive for high fan-out CMOS or TTL compatible signal levels. The network driving IOB pin .o becomes the registered or direct data source for the output buffer. The three-state control signal, IOB pin .ts, can control output activity. An open-drain type output can be obtained by using the same signal for driving the output and three-state signal nets, so that the buffer output is enabled only for a LOW.

The configuration memory cells, shown in the figure above, control the optional output register and logical signal inversion, as well as the three-state and slew rate configuration bits. A choice of two clocks is available on each die edge. All user inputs are programmed for TTL or CMOS thresholds.

The IOB includes input and output storage elements and the following I/O options selected by configuration memory cells.

- Logical **inversion of the output** is controlled by one configuration bit per IOB.
- Logical **three-state control** of each IOB output buffer is determined by the states of the configuration data bits that turn the buffer on/off or select the output buffer three-state control interconnection, pin .ts. When this IOB output control signal is HIGH, or logic 1, the buffer is disabled and the package pin is high impedance. Inversion of the buffer three-state control logic sense, output enable, is controlled by an additional configuration data bit.
- **Direct or registered output** is selectable for each IOB . The register uses a positive-edge, clocked flip-flop. The clock source, IOB pin .ok, can be supplied by either of two metal lines, which are available along each die edge. Each of these lines is driven by an invertible buffer.
- Increased **output transition speed** can be selected to satisfy critical nets. Slower transitions reduce capacitive load peak currents of non-critical outputs and minimize system noise.
- A high impedance **pull-up resistor** can be used to prevent floating, unused inputs.

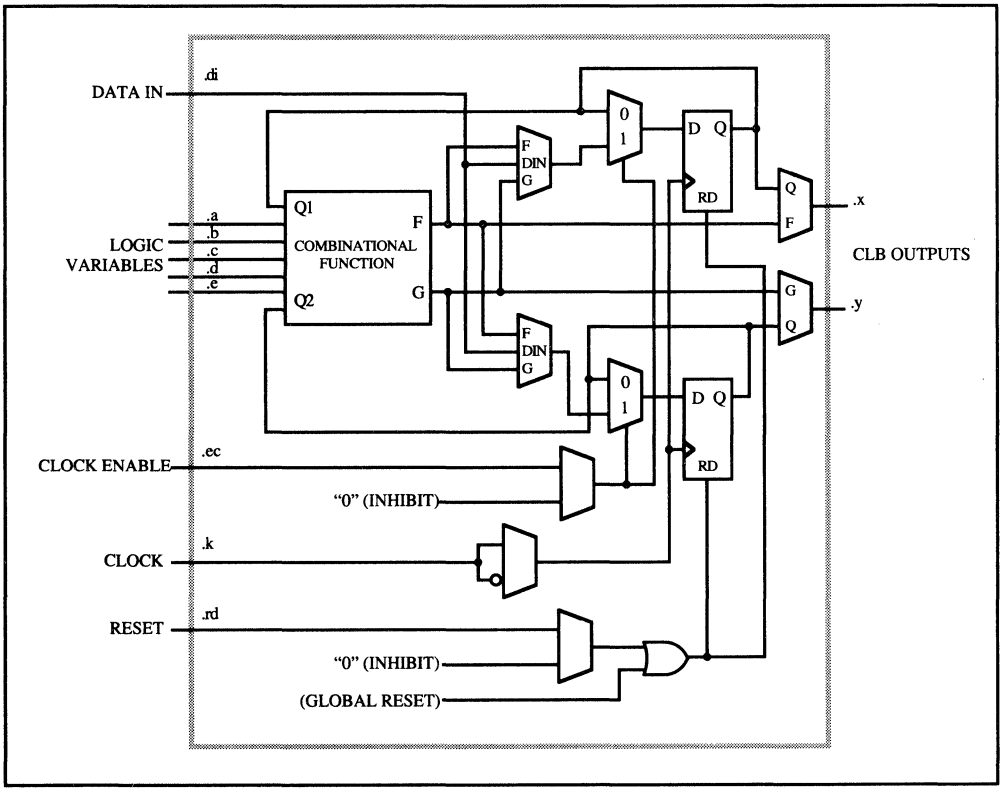
The following table is summarizes the I/O options.

INPUTS	OUTPUTS
Direct	Direct/registered
Flip-flop/latch	Inverted/true
CMOS/TTL threshold (chip inputs)	Full speed/slew limited
Pull-up resistor	Optional three-state control

Configurable Logic Blocks

CLBs are the functional elements from which the user's logic is constructed. The logic blocks are arranged in a matrix within the perimeter of IOBs. The 3020 has 64 such blocks arranged in eight rows and eight columns. The XACT LCA Development System compiles the configuration data, which defines the operation and interconnection of each block. Users can define CLBs and their interconnecting networks by automatic translation from a schematic capture logic diagram or, optionally, by installing library or user macros.

Each CLB has a combinational logic section, two flip-flops, and an internal control section. As shown in the following figure, there are five logic inputs (.a, .b, .c, .d, and .e), a common clock input (.k), an asynchronous direct reset input (.rd), and a clock enable (.ec). All can be driven from the interconnection resources adjacent to the blocks. Each CLB also has two outputs (.x and .y) that can drive interconnection networks.

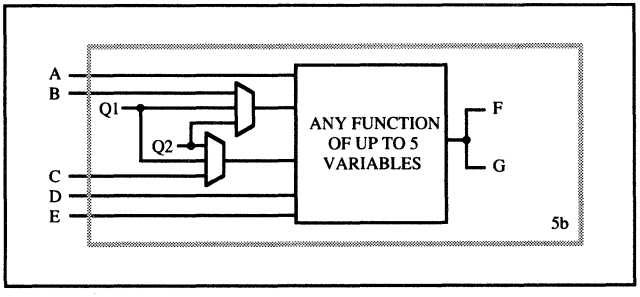


Data input for either flip-flop within a CLB is supplied from the F or G function outputs of the combinational logic, or the direct data input, .di. Both flip-flops in each CLB share the active HIGH

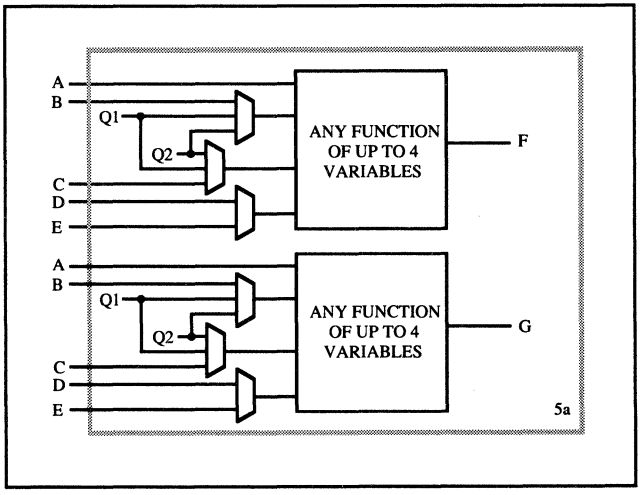
asynchronous reset (.rd), which is dominant over clocked inputs. All flip-flops are reset by the active LOW chip input, ~RESET, or during the configuration process. The flip-flops share the clock enable (.ec), which, when LOW, recirculates the present states of the flip-flops and inhibits response to the data-in or combinational function inputs on a CLB. The user can enable these control inputs and select their sources. The user also can select the clock net input (.k) and its active sense in each logic block. This programmable inversion eliminates the need to route both phases of a clock signal throughout the device. Flexible routing allows use of common or individual CLB clocking.

The combinational logic portion of the CLB uses a 32-by-1 look-up table to perform Boolean functions. Variables selected from the five logic inputs and two internal block flip-flops are used as table address inputs. The combinational propagation delay through the network is independent of the logic function generated and is spike free for changes in single input variables.

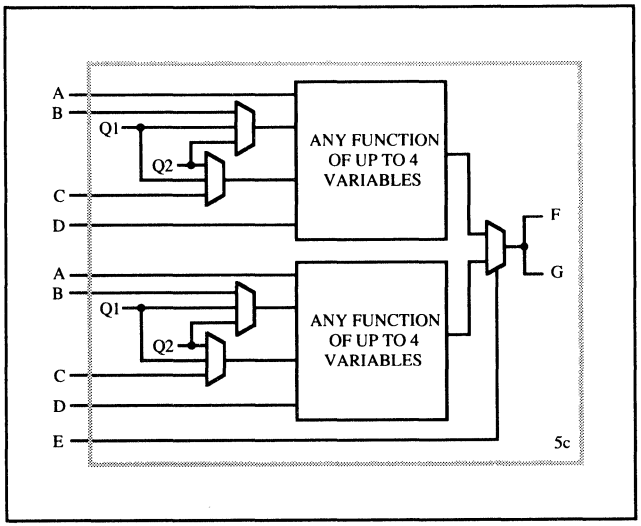
This technique can generate a single function of five variables, as shown below.



It can also generate any two logic functions of up to four variables each.

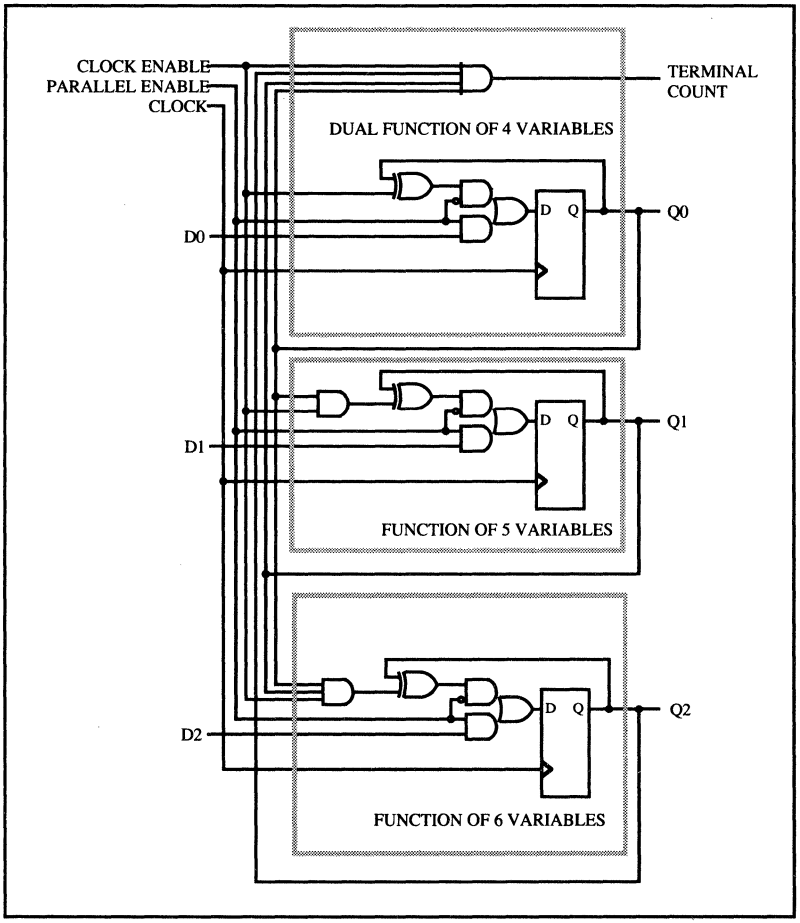


It can also generate some functions of seven variables, as shown in the next figure.



The partial functions of six or seven variables are generated by the input variable,.e, which dynamically selects between two functions of four different variables. For the two functions of four variables each, the independent results,F and G, can be used as data inputs to either flip-flop or either logic block output. For the single function of five variables and merged functions of six or seven variables, the F and G outputs are identical. Symmetry of the F and G functions and the flip-flops helps optimize the routing of the networks connecting the logic blocks and IOBs.

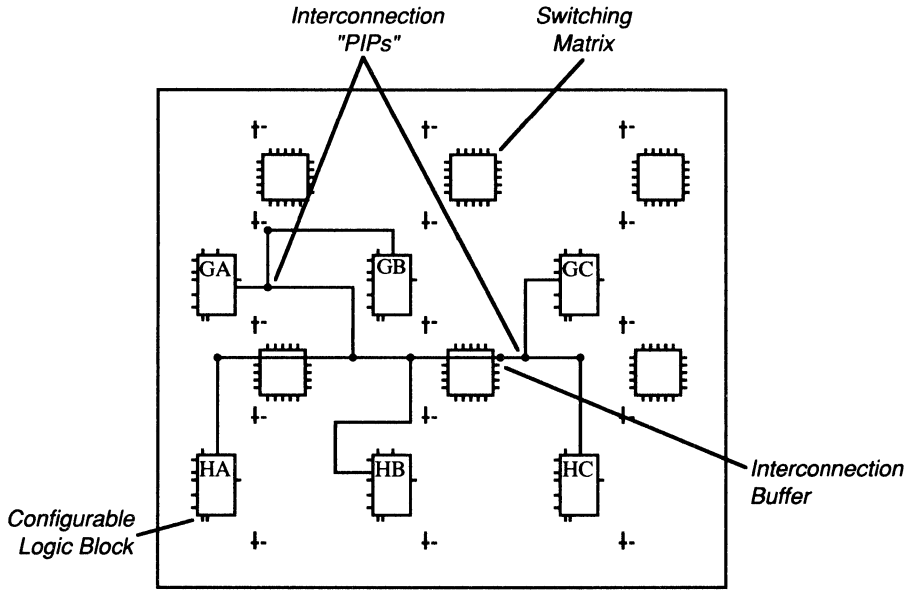
The next figure shows a modulo 8 binary counter with parallel enable. It uses one CLB of each type.



INTERCONNECTIONS

Programmable Interconnections

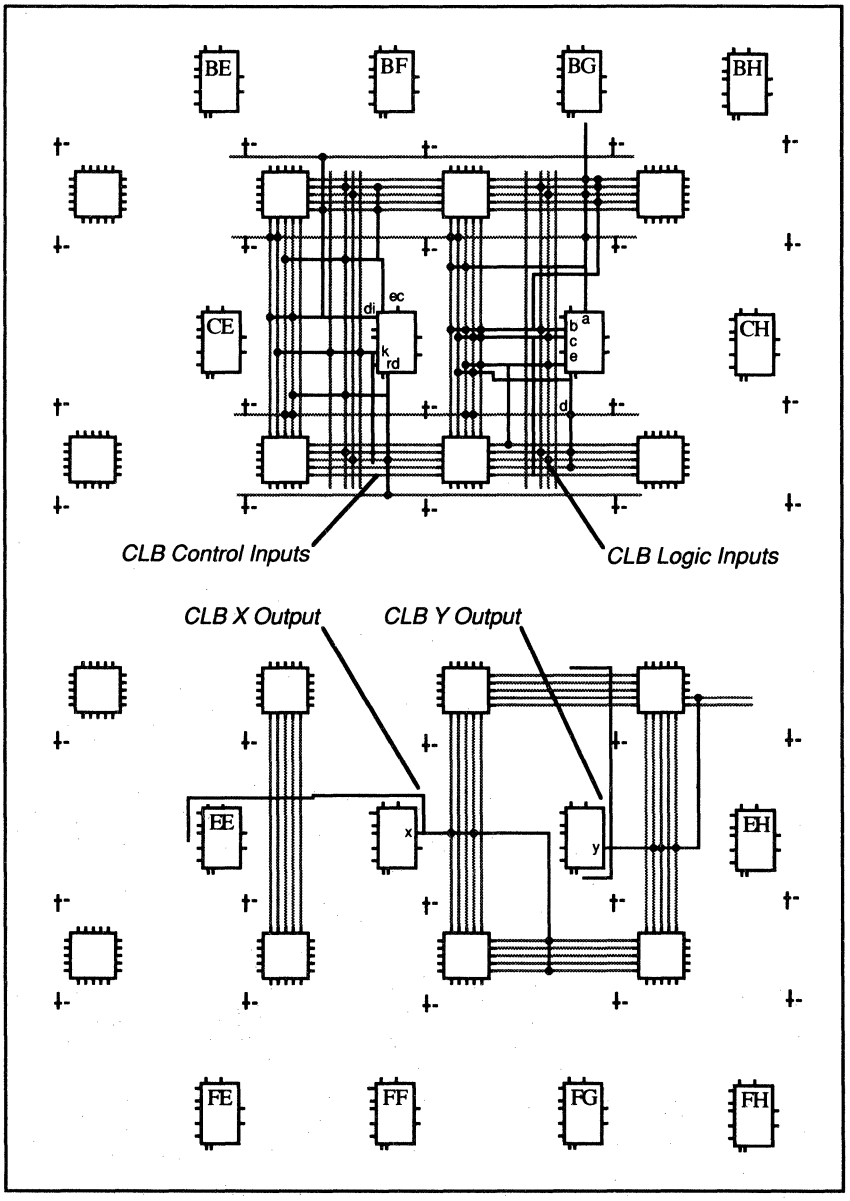
Programmable interconnection resources in the LCA device provide routing paths to connect inputs and outputs of the I/O and logic blocks into logical networks. Interconnections between blocks are composed of two-layer grid of metal segments. Specially designed pass transistors, each controlled by a configuration bit, form programmable interconnect points (PIPs) and switching matrices used to implement the necessary connections between selected metal segments and block pins. The figure below provides an example of a routed net.



The XACT LCA Development System automatically routes these interconnections. Interactive routing using Editnet can also be done to optimize the design. The inputs of the logic or IOB are multiplexers that can be programmed to select an input network from the adjacent interconnection segments.

Note: The switch connections to block inputs are usable only for input connection, and not for routing, because they are unidirectional (as are block outputs).

The following figure illustrates routing access to logic block input variables, control inputs, and block outputs.

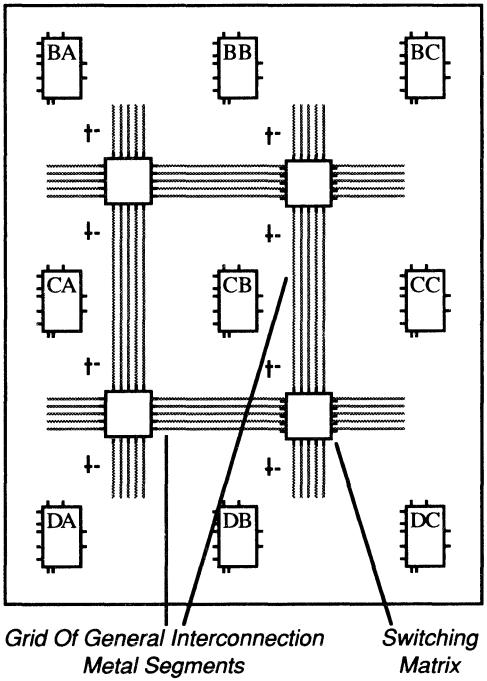

BE
BF
BG
BH
CE
CH
di
ec
k
rd
b
a
c
e
d
CLB Control Inputs
CLB Logic Inputs
CLB X Output
CLB Y Output
EE
EH
x
y
FE
FF
FG
FH

Three types of metal resources are available for network interconnections.

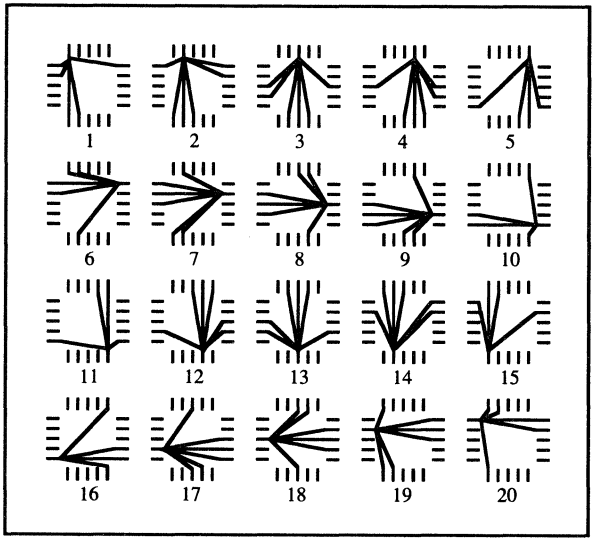
- General Purpose Interconnection
- Direct Connection
- Long Lines

General Purpose Interconnections

A general purpose interconnection, as shown below, consists of a grid of five horizontal and five vertical metal segments located between the rows and columns of logic and I/O Blocks. These segments can be connected through switch matrices to form networks for CLB and IOB inputs and outputs.



Each segment is the height or width of a logic block. Switching matrices join the ends of these segments and allow programmed interconnections between the metal grid segments of adjoining rows and columns. The switches of an unprogrammed device are all non-conducting. The connections through the switch matrix can be made by automatic routing, or by using Editnet to select the desired pairs of matrix pins that are to be connected or disconnected. The legitimate switching matrix combinations for each pin are shown in the next figure.

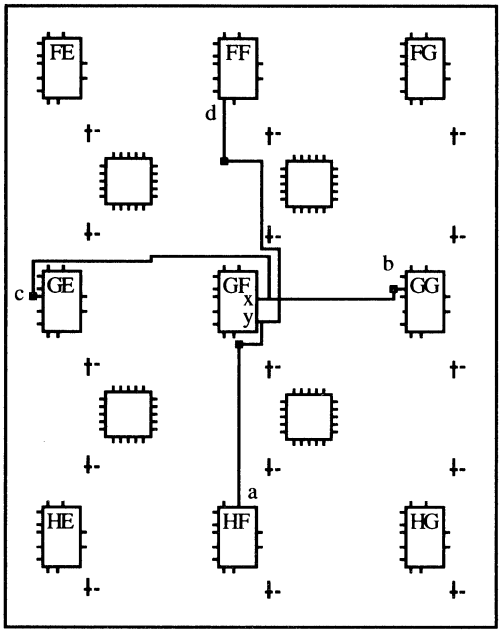


Special buffers in the general interconnection areas provide periodic signal isolation and restoration, thus improving performance of lengthy nets. The interconnection buffers can propagate signals in either direction on a general interconnection segment. These bidirectional buffers are above and to the right of the switching matrices. The other PIPs adjacent to the matrices are gateways to and from long lines.

The XACT LCA Development System automatically defines the buffer direction based on the location of the interconnection network source. The delay calculator of the XACT LCA Development System automatically calculates and displays the block, interconnection, and buffer delays for the selected paths. It can also generate the simulation net list with a worst-case delay model.

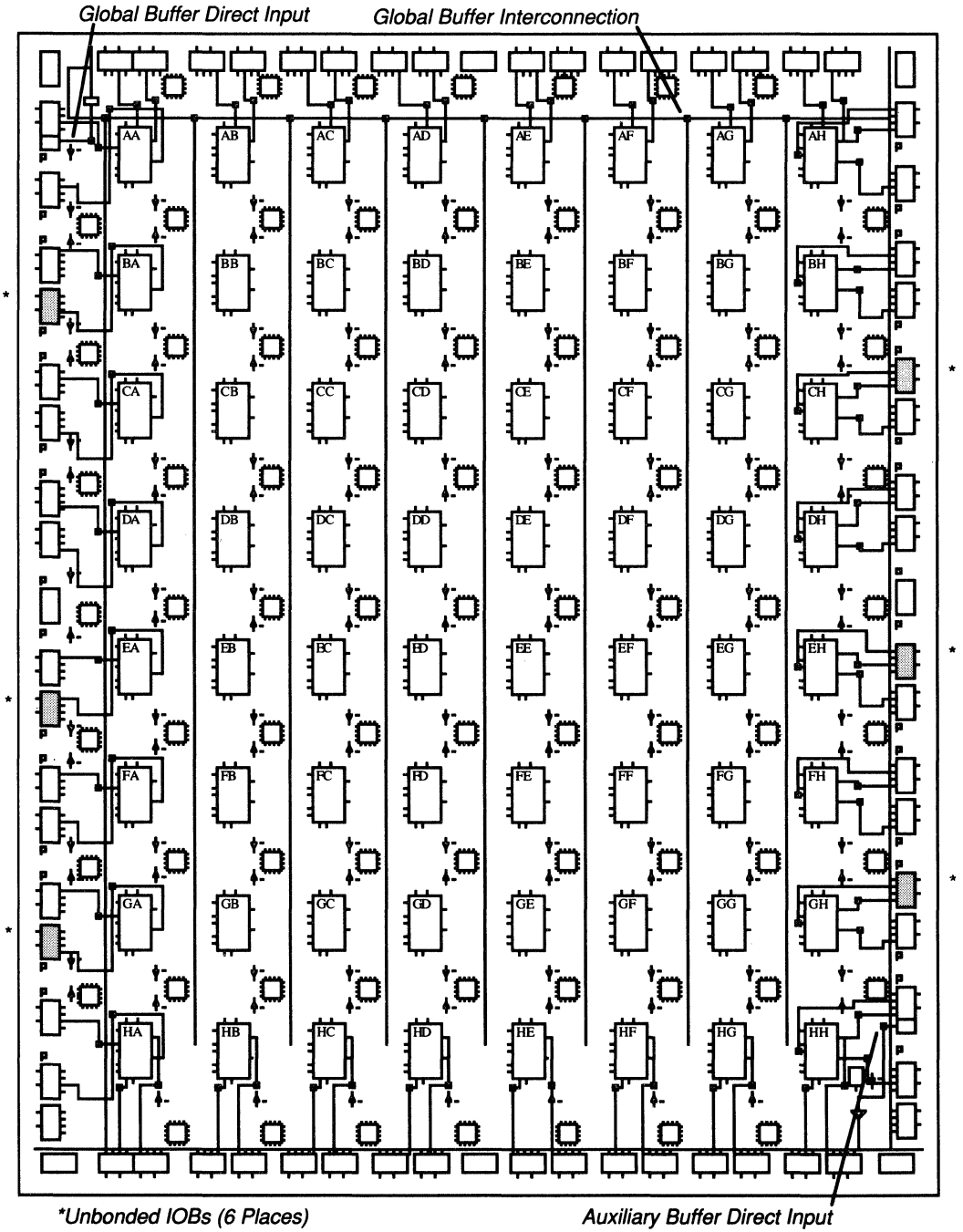
Direct Interconnections

A direct interconnection, shown below, provides the most efficient implementation of networks between adjacent logic or IOBs. The .x and .y outputs of each CLB have single contact, direct access to inputs of adjacent CLBs.



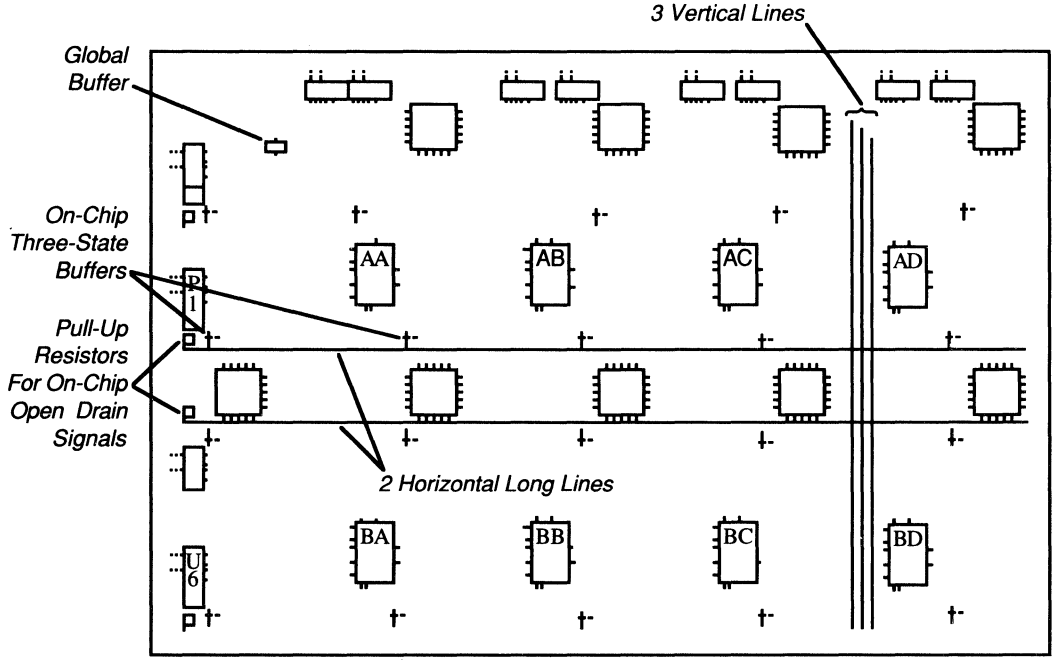
Signals routed from block to block by direct interconnection show minimum interconnection propagation and use no general interconnection resources. For each CLB, the .x output can be connected directly to the .b input of the CLB to its right, and to the .c input of the CLB to its left. The .y output can use a direct interconnection to drive the .d input of the block above, and the .a input of the block below.

Direct interconnection should be used to maximize the speed of high performance portions of logic. Where logic blocks are adjacent to IOBs, a direct connection is provided alternately to the IOB inputs (.i) and outputs (.o) of the left, top, and bottom edges of the die. The right edge provides alternate direct input and output of CLBs and IOBs. Direct interconnections of IOBs and CLBs are shown in the next figure.

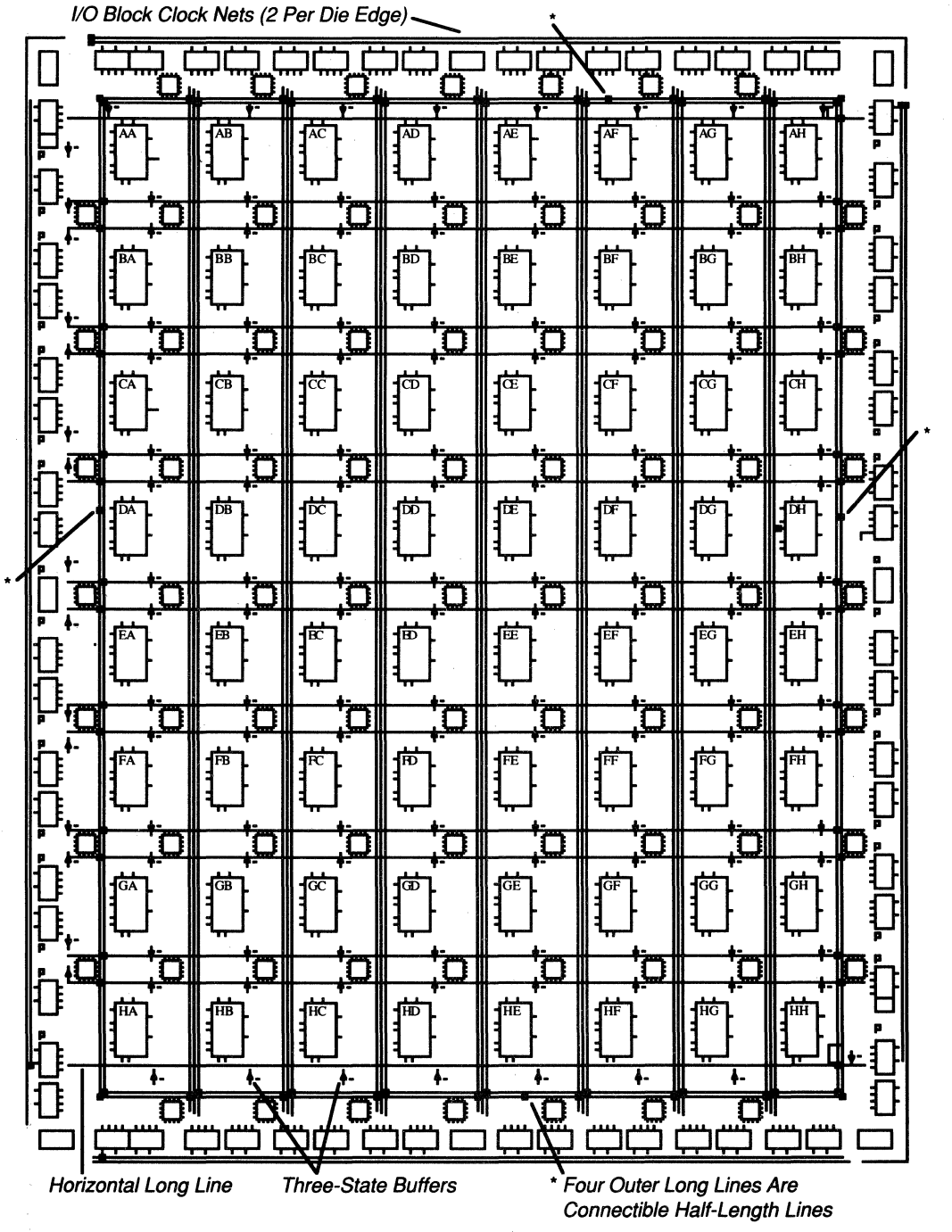

Global Buffer Direct Input
Global Buffer Interconnection
AA
AB
AC
AD
AE
AF
AG
AH
BA
BB
BC
BD
BE
BF
BG
BH
CA
CB
CC
CD
CE
CF
CG
CH
DA
DB
DC
DD
DE
DF
DG
DH
EA
EB
EC
ED
EE
EF
EG
EH
FA
FB
FC
FD
FE
FF
FG
FH
GA
GB
GC
GD
GE
GF
GG
GH
HA
HB
HC
HD
HE
HF
HG
HH
*Unbonded IOBs (6 Places)
Auxiliary Buffer Direct Input

Long Lines

Long lines, which bypass the switch matrices, are intended primarily for signals that must travel a long distance, or that must have minimum skew among multiple destinations. Long lines, shown below, run the height or width of the interconnection area. Each interconnection column has three vertical long lines, and each interconnection row has two horizontal long lines. Two additional long lines are adjacent to the outer sets of switching matrices. The outermost are connectible half-length lines.



Horizontal and vertical long lines provide high fan-out, low-skew signal distribution in each row and column. The global buffer in the upper left die corner drives a common line throughout the LCA device. The programmable interconnection of long lines is provided at the edges of the routing area. Long lines can be driven by a CLB or IOB output on a column-by-column basis. This provides a common low skew control or clock line within each column of logic blocks. Interconnections of these long lines are shown in the following figure. Isolation buffers are provided at each input to a long line and are enabled automatically by the XACT LCA Development System when a connection is made.

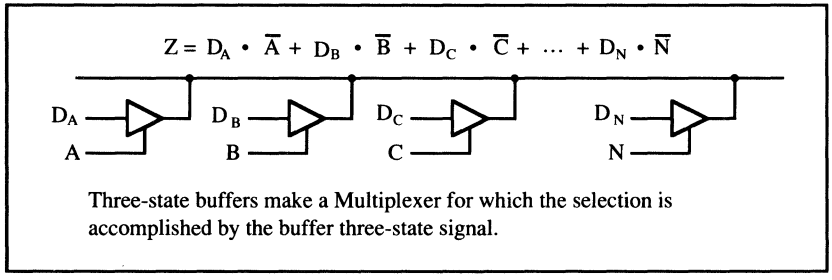

I/O Block Clock Nets (2 Per Die Edge)
*
AA
AB
AC
AD
AE
AF
AG
AH
BA
BB
BC
BD
BE
BF
BG
BH
CA
CB
CC
CD
CE
CF
CG
CH
DA
DB
DC
DD
DE
DF
DG
DH
EA
EB
EC
ED
EE
EF
EG
EH
FA
FB
FC
FD
FE
FF
FG
FH
GA
GB
GC
GD
GE
GF
GG
GH
HA
HB
HC
HD
HE
HF
HG
HH
Horizontal Long Line
Three-State Buffers
* Four Outer Long Lines Are Connectible Half-Length Lines

A buffer in the upper left corner of the LCA chip drives a global net available to all .k inputs of logic blocks. Using the global buffer for a clock signal provides a skew free, high fan-out, synchronized clock for use at any, or all, of the I/O and logic blocks. Three-state buffers let you use horizontal long lines to form wired-AND and multiplexed busses. Configuration bits for the .k input to each logic block can select this global line or another routing resource as the clock source for its flip-flops. This net can also be programmed to drive the die edge clock lines for IOB use. An enhanced speed, CMOS threshold direct access to this buffer is available at the second pad from the top of the left die edge.

A buffer in the lower right corner of the array drives a horizontal long line, which can drive programmed connections to a vertical long line in each interconnection column. This alternate buffer also is low skew and high fan-out. The network formed by this alternate buffer's long lines can be selected to drive the .k inputs of the logic blocks. The CMOS threshold, high-speed access to this buffer is at the third pad from the bottom of the right die edge.

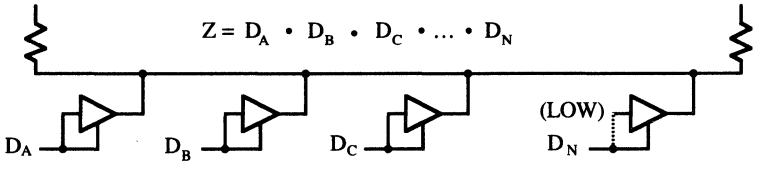
Internal Busses

A pair of three-state buffers are located adjacent to each CLB. These let logic drive the horizontal long lines. Logical operation of the three-state buffer controls lets them implement wide multiplexing functions. Any three-state buffer input can be selected to drive the horizontal long line bus by applying a low logic level on its three-state control line, as shown in the next figure.



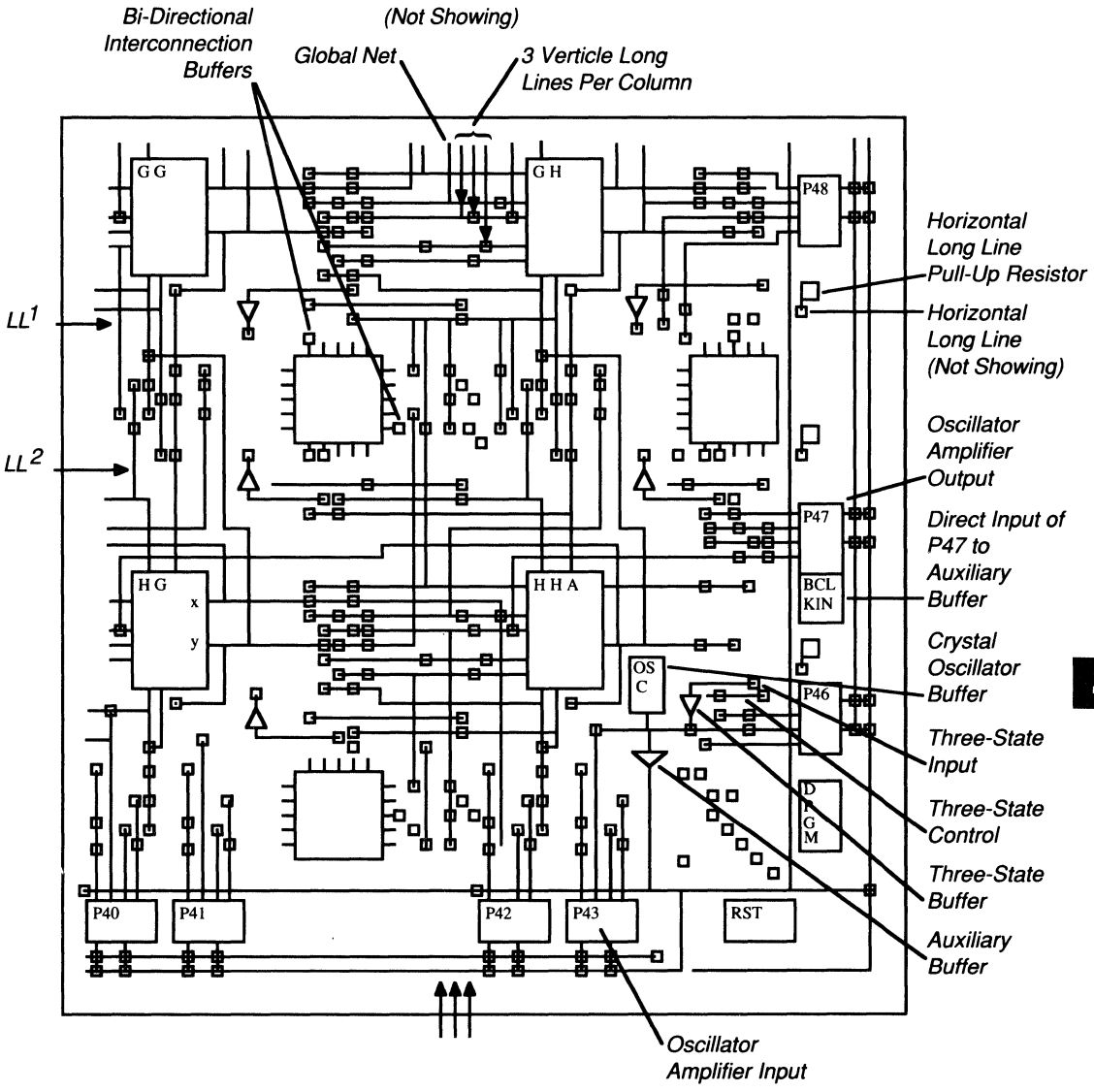
Three-state buffers make a Multiplexer for which the selection is accomplished by the buffer three-state signal.

The user must avoid contention resulting from multiple drivers with opposing logic levels. Control of the three-state input by the same signal that drives the buffer input creates an open drain wired-AND function. A logical HIGH on both buffer inputs creates a high impedance with no contention. A logical LOW enables the buffer to drive the long line low, as shown below.



Three-state buffers make a Wired-AND function. When all the three-state lines are HIGH (high impedance), the pull-up resistor(s) provide the HIGH output. The buffer inputs are driven by the control signals or a LOW.

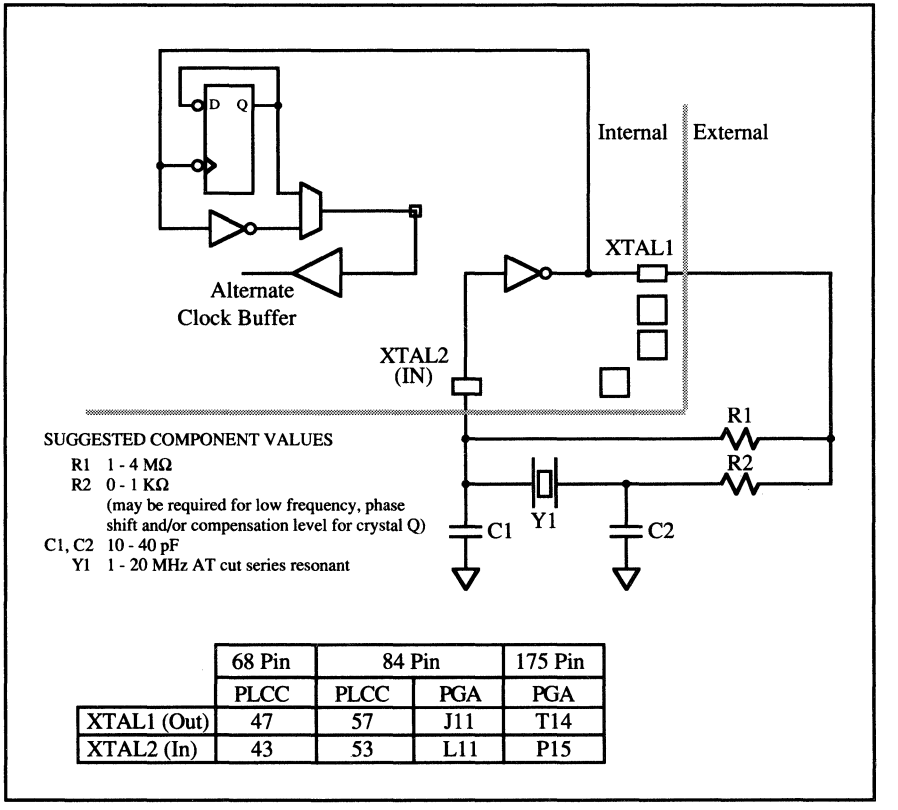
Pull-up resistors are available at each end of the long line to provide a HIGH output when all connected buffers are non-conducting. These buffers allow fast, wide gating, optimum speed, and efficient routing of high fan-out signals. The following figure shows three-state buffers, long lines, and pull-up resistors.


Bi-Directional Interconnection Buffers
(Not Showing)
Global Net
3 Verticle Long Lines Per Column
LL1
LL2
GG
GH
HG
x
y
HHA
OSC
P40
P41
P42
P43
RST
P46
P47
P48
BCL KIN
DFGM
Horizontal Long Line Pull-Up Resistor
Horizontal Long Line (Not Showing)
Oscillator Amplifier Output
Direct Input of P47 to Auxiliary Buffer
Crystal Oscillator Buffer
Three-State Input
Three-State Control
Three-State Buffer
Auxiliary Buffer
Oscillator Amplifier Input

Crystal Oscillator

The previous figure also shows the location of an internal high-speed inverting amplifier that can be used as an on-chip crystal oscillator. It is associated with the auxiliary buffer in the lower right corner of the die. When the oscillator is configured as a signal source, two special user IOBs are also configured to connect the oscillator amplifier with external crystal oscillator components, as shown below.

When activated by selecting an output network for its buffer, the crystal oscillator inverter uses two of the package pins and external components to make an oscillator. An optional divide-by-two mode is available to ensure symmetry.



	68 Pin	84 Pin		175 Pin
	PLCC	PLCC	PGA	PGA
XTAL1 (Out)	47	57	J11	T14
XTAL2 (In)	43	53	L11	P15

A divide-by-two option is available to assure symmetry. The oscillator circuit becomes active before configuration is complete so the oscillator can stabilize. Actual internal connection is delayed until completion of configuration. In the preceding figure the feedback resistor, R1, between output and input biases the amplifier at threshold. The value should be as large as practical to minimize loading the

crystal. The inversion of the amplifier, together with the R-C networks and an AT cut series resonant crystal, produce the 360 degree phase shift of the Pierce oscillator. A series resistor, R2, can be included to increase the amplifier output impedance. This may be needed for phase shift control or crystal resistance matching, or to limit the amplifier input swing to control clipping at large amplitudes.

Excess feedback voltage can be corrected by the ratio of C2/C1. The amplifier can be used from 1 MHz to one-half the specified CLB toggle frequency. Used at frequencies below 1 MHz, the amplifier may require individual characterization with respect to a series resistance. Crystal oscillators operating at frequencies above 20 MHz usually require a crystal that operates in a third overtone mode, in which the fundamental frequency must be suppressed by the R-C networks. When the oscillator inverter is not used, these IOBs and their package pins are available for general user I/O.

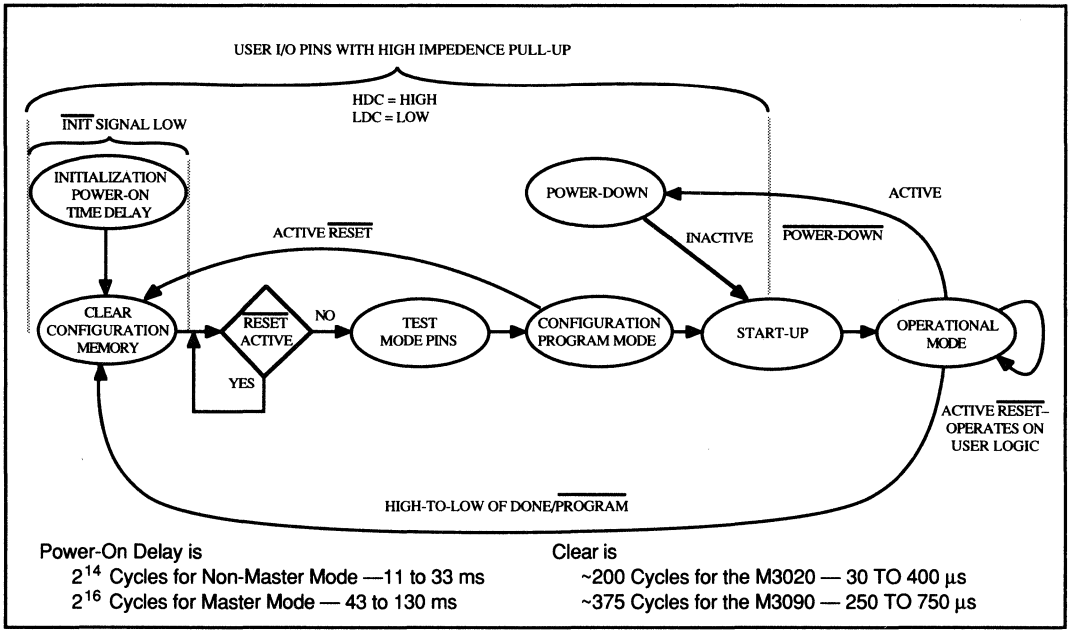
PROGRAMMING

Initialization

An internal power-on reset circuit is triggered when power is applied. When Vcc reaches the voltage at which portions of the LCA device begin to operate (2.5 to 3 V), the programmable I/O output buffers are disabled and a high impedance pull-up resistor is provided for the user I/O pins. A time-out delay is initiated to let the power supply voltage stabilize. During this time, the power-down mode is inhibited. The initialization state time-out (about 20 to 30 ms) is determined by a 14-bit counter driven by a self-generated, internal timer. This nominal 1 MHz timer is subject to variations as a result of process, temperature, and power supply from 0.5 to 1.5 MHz. As shown in the following table, five configuration modes are available, as determined by the input levels of three mode pins M0, M1, and M2.

M0	M1	M2	CLOCK	MODE	DATA
0	0	0	output	Master	Bit Serial
0	0	1	output	Master	Byte Wide (0000 up)
0	1	0	—	—	—
0	1	1	output	Master	Byte Wide (FFFF down)
1	0	0	—	—	—
1	0	1	input	Peripheral	Byte Wide
1	1	0	—	—	—
1	1	1	input	Slave	Bit Serial

In master configuration modes, the LCA device becomes the source of the configuration clock (CCLK). The beginning of configuration of devices using peripheral or slave modes must be delayed until they are initialized. An LCA device with mode lines selecting a master configuration mode extends its initialization state using four times the delay (80 to 120 ms). This ensures that all daisy-chained slave devices it may be driving will be ready. The next figure shows the sequence of states.

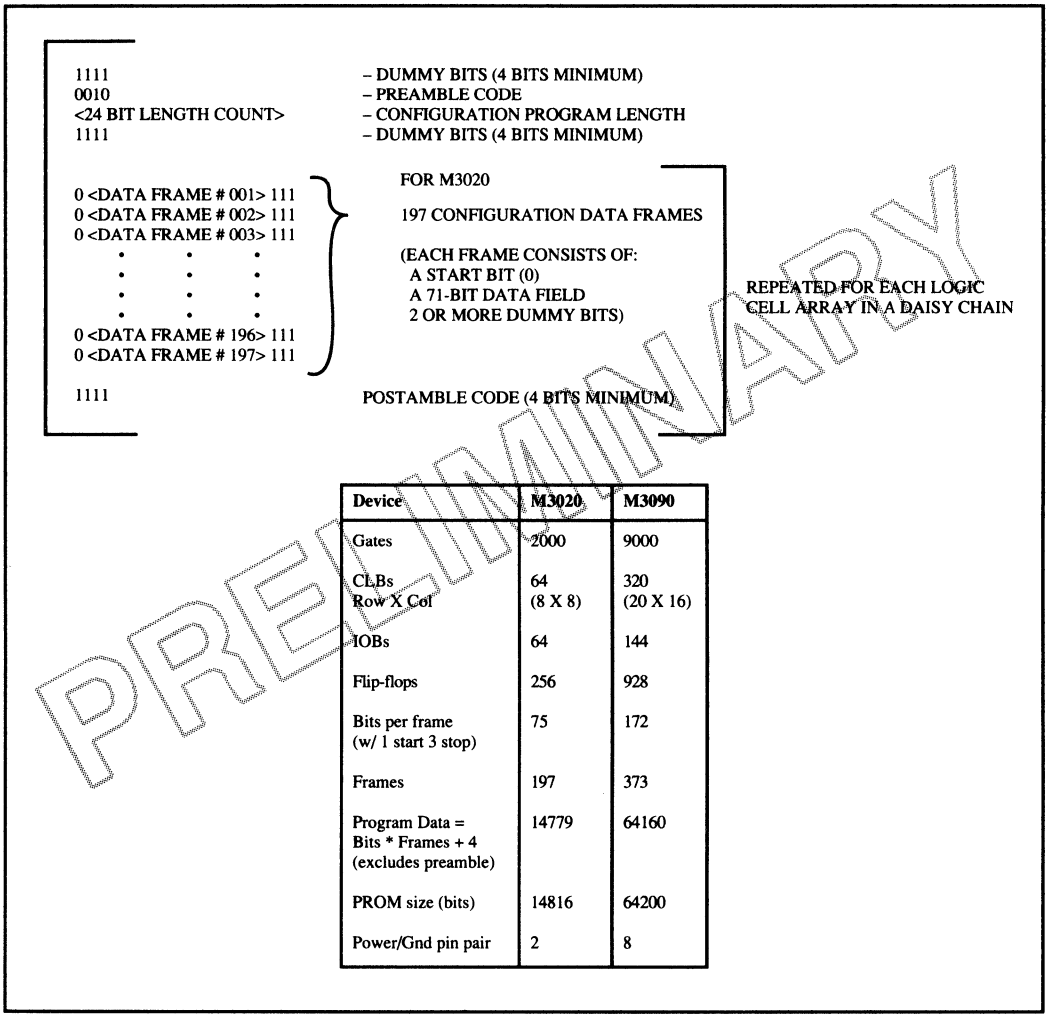


At the end of initialization, the LCA device enters the clear state, in which it clears the configuration memory. The active LOW, open-drain initialization signal, RDY/~INIT, indicates completion of the initialization and clear states. The LCA device tests for the absence of an external active LOW ~RESET before it makes a final sample of the mode lines and enters the configuration state. An external wired-AND of one or more ~INIT pins can be used to control configuration by the assertion of the active LOW ~RESET of a master mode device or to signal a processor that the LCA devices are not yet initialized.

If a configuration has begun, re-asserting ~RESET for at least three internal timer cycles is recognized, and the LCA device will abort the program, clearing the partially loaded configuration memory words. The LCA device will then re-sample ~RESET and the mode lines before re-entering the configuration state.

The configuration bit stream is initiated again when a configured LCA device senses a HIGH to LOW transition on the DONE/~PROG package pin. The LCA device returns to the clear state, in which the configuration memory is cleared and mode lines re-sampled, as for an aborted configuration. The complete configuration bit stream is cleared and loaded during each configuration cycle.

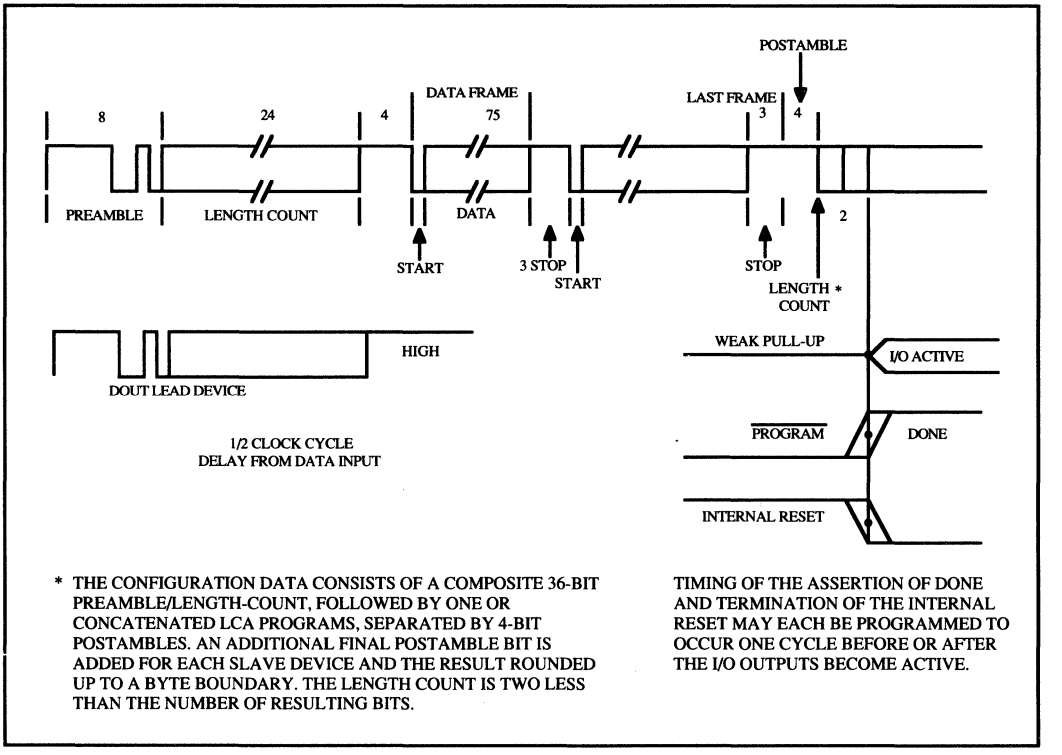
Length count control lets a system of multiple LCA devices, of various sizes, begin synchronized operation. The configuration bit stream generated by the MakePROM software of the XACT LCA Development System begins with a preamble of 11110010. This is followed by a 24-bit length count representing the total number of configuration clocks needed to complete loading of the configuration program(s). The data framing is shown in the following figure.



Device	M3020	M3090
Gates	2000	9000
CLBs Row X Col	64 (8 X 8)	320 (20 X 16)
IOBs	64	144
Flip-flops	256	928
Bits per frame (w/ 1 start 3 stop)	75	172
Frames	197	373
Program Data = Bits * Frames + 4 (excludes preamble)	14779	64160
PROM size (bits)	14816	64200
Power/Gnd pin pair	2	8

All LCA devices connected in series read and shift preamble and length count in on positive, and out on negative, configuration clock edges. An LCA device that has received the preamble and length count then presents a HIGH Data Out until it has intercepted the appropriate number of data frames. When the configuration memory of an LCA device is full and the length count does not compare, the LCA device shifts any additional data through in the same way it did for preamble and length count.

When the LCA configuration memory is full and the length count compares, the LCA device executes a synchronous start-up sequence and becomes operational, as shown below.



* THE CONFIGURATION DATA CONSISTS OF A COMPOSITE 36-BIT PREAMBLE/LENGTH-COUNT, FOLLOWED BY ONE OR CONCATENATED LCA PROGRAMS, SEPARATED BY 4-BIT POSTAMBLES. AN ADDITIONAL FINAL POSTAMBLE BIT IS ADDED FOR EACH SLAVE DEVICE AND THE RESULT ROUNDED UP TO A BYTE BOUNDARY. THE LENGTH COUNT IS TWO LESS THAN THE NUMBER OF RESULTING BITS.

TIMING OF THE ASSERTION OF DONE AND TERMINATION OF THE INTERNAL RESET MAY EACH BE PROGRAMMED TO OCCUR ONE CYCLE BEFORE OR AFTER THE I/O OUTPUTS BECOME ACTIVE.

Two CCLK cycles after the completion of loading configuration data, the user I/O pins are enabled as configured. As selected in MAKEBITS, the internal user-logic reset is released either one clock cycle before, or one clock cycle after the I/O pins become active. A similar timing selection is programmable for the DONE/~PROG output signal. DONE/~PROG can also be programmed to be an open drain or to include a pull-up resistor to accommodate wired ANDing. High During Configuration (HDC) and Low During Configuration (LDC) are two user I/O pins driven active when an LCA is initializing, clearing, or configuring. These pins and the DONE/~PROG commands provide signals for control of external logic signals such as reset, bus enable, or PROM enable during configuration. For parallel master configuration modes, these signals provide PROM enable control and let the data pins be shared with user logic signals.

User I/O inputs can be programmed for either TTL or CMOS compatible thresholds. At power-up, all inputs have TTL thresholds and can change to CMOS thresholds at the completion of configuration, if the user has selected CMOS thresholds. The threshold of ~PWRDWN and the direct clock inputs are fixed at the CMOS level.

If the crystal oscillator is used, it will begin operation before configuration is completed; this allows time for stabilization before the oscillator is connected to the internal circuitry.

Configuration Data

Configuration data to define the function and interconnection within an LCA device are loaded from an external storage at power-up and, if not inhibited, on a reprogram signal. Several methods of automatic and controlled loading of the required data are designed into the LCA device. Logic levels applied to mode selection pins at the start of configuration determine the method to be used. See the mode selection table above.

The format of the data can be either bit-serial or byte-parallel, depending on the configuration mode. Various AMD programmable gate arrays will have different sizes and numbers of data frames. To maintain compatibility between various device types of the AMD product line, the 3000 series LCA devices use formats compatible with the 2000 series. For the 3020, configuration requires 14779 bits, arranged in 197 data frames, for each device. An additional 36 bits are used in header, as shown in the previous figure. The specific data format for each device is produced by the MAKEBITS command of the XACT LCA Development System.

One or more of these files can then be combined and appended to a length count preamble and be transformed into a PROM format file by the MAKE PROM command of the XACT LCA Development System. An exception to the compatibility of the devices is that a 2000 series device cannot be used as the master for a 3000 series device if their DONE or RESET are programmed to occur after their outputs become active. The TIE option of MAKEBITS causes the unused block outputs to be defined as constant LOW levels that are used to drive the unused routing and block resources. Resources that might not be accessible to unused block outputs will then be added to FLAGNET, non-critical user nets. NORESTORE will retain the results of TIE for timing analysis with QUERYNET, before RESTORE returns the design to the untied condition. TIE can be omitted for quick breadboard iterations where a few additional mA of I_{CC} are acceptable.

The configuration bit stream begins with HIGH preamble bits, a four-bit preamble code, and a 24-bit length count. When configuration is initiated, a counter in the LCA device is set to 0 and begins to count the total number of configuration clock cycles applied to the device. As each configuration data frame is supplied to the LCA device, it is internally assembled into a data word. As each data word is completely assembled, it is loaded in parallel into one word of the internal configuration memory array. The configuration loading process is completed when the current length count equals the loaded length count, and the required configuration bit stream data frames have been written. Internal user flip-flops are held reset during configuration.

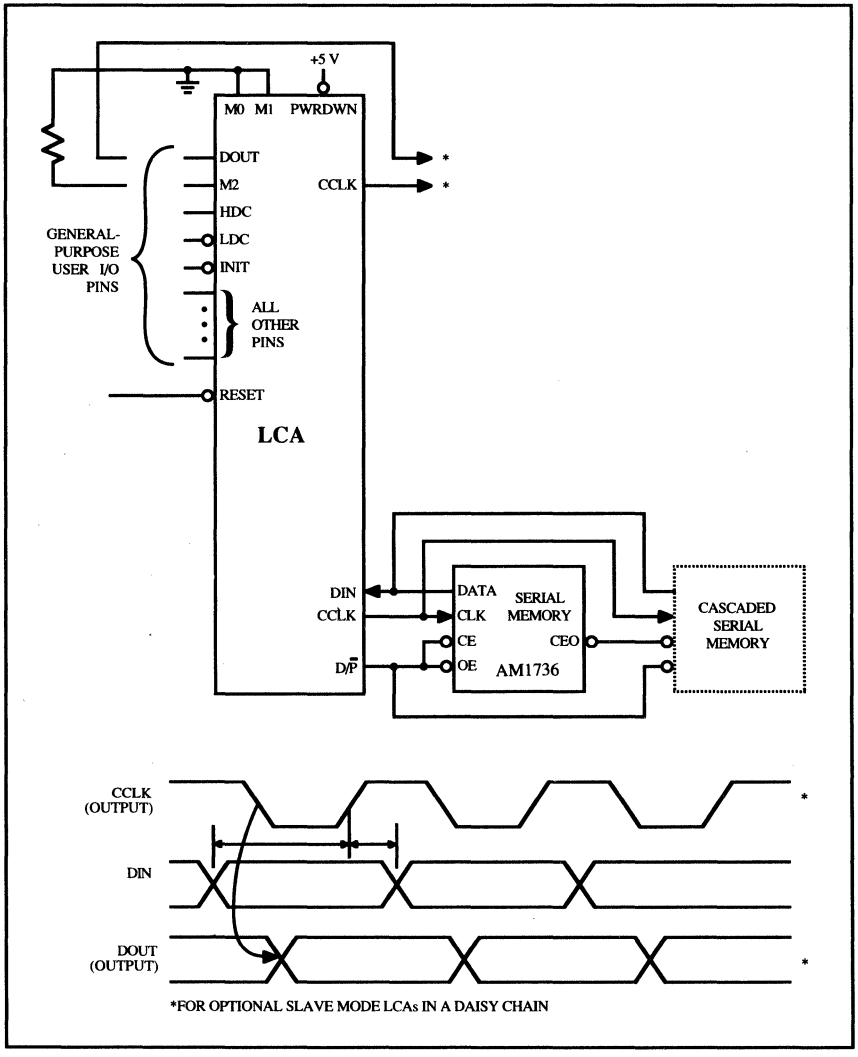
Two user-programmable pins are defined in the unconfigured LCA device. HDC and LDC, as well as DONE/~PROG, can be used as external control signals during configuration. In master mode configurations it is convenient to use LDC as an active-LOW EPROM Chip Enable. After the last configuration data-bit is loaded and the length count compares, the user I/O pins become active. Options in the MAKEBITS software allow timing choices of one clock earlier or later for the timing of the end of the internal logic reset and the assertion of the DONE signal. The open-drain DONE/~PROG output can be AND-tied with multiple LCA devices and used as an active HIGH READY, an active LOW PROM enable, or a RESET to other portions of the system.

Master Mode

In master mode, the LCA device automatically loads configuration data from an external memory device. There are three master modes that use the internal timing source to time the incoming data supplying the

configuration clock (CCLK). Serial master mode uses serial configuration data supplied to data-in (DIN) from a synchronous serial source such as the AMD Serial Configuration PROM (Am1736) shown below.

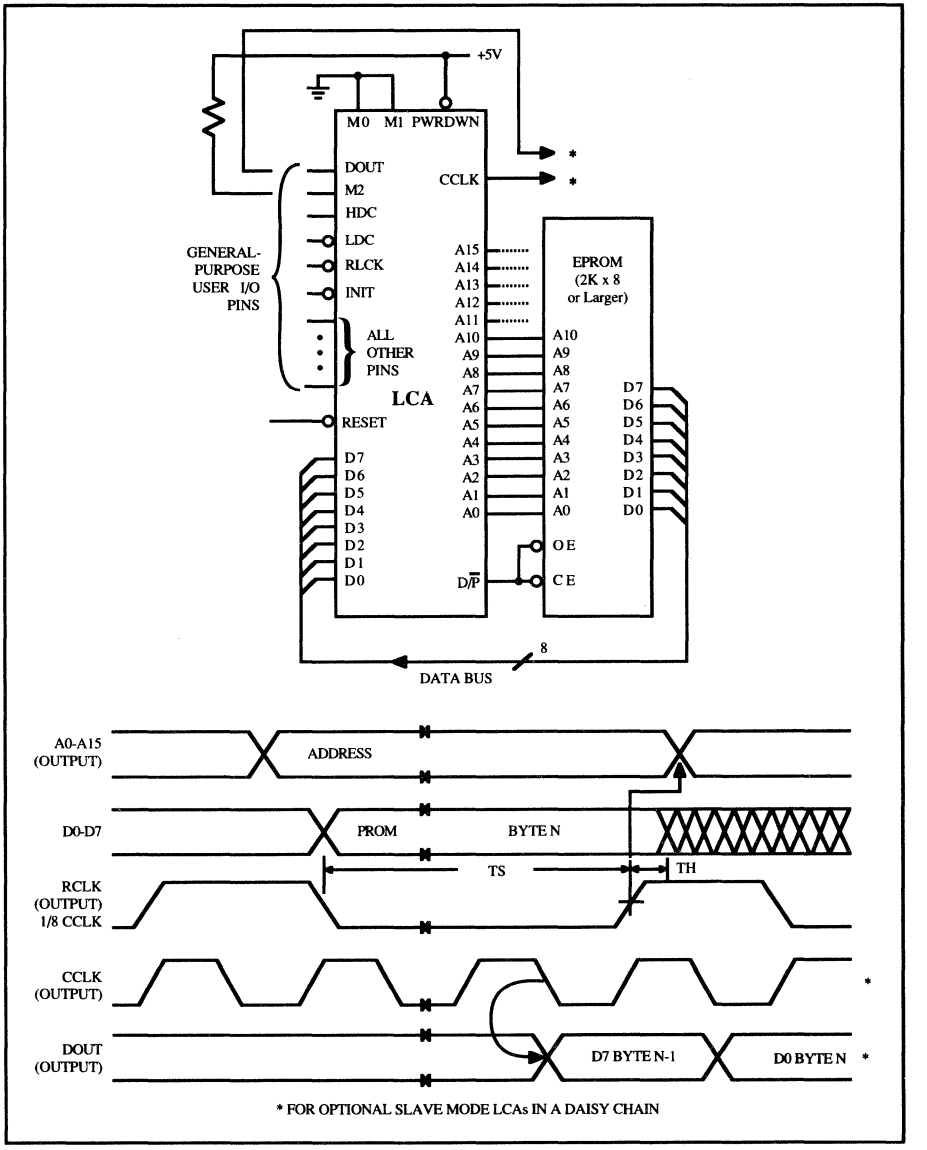
The one-time-programmable Am1736 Serial Configuration PROM supports automatic loading of configuration programs up to 36K bits. Multiple devices can be cascaded to support additional LCA devices. An early DONE inhibits the Am1736 data output one CCLK cycle before the LCA I/O becomes active.



Master LOW and master HIGH modes automatically use parallel data supplied to the D0-D7 pins in response to the 16-bit address generated by the LCA device. The next figure shows an example of the parallel master mode connections required. The LCA HEX starting address is 0000 and increments for master LOW mode. It is FFFF and decrements for master HIGH mode. These two modes provide address compatibility with microprocessors beginning execution from opposite ends of memory.

For master HIGH or LOW, data bytes are read in parallel by each read clock (RCLK) and internally serialized by the configuration clock. As each data byte is read, the least significant bit of the next byte, D0, becomes the next bit in the internal serial configuration word. One master mode LCA device can be used to interface the configuration program-store and pass additional concatenated configuration data to additional LCA devices in a serial daisy-chain fashion. CCLK is provided for the slaved devices and their serialized data is supplied from DOUT to DIN - DOUT to DIN, etc.

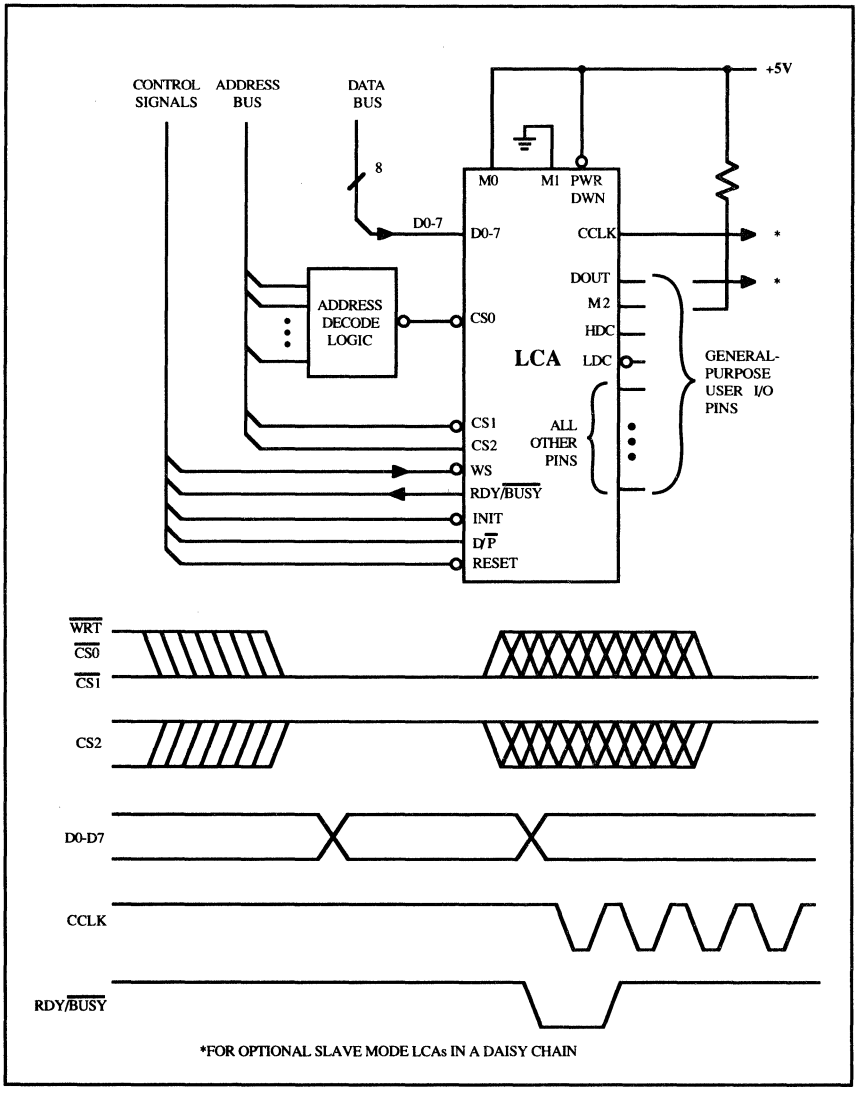
Configuration data are loaded automatically from an external byte wide PROM. An early DONE inhibits the PROM outputs a CCLK before the LCA I/O becomes active.


+5V
M0 M1 PWRDWN
DOUT
M2
HDC
LDC
RLCK
INIT
CCLK
GENERAL-
PURPOSE
USER I/O
PINS
ALL
OTHER
PINS
LCA
RESET
D7
D6
D5
D4
D3
D2
D1
D0
A15
A14
A13
A12
A11
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
D/P
EPROM
(2K x 8
or Larger)
A10
A9
A8
A7
A6
A5
A4
A3
A2
A1
A0
D7
D6
D5
D4
D3
D2
D1
D0
OE
CE
8
DATA BUS
A0-A15
(OUTPUT)
ADDRESS
D0-D7
PROM
BYTE N
TS
TH
RCLK
(OUTPUT)
1/8 CCLK
CCLK
(OUTPUT)
DOUT
(OUTPUT)
D7 BYTE N-1
D0 BYTE N
* FOR OPTIONAL SLAVE MODE LCAs IN A DAISY CHAIN

Peripheral Mode

Peripheral mode provides a simplified interface through which the device can be loaded byte-wide, as a processor peripheral. The next figure shows the peripheral mode connections. Processor write cycles are decoded from the common assertion of the active LOW Write Strobe (~WRT), and two active LOW and one active HIGH Chip Selects (~CS0, ~CS1, CS2). If all these signals are not available, the unused inputs should be driven to their respective active levels. The LCA device accepts one byte of configuration data on the D0-D7 inputs for each selected processor write cycle. Each byte of data is loaded into a buffer register.

The LCA device generates a configuration clock from the internal timing generator and serializes the parallel input data for internal framing or for succeeding slaves on Data Out (DOUT). An output HIGH on the READY/~BUSY pin indicates completion of loading for each byte and that the input register is ready for a new byte. As with master modes, peripheral mode can also be used as a lead device for a daisy-chain of slave devices.

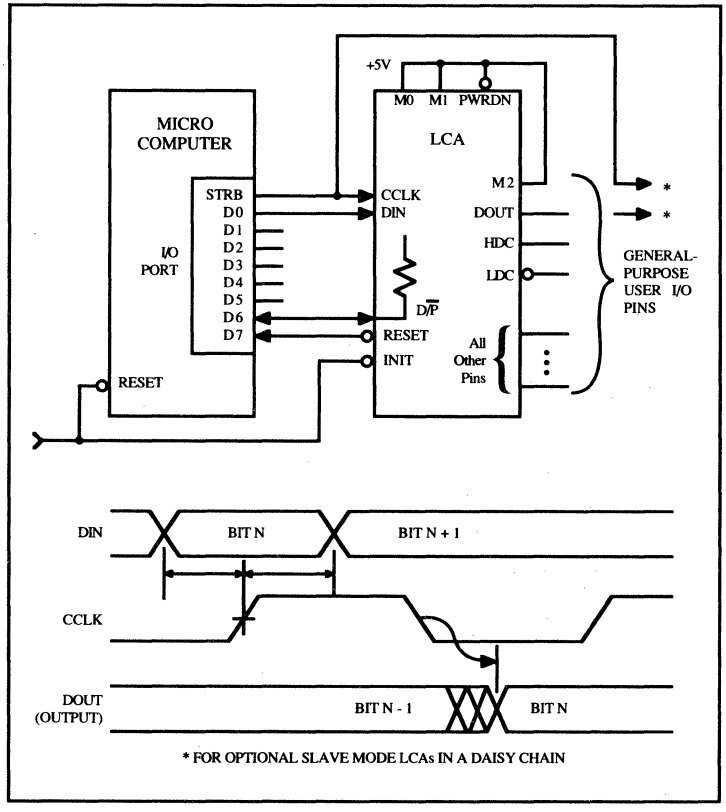


*FOR OPTIONAL SLAVE MODE LCAs IN A DAISY CHAIN

Slave Mode

Slave mode provides a simple interface for loading the LCA device configuration, as shown below. Data are supplied in conjunction with a synchronizing input clock. Bit-serial data configuration are read at rising edge of the CCLK. Data on DOUT are provided on the falling edge of CCLK.

Most slave mode applications are in daisy-chain configurations in which the data input are supplied by the previous LCA's data out, while the clock is supplied by a lead device in master or peripheral mode. Data can also be supplied by a processor or other special circuits.

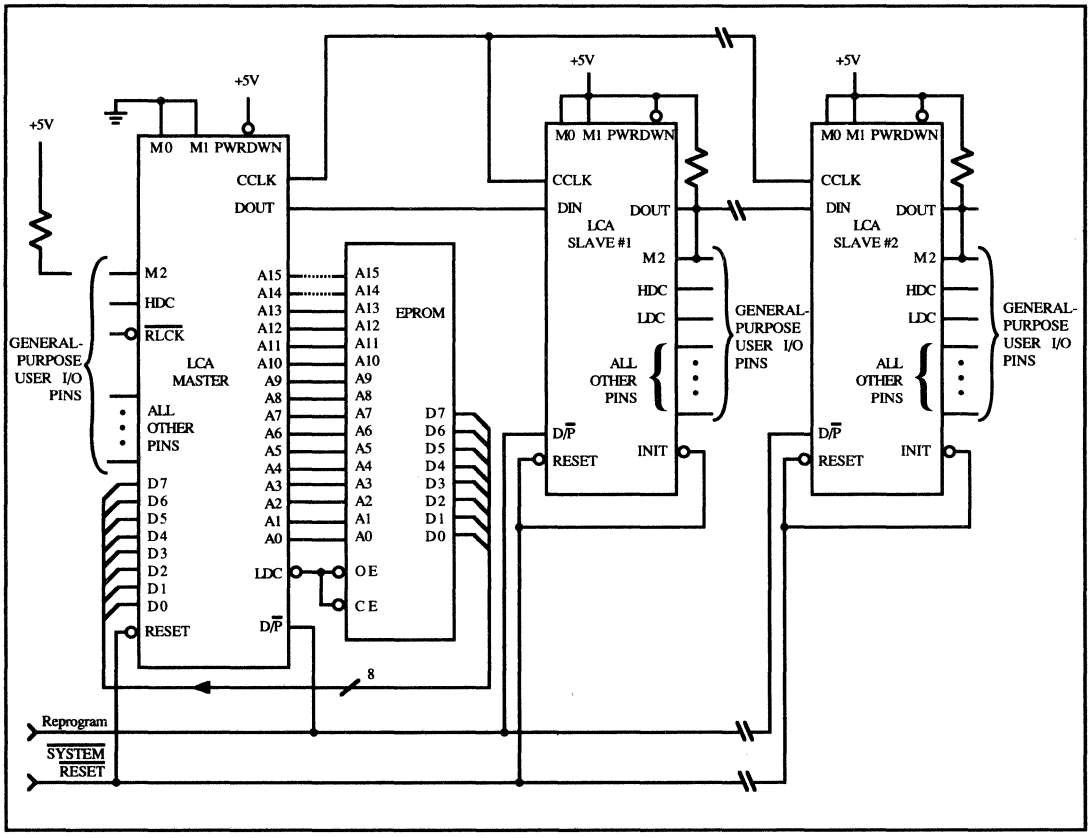


Daisy-Chain

The AMD XACT LCA Development System is used to create a composite configuration bit stream for selected LCA devices. This configuration includes the following:

- A preamble
- A length count for the total bit stream
- Multiple concatenated data programs
- A postamble
- An additional fill bit per device in the serial chain

After loading and passing on the preamble and length count to a possible daisy-chain, a lead device will load its configuration data frames while providing a HIGH DOUT to possible down-stream devices as shown below. In this figure, all are configured from the common EPROM source. The slave mode device ~INIT signals delay the master device configuration until they are initialized. A well defined termination of SYSTEM RESET is needed when controlling multiple LCA devices.



Loading continues until the current length count has reached the full value. The additional data are passed through the lead device and appear on the DOUT pin in serial form. The lead device also

generates the CCLK to synchronize the serial output data and data in of LCA devices further attached. Data are read in on DIN of slave devices by the positive edge of CCLK and shifted out the DOUT on the negative edge of CCLK. A parallel master mode device uses its internal timing generator to produce an internal CCLK of eight times its EPROM address rate, while a peripheral mode device produces a burst of eight CCLKs for each chip select and write-strobe cycle. The internal timing generator continues to operate for general timing and synchronization of inputs in all modes.

SPECIAL CONFIGURATION FUNCTIONS

The configuration data include control over several special functions, in addition to the normal user logic functions and interconnections.

- Input thresholds
- Readback enable
- DONE pull-up resistor
- DONE timing
- RESET timing
- Oscillator frequency divided-by-two

Each of these functions is controlled by configuration data bits selected as part of the normal XACT LCA Development System bit-stream generation process.

Input Thresholds

Prior to the completion of configuration, all LCA input thresholds are TTL compatible. Upon completion of configuration, the input thresholds become either TTL or CMOS compatible, as programmed. The use of the TTL threshold option requires some additional supply current for threshold shifting. The exception is the threshold of the ~PWRDWN input and direct clocks that always have a CMOS input. Prior to the completion of configuration, the user I/O pins have a high impedance pull-up. The configuration bit stream can be used to enable the IOB pull-up resistors in the operational mode to act either as an input load or to avoid a floating input on an otherwise unused pin.

Readback

The contents of an LCA device can be read back if it has been programmed with a bit stream in which the Readback option has been enabled. Readback can be used for verification of configuration, as well as a method of determining the state of internal logic nodes during debugging with the XACTOR In-Circuit debugger. There are three options in generating the configuration bit stream.

- **Never** inhibits the Readback capability.
- **One-time** inhibits Readback after one Readback has been executed to verify the configuration.
- **On-command** permits unrestricted use of Readback.

Readback is done without the use of any of the user I/O pins; only M0, M1, and CCLK are used. The initiation of readback is produced by a LOW to HIGH transition of the M0/RTRIG (Read Trigger) pin. Once the READBACK command has been given, the input CCLK is driven by external logic to read back each data bit in a format similar to loading. After two dummy bits, the first data frame is shifted out, in inverted sense, on the M1/~RDATA (Read Data) pin. All data frames must be read back to complete the process and return the mode select and CCLK pins to their normal functions.

The readback data includes the current state of each internal logic block storage element, and the state of the .i and .ri connection pins on each IOB. These data are imbedded into unused configuration bit positions during readback. This state information is used by the XACT LCA Development System In-Circuit Verifier to provide visibility into the internal operation of the logic while the system is operating. To readback a uniform time-sample of all storage elements it may be necessary to inhibit the system clock.

Reprogram

To initiate a reprogramming cycle, the dual function package pin DONE/~PROG must transition from HIGH to LOW . To reduce noise sensitivity, the input signal is filtered for two cycles of the LCA's internal timing generator. When reprogram begins, the user programmable I/O output buffers are disabled, and high impedance pull-ups are provided for the package pins. The device returns to the clear state and clears the configuration memory before it is initialized.

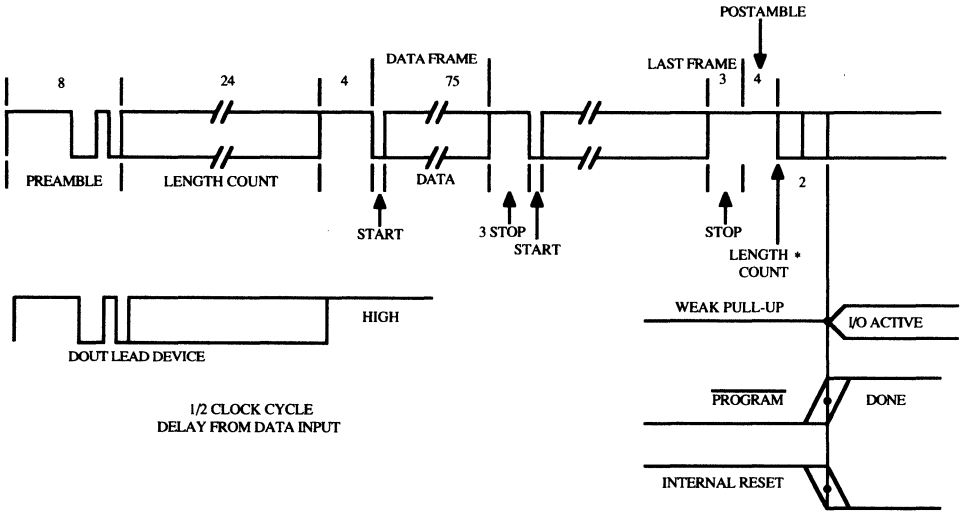
Reprogram control is often exercised using an external open collector driver that pulls DONE/~PROG LOW. Once it recognizes a stable request, the LCA device holds a LOW until the new configuration has been completed. Even if the reprogram request is externally held LOW beyond the configuration period, the LCA device will begin operation upon completion of configuration.

DONE Pull-up

DONE/~PROG is an open drain I/O pin indicating that the LCA device is operational. An optional internal pull-up resistor can be enabled by the user of the XACT LCA Development System when MakeBits is executed. The DONE/~PROG pins of multiple LCA devices in a daisy-chain can be connected to indicate all are DONE or to direct them to reprogram.

DONE Timing

By a selection in the MakeBits program, the timing of the DONE status signal can be controlled to occur one CCLK cycle before, or one cycle after, the timing of outputs are activated. This is shown below. This facilitates control of external functions such as a PROM enable or holding a system in a wait state.



* THE CONFIGURATION DATA CONSISTS OF A COMPOSITE 36-BIT PREAMBLE/LENGTH-COUNT, FOLLOWED BY ONE OR CONCATENATED LCA PROGRAMS, SEPARATED BY 4-BIT POSTAMBLES. AN ADDITIONAL FINAL POSTAMBLE BIT IS ADDED FOR EACH SLAVE DEVICE AND THE RESULT ROUNDED UP TO A BYTE BOUNDARY. THE LENGTH COUNT IS TWO LESS THAN THE NUMBER OF RESULTING BITS.

TIMING OF THE ASSERTION OF DONE AND TERMINATION OF THE INTERNAL RESET MAY EACH BE PROGRAMMED TO OCCUR ONE CYCLE BEFORE OR AFTER THE I/O OUTPUTS BECOME ACTIVE.

RESET Timing

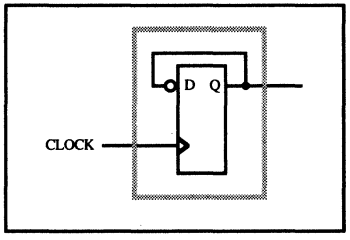
As with DONE timing, the timing of the release of the internal RESET can be controlled by a selection in the MakeBits program. It then occurs one CCLK cycle before, or one cycle after, the timing of outputs are enabled, as shown above. This reset maintains all user-programmable flip-flops and latches in a zero state during configuration.

Crystal Oscillator Division

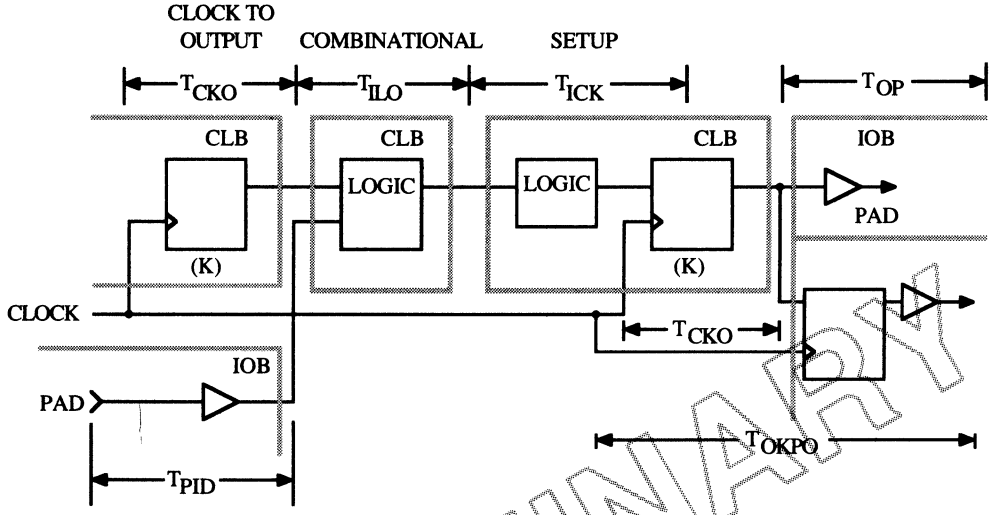
A selection in the MakeBits software lets the user incorporate a dedicated divide-by-two flip-flop in the crystal oscillator function. This helps ensure a symmetrical timing signal. Although the frequency stability of crystal oscillators is high, the symmetry of the waveform can be affected by bias or feedback drive.

Device Performance

The high performance of the LCA device is due in part to the manufacturing process, which is similar to that used for high-speed CMOS static memories. Performance can be measured in terms of minimum propagation times for logic elements. The parameter that traditionally describes the overall performance of a gate array is the toggle frequency of a flip-flop. The configuration for determining the toggle performance of the LCA device is shown below. The flip-flop output Q is fed back through the combinational logic as ~Q to form the toggle flip-flop.



Actual LCA device performance is determined by the timing of critical paths, including both the fixed timing for the logic and storage elements in that path, as well as the timing associated with the network routing. Examples of internal worst-case timing are included in the performance data to let the user to make the best use of the device's capabilities. The XACT LCA Development System timing calculator, or LCA generated simulation models, should be used to calculate worst-case paths by using actual impedance and loading information. The following figure shows a variety of elements used involved in determining system performance.



			Speed Grade	-50		-70		Units
	Description	Symbol	Min	Max	Min	Max		
Logic input to Output	Combinational	T_{ILO}		14		9		ns
K Clock	To output	T_{CKO}		12		8		ns
	Logic-input setup	T_{ICK}	12		8			ns
	Logic-input hold	T_{CKI}	0		0			ns
Input/Output	Pad to input (direct)	T_{PID}		10		7		ns
	Output to pad (enabled)	T_{OP}		14		10		ns
	I/O clock to pad	T_{OKPO}		18		13		ns
FF toggle frequency		F_{CLK}		50		70		MHz

The speed of internal elements is determined by differential measurements of package pins. The performance of a user's design can be predicted by the XACT LCA Development System delay calculator.

Actual measurement of internal timing is not practical; often only the sum of component timing is relevant, as in the case of input to output. The relationship between input and output timing is arbitrary; only the total determines performance. Timing components of internal functions can be determined by measuring the differences at the pins of the package. A synchronous logic function with a clock-to-block output, and a block-input to clock set-up, is capable of higher speed operation than a logic configuration of two synchronous blocks with an extra combinational block level between them. System clock rates to 60% of the toggle frequency are practical for logic in which there is an extra combinational level located between synchronized blocks. This permits implementation of functions of up to 25 variables. The use of the wired-AND is also available for wide, high-speed functions.

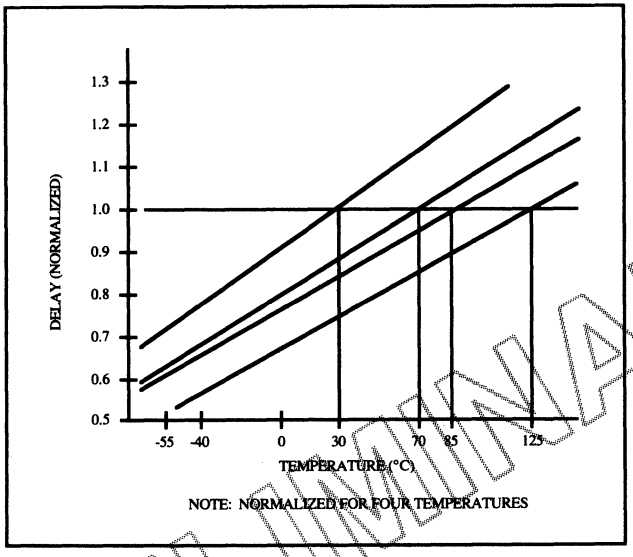
Logic Block Performance

Logic block performance is expressed as the propagation time from the interconnection point at the input of the combinational logic, to the output of the block in the interconnection area. Combinational performance is independent of the specific logic function, which is based on look-up tables.

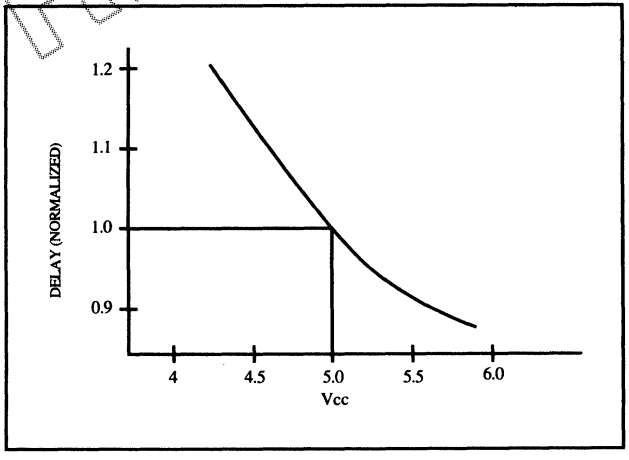
The only parameter for all logic functions is the Logic Input to Output delay. For combinational logic used in conjunction with the storage element, however, there are two critical parameters. First, for the combinational logic function driving the data input of the storage element, the critical timing is data setup relative to the clock edge provided to the flip-flop. Second, for the signals then produced by the storage elements, the critical timing is the Clock to Output delay. These parameters are shown in the previous figure.

Loading of a logic block output is limited only by the resulting propagation delay of the larger interconnection network. Speed performance of the logic block is a function of the supply voltage and the temperature, as shown in the next two figures.

The following figure shows the change in speed performance as a function of temperature. The variation is normalized for 30° C, 70° C, 85° C, and 125° C.



The following figure shows how the speed performance of a CMOS device increases with Vcc within the operating range.



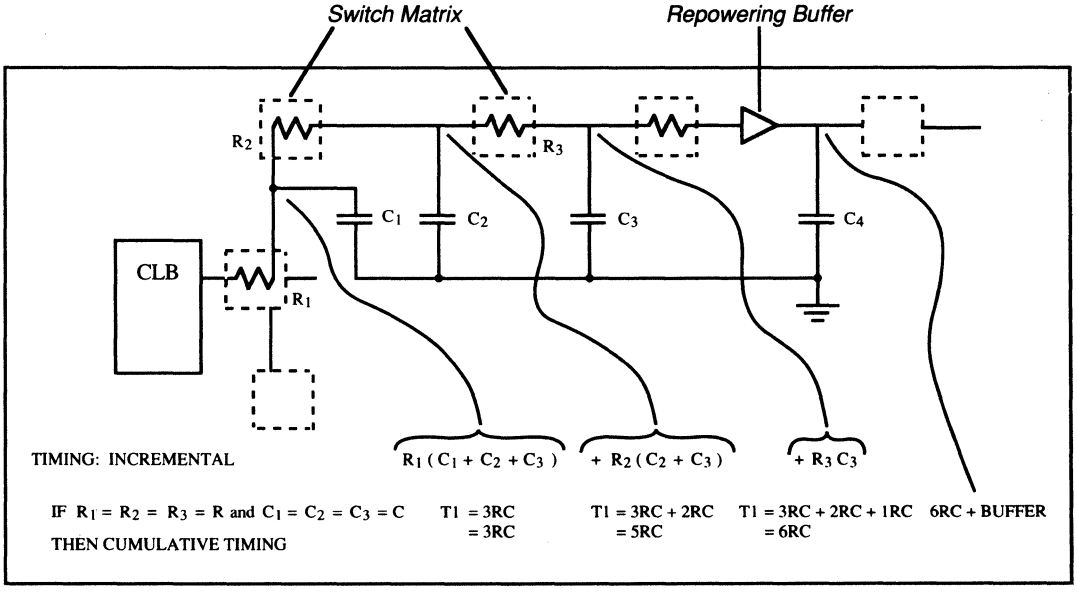
Interconnection Performance

Interconnection performance depends on the routing resource used for the signal path. As discussed earlier, direct interconnection from block to block provides a fast path for a signal. The single metal segment used for long lines exhibits low resistance from end to end, but relatively high capacitance. Signals driven through a programmable switch will have the additional impedance of the switch added to their normal drive impedance.

General purpose interconnection performance depends on the number of switches and segments used, the presence of the bidirectional re-powering buffers, and the loading at all points on the signal path. In calculating the worst-case timing for a general interconnection path, the timing calculator portion of the XACT LCA Development System takes all of these elements into account.

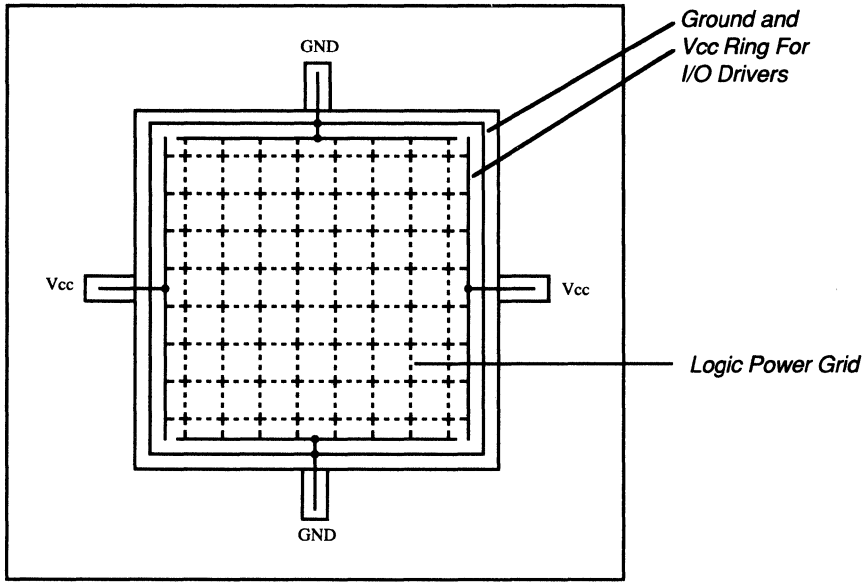
As an approximation, interconnection timing is proportional to the summation of totals of local metal segments beyond each programmable switch. In effect, the time is a sum of R-C time, each approximated by an R times the total C it drives. The R of the switch and the C of the interconnection are functions of the particular device performance grade.

For a string of three local interconnections, the approximate time at the first segment (after the first switch resistance) would be three units; after the next switch there are an additional two units; and there is an additional unit after the last switch in the chain. The interconnection R-C chain terminates at each re-powering buffer. The capacitance of the block inputs is not significant; the capacitance is in the interconnection metal and switches, as shown in the following figure.



Power Distribution

Power for the LCA device is distributed through a grid to achieve high noise immunity and isolation between the logic and I/O. Inside the LCA device, a dedicated Vcc and ground ring surrounding the logic array provide power to the I/O drivers, as shown below. An independent matrix of Vcc and ground lines supplies the interior logic of the device. This power distribution grid provides a stable supply and ground for all internal logic; this assumes that the external package power pins are all connected and appropriately decoupled. Usually, a 0.1 µF capacitor connected near the Vcc and ground pins of the package will provide adequate decoupling.

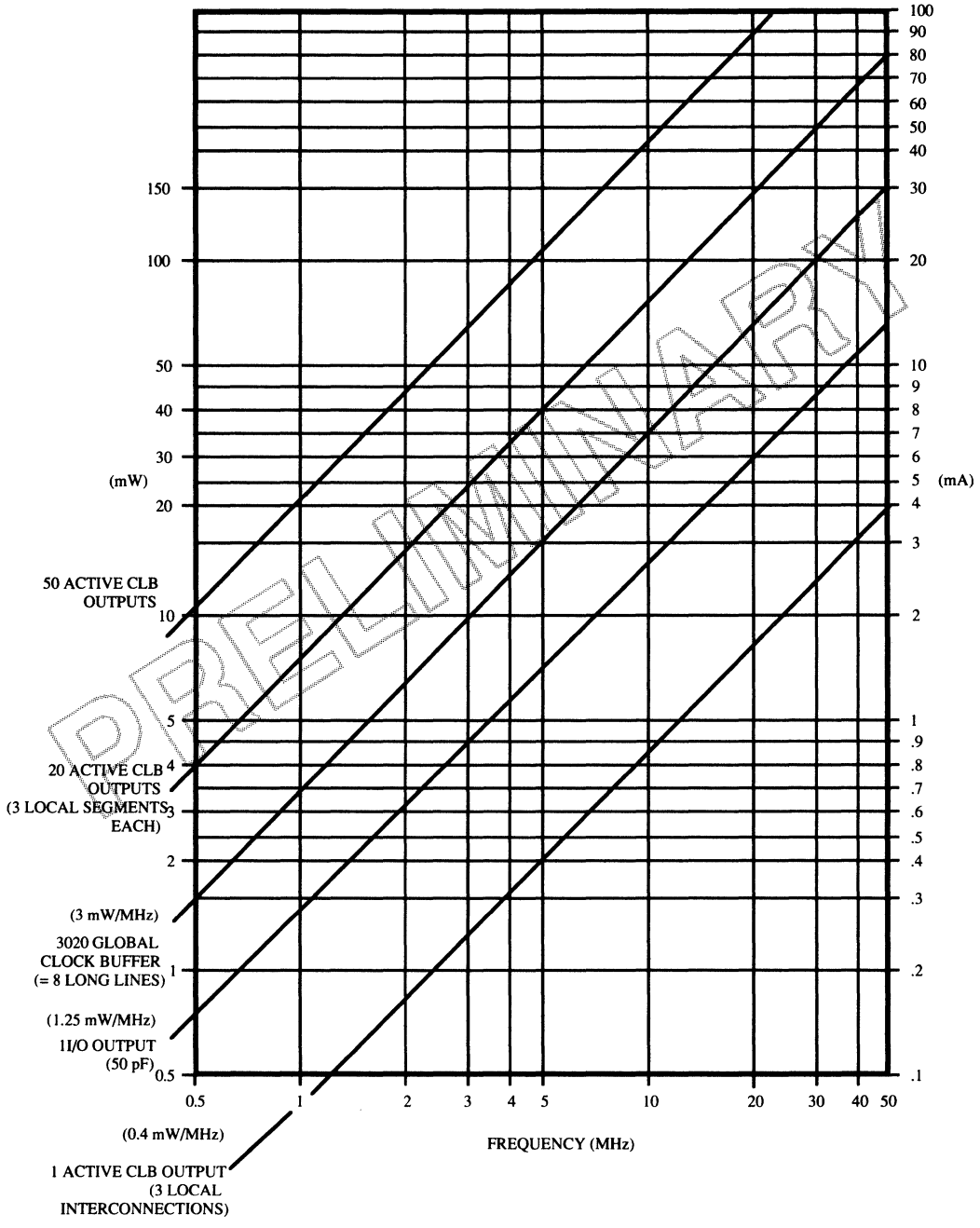


Output buffers capable of driving the specified 4 mA loads under worst-case conditions can be capable of driving 25 to 30 times that current in a best case. Noise can be reduced by minimizing external load capacitance and reducing simultaneous output transitions in the same direction. Also, it may be beneficial to locate heavily loaded output buffers near the ground pads. The IOB output buffers have a slew-limited mode that should be used where output rise and fall times are not speed critical. A lower AC drive current reduces transition and supply noise without a corresponding reduction in DC drive. A maximum of 32 simultaneously switching outputs is allowed.

Power Dissipation

The LCA device exhibits the low power consumption characteristic of CMOS ICs. For any design, the user can use the figure below to calculate the total power requirement based on the sum of the external and internal capacitive and DC loads. The total chip power is the sum of $V_{CC} \cdot I_{CCO}$, plus internal and external values of capacitive charging currents and resistive loads.

The configuration options of TTL chip input threshold requires power for the threshold reference. The power required by the static memory cells holding the configuration data is very low and can be maintained in a power-down mode.



Usually, most power dissipation is produced by external capacitive loads on the output buffers. The load- and frequency-dependent power is 25 µW/pF/MHz per output. Another component of I/O power is the DC loading on each output pin by LCA-driven devices.

Internal power dissipation is a function of the number and size of the nodes, and the frequency at which they change. In an LCA device, the fraction of nodes changing on a given clock is typically low (10-20%). For example, in a large binary counter, the average clock cycle produces changes equal to one CLB output at the clock frequency. In a 4-input AND gate, there will be two transitions in 16 states. Typical global clock buffer power is about 3 mW/MHz. The internal capacitive load is more a function of interconnection than fanout. With typical load of three general interconnection segments, each CLB output requires about 0.4 mW/MHz of its output frequency.

Total power = V_{CC} • I_{CCO} + external (DC + capacitive) + internal (CLB + IOB+ Long Line+ pull-up)

Because the control storage of the LCA device is CMOS static memory, its cells require a very low standby current for data retention. In some systems, this characteristic can be used as a method of preserving configurations in the event of a primary power loss. The LCA device has built in power-down logic that, when activated, will disable normal operation of the device and retain only the configuration data. All internal operation is suspended and output buffers are placed in a high impedance state with no pull-ups. Power-down data retention is possible with a simple battery-backup circuit because the power requirement is extremely low. For retention at 2.4 V, the required current is typically on the order of 50 nA.

To force the LCA device into the power-down state, the user must pull the ~PWRDWN pin LOW and continue to supply a retention voltage to the V_{CC} pins of the package. When normal power is restored, V_{CC} is increased to its normal operating voltage and ~PWRDWN is returned to HIGH. The LCA device resumes operation with the same internal sequence that occurs at the conclusion of configuration. Internal I/O and logic block storage elements will be reset, the outputs will become enabled, and the DONE/~PROG pin will be released. No configuration programming is required.

When the power supply is removed from a CMOS device, it is possible to supply some power from an input signal. The conventional electrostatic input protection is provided by diodes to the supply and ground. A positive voltage applied to an input or output will cause the positive protection diode to conduct and drive the power pin. This condition can produce invalid power conditions and should be avoided. A large series resistor can be used to limit the current, or a bi-polar buffer can be used to isolate the input signal.

DEVELOPMENT SYSTEMS

To implement your system application on the LCA device, AMD provides a wide host of software packages. These packages are primarily IBM-PC/AT based and allow the entire programmable gate array design cycle to be completed inexpensively and quickly at the system designer's desk. The packages provide the following capabilities.

- Schematic capture
- PALASM™ Boolean entry
- Large number of predefined macro library elements
- Automatic logic conversion and reduction
- Logic and electrical rule checking

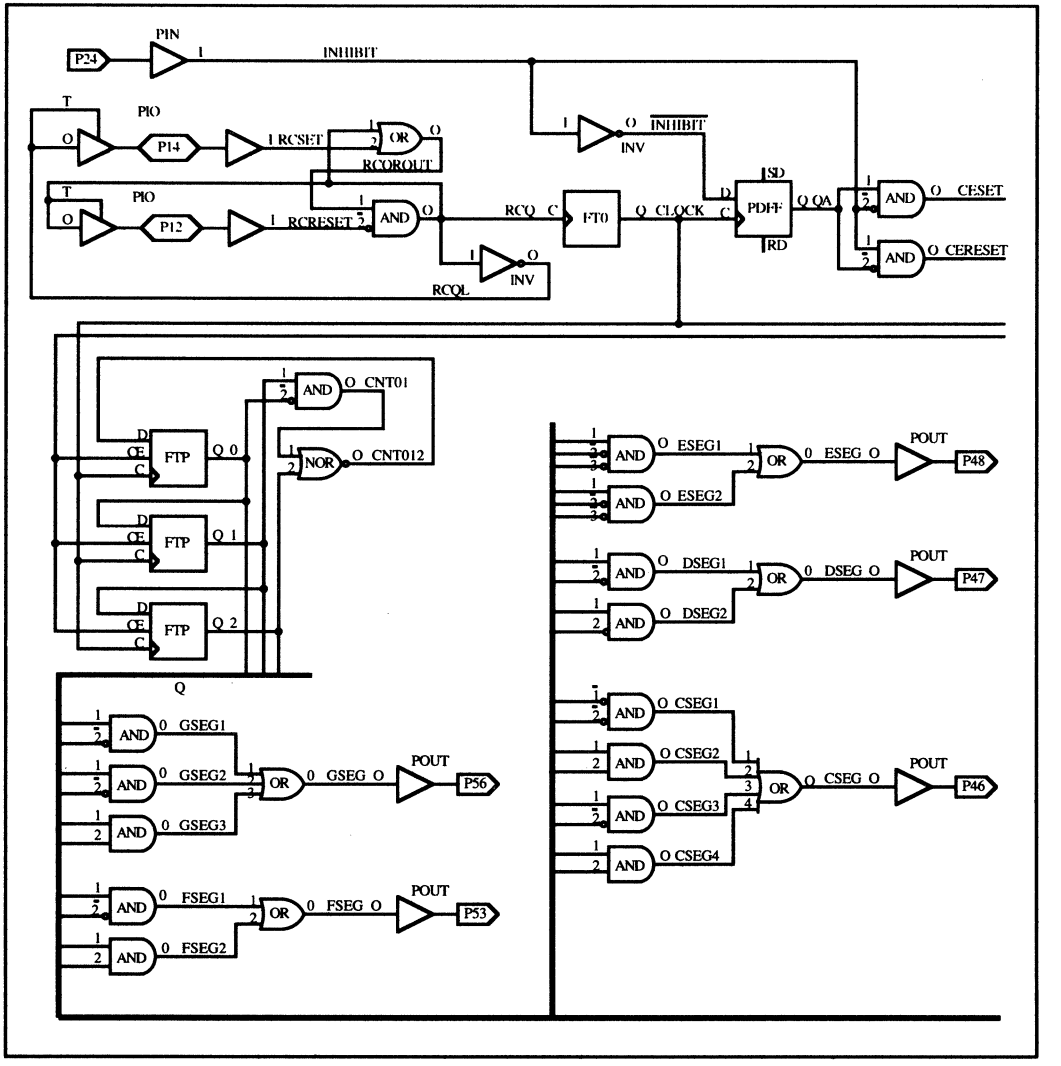
- Logic partitioning
- Automatic placement and routing
- Interactive timing calculator
- Logic and timing simulation interfaces
- Automatic design documentation
- Interactive design editing and optimization
- PROM programmer format output capabilities
- In-system design verification for multiple arrays

The XACT LCA Design Editor is hosted on an IBM-PC/AT system with DOS 3.0 or higher. The system requires 640K bytes of internal RAM, 1.5 MBytes of extended memory, color graphics, and a mouse. The complete system requires one parallel I/O port and two serial ports for the mouse and in-system emulator.

Design Entry

Design entry can be accomplished with popular schematic editors. Popular engineering workstations such as Daisy or Mentor are also supported. Additionally, designs can be entered and configured manually with the XACT Design Editor, or through Boolean expressions via PALASM programmable logic language.

The following figure shows a partial sample of a entering a design via schematic capture.



Design Implementation

Following design entry, logic designs can be automatically converted, reduced, partitioned, placed, and routed with the Automatic Design Implementation (ADI) software packages. For those designs that are not completely automatic, the XACT Design Editor can be used to manually complete or optimize the design. Following layout, various design and electrical rules are checked automatically by the software, producing a valid design file. This file contains all the programming data used to download directly into

an LCA device in the user's target system. The programming information can be used to program PROM, EPROM, or ROM devices, or stored in other media, as needed by the final system.

Design Verification

Design verification can be accomplished by using the AMD XACTOR In-System Emulator directly in the target system. Also, the ADI packages provide output data that can be accepted by popular simulators such as P-SILOS for complete logic and timing simulation. If design changes are required, the changes can be implemented in minutes at the designer's desk.

PIN DESCRIPTIONS

~PWRDWN

An active LOW power down input stops all internal activity to minimize Vcc power and puts all output buffers in a high impedance state. Configuration is retained, however, internal storage elements are Reset. When the ~PWRDWN pin returns HIGH, the device returns to operation with the same sequence of Reset, buffer enable, and DONE/~PROGRAM as at the completion of configuration.

M 0

As Mode 0, this input and M1 M2 are sampled before the start of configuration to establish the configuration mode to be used.

o r

RTRIG

As a Read Trigger, after configuration is complete, an input transition to a HIGH will initiate a Readback of configuration and storage element data by CCLK. This operation can be limited to a single request, or can be inhibited altogether, by selecting the appropriate readback option when generating the bit stream.

M 1

As Mode 1, this input, M0, and M2 are sampled before the start of configuration to establish the configuration mode.

o r

~RDATA

As an active LOW Read Data, this pin is the output of the readback data after configuration is complete.

M 2

As Mode 2 this input, M0, and M1 are sampled before the start of configuration to establish the configuration mode. After configuration, this pin becomes a user-programmable I/O pin.

HDC

High During Configuration is held at a HIGH level by the LCA device until after configuration. It is intended to be available as a control indication that configuration is not completed. After configuration, this pin is a user I/O pin.

LDC

Low During Configuration is held at a LOW level by the LCA device until after configuration. It is intended to be available as a control indication that configuration is not completed. It is particularly useful in master mode as a LOW enable for an EPROM. After configuration, this pin is a user I/O pin. If used as a LOW EPROM enable, it would need to be programmed as a HIGH after configuration.

~INIT

This is active LOW open collector output is held LOW during the power stabilization and internal clearing of the configuration memory. It can be used to indicate status to a configuring microprocessor or, as a wired-AND of several slave mode devices, a hold-off signal for a master mode device. After configuration, this pin becomes a user programmable I/O pin.

~RESET

This active LOW input has three functions. Prior to the start of configuration, a LOW input will delay the start of configuration. An internal circuit senses the application of power and begins a minimal time-out cycle. When the time-out and ~RESET are complete, the levels of the M lines are sampled and configuration begins. If ~RESET is asserted during a configuration , the LCA device is re-initialized and will restart the configuration at the termination of ~RESET. If ~RESET is asserted after configuration is complete, it will provide an asynchronous reset of all IOB and CLB storage elements of the LCA device.

DONE

The DONE output is configurable as an open drain with or without a pull-up resistor. At the completion of configuration, the circuitry of the LCA device becomes active in a synchronous order, and DONE can be programmed to occur one cycle before or after.

~PROG

Once configuration is completed, a HIGH-to-LOW transition of this pin will cause an initialization of the LCA device and start a reconfiguration.

XTL1

This user I/O pin can be configured to operate as the output of an amplifier usable with an external crystal and bias circuitry.

XTL2

This user I/O pin can be configured to operate as the input of an amplifier usable with an external crystal and bias circuity.

CCLK

During configuration, Configuration Clock is an output of an LCA device in master mode or peripheral mode. LCA devices in slave mode use it as a clock input. During a Readback operation, it is an input clock for the configuration data being output.

DOUT

This user I/O pin is used during configuration to output serial configuration data for the Data In of daisy-chained slaves.

DIN

This user I/O pin is used as serial Data In during slave or master serial configuration. This pin is D0 in master or peripheral configuration mode.

~CS0, ~CS1, CS2, ~WRT

These four inputs represent a set of signals, three active LOW and one active HIGH, which are used in peripheral mode to control configuration data entry. The assertion of all four generates a write to the internal data buffer. The removal of any assertion results in the present data of D0-D7 being clocked in.

~RCLK

During master parallel mode configuration, this pin represents a read of an external memory device.

RDY/~BUSY

During peripheral parallel mode configuration, this pin indicates when the chip is ready for another byte of data to be written to it. After configuration is complete, this pin becomes a user-programmed I/O pin.

D0-D7

This set of eight pins represents the parallel configuration byte for the parallel master and peripheral modes. After configuration is complete, they are user-programmable I/O pins.

A0-A15

This set of 16 pins presents an address output for a configuration EPROM during master parallel mode. After configuration is complete, they are user-programmable I/O pins.

I/O

A pin that, after configuration, can be programmed by the user to be an input and/or output pin. Some of these pins present a high impedance pull-up or perform other functions before configuration is complete.

Table 2a. 3000 Family Configuration Pin Assignments

CONFIGURATION MODE: <M2:M1:M0>					68	84	84	175	USER
SLAVE <1:1:1>	MASTER-SER <0:0:0>	PERIPHERAL <1:0:1>	MASTER-HIGH <1:1:0>	MASTER-LOW <1:0:0>	PLCC	PLCC	PGA	PGA	OPERATION
PWRDWN (I)	PWRDWN (I)	PWRDWN (I)	PWRDWN (I)	PWRDWN (I)	10	12	B2	B2	PWRDWN (I)
M1 (HIGH) (I)	M1 (LOW) (I)	M1 (LOW) (I)	M1 (HIGH) (I)	M1 (LOW) (I)	25	31	J2	B14	RDATA (O)
M0 (HIGH) (I)	M0 (LOW) (I)	M0 (HIGH) (I)	M0 (LOW) (I)	M0 (LOW) (I)	26	32	L1	B15	RTRIG (I)
M2 (HIGH) (I)	M2 (LOW) (I)	M2 (HIGH) (I)	M2 (HIGH) (I)	M2 (HIGH) (I)	27	33	K2	C15	I/O
HDC (HIGH)	HDC (LOW)	HDC (HIGH)	HDC (HIGH)	HDC (HIGH)	28	34	K3	E14	I/O
LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	LDC (LOW)	30	36	L3	D16	I/O
INIT*	INIT*	INIT*	INIT*	INIT*	34	42	K6	H15	I/O
					43	53	L11	P15	XTL2 OR I/O
RESET (I)	RESET (I)	RESET (I)	RESET (I)	RESET (I)	44	54	K10	R15	RESET (I)
DONE	DONE	DONE	DONE	DONE	45	55	J10	R14	PROGRAM (I)
		DATA 7 (I)	DATA 7 (I)	DATA 7 (I)	46	56	K11	N13	I/O
					47	57	J11	T14	XTL1 OR I/O
		DATA 6 (I)	DATA 6 (I)	DATA 6 (I)	48	58	H10	P12	I/O
		DATA 5 (I)	DATA 5 (I)	DATA 5 (I)	49	60	F10	T11	I/O
		CS0 (I)			50	61	G10	R10	I/O
		DATA 4 (I)	DATA 4 (I)	DATA 4 (I)	51	62	G11	R9	I/O
		DATA 3 (I)	DATA 3 (I)	DATA 3 (I)	53	65	F11	R8	I/O
		CS1 (I)			54	66	E11	P8	I/O
		DATA 2 (I)	DATA 2 (I)	DATA 2 (I)	55	67	E10	R8	I/O
		DATA 1 (I)	DATA 1 (I)	DATA 1 (I)	56	70	D10	R7	I/O
		RDY/BUSY (I)	RCLK	RCLK	57	71	C11	R5	I/O
DIN (I)	DIN (I)	DATA 0 (I)	DATA 0 (I)	DATA 0 (I)	58	72	B11	P5	I/O
DOUT (O)	DOUT (O)	DOUT	DOUT	DOUT	59	73	C10	R3	I/O
CCLK	CCLK	CCLK	CCLK	CCLK	60	74	A11	N4	I/O
		WS (I)	A0	A0	61	75	B10	R2	CCLK (I)
		CS2 (I)	A1	A1	62	76	B9	P2	I/O
			A2	A2	63	77	A10	M3	I/O
			A3	A3	64	78	A9	P1	I/O
			A15	A15	65	81	B6	N1	I/O
			A4	A4	66	82	B7	M1	I/O
			A14	A14	67	83	A7	L2	I/O
			A5	A5	68	84	C7	K2	I/O
			A13	A13	2	2	A6	K1	I/O
			A6	A6	3	3	A5	H2	I/O
			A12	A12	4	4	B5	H1	I/O
			A7	A7	5	5	C5	F2	I/O
			A11	A11	6	8	A3	E1	I/O
			A8	A8	7	9	A2	D1	I/O
			A10	A10	8	10	B3	E3	I/O
			A9	A9	9	11	A1	C2	I/O

☐ REPRESENTS A 50K TO 100K OHM PULL-UP DURING CONFIGURATION
* INIT IS AN OPEN DRAIN OUTPUT DURING CONFIGURATION

Note: Pin assignments of "PGA Footprint" PLCC sockets and PGA packages are not electrically identical.

Table 2b. 3000 Family 68-Pin PLCC Pinouts

PLCC Pin Numbers	3020	PLCC Pin Numbers	3020
10	~PWRDN	44	~RESET
11	I/O	45	DONT-~PG
12	I/O	46	D7-I/O
13	I/O*	47	XTL1-I/O
14	I/O	48	D6-I/O*
15	I/O	49	D5-I/O
16	I/O	50	~CS0-I/O
17	I/O	51	D4-I/O*
18	VCC	52	VCC
19	I/O*	53	D3-I/O
20	I/O	54	~CS1-I/O
21	I/O*	55	D2-I/O*
22	I/O	56	D1-I/O
23	I/O	57	RDY/~BUSY-~RCLK-I/O
24	I/O	58	D0-DIN-I/O
25	M1-~RDATA	59	DOUT-I/O
26	M0-RTRIG	60	CCLK
27	M2-I/O	61	A0-~WS-I/O
28	HDC-I/O	62	A1-CS2-I/O
29	I/O	63	A2-I/O
30	~LDC-I/O	64	A3-I/O
31	I/O	65	A15-I/O
32	I/O	66	A4-I/O
33	I/O	67	A14-I/O
34	~INIT-I/O	68	A5-I/O
35	GND	1	GND
36	I/O	2	A13-I/O
37	I/O	3	A6-I/O
38	I/O	4	A12-I/O
39	I/O	5	A7-I/O
40	I/O	6	A11-I/O
41	I/O	7	A8-I/O
42	I/O	8	A10-I/O
43	XTL2-I/O	9	A9-I/O

*6 Unbonded IOBS 3020

The default configuration of IOBs is input with pull-up. This can be used to prevent an undefined pad level for unbonded or unused IOBs.

Table 2c. 3000 Family 84-Pin PLCC and PGA Pinouts

PLCC Pin Number	PGA Pin Number	3020	3030	PLCC Pin Number	PGA Pin Number	3020	3030
12	B2	~PWRDN	~PWRDN	54	K10	~RESET	~RESET
13	C2	I/O	I/O	55	J10	DONE-~PG	DONE-~PG
14	B1	N/C	I/O	56	K11	D7-I/O	D7-I/O
15	C1	I/O	I/O	57	J11	XTL1-I/O	XTL1-I/O
16	D2	I/O	I/O	58	H10	D6-I/O	D6-I/O
17	D1	I/O	I/O	59	H11	I/O	I/O
18	E3	I/O	I/O	60	F10	D5-I/O	D5-I/O
19	E2	I/O	I/O	61	G10	~CS0-I/O	~CS0-I/O
20	E1	I/O	I/O	62	G11	D4-I/O	D4-I/O
21	F2	I/O	I/O	63	G9	I/O	I/O
22	F3	VCC	VCC	64	F9	VCC	VCC
23	G3	I/O	I/O	65	F11	D3-I/O	D3-I/O
24	G1	I/O	I/O	66	E11	~CS1-I/O	~CS1-I/O
25	G2	I/O	I/O	67	E10	D2-I/O	D2-I/O
26	F1	I/O	I/O	68	E9	I/O	I/O
27	H1	I/O	I/O	69	D11	N/C	I/O
28	H2	I/O	I/O	70	D10	D1-I/O	D1-I/O
29	J1	I/O	I/O	71	C11	RDY/~BUSY-~RCLK-I/O	RDY/~BUSY-~RCLK I/O
30	K1	I/O	I/O	72	B11	D0-DIN-I/O	D0-DIN-I/O
31	J2	M1-~RDATA	M1-~RDATA	73	C10	DOUT-I/O	DOUT-I/O
32	L1	M0-~RTRIG	M0-~RTRIG	74	A11	CCLK	CCLK
33	K2	M2-I/O	M2-I/O	75	B10	A0-~WS-I/O	A0-~WS-I/O
34	K3	HDC-I/O	HDC-I/O	76	B9	A1-CS2-I/O	A1-CS2-I/O
35	L2	I/O	I/O	77	A10	A2-I/O	A2-I/O
36	L3	~LDC-I/O	~LDC-I/O	78	A9	A3-I/O	A3-I/O
37	K4	I/O	I/O	79	B8	N/C	I/O
38	L4	N/C	I/O	80	A8	N/C	I/O
39	J5	I/O	I/O	81	B6	A15-I/O	A15-I/O
40	K5	I/O	I/O	82	B7	A4-I/O	A4-I/O
41	L5	N/C	I/O	83	A7	A14-I/O	A14-I/O
42	K6	~INIT-I/O	~INIT-I/O	84	C7	A5-I/O	A5-I/O
43	J6	GND	GND	1	C6	GND	GND
44	J7	I/O	I/O	2	A6	A13-I/O	A13-I/O
45	L7	I/O	I/O	3	A5	A6-I/O	A6-I/O
46	K7	I/O	I/O	4	B5	A12-I/O	A12-I/O
47	L6	I/O	I/O	5	C5	A7-I/O	A7-I/O
48	L8	I/O	I/O	6	A4	N/C	I/O
49	K8	I/O	I/O	7	B4	N/C	I/O
50	L9	N/C	I/O	8	A3	A11-I/O	A11-I/O
51	L10	N/C	I/O	9	A2	A8-I/O	A8-I/O
52	K9	I/O	I/O	10	B3	A10-I/O	A10-I/O
53	L11	XTL2-I/O	XTL2-I/O	11	A1	A9-I/O	A9-I/O

The default configuration of IOBs is input with pull-up. This can be used to prevent an undefined pad level for unbonded or unused IOBs.

Table 2d. SC3000 Family 175-Pin PGA Pinouts

PGA Pin Number	3090	PGA Pin Number	3090	PGA Pin Number	3090	PGA Pin Number	3090
B2	~PWRDN	D13	I/O	R14	DONE-~PG	R3	D0-DIN-I/O
D4	I/O	B14	M1-~RDATA	N13	D7-I/O	N4	DOUT-I/O
B3	I/O	C14	VSS	T14	XTAL1-I/O	R2	CCLK
C4	I/O	B15	M0-~RTRIG	P13	I/O	P3	VCC
B4	I/O	D14	VCC	R13	I/O	N3	VSS
A4	I/O	C15	M2-I/O	T13	I/O	P2	A0-~WS-I/O
D5	I/O	E14	HDC-I/O	N12	I/O	M3	A1-CS2-I/O
C5	I/O	B16	I/O	P12	D6-I/O	R1	I/O
B5	I/O	D15	I/O	R12	I/O	N2	I/O
A5	I/O	C16	I/O	T12	I/O	P1	A2-I/O
C6	I/O	D16	~LDC-I/O	P11	I/O	N1	A3-I/O
D6	I/O	F14	I/O	N11	I/O	L3	I/O
B6	I/O	E15	I/O	R11	I/O	M2	I/O
A6	I/O	E16	I/O	T11	D5-I/O	M1	A15-I/O
B7	I/O	F15	I/O	R10	~CS0-I/O	L2	A4-I/O
C7	I/O	F16	I/O	P10	I/O	L1	I/O
D7	I/O	G14	I/O	N10	I/O	K3	I/O
A7	I/O	G15	I/O	T10	I/O	K2	A14-I/O
A8	I/O	G16	I/O	T9	I/O	K1	A5-I/O
B8	I/O	H16	I/O	R9	D4-I/O	J1	I/O
C8	I/O	H15	~INIT-I/O	P9	I/O	J2	I/O
D8	VSS	H14	VCC	N9	VCC	J3	VSS
D9	VCC	J14	VSS	N8	VSS	H3	VCC
C9	I/O	J15	I/O	P8	D3-I/O	H2	A13-I/O
B9	I/O	J16	I/O	R8	~CS1-I/O	H1	A6-I/O
A9	I/O	K16	I/O	T8	I/O	G1	I/O
A10	I/O	K15	I/O	T7	I/O	G2	I/O
D10	I/O	K14	I/O	N7	I/O	G3	I/O
C10	I/O	L16	I/O	P7	I/O	F1	I/O
B10	I/O	L15	I/O	R7	D2-I/O	F2	A12-I/O
A11	I/O	M16	I/O	T6	I/O	E1	A7-I/O
B11	I/O	M15	I/O	R6	I/O	E2	I/O
D11	I/O	L14	I/O	N6	I/O	F3	I/O
C11	I/O	N16	I/O	P6	I/O	D1	A11-I/O
A12	I/O	P16	I/O	T5	I/O	C1	A8-I/O
B12	I/O	N15	I/O	R5	D1-I/O	D2	I/O
C12	I/O	R16	I/O	P5	RDY/~BUSY-~RCLK-I/O	B1	I/O
D12	I/O	M14	I/O	N5	I/O	E3	A10-I/O
A13	I/O	P15	XTAL2-I/O	T4	I/O	C2	A9-I/O
B13	I/O	N14	VSS	R4	I/O	D3	VCC
C13	I/O	R15	~RESET	P4	I/O	C3	VSS
A14	I/O	P14	VCC				

The default configuration of IOBs is input with pull-up. This can be used to prevent an undefined pad level for unbonded or unused IOBs. Pins A2, A3, A15, A16, T1, T2, T3, T15 and T16 are not connected. Pin A1 does not exist.

PARAMETRICS

Absolute Maximum Ratings			Units
V_{CC}	Supply voltage relative to GND	–0.5 to 7.0	V
V_{IN}	Input voltage with respect to GND	–0.5 to V_{CC} + 0.5	V
V_{TS}	Voltage applied to three-state output	–0.5 to V_{CC} + 0.5	V
V_{STG}	Storage temperature (ambient)	–65 to +150	° C
T_{SOL}	Maximum soldering temperature (10 sec @ 1/16 in.)	+260	° C
T_J	Junction temperature plastic	+125	° C
	Junction temperature ceramic	+150	° C

*Note: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those listed under Recommended Operating Conditions is not implied. Exposure to Absolute Maximum Ratings conditions for extended periods of time may affect device reliability.

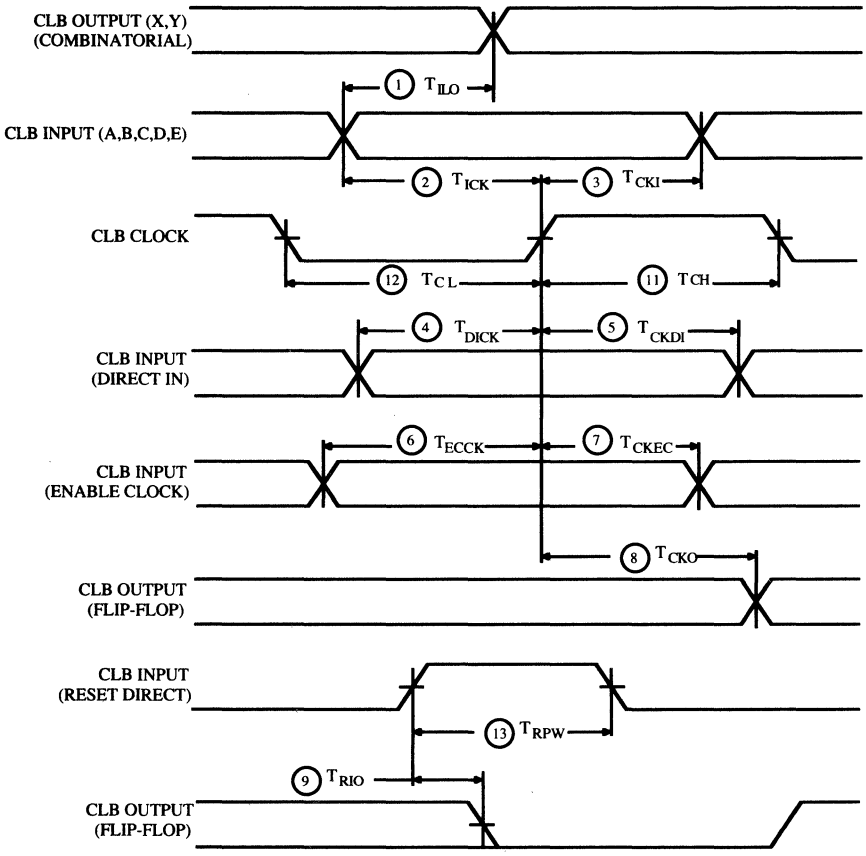
RECOMMENDED OPERATING CONDITIONS				MIN	MAX	UNITS
V_{CC}	Supply voltage relative to GND	Commercial	0° C to 70° C	4.75	5.25	V
	Supply voltage relative to GND	Industrial	–40° C to 85° C	4.5	5.5	V
	Supply voltage relative to GND	Military	–55° C to 125° C	4.5	5.5	V
V_{IHT}	High-level input voltage — TTL configuration			2.0	V_{CC}	V
V_{ILT}	Low-level input voltage — TTL configuration			0	0.8	V
V_{IHC}	High-level input voltage — CMOS configuration			.7 V_{CC}	V_{CC}	V
V_{ILC}	Low-level input voltage — CMOS configuration			0	.2 V_{CC}	V
T_{IN}	Input signal transition time				250	ns

ELECTRICAL CHARACTERISTICS OVER OPERATING CONDITIONS			MIN	MAX	UNITS
V_{OH}	High-level output voltage (@ I_{OH} = –4.0 ma V_{CC} min)	Commercial	3.86		V
V_{OL}	Low-level output voltage (@ I_{OL} = 4.0 ma V_{CC} min)			0.32	V
V_{OH}	High-level output voltage (@ I_{OH} = –4.0 ma V_{CC})	Industrial	3.76		V
V_{OL}	Low-level output voltage (@ I_{OL} = 4.0 ma V_{CC})			0.37	V
V_{OH}	High-level output voltage (@ I_{OH} = –4.0 ma V_{CC})	Military	3.7		V
V_{OL}	Low-level output voltage (@ I_{OH} = 4.0 ma V_{CC})			0.4	V
I_{CCPD}	Power-down supply current (V_{CC} = 5.0 V @ 70°C)[1]			500	µA
I_{CCO}	Quiescent LCA supply current in addition to I_{CCPD}[2]				
	CMOS chip thresholds			500	µA
	TTL chip thresholds			10	mA
I_{IL}	Leakage Current Commercial/Industrial Temperature		–10	+10	µA
	Leakage Current Military –55° C to 125° C		–20	+20	µA
C_{IN}	Input capacitance (sample tested)			10	pF
I_{RIN}	Pad pull-up (when selected) @ V_{IN} = 0V (sample tested)		0.02	0.17	mA
I_{RLL}	Horizontal long line pull-up (when selected) @ logic LOW		0.4	3.4	mA

Notes: 1. Based on a 3020, I_{CCPD} ratio for 3090, 5.0

2. With no output current loads, no active input or long line pull-up resistors, all package pins at V_{CC} or GND, and the LCA configured with a MAKEBITS "tie" option. See LCA power chart for additional activity dependent operating component.

CLB SWITCHING CHARACTERISTIC GUIDELINES



CLB SWITCHING CHARACTERISTIC GUIDELINES (Continued)

Testing of the switching characteristic guidelines is modeled after testing specified by MIL-M-38510/605. Devices are 100% functionally tested. Benchmark timing patterns are used to provide correlation to the switching characteristic guideline values. Actual worst-case timing is provided by the XACT Timing calculator or Simulation modeling.

Speed Grade				-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	
CLB Logic input	Combinatorial	1	T_{ILO}		14		9	ns
Reset direct	CLB output	9	T_{RIO}		15		10	ns
	Reset Direct width*	13	T_{RPW}		10		7	ns
	Master Rest pin to CLB out		T_{MRQ}		35		25	ns
CLB K Clock input	To CLB output	8	T_{CKO}		12		8	ns
	Additional for Q returning through F or G to CLB out		T_{QLO}		11		7	ns
	Logic-input setup	2	T_{ICK}	12		8		ns
	Logic-input hold	3	T_{CKI}	0		0		ns
	Data In setup	4	T_{DICK}	8		5		ns
	Data In hold	5	T_{CKDI}	6		4		ns
	Enable Clock setup	6	T_{ECCK}	10		7		ns
	Enable Clock hold	7	T_{CKEC}	0		0		ns
	Clock (high)*	11	T_{CH}	9		7		ns
	Clock (low)*	12	T_{CL}	9		7		ns
Flip-flop Toggle rate	Q through F/G to flip-flop in*		F_{CLK}	50		70		MHz

* These timing limits are based on calculations.

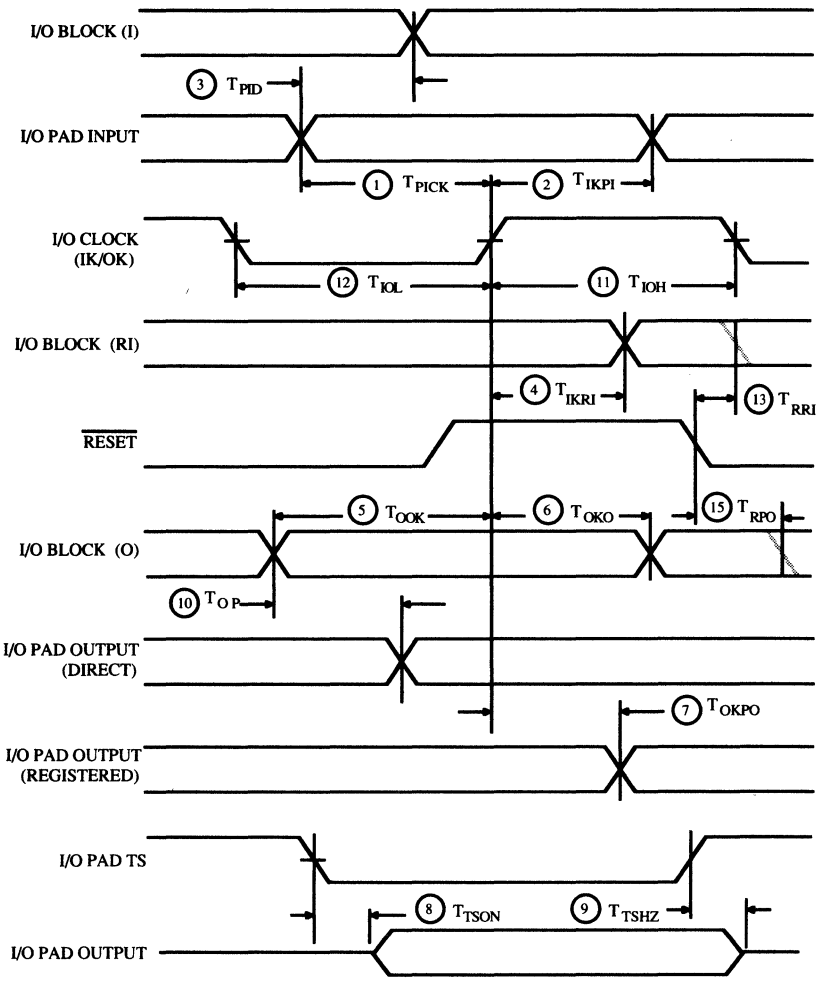
The speed of block inputs is a function of interconnect.

BUFFER (Internal) SWITCHING CHARACTERISTIC GUIDELINES

Speed Grade		-50		-70		Units
	Description	Min	Max	Min	Max	
Clock Buffer**	GCLK, ACLK		9		6	ns
TBUF**	Data to Output (Long line buffer)		8		5	ns
	Three-state to Output					
	Single pull-up resistor		34		22	ns
	Pair of pull-up resistors		17		11	ns
Bi-directional	BIDI		6		4	ns

** Timing is based on the 3020, for other devices see XACT timing calculator.

IOB SWITCHING CHARACTERISTIC GUIDELINES



IOB SWITCHING CHARACTERISTIC GUIDELINES (Continued)

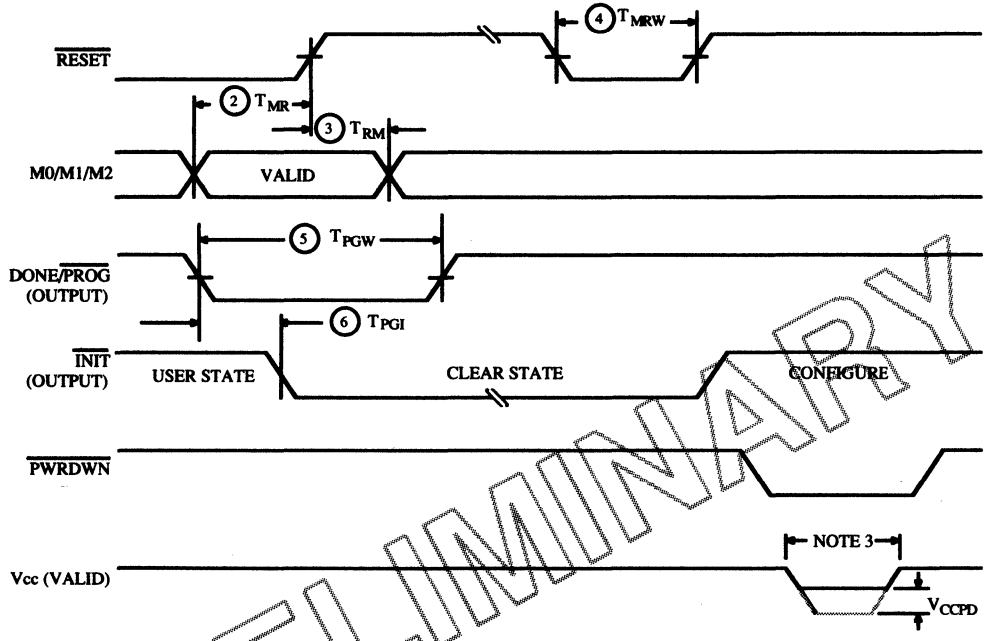
Testing of the switching characteristic guidelines is modeled after testing specified by MIL-M-38510/605. Devices are 100% functionally tested. Benchmark timing patterns are used to provide correlation to the switching characteristic guideline values. Actual worst-case timing is provided by the XACT Timing calculator or Simulation modeling.

Speed Grade				-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	
Pad (package pin)	To input (direct CLKIN)		T_{PIDC}		5		3	ns
	To input (direct)	3	T_{PID}		10		7	ns
I/O Clock	To I/O RI input (FF)	4	T_{IKRI}		10		7	ns
	I/O pad-input setup	1	T_{PICK}	30		20		ns
	I/O pad-input hold	2	T_{IKPI}	0		0		ns
	To I/O pad (fast)	7	T_{OKPO}		18		13	ns
	I/O pad output setup	5	T_{OOK}	15		10		ns
	I/O pad output hold	6	T_{OKO}	0		0		ns
	Clock (high)	11	T_{IOH}	9		7		ns
	Clock (low)	12	T_{IOL}	9		7		ns
Output	To pad (enabled fast)	10	T_{OPF}		14		10	ns
	To pad (enabled slow)	10	T_{OPS}		33		25	ns
Three-state	To pad begin hi-Z (fast)	9	T_{TSHZ}		12		8	ns
	To pad valid (fast)	8	T_{TSON}		20		14	ns
Master Reset	To input RI	13	T_{RRI}		45		30	ns
(Package Pin)	To output (FF)	15	T_{RPO}		55		37	ns

Notes: 1. Timing is measured at pin threshold, with 50 pF external capacitive loads (incl. test fixture). Typical fast mode output rise/fall times are 5 ns and will increase approximately 2%/pF. A maximum total external capacitive load for simultaneous fast mode switching in the same direction is 500 pF per power/ground pin pair.

2. Voltage levels of unused (bonded and unbonded) pads must be valid logic levels. Each can be configured with the internal pull-up resistor or alternatively configured as a driven output or driven from an external source.

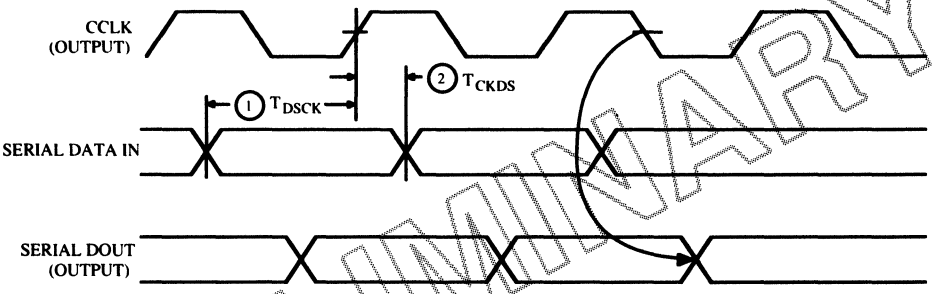
GENERAL LCA SWITCHING CHARACTERISTICS



				–50		–70		Units
	Description	Symbol		Min	Max	Min	Max	
~RESET (2)	M2, M1, M0 setup	2	T_{MR}	1		1		µs
	M2, M1, M0 hold	3	T_{RM}	1		1		µs
	Width (low) Abort	4	T_{MRW}	6		6		µs
DONE/~PROG	Program width (low)	5	T_{PGW}	6		6		µs
	Start ~INIT	6	T_{PGI}		7		7	µs
~PWRDWN (3)	Power Down V_{CC}		V_{CCPD}		2.0		2.0	V

Notes: 1. V_{CC} must rise from 2.0 Volts to V_{CC} minimum in less than 10 ms for master modes.
2. RESET timing relative to valid mode lines (M0, M1, M2) is relevant when ~RESET is used to delay configuration.
3. ~PWEDWN transitions must occur during operational V_{CC} levels.

MASTER SERIAL MODE SWITCHING CHARACTERISTICS GUIDELINES

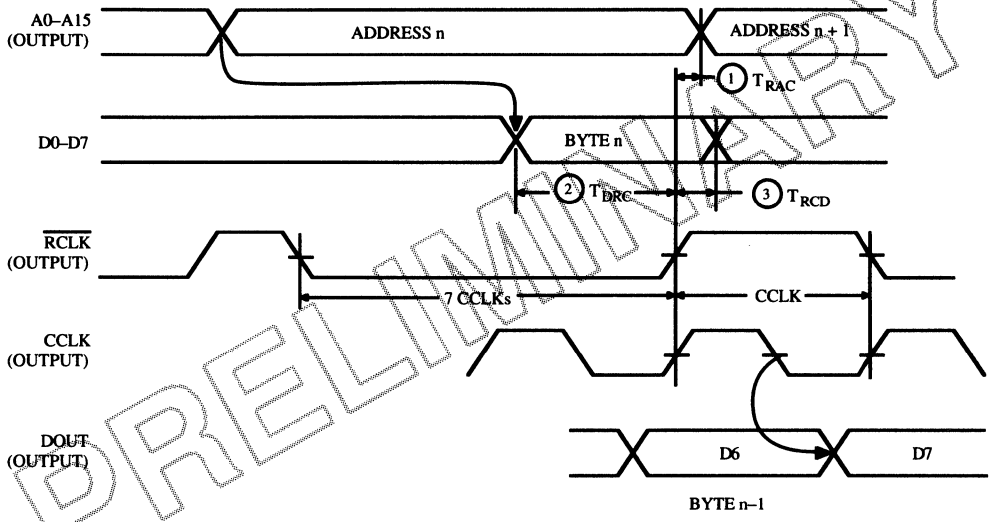


				-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	
CCLK[3]	Data In setup	1	T_{DSCK}	60		60		ns
	Data In hold	2	T_{CKDS}	0		0		ns

Notes:
1. At power-up, V_{CC} must rise from 2.0 V to V_{CC} min. in less than 10 ms, otherwise delay configuration using ~RESET until VCC is valid.
2. Configuration can be controlled by holding ~RESET low with or until after the ~INIT of all daisy-chain slave mode devices is HIGH.
3. Master serial mode timing is based on slave mode testing.

MASTER PARALLEL MODE PROGRAMMING SWITCHING CHARACTERISTIC GUIDELINES

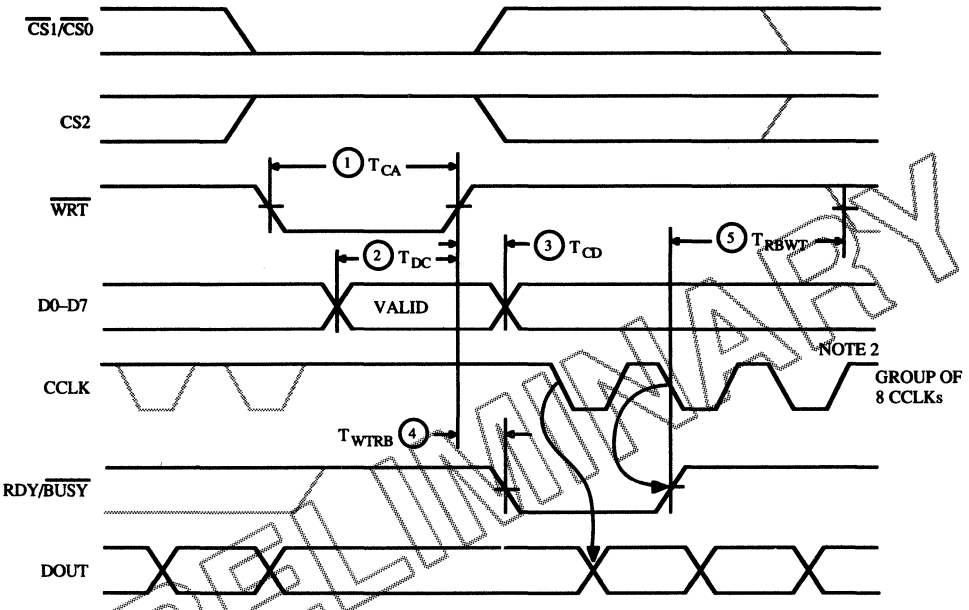
Testing of the switching characteristic guidelines is modeled after testing specified by MIL-M-38510/605. Devices are 100% functionally tested. Benchmark timing patterns are used to provide correlation to the switching characteristic guideline values. Actual worst-case timing is provided by the XACT Timing calculator or Simulation modeling.



				–50		–70		Units
	Description	Symbol		Min	Max	Min	Max	
RCLK	To address valid	1	T_{RAC}	0	200	0	200	ns
	To data setup	2	T_{DRC}	60		60		ns
	To data hold	3	T_{RCD}	0		0		ns
	RCLK high		T_{RCH}	600		600		ns
	RCLK low		T_{RCL}	4.0		4.0		µs

Notes: 1. At power-up, V_{CC} must rise from 2.0 V to V_{CC} min. in less than 10 ms, otherwise delay configuration using ~RESET until V_{CC} is valid.
2. Configuration can be controlled by holding ~RESET low with or until after the ~INIT of all daisy-chain slave mode devices is HIGH.

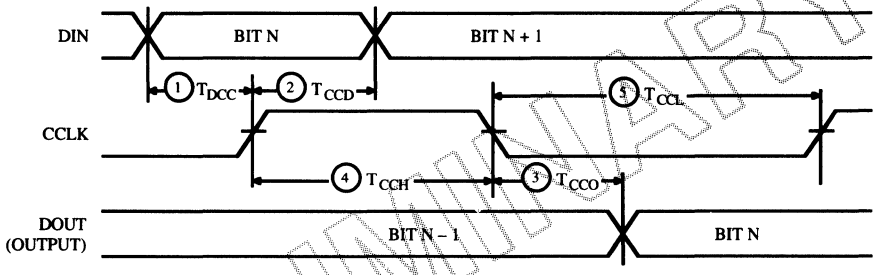
PERIPHERAL MODE PROGRAMMING SWITCHING CHARACTERISTICS



				–50		–70		Units
	Description	Symbol		Min	Max	Min	Max	
Write	WRT Low	1	T_{CA}	0.5		0.5		µs
	DIN setup	2	T_{DC}	60		60		ns
	DIN hold	3	T_{CD}	0		0		ns
	Ready/Busy	4	T_{WTRB}		60		60	ns
RDY	WRT Active	5	T_{RBWT}	0		0		ns

Notes: 1. Configuration must be delayed until the ~INIT of all LCAs is HIGH.
2. Time from end of WRT to CCLK cycle for the new byte of data depends on completion of previous byte processing and the phase of the internal timing generator for CCLK.
3. CCLK and DOUT timing is tested in slave mode.

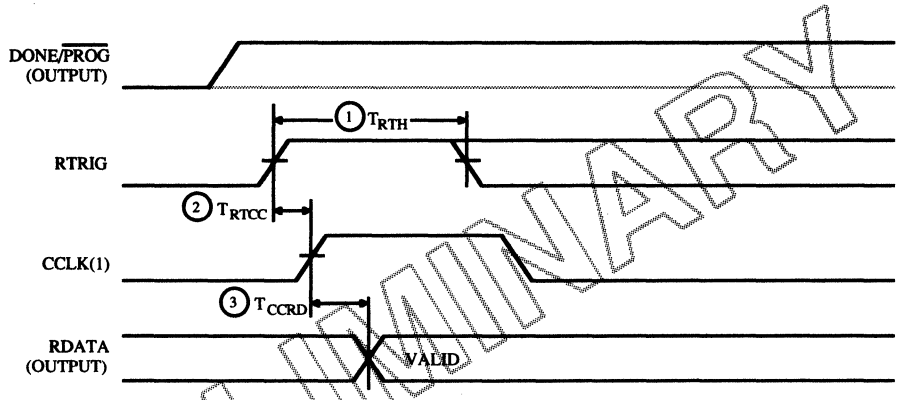
SLAVE MODE PROGRAMMING SWITCHING CHARACTERISTICS



				–50		–70		Units
	Description	Symbol		Min	Max	Min	Max	
CCLK	To DOUT	3	T_{CCO}		100		100	ns
	DIN setup	1	T_{DCC}	60		60		ns
	DIN hold	2	T_{CCD}	0		0		ns
	High time	4	T_{CCH}	0.5		0.5		µs
	Low time	5	T_{CCL}	0.5	5.0	0.5	5.0	µs
	Frequency		F_{CC}		1		1	MHz

Note: Configuration must be delayed until the INIT of all LCAs is HIGH.

PROGRAM READBACK SWITCHING CHARACTERISTICS



				-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	
RTRIG	RTRIG high	1	T_{RTH}	250		250		ns
CCLK	RTRIG setup	2	T_{RTCC}	10		10		µs
	~RDATA delay	3	T_{CCRD}		100		100	ns

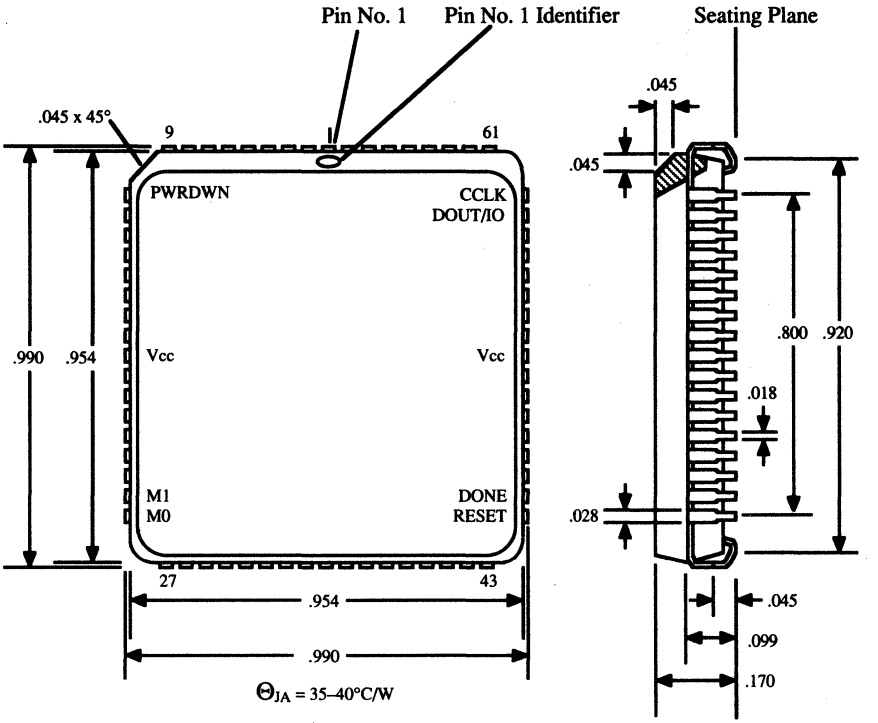
Notes:
1. CCLK and DOUT timing are the same as for slave mode.
2. RETRIG (M0 positive transition) shall not be done until after one clock following active I/O pins.
3. Readback should not be initiated until configuration is complete.

SOCKET INFORMATION

The following sockets, with matching hole patterns, are available for PLCC devices.

DESCRIPTION 68 PIN	VENDOR	PART NUMBER
PCB solder tail, tin plate	AMP	821574-1
Surface mount, tin plate	AMP	821542-1
PCB solder tail, tin plate*	Burndy	QILE68P-410T
PCB solder tail, tin plate*	Midland-Ross	709-2000-068-4-1-1
PCB solder tail, tin plate*	Methode	213-068-001
Surface mount, tin plate	Methode	213-068-002

*Sockets will plug into pin-grid array wire-wrap sockets for breadboard use. However, the physical translation of pins in a PLCC socket does not result in electrical equivalence to the pin locations of a PGA package.

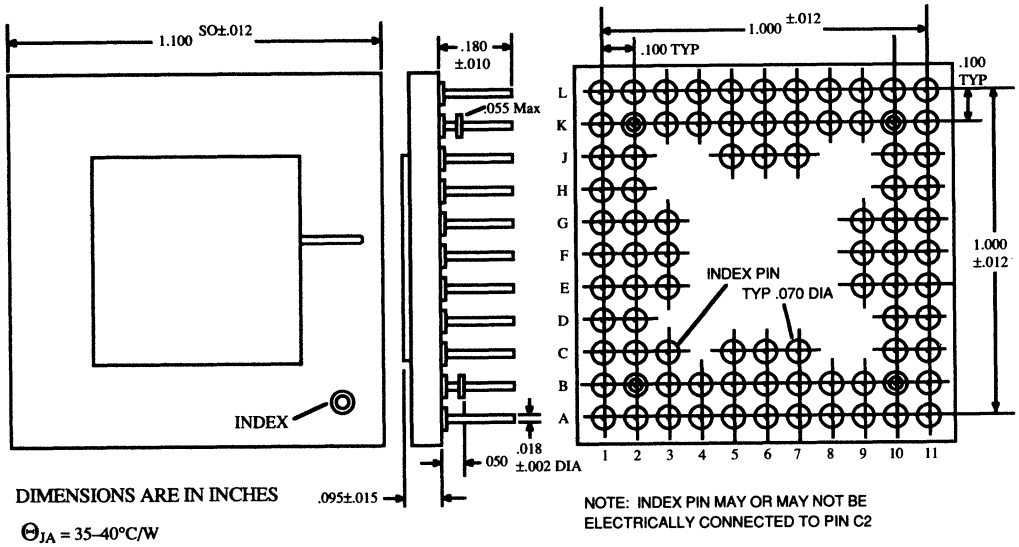


PIN SPACING
.050 TYPICAL

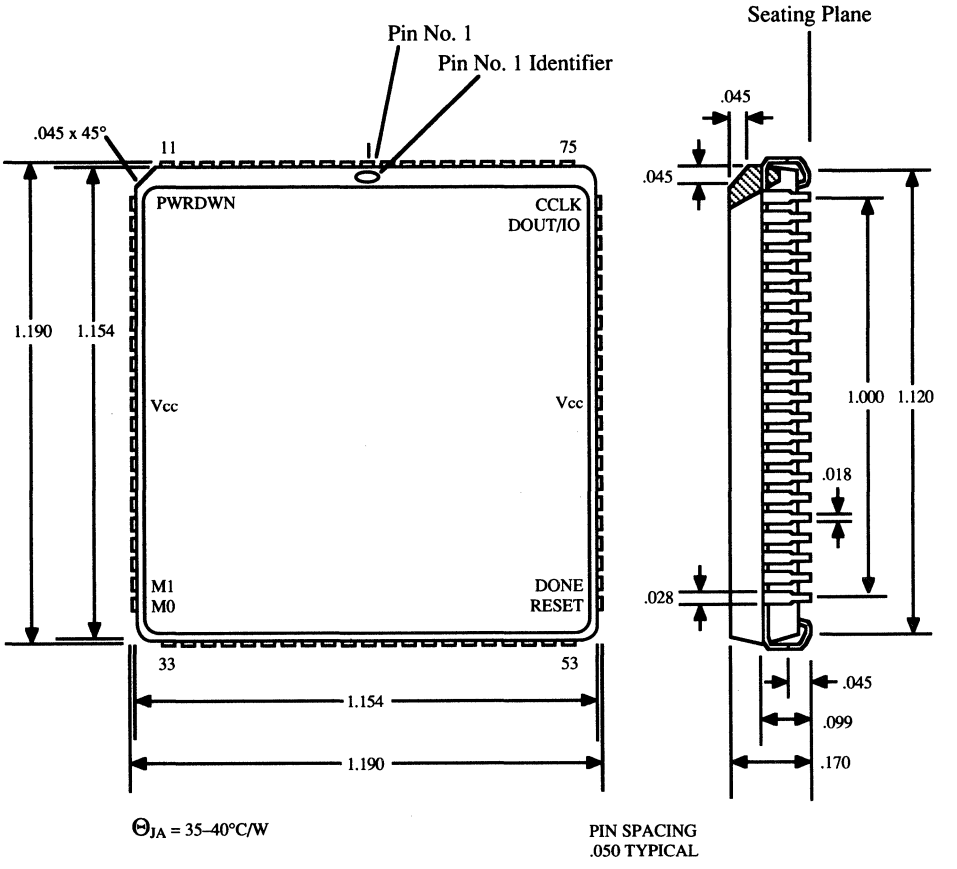
LEAD CO-PLANARITY ± 0.002 IN

ALL OTHER DIMENSIONS ARE
IN INCHES ± 0.005

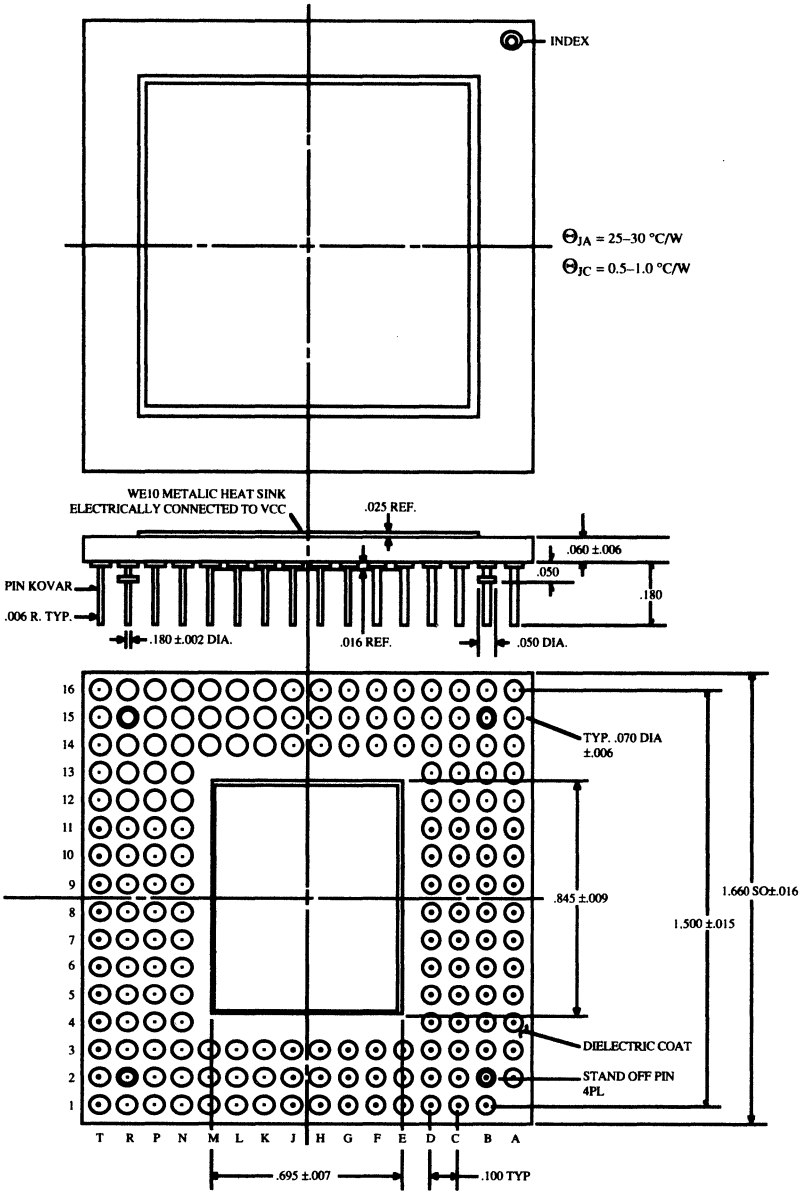
68-Pin PLCC Package



84-Pin PGA Package



84-Pin PLCC Package



175-Pin PGA Package

LCA-MDS21 XACT Design Editor System

Features

- **Runs on an IBM® PC-XT™ or compatible computer**
- **Complete basic system for designing with Logic Cell Arrays**
- **Interactive graphical design editor**
- **Simplified definition, placement and interconnection capability for logic design and implementation**
- **Macro library of 113 standard logic family equivalents**
- **Utility for user-defined macros**
- **Boolean equation or Karnaugh map alternatives to specify logic functions**
- **Point-to-point timing calculations for critical path analysis**
- **Automatic design consistency checking for connectivity and design violations**
- **Documentation support with hardcopy output of logic and physical configuration information**
- **Download cable to transfer configuration programs from personal computer to LCA in target system**
- **Compatible hardware and software options to enhance design productivity**
- **File formatter for EPROM programmer**

General

The XACT Design Editor provides users with a complete design and development system for specification and implementation of designs using Monolithic Memories' Logic Cell Arrays. Functional definition of Configurable Logic Blocks (CLBs), Input/Output Blocks (IOBs) and interconnection is performed with a menu-driven interactive graphics editor. An automatic router greatly reduces the effort to interconnect logic.

Designs are captured with a graphics-based design editor using either a mouse for menu-driven entry, or a keyboard for command-driven entry. Functions are specified by CLB and IOB definitions plus their interconnections. The macro library and user-defined macros enable the user to easily implement complex functions.

The check for logic connectivity and design rule violation is easily performed. All unused internal nodes are automatically configured to minimize power dissipation.

Interactive point-to-point timing delay calculation is provided for timing analysis and critical path determination. This ability enables the user to quickly identify and correct timing problems while the design is in progress.

Automatic generation of similar input netlist files with timing parameters simplifies the use of P-SILOS for logic and timing simulation.

The XACT Design Editor includes hardcopy generation to document a design and automatically track design changes. Logic Cell Array configuration programs can be automatically translated into standard EPROM programming bit pattern formats.

A download cable included with XACT is useful for transferring configuration programs serially from the PC workstation to a Logic Cell Array installed in a system. During product development and debug this capability can be used to save the time required to write a modified configuration program into an EPROM.

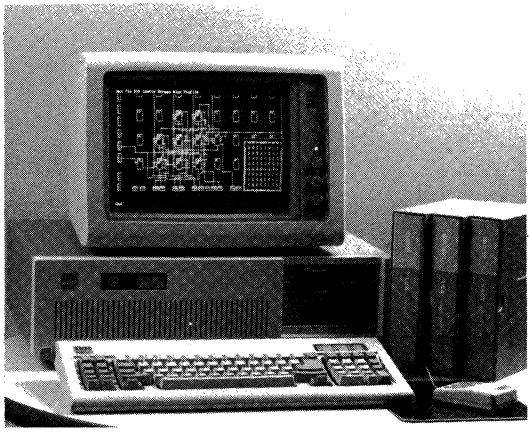
Monolithic Memories provides ongoing support for XACT users. For the first year, software updates are included. After that the user may purchase the LCA-MSC21 Annual Support Agreement to continue to receive the latest software releases. XACT users also receive Monolithic Memories' technical information, which includes information about Logic Cell Arrays and PAL® devices, as well as software updates and application notes for designers. In addition, Monolithic Memories provides comprehensive field and factory support.

System Requirements

Minimum System Configuration

IBM PC-XT, PC-AT or compatible computer with:

- MS-DOS™ 2.1 or higher
- 1M Bytes RAM
- 1 Diskette Drive
- 10-MB Hard Disk
- IBM compatible Color Graphic Adapter and Display
- 1 Serial Interface Port
- 1 Parallel Interface Port
- Mouse System™, Microsoft® or compatible mouse



Design Editor with Routed Design

Publication #	Rev.	Amendment
10853	A	/0

Issue Date: **January 1988**

General		CLBs
GADD	Adder	1
GCOMP	Compare	1
GEQGT	Equal or Greater	1
GMAJ	Majority	1
GMux	2-to-1 Mux	1
GPAR	Parity	1
GXOR	Exclusive-OR	1
GXOR2	Dual Exclusive-OR	1
GXTL	Crystal Oscillator	0 + 2IOB
GOSC	Low Frequency Resistor-Capacitor Oscilator	1 + 2IOB

Pads		IOBs
PIN	Input Pad	1
PINQ	Input Pad with Storage	1
PIO	Input/Output Pad	1
PIOQ	Input/Output Pad with Input Storage	1
PIOC	Input/Output Pad with 'Open Collector'	1
PIOQC	Input/Output Pad with Input Storage, 'Open Collector'	1
POUT	Output Pad	1
POUTC	Output Pad with 'Open Collector'	1
POUTZ	Output Pad with 3-State Control	1
PREG	Output Pad with Input Storage	1

Latches		CLBs
LD	Data Latch	1
LC-rd	Data Latch with ResetDir	1
LC-sd	Data Latch with SetDir	1
LD-srd	Data Latch with SetDir, ResetDir	1
LDM	Data Latch with 2-Input Data Mux	1
LDM-rd	Data Latch with 2-Input Data Mux, ResetDir	1
LDM-sd	Data Latch with 2-Input Data Mux, SetDir	1

Flip-Flops		CLBs
FD	D Flip-Flop	1
FDR	D Flip-Flop with Reset	1
FDS	D Flip-Flop with Set	1
FD-rd	D Flip-Flop with ResetDir	1
FD-sd	D Flip-Flop with SetDir	1
FD-srd	D Flip-Flop with SetDir, ResetDir	1
FDC	D Flip-Flop with ClkEna	1
FDCR	D Flip-Flop with ClkEna, Reset	1
FDCS	D Flip-Flop with ClkEna, Set	1
FDM	D Flip-Flop 2-Input Data Mux	1
FDMR	D Flip-Flop 2-Input Data Mux, Reset	1
FDMS	D Flip-Flop 2-Input Data Mux, Set	1
FDM-rd	D Flip-Flop 2-Input Data Mux, ResetDir	1
FDM-sd	D Flip-Flop 2-Input Data Mux, SetDir	1
FSR	Set-Reset Flip-Flop with Set Dominate	1
FRS	Set-Reset Flip-Flop with Reset Dominate	1
FJK	J-K Flip Flop	1
FJKS	J-K Flip Flop with Synchronous Set	1
FJK-rd	J-K (Set-Reset) Flip Flop with ResetDir	1
FJK-sd	J-K (Set-Reset) Flip Flop with SetDir	1
FJK-srd	J-K (Set-Reset) Flip Flop with SetDir, ResetDir	1
FT0	Self Toggle Flip-Flop	1
FT0R	Self Toggle Flip-Flop with Reset	1
FT	Toggle Flip-Flop	1
FTP	Toggle Flip-Flop with ParEna	1
FTP-rd	Toggle Flip-Flop with ParEna, ResetDir	1
FTR	Toggle Flip-Flop with Reset	1
FTS	Toggle Flip-Flop with Set	1
FT2	2-Input Toggle Flip-Flop	1
FT2R	2-Input Toggle Flip-Flop with Reset	1

Decoders		CLBs
D2-4	1-of-4 Decoder	2
D2-4E	1-of-4 Decoder, with Ena	2
74-139	1-of-4 Single Decoder with Low Output, Ena	4
D3-8	1-of-8 Decoder	5
D3-8E	1-of-8 Decoder with Ena	6
74-138	1-of-8 Decoder with Enables, Low Output	7
74-42	1-of-10 Decoder with Low Output	8

Multiplexers		CLBs
M3-1	3-to-1 Mux	2
M3-1E	3-to-1 Mux with Ena	2
M4-1	4-to-1 Mux	3
M4-1E	4-to-1 Mux with Ena	3
74-352	4-to-1 Mux with Low Output, Ena	3
M8-1	8-to-1 Mux	7
M8-1E	8-to-1 Mux with Ena	7
74-151	8-to-1 Mux with Ena Complementary Outputs	7
74-152	8-to-1 Mux with Low Output	7

Registers

		CLBs
Data Registers		
RD4	4-Bit Data Register	4
RD8	8-Bit Data Register	8
RE8CR	8-Bit Data Register with ClkEna, Reset	8
Serial to Parallel		
RS4	4-Bit Shift Register	4
74-195	4-Bit Serial to Parallel Shift Register with ParEna, Reset	5
74-194	4-Bit Bidirectional Shift Register with ClkEna, ParEna, ResetDir	12
RS8	8-Bit Shift Register	8
RS8CR	8-Bit Shift Register with ClkEna, Reset	8
RS8PR	8-Bit Shift Register with ParEna, Reset	8
RS8R	8-Bit Shift Register with Reset	8
74-164	8-Bit Serial to Parallel Shift Register with ResetDir	8

Counters

		CLBs
Modulo 2		
C2BCR	1-Bit Binary Counters with ClkEna, Reset	1
C2BC-rd	1-Bit Binary Counters with ClkEna, ResetDir	1
C2BP	1-Bit Binary Counters with ParEna	1
C2BR	1-Bit Binary Counters with Reset	1
C2B-rd	1-Bit Binary Counters with ResetDir	1
Modulo 4		
C4BCP	2-Bit Binary Counters with ClkEna, ParEna	3
C4BCR	2-Bit Binary Counters with ClkEna, Reset	2
C4BC-rd	2-Bit Binary Counters with ClkEna, ResetDir	2
C4JCR	2-Bit Johnson Counters with ClkEna, Reset	2
Modulo 6		
C6JCR	3-Bit Johnson Counter with ClkEna, Reset	3
Modulo 8		
C8BCP	3-Bit Binary Counters with ClkEna, ParEna	5
C8BCR	3-Bit Binary Counters with ClkEna, Reset	4
C8BC-rd	3-Bit Binary Counters with ClkEna, ResetDir	4
C8JCR	3-Bit Johnson Counters with ClkEna, Reset	4
Modulo 10		
C10BC-rd	4-Bit BCD Counter with ClkEna, ResetDir	4
C10BCP-rd	4-Bit BCD Counter with ClkEna, ParEna, ResetDir	7
74-160	4-Bit BCD Counter with ClkEna, ParEna, ResetDir	8
C10BP-rd	4-Bit BCD Counter with ParEna, ResetDir	6
C10JCR	5-Bit Johnson Counter with ClkEna, Reset	5
Modulo 12		
C12JCR	6-Bit Johnson Counter with ClkEna, Reset	6
Modulo 16		
C16BA-rd	4-Bit Binary Ripple Counter with ResetDir	4
C16BC-rd	4-Bit Binary Counter with ClkEna, ResetDir	4
C16BCPR	4-Bit Binary Counter with ClkEna, ParEna, Reset	10
C16BCP-rd	4-Bit Binary Counter with ClkEna, ParEna, ResetDir	6
74-161	4-Bit Binary Counter with ResetDir	8
C16BP-rd	4-Bit Binary Counter with ParEna, ResetDir	5
C16BUD-rd	4-Bit Binary Up-Down Counter with ParEna, ResetDir	8
C16JCR	8-Bit Johnson Counter with ClkEna, Reset	8
Modulo 256		
C256FC-rd	8-Bit Modulo 256 Feedback Shift Register with ClkEna, ResetDir	9

LCA-MDS22 P-SILOS Simulator

Features

- **Event-driven logic and timing simulator**
- **Logic network input automatically generated by XACT Design Editor**
- **Control and observation of any physical circuit node**
- **Multiple file input for vectors and commands**
- **Interactive or batch mode operation**
- **Output available in printed or tabular formats**
- **Runs on an IBM PC-XT, PC-AT or compatible personal computer**

General

P-SILOS is a powerful PC-based simulator that provides event-driven logic and timing simulation of Logic Cell Array designs. Simulation is particularly useful for testing logic or logic segments as well as for verifying critical timing over worst case power supply, temperature and process conditions.

Simulation is useful in several stages of the design cycle. After design entry, simulation may be used to debug logic in an unplaced and unrouted design. This saves design time because logic errors can be detected and corrected prior to final placement and routing. After a circuit has been placed, routed, and then fully debugged using in-circuit emulation, worst case timing may be verified. This enables the user to select the correct Logic Cell Array speed for a particular application.

Network inputs for Logic Cell Array designs are automatically created by the Simgen utility in the XACT system. The network includes logic and routing delay parameters and setup and hold times based upon the selected speed grade operating under worst case conditions. Simulation stimuli are created with a set of clock statements or with an input pattern for either pad inputs or internal nodes. Simulation results are available in tabular, plotted, and graphic formats. This flexibility makes debugging easy for both the circuit function and timing.

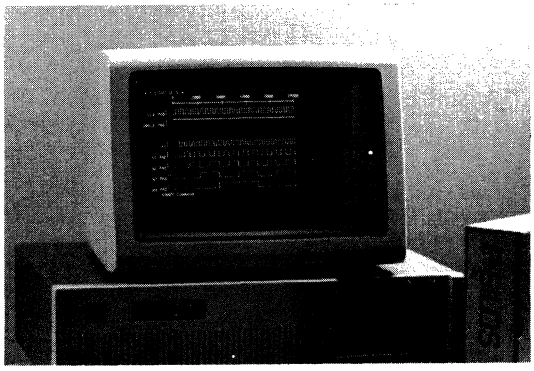
System Requirements

Minimum System Configuration

IBM PC-XT, PC-AT or compatible computer with:

- MS-DOS 2.1 or higher
- 640 K Bytes RAM
- 1 Diskette Drive
- 10-MB Hard Disk
- 1 Parallel Interface Port

Refer to the MDS21 XACT Design Editor Product Datasheet for additional equipment required for systems which will also run the XACT Design Editor.



P-SILOS Waveform Output

Publication #	Rev.	Amendment
10854	A	/0

Issue Date: January 1988

LCA-MDS23

Automatic Design Implementation

Distinctive Characteristics

- Automatic elimination of disabled and unused logic
- Automatic partitioning of the schematic into LCA resources.
- Automatic placement and routing of logic to minimize design cycle time
- User control over placement of logic blocks
- User specification of critical paths
- May be used in conjunction with schematic capture or with the XACT Design Editor
- Runs on an IBM PC/AT or compatible personal computer

General Description

The automatic logic reduction and partitioning software included with in the Automatic Design Implementation package eliminates unused and redundant logic, then partitions the schematic into LCA physical resources, logic and I/O blocks. Automated reduction and partitioning allows designers to immediately determine the programmable gate array size required, during design definition and entry.

The Automatic Design Implementation package enhances the productivity of designers using Programmable Gate Arrays by reducing design placement and routing time, whether the design logic is entered from a schematic capture package or from the XACT Design Editor.

Designs that are development incrementally can also take advantage of automatic placement and routing. Partial LCA layouts can be locked in place while additions to the design are automatically placed and routed, or the design can be completely rearranged to yield a new placement.

The automatic placement and routing program is extremely flexible. Through placement directives, the user can control the placement process to achieve the best placement for a particular design. Routing resources can be specified to minimize clock skews and signal delays for critical paths. The result is faster product development.

Publication #	Rev.	Amendment
10851	A	/0
Issue date: June 1988		

LCA-MDS24, LCA-MDS26, LCA-MDS27 XACTOR In-Circuit Emulator

Features

- **Real-time in-circuit emulation in user's target system**
- **Concurrent emulation of up to four devices**
- **Readback and display of Logic Cell Array internal storage element states**
- **Device status display with automatic update of asynchronous events**
- **Control and I/O pin isolation from target system**
- **Support for daisy chain programming of up to seven devices in a daisy chain**
- **On-chip crystal oscillator support during emulation**
- **Support for multiple device and package types**
- **Runs on an IBM PC-XT, PC-AT or compatible personal computer**

General

The XACTOR real-time in-circuit emulator provides interactive target-system emulation of up to four Logic Cell Arrays from the host PC system. In-circuit emulation provides a powerful productivity enhancement to simulation, providing capabilities to verify functionality in the target system at full speed with all other circuits and system software.

The emulation system is composed of a microcomputer-based controller (LCA-MDS24), and from one to four universal emulation pods (LCA-MDS26), each with a package-specific emulation header (LCA-MDS27). One universal emulation pod is included with the system. The controller is connected to the host PC through a serial port and provides local storage of configuration programs, control of individual device configurations and control of the isolation of the pod device(s) from the target system. The user can set the state and isolation for each of the control signals to provide debugging of target hardware. Four general I/O pins are available to provide test points which may also be isolated from the target system.

Target Logic Cell Arrays can be programmed individually or in a daisy chain. Daisy chains of up to seven devices may be supported from any of the four pods. Individual device isolation and configuration is controlled with mouse or keyboard commands and may be supplemented with user-defined setup files for easy system debugging.

Readback of device configuration may be performed on command for verification of the configuration process and interrogation of the internal states. The state of all internal storage elements is displayed after readback has been performed. Status displays showing the state of all isolation switches and control signal states are provided. The status display includes automatic reporting of asynchronous status changes in the target system.

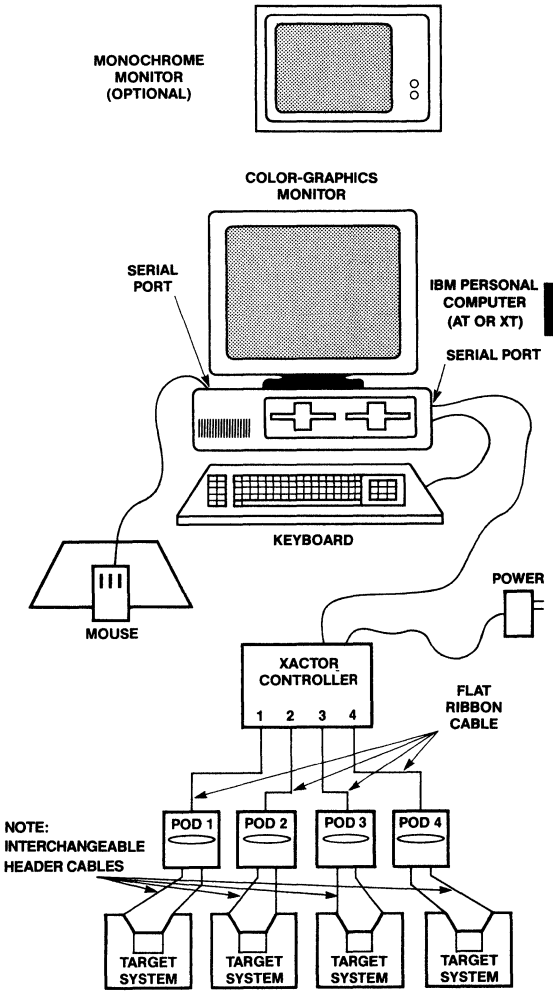
Universal In-Circuit Emulator Pod (LCA-MDS26)

Additional pods may be connected to the XACTOR in-circuit emulator controller, up to a maximum of four pods per controller. Pod headers (LCA-MDS27) are interchangeable for different device and package types. Each pod provides a direct in-socket connection for a minimum disruption of the target system. Test points are provided to allow connection of a logic analyzer or other test equipment to aid in the system debugging.

System Requirements

Minimum System Configuration

IBM PC-XT, PC-AT or compatible computer as configured for MDS21 XACT Design Editor, plus second serial interface port.



XACTOR Hardware

LCA-MDS31/LCA-MDS33/LCA-MDS34/LCA-MDS35

Schematic Design Entry Interface for FutureNet, Daisy, Mentor, OrCAD

Distinctive Characteristics

- Use of familiar design entry methodology through the choice of schematic entry package.
- Macro library of over 100 TTL and standard logic family equivalents.
- User-control of flagging critical paths for later use of Automatic Design Implementation (LCA-MDS23)
- Output compatibility with other parts of the PGA Development System software.

General Description

The Schematic Design Entry Interfaces are software packages specifically designed to enable users of FutureNet, Daisy, Mentor and OrCAD workstations to create Logic Cell Array designs using the design entry editors supplied with their schematic capture software.

The user creates a design file on the workstation with the LCA symbol library. A conversion routine automatically eliminates unused and disabled logic and converts the design file into a format that can be read by other PGA Development System tools.

Product Selector Guide

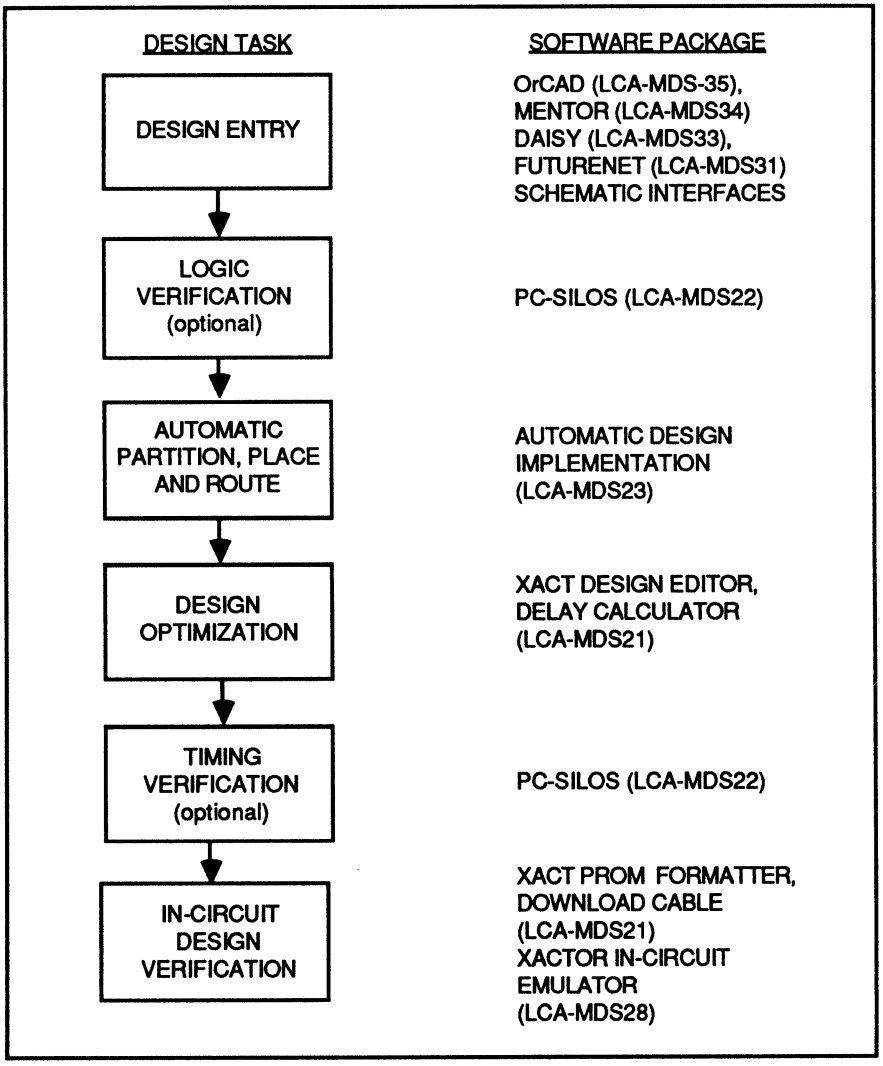
Product Number	Product Name
LCA-MDS31	FutureNet DASH Schematic Entry Interface and LCA Library
LCA-MDS33	Daisy Schematic Entry Interface and LCA Library
LCA-MDS34	Mentor Schematic Entry Interface and LCA Library
LCA-MDS35	OrCAD Schematic Entry Interface and LCA Library

Publication #	Rev.	Amendment
10849	A	/0

Issue date: June 1988

PGA Design Cycle

The PGA Development System tools includes software for all phases of the PGA design cycle. The diagram below indicates the design tasks needed to complete a PGA design and the corresponding software products used to complete the tasks.



LCA-MDS151/LCA-MDS152

PGA Development System/PGA Design Entry System

Distinctive Characteristics

- Complete systems for entry and implementation of Programmable Gate Array designs.
- Includes OrCAD Schematic Design Tools and Schematic Design Interface (LCA-MDS135) and Automatic Design Implementation (LCA-MDS23). LCA-MDS151 also includes XACT PGA Design Editor (LCA-MDS21).
- Schematic editor provides hierarchical PGA design capability.
- LCA macro library of over 100 TTL and standard family equivalents.
- Automatic logic reduction and partitioning removes unused and disabled logic.
- Automatic placement and routing minimizes design cycle time.
- PGA Design Editor provides interactive physical editing, timing calculator and design consistency checking.
- Download cable transfers configuration bitstreams directly from the PC to the PGA during design debugging.
- Runs on an IBM PC-AT or compatible computer.

General Description

The PGA Development System offers complete capabilities for schematic entry and implementation of Programmable Gate Array design. Packaged together are the OrCAD SDT III Schematic Design Entry software, Automatic Design Implementation and the PGA Design Editor. The general purpose OrCAD schematic editor provides powerful design entry and documentation capabilities. An extensive macro library of TTL and standard logic equivalents is included. Designers can implement their own macro functions as desired. Designs can be entered in a true hierarchical structure to manage the complexity of large designs. In additon, designers can flag critical timing paths to ensure critical signals are routed with minimum delay.

The Automatic Design Implementation software eliminates unused and redundant logic, then partitions the schematic into LCA resources, logic and I/O blocks.

Automated reduction and partitioning allows designers to immediately determine the programmable gate array size required, during design definition and entry. Automatic placement and routing can be used to arrange logic blocks and calculate routes.

Designers can interactively document, design-check and edit the resulting physical design with the graphics-based XACT Design Editor. A built-in timing calculator permits point-to-point timing calculations for critical path analysis.

When a design is ready for debugging or production, the design is compiled in to an LCA configuration program using the XACT Design Editor. During design debug, designers can save time by using the download cable to transfer the configuration program from the PC directly into an LCA under development in a system.

Product Selector Guide

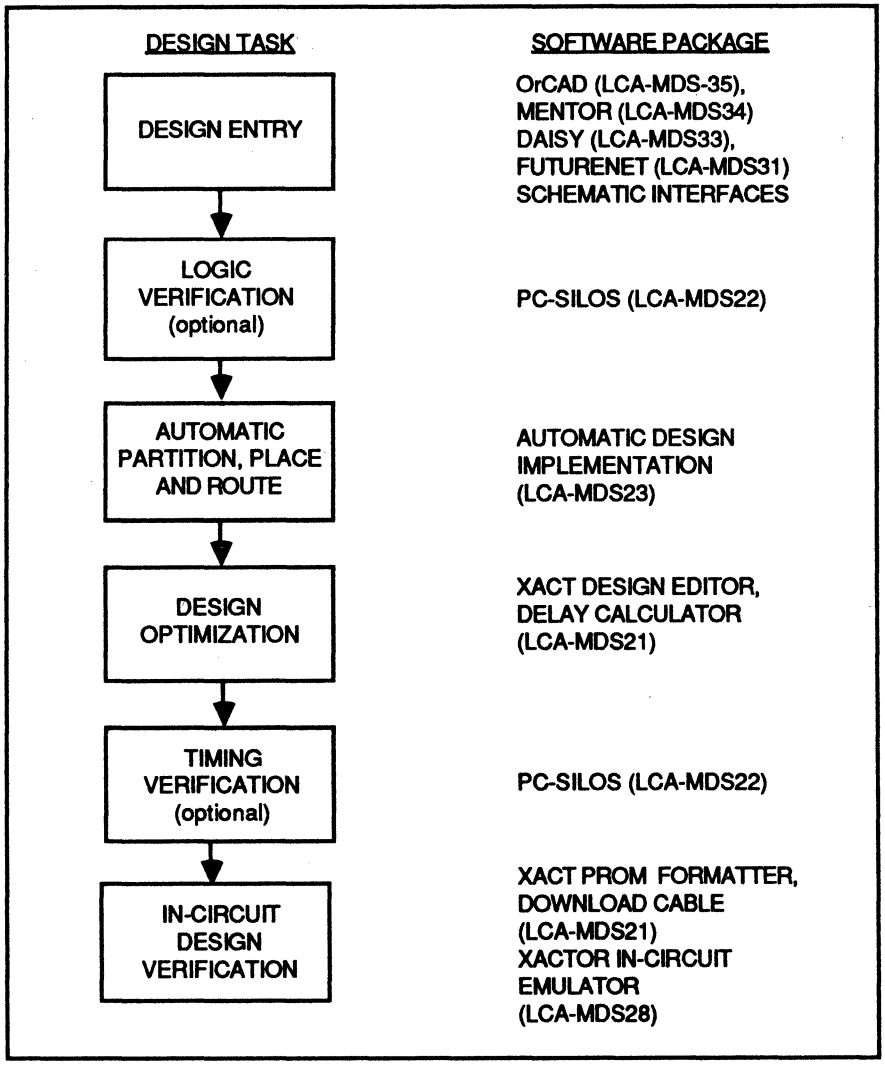
Product Number	Product Name
LCA-MDS151	Programmable Gate Array Development System (Includes LCA-MDS21, LCA-MDS23, LCA-MDS135)
LCA-MDS152	Programmable Gate Array Design Entry System (Includes LCA-MDS23, LCA-MDS135)

Publication #	Rev.	Amendment
10850	A	/0

Issue date: June 1988

PGA Design Cycle

The PGA Development System tools includes software for all phases of the PGA design cycle. The diagram below indicates the design tasks needed to complete a PGA design and the corresponding software products used to complete the tasks.



Symbol Library

Each schematic interface package includes the LCA library, which must be used when entering designs to be implemented in an AMD Programmable Gate Array. The library contains both combinatorial logic primitives and macros. Primitives represent the lowest level of logic symbols. Macros are logic functions implemented by the use of primitives and/or other macros. The following table lists the functions that are available in the four interface libraries. An **X** denotes that the function is available in the LCA library for that interface. If the function is available under a different macro name, that name appears in the column instead of an X.

NAME	DESCRIPTION	FUTURENET	DAISY	MENTOR	ORCAD
Buffers:					
ACLK	Alternate clock buffer	X	X	X	X
GCLK	Global clock buffer	X	X	X	X
BUF1	Buffer	X	X	X	X
INV1	Inverter	X	X	X	X
AND Gates:					
AND2	2-input AND gate	X	X	X	X
AND2B	2-input AND gate; all inputs inverted	AND20	X	X	X
AND2B1	2-input AND gate; 1 input inverted	AND201	X	X	X
AND3	3-input AND gate	X	X	X	X
AND3B	3-input AND gate; all inputs inverted	AND30	X	X	X
AND3B1	3-input AND gate; 1 input inverted	AND301	X	X	X
AND3B2	3-input AND gate; 2 inputs inverted	AND302	X	X	X
AND4	4-input AND gate	X	X	X	X
AND4B	4-input AND gate; all inputs inverted	AND40	X	X	X
AND4B1	4-input AND gate; 1 input inverted	AND401	X	X	X
AND4B2	4-input AND gate; 2 inputs inverted	AND402	X	X	X
AND4B3	4-input AND gate; 3 inputs inverted	AND403	X	X	X
AND5	5-input AND gate				X
AND5B	5-input AND gate, all inputs inverted				X
AND5B1	5-input AND gate; 1 input inverted				X
AND5B2	5-input AND gate; 2 inputs inverted				X
AND5B3	5-input AND gate; 3 inputs inverted				X
AND5B4	5-input AND gate; 4 inputs inverted				X
NAND Gates:					
NAND2	2-input NAND gate	X	X	X	X
NAND2B	2-input NAND gate; all inputs inverted	NAND20	X	X	X
NAND2B1	2-input NAND gate; 1 input inverted	NAND201	X	X	X
NAND3	3-input NAND gate	X	X	X	X
NAND3B	3-input NAND gate; all inputs inverted	NAND30	X	X	X
NAND3B1	3-input NAND gate; 1 input inverted	NAND301	X	X	X
NAND3B2	3-input NAND gate; 2 inputs inverted	NAND302	X	X	X

NAME	DESCRIPTION	FUTURENET	DAISY	MENTOR	ORCAD
NAND Gates (con 't.):					
NAND4	4-input NAND gate	X	X	X	X
NAND4B	4-input NAND gate; all inputs inverted	NAND40	X	X	X
NAND4B1	4-input NAND gate; 1 input inverted	NAND401	X	X	X
NAND4B2	4-input NAND gate; 2 inputs inverted	NAND402	X	X	X
NAND4B3	4-input NAND gate; 3 inputs inverted	NAND403	X	X	X
NAND5	5-input NAND gate				X
NAND5B	5-input NAND gate; all inputs inverted				X
NAND5B1	5-input NAND gate; 1 input inverted				X
NAND5B2	5-input NAND gate; 2 inputs inverted				X
NAND5B3	5-input NAND gate; 3 inputs inverted				X
NAND5B4	5-input NAND gate; 4 inputs inverted				X
NOR Gates:					
NOR2	2-input NOR gate	X	X	X	X
NOR2B	2-input NOR gate; all inputs inverted	NOR20	X	X	X
NOR2B1	2-input NOR gate; 1 input inverted	NOR201	X	X	X
NOR3	3-input NOR gate	X	X	X	X
NOR3B	3-input NOR gate; all inputs inverted	NOR30	X	X	X
NOR3B1	3-input NOR gate; 1 input inverted	NOR301	X	X	X
NOR3B2	3-input NOR gate; 2 inputs inverted	NOR302	X	X	X
NOR4	4-input NOR gate	X	X	X	X
NOR4B	4-input NOR gate; all inputs inverted	NOR40	X	X	X
NOR4B1	4-input NOR gate; 1 input inverted	NOR401	X	X	X
NOR4B2	4-input NOR gate; 2 inputs inverted	NOR402	X	X	X
NOR4B3	4-input NOR gate; 3 inputs inverted	NOR403	X	X	X
NOR5	5-input NOR gate				X
NOR5B	5-input NOR gate; all inputs inverted				X
NOR5B1	5-input NOR gate; 1 input inverted				X
NOR5B2	5-input NOR gate; 2 inputs inverted				X
NOR5B3	5-input NOR gate; 3 inputs inverted				X
NOR5B4	5-input NOR gate; 4 inputs inverted				X
OR Gates:					
OR2	2-input OR gate	X	X	X	X
OR2B	2-input OR gate; all inputs inverted	OR20	X	X	X
OR2B1	2-input OR gate; 1 input inverted	OR201	X	X	X
OR3	3-input OR gate	X	X	X	X
OR3B	3-input OR gate; all inputs inverted	OR30	X	X	X
OR3B1	3-input OR gate; 1 input inverted	OR301	X	X	X
OR3B2	3-input OR gate; 2 inputs inverted	OR302	X	X	X
OR4	4-input OR gate	X	X	X	X
OR4B	4-input OR gate; all inputs inverted	OR40	X	X	X
OR4B1	4-input OR gate; 1 input inverted	OR401	X	X	X
OR4B2	4-input OR gate; 2 inputs inverted	OR402	X	X	X
OR4B3	4-input OR gate; 3 inputs inverted	OR403	X	X	X
OR5	5-input OR gate				X
OR5B	5-input OR gate; all inputs inverted				X
OR5B1	5-input OR gate; 1 input inverted				X

Name	Description	FutureNet	Daisy	Mentor	OrCAD
OR Gates (con't.):					
OR5B2	5-input OR gate; 2 inputs inverted				X
OR5B3	5-input OR gate; 3 inputs inverted				X
OR5B4	5-input OR gate; 4 inputs inverted				X
Exclusive OR/NOR Gates:					
XOR2	2-input XOR gate	X	X	X	X
XOR3	3-input XOR gate	X	X	X	X
XOR4	4-input XOR gate	X	X	X	X
XOR5	5-input XOR gate				X
XNOR2	2-input XNOR gate	X	X	X	X
XNOR3	3-input XNOR gate	X	X	X	X
XNOR4	4-input XNOR gate	X	X	X	X
XNOR5	5-input XNOR gate				X
General:					
GADD	Adder	X	X	X	X
GCOMP	Compare	X	X	X	X
GEQGT	Equal or greater-than comparator	X	X	X	X
GMAJ	4-to-1 majority gate	X	X	X	X
GMUX	2-to-1 mux	X	X	X	X
GXTL	Crystal oscillator	X	X	X	X
GLTGT	Less-than and greater-than comparator				X
GPAR	Parity	X	X		
GXOR	Exclusive-or	X	X		
Input/Output buffers and Pads:					
IBUF	Input buffer		X	X	X
OBUF	Output buffer		X	X	X
OBUFZ	Output buffer with 3-state control		X	X	X
INFF	IOB configured as a flip-flop		X	X	X
OSC	Oscillator		X	X	X
INLAT	IOB configured as an input latch				X
BPAD	Bi-directional pad				X
IPAD	Input pad				X
UPAD	Unbonded pad				X
OPAD	Output pad				X
OUTFF	Output flip-flop				X
OUTFFZ	Output flip-flop with 3-state control				X
TBUF	Internal 3-state driver				X
Latches:					
LD	Data latch with load input	X	X	X	X
LDRD	Data latch with load input and reset direct	X	X	X	X
LDSD	Data latch with load input and set direct	X	X	X	X
LDSRD	Data latch with load input, reset and set direct	X	X	X	X
LDM	Data latch with 2-input data multiplexer	X	X	X	X
LDMRD	Data latch with 2-input data multiplexer and reset direct	X	X	X	X

NAME	DESCRIPTION	FUTURENET	DAISY	MENTOR	ORCAD
Latches: (con't.)					
LDMSD	Data latch with 2-input data multiplexer and set direct	X	X	X	X
DLAT	Data latch with set direct and reset direct		X	X	X
LRS	Data latch with set and reset				X
Flip-Flops:					
DFF	Positive edge-tiggered D flip-flop				X
FD	D flip-flop with one data line	X	X	X	X
FDR	D flip-flop with reset	X	X	X	X
FDS	D flip-flop with set	X	X	X	X
FDRD	D flip-flop with reset direct	X	X	X	X
FDSD	D flip-flop with set direct	X	X	X	X
FDSRD	D flip-flop with set and reset direct	X	X	X	X
PDFF	D flip-flop with positive edge clock	X	X		
NDFF	D flip-flop with negative edge clock	X	X		
FDC	D flip-flop with clock enable	X	X	X	X
FDCR	D flip-flop with clock enable and reset	X	X	X	X
FDCS	D flip-flop with clock enable and set	X	X	X	X
FDM	D flip-flop with 2-input data multiplexer	X	X	X	X
FDMR	D flip-flop with 2-input data multiplexer and reset	X	X	X	X
FDMS	D flip-flop with 2-input data multiplexer and set	X	X	X	X
FDMRD	D flip-flop with 2-input data multiplexer and reset direct	X	X	X	X
FDMSD	D flip-flop with 2-input data multiplexer and set direct	X	X	X	X
FDCRD	D flip-flop with clock enable and reset direct				X
DFF1	D flip-flop with set direct and reset direct		X	X	
FSR	Set and reset flip-flop with set dominant over reset	X	X	X	X
FRS	Set and reset flip-flop with reset dominance over set	X	X	X	X
FJK	J-K flip-flop	X	X	X	X
FJKS	J-K flip-flop with synchronous set	X	X	X	X
FJKRD	J-K flip-flop with reset direct	X	X	X	X
FJKSD	J-K flip-flop with set direct	X	X	X	X
FJKSRD	J-K flip-flop with set and reset direct	X	X	X	X
FT0	Self-toggle flip-flop	X	X	X	X
FT0R	Self-toggle flip-flop with reset	X	X	X	X
FT0RD	2-input toggle flip-flop with reset direct				X
FT	Toggle flip-flop	X	X	X	X
FTP	Toggle flip-flop with parity enable	X	X	X	X
FTPRD	Toggle flip-flop with parity enable and reset direct	X	X	X	X
FTR	Toggle flip-flop with reset	X	X	X	X
FTRD	2-input toggle flip-flop		X	X	X
FTS	Toggle flip-flop with set	X	X	X	X

NAME	DESCRIPTION	FUTURENET	DAISY	MENTOR	ORCAD
Flip-flops (con't.)					
FT2	2-input toggle flip-flop	X	X	X	X
FT2R	2-input toggle flip-flop with reset	X	X	X	X
Decoders:					
D2_4	1-of-4 decoder	X	X	X	X
D2_4E	1-of-4 decoder with enable	X	X	X	X
74_139	1-of-4 single decoder with low output and enable	X	X	$74_139	X
D3_8	1-of-8 decoder	X	X	X	X
D3_8E	1-of-8 decoder with enable	X	X	X	X
74_138	1-of-8 decoder with enables and low output	X	X	$74_138	X
74_42	1-of-10 decoder with low output	X	X	$74_42	X
Multiplexers:					
M3_1	3-to-1 multiplexer	X	X	X	X
M3_1E	3-to-1 multiplexer with enable	X	X	X	
M4_1	4-to-1 multiplexer	X	X	X	X
M4_1E	4-to-1 multiplexer with enable	X	X	X	X
74_352	4-to-1 multiplexer with low output and enable	X	X	$74_352	X
M4_2	4-to-2 multiplexer				X
M8_1	8-to-1 multiplexer	X	X	X	X
M8_1E	8-to-1 multiplexer with enable	X	X	X	X
74_151	8-to-1 multiplexer with enable and complementary outputs	X	X	$74_151	X
74_152	8-to-1 multiplexer with low output	X	X	$74_152	X
Registers:					
RD4	4-bit data register with clock	X	X	X	X
RD8	8-bit data register with clock	X	X	X	X
RD8CR	8-bit data register with clock enable and reset	X	X	X	X
RS4	4-bit shift register with clock	X	X	X	X
RS8	8-bit shift register	X	X	X	X
RS8CR	8-bit shift register with clock enable and reset	X	X	X	X
74_194	4-bit bidirectional shift register with clock, parity enable and reset direct	X	X	$74_194	X
74_195	4-bit serial-to-parallel shift register with clock, parity enable and reset direct	X	X	$74_195	X
RS8PR	8-bit shift register with clock, parity enable, and reset	X	X	X	X
RS8R	8-bit shift register with clock and reset	X	X	X	X
74_164	8-bit serial-to-parallel shift register with clock and reset direct	X	X	$74_164	X
RD4RD	4-bit data register with clock and reset direct				X
RD8RD	8-bit data register with clock and reset direct				X
RS4C	4-bit shift register with clock enable				X
RS4CR	4-bit shift register with clock enable and reset				X
RS4CRD	4-bit shift register with clock enable and reset direct				X
RS4RD	4-bit shift register with reset direct				X

NAME	DESCRIPTION	FUTURENET	DAISY	MENTOR	ORCAD
Registers (con't.)					
RS8C	8-bit shift register with clock enable				X
RS8CRD	8-bit shift register with clock enable and reset direct				X
RS8RD	8-bit shift register with reset direct				X
Counters:					
C2BCR	1-bit binary counter with clock and reset	X	X	X	X
C2BCRD	1-bit binary counter with clock and reset direct	X	X	X	X
C2BP	1-bit binary counter with clock and parity enable	X	X	X	X
C2BR	1-bit binary counter with clock and reset	X	X	X	X
C2BRD	1-bit binary counter with reset direct	X	X	X	X
C2BCP	1-bit binary counter with clock and parity enable				X
C2BCPRD	1-bit binary counter with clock, parity enable and reset direct				X
C4BCP	2-bit binary counter with clock and parity enable	X	X	X	X
C4BCR	2-bit binary counter with clock enable and reset	X	X	X	X
C4BCRD	2-bit binary counter with clock enable and reset direct	X	X	X	X
C4JCR	2-bit Johnson counter with clock enable and reset	X	X	X	X
C4BCPRD	2-bit binary counter with clock enable and reset direct				X
C4JX	2-bit shift expandable Johnson counter				X
C4JXC	2-bit expandable Johnson counter with clock enable				X
C4JXCR	2-bit expandable Johnson counter with clock enable and reset				X
C4JXCRD	2-bit expandable Johnson counter with clock enable and reset direct				X
C4JXRD	2-bit expandable Johnson counter with reset direct				X
C6JCR	3-bit Johnson counter with clock enable and reset	X	X	X	X
C8BCP	3-bit binary counter with clock and parity enable	X	X	X	X
C8BCR	3-bit binary counter with clock enable and reset	X	X	X	X
C8BCRD	3-bit binary counter with clock enable and reset direct	X	X	X	X
C8JCR	4-bit Johnson counter with clock enable and reset	X	X	X	X
C8BCPRD	8-bit binary counter with clock, parity enable and reset direct				X
C10BCRD	4-bit BCD counter with clock enable and reset direct	X	X	X	X
C10BCPRD	4-bit BCD counter with clock, parity enable, and reset direct	X	X	X	X
74_160	4-bit BCD counter with clock, parity enable, and reset direct	X	X	$74_160	X
C10BPRD	4-bit BCD counter with clock, parity enable, and reset direct	X	X	X	X
C10JCR	5-bit Johnson counter with clock enable and reset	X	X	X	X

NAME	DESCRIPTION	FUTURENET	DAISY	MENTOR	ORCAD
Counters (con't.)					
C12JCR	6-bit Johnson counter with clock enable and reset	X	X	X	X
C16BARD	4-bit binary ripple counter with clock and reset direct	X	X	X	X
C16BCRD	4-bit binary counter with clock enable and reset direct	X	X	X	X
C16BCPR	4-bit binary counter with clock, parity enable, and reset	X	X	X	X
C16BCPRD	4-bit binary counter with clock, parity enable, and reset direct	X	X	X	X
C16BCP	4-bit binary counter with clock, parity enable				X
74_161	4-bit binary counter with clock and reset direct	X	X	$74_161	X
74_162	4-bit binary counter with reset				X
74_163	4-bit binary counter with reset				X
C16BPRD	4-bit binary up counter with clock, parity enable, and reset direct	X	X	X	X
C16BUDRD	4-bint binary up-down counter with clock, parity enable, and reset direct	X	X	X	X
C16JCR	8-bit Johnson counter with clock enable and reset	X	X	X	X
C256FCRD	8-bit modulo 256 feedback shift register with clock enable and reset direct	X	X	X	X
C256BCP	8-bit binary counter with clock, parity enable				X
C256BCPR	8-bit binary counter with clock, parity enable, and reset direct				X
C256BCR	8-bit binary counter with clock enable and reset				X
C256BCRD	8-bit binary counter with clock enable and reset direct				X
Flags:					
C	Critical signal flag	X			X
L	Longline signal flag	X			X
N	Non-critical signal flag	X			X
X	Explicit signal flag	X			X
D	External debug net signal flag	X			
K	K pin for clock	X			X
G	G pin for clock	X			X
I	Input (C) pin for clock	X			X
CLB					
CLB	CLB primitive	X			X
IOB Primitives:					
IOB	IOB primitive				X
PIN	Input pad	X			
POUT	Output pad	X			
POUTZ	Output pad with 3-state control	X			
PBUF	Buffered input	X			
PINQ	Input pad with storage	X			
PIO	Input/output pad	X			

NAME	DESCRIPTION	FUTURENET	DAISY	MENTOR	ORCAD
IOB Primitives (con't.)					
PREG	Outpu pad with input storage	X			
PIOQ	Input/output pad with input storage	X			
IOFF	IOB configured as a flip-flop	X			
Resistors:					
Pullup	Pullup resistor				X

Programmable Gate Array Software

PART NUMBER	PRODUCT DESCRIPTION
LCA-MDS21	XACT LCA Development System
LCA-MDS22	P-Silos Simulator
LCA-MDS23	Automatic Design Implementation
LCA-MDS24	XACTOR In-Circuit Emulator
LCA-MDS26	XACTOR Universal POD
LCA-MDS27	XACTOR POD Header
LCA-MDS31	FutureNet Schematic Interface and LCA Library
LCA-MDS33	Daisy Schematic Interface and LCA Library
LCA-MDS34	Mentor Schematic Interface and LCA Library
LCA-MDS35	OrCAD Schematic Interface and LCA Library
LCA-MDS135	OrCAD Schematic Entry Software, Interface and LCA Library (Includes LCA-MDS35)
LCA-MDS151	LCA Bundled Development System (Includes LCA-MDS21, LCA-MDS23, and LCA-MDS135)
LCA-MDS152	LCA Bundled Design Entry Package (Includes LCA-MDS23 and LCA-MDS135)

LCA-MDS135

OrCAD/SDT III PGA Design Entry System and Interface

Distinctive Characteristics

- PGA design entry via the OrCAD/SDT III Schematic Editor
- Easy schematic creation using simple keyboard or mouse commands
- Fast graphical editing
- Powerful keyboard macros simplify schematic capture
- More than 200 hierarchical levels help manage design complexity
- Compatible with both 2000 and 3000 series LCAs
- Extensive SSI/MSI library contains over 3600 components
- LCA library of over 100 macros, with TTL and standard family equivalents
- Pop-up menus
- User control for flagging critical paths for the LCA-MDS23 Automatic Design Implementation program
- Includes OrCAD/SDT III Schematic Editor and LCA libraries and interface.
- Runs on an IBM PC/AT or compatible personal computer

General Description

Schematic entry and automatic partitioning of Programmable Gate Array designs shortens logic reduction and product development times. Complex designs can be specified schematically and quickly implemented for in-circuit design verification.

Developed specifically to run on IBM personal computers and compatibles, OrCAD/SDT supports most of the popular graphics boards and printers. This gives you the flexibility to use standard equipment rather than special proprietary hardware. OrCAD/SDT also includes an array of sophisticated utility programs, including schematic output, net list, design check and bill of materials. Other helpful features include runbberbanding of wires and buses, on-line part browsing, auto panning of worksheet, string searching, visible grids

The AMD version of ORCAD/SDT III includes a complete library of gates and macros that designers can use to enter their LCA schematics. The AMD library provides the logic, I/O, and macro symbols to be used in the schematic. An AMD conversion utility, part of the Automatic Design Implementation package (LCA-MDS 23), automatically eliminates unused and disabled logic, then partitions the schematic into a Logic Cell Array design.

Publication #	Rev.	Amendment
10852	A	/0

Issue date: June 1988

Symbol Library

The OrCAD schematic interface package includes the LCA library, which must be used when entering designs to be implemented in an AMD Programmable Gate Array. The library contains both combinatorial logic primitives and macros. Primitives represent the lowest level of logic symbols. Macros are logic functions implemented by the use of primitives and/or other macros. The following table lists the functions that are available in the OrCAD interface library. Note that some functions are available for the 3000 family only.

NAME	DESCRIPTION
Buffers:	
ACLK	Alternate clock buffer
GCLK	Global clock buffer
BUF1	Buffer
INV1	Inverter
AND Gates:	
AND2	2-input AND gate
AND2B	2-input AND gate; all inputs inverted
AND2B1	2-input AND gate; 1 input inverted
AND3	3-input AND gate
AND3B	3-input AND gate; all inputs inverted
AND3B1	3-input AND gate; 1 input inverted
AND3B2	3-input AND gate; 2 inputs inverted
AND4	4-input AND gate
AND4B	4-input AND gate; all inputs inverted
AND4B1	4-input AND gate; 1 input inverted
AND4B2	4-input AND gate; 2 inputs inverted
AND4B3	4-input AND gate; 3 inputs inverted
AND5	5-input AND gate
AND5B	5-input AND gate, all inputs inverted
AND5B1	5-input AND gate; 1 input inverted
AND5B2	5-input AND gate; 2 inputs inverted
AND5B3	5-input AND gate; 3 inputs inverted
AND5B4	5-input AND gate; 4 inputs inverted
NAND Gates:	
NAND2	2-input NAND gate
NAND2B	2-input NAND gate; all inputs inverted
NAND2B1	2-input NAND gate; 1 input inverted
NAND3	3-input NAND gate
NAND3B	3-input NAND gate; all inputs inverted
NAND3B1	3-input NAND gate; 1 input inverted
NAND3B2	3-input NAND gate; 2 inputs inverted
NAND4	4-input NAND gate
NAND4B	4-input NAND gate; all inputs inverted
NAND4B1	4-input NAND gate; 1 input inverted
NAND4B2	4-input NAND gate; 2 inputs inverted

NAME	DESCRIPTION
NAND Gates (con't.):	
NAND4B3	4-input NAND gate; 3 inputs inverted
NAND5	5-input NAND gate
NAND5B	5-input NAND gate; all inputs inverted
NAND5B1	5-input NAND gate; 1 input inverted
NAND5B2	5-input NAND gate; 2 inputs inverted
NAND5B3	5-input NAND gate; 3 inputs inverted
NAND5B4	5-input NAND gate; 4 inputs inverted
NOR Gates:	
NOR2	2-input NOR gate
NOR2B	2-input NOR gate; all inputs inverted
NOR2B1	2-input NOR gate; 1 input inverted
NOR3	3-input NOR gate
NOR3B	3-input NOR gate; all inputs inverted
NOR3B1	3-input NOR gate; 1 input inverted
NOR3B2	3-input NOR gate; 2 inputs inverted
NOR4	4-input NOR gate
NOR4B	4-input NOR gate; all inputs inverted
NOR4B1	4-input NOR gate; 1 input inverted
NOR4B2	4-input NOR gate; 2 inputs inverted
NOR4B3	4-input NOR gate; 3 inputs inverted
NOR5	5-input NOR gate
NOR5B	5-input NOR gate; all inputs inverted
NOR5B1	5-input NOR gate; 1 input inverted
NOR5B2	5-input NOR gate; 2 inputs inverted
NOR5B3	5-input NOR gate; 3 inputs inverted
NOR5B4	5-input NOR gate; 4 inputs inverted
OR Gates:	
OR2	2-input OR gate
OR2B	2-input OR gate; all inputs inverted
OR2B1	2-input OR gate; 1 input inverted
OR3	3-input OR gate
OR3B	3-input OR gate; all inputs inverted
OR3B1	3-input OR gate; 1 input inverted
OR3B2	3-input OR gate; 2 inputs inverted
OR4	4-input OR gate
OR4B	4-input OR gate; all inputs inverted
OR4B1	4-input OR gate; 1 input inverted
OR4B2	4-input OR gate; 2 inputs inverted
OR4B3	4-input OR gate; 3 inputs inverted
OR5	5-input OR gate
OR5B	5-input OR gate; all inputs inverted
OR5B1	5-input OR gate; 1 input inverted
OR5B2	5-input OR gate; 2 inputs inverted
OR5B3	5-input OR gate; 3 inputs inverted
OR5B4	5-input OR gate; 4 inputs inverted

NAME	DESCRIPTION
Exclusive OR/NOR Gates:	
XOR2	2-input XOR gate
XOR3	3-input XOR gate
XOR4	4-input XOR gate
XOR5	5-input XOR gate
XNOR2	2-input XNOR gate
XNOR3	3-input XNOR gate
XNOR4	4-input XNOR gate
XNOR5	5-input XNOR gate
General:	
GADD	Adder
GCOMP	Compare
GEQGT	Equal or greater-than comparator
GMAJ	4-to-1 majority gate
GMUX	2-to-1 mux
GXTL	Crystal oscillator
GLTGT	Less-than and greater-than comparator
Input/Output buffers and Pads:	
IBUF	Input buffer
OBUF	Output buffer
OBUFZ	Output buffer with 3-state control
INFF	IOB configured as a flip-flop
OSC	Oscillator
INLAT	IOB configured as an input latch
BPAD	Bi-directional pad
IPAD	Input pad
UPAD	Unbonded pad
OPAD	Output pad
OUTFF	Output flip-flop
OUTFFZ	Output flip-flop with 3-state control
TBUF	Internal 3-state driver
Latches:	
LD	Data latch with load input
LDRD	Data latch with load input and reset direct
LDSD	Data latch with load input and set direct
LDSRD	Data latch with load input, reset and set direct
LDM	Data latch with 2-input data multiplexer
LDMRD	Data latch with 2-input data multiplexer and reset direct
LDMSD	Data latch with 2-input data multiplexer and set direct
DLAT	Data latch with set direct and reset direct
LRS	Data latch with set and reset
Flip-Flops:	
DFF	Positive edge-tiggered D flip-flop
FD	D flip-flop with one data line
FDR	D flip-flop with reset

NAME	DESCRIPTION
Flip-flops: (con't.)	
FDS	D flip-flop with set
FDRD	D flip-flop with reset direct
FDSD	D flip-flop with set direct
FDSRD	D flip-flop with set and reset direct
FDC	D flip-flop with clock enable
FDCR	D flip-flop with clock enable and reset
FDCS	D flip-flop with clock enable and set
FDM	D flip-flop with 2-input data multiplexer
FDMR	D flip-flop with 2-input data multiplexer and reset
FDMS	D flip-flop with 2-input data multiplexer and set
FDMRD	D flip-flop with 2-input data multiplexer and reset direct
FDMSD	D flip-flop with 2-input data multiplexer and set direct
FDCRD	D flip-flop with clock enable and reset direct
FSR	Set and reset flip-flop with set dominant over reset
FRS	Set and reset flip-flop with reset dominance over set
FJK	J-K flip-flop
FJKS	J-K flip-flop with synchronous set
FJKRD	J-K flip-flop with reset direct
FJKSD	J-K flip-flop with set direct
FJKSRD	J-K flip-flop with set and reset direct
FT0	Self-toggle flip-flop
FT0R	Self-toggle flip-flop with reset
FT0RD	2-input toggle flip-flop with reset direct
FT	Toggle flip-flop
FTP	Toggle flip-flop with parity enable
FTPRD	Toggle flip-flop with parity enable and reset direct
FTR	Toggle flip-flop with reset
FTRD	2-input toggle flip-flop
FTS	Toggle flip-flop with set
FT2	2-input toggle flip-flop
FT2R	2-input toggle flip-flop with reset
Decoders:	
D2_4	1-of-4 decoder
D2_4E	1-of-4 decoder with enable
74_139	1-of-4 single decoder with low output and enable
D3_8	1-of-8 decoder
D3_8E	1-of-8 decoder with enable
74_138	1-of-8 decoder with enables and low output
74_42	1-of-10 decoder with low output
Multiplexers:	
M3_1	3-to-1 multiplexer
M3_1E	3-to-1 multiplexer with enable
M4_1	4-to-1 multiplexer
M4_1E	4-to-1 multiplexer with enable
74_352	4-to-1 multiplexer with low output and enable
M4_2	4-to-2 multiplexer

NAME	DESCRIPTION
Multiplexers: (con't.)	
M8_1	8-to-1 multiplexer
M8_1E	8-to-1 multiplexer with enable
74_151	8-to-1 multiplexer with enable and complementary outputs
74_152	8-to-1 multiplexer with low output
Registers:	
RD4	4-bit data register with clock
RD8	8-bit data register with clock
RD8CR	8-bit data register with clock enable and reset
RS4	4-bit shift register with clock
RS8	8-bit shift register
RS8CR	8-bit shift register with clock enable and reset
74_194	4-bit bidirectional shift register with clock, parity enable and reset direct
74_195	4-bit serial-to-parallel shift register with clock, parity enable and reset direct
RS8PR	8-bit shift register with clock, parity enable, and reset
RS8R	8-bit shift register with clock and reset
74_164	8-bit serial-to-parallel shift register with clock and reset direct
RD4RD	4-bit data register with clock and reset direct
RD8RD	8-bit data register with clock and reset direct
RS4C	4-bit shift register with clock enable
RS4CR	4-bit shift register with clock enable and reset
RS4CRD	4-bit shift register with clock enable and reset direct
RS4RD	4-bit shift register with reset direct
RS8C	8-bit shift register with clock enable
RS8CRD	8-bit shift register with clock enable and reset direct
RS8RD	8-bit shift register with reset direct
Counters:	
C2BCR	1-bit binary counter with clock and reset
C2BCRD	1-bit binary counter with clock and reset direct
C2BP	1-bit binary counter with clock and parity enable
C2BR	1-bit binary counter with clock and reset
C2BRD	1-bit binary counter with reset direct
C2BCP	1-bit binary counter with clock and parity enable
C2BCPRD	1-bit binary counter with clock, parity enable and reset direct
C4BCP	2-bit binary counter with clock and parity enable
C4BCR	2-bit binary counter with clock enable and reset
C4BCRD	2-bit binary counter with clock enable and reset direct
C4JCR	2-bit Johnson counter with clock enable and reset
C4BCPRD	2-bit binary counter with clock enable and reset direct
C4JX	2-bit shift expandable Johnson counter
C4JXC	2-bit expandable Johnson counter with clock enable
C4JXCR	2-bit expandable Johnson counter with clock enable and reset
C4JXCRD	2-bit expandable Johnson counter with clock enable and reset direct
C4JXRD	2-bit expandable Johnson counter with reset direct
C6JCR	3-bit Johnson counter with clock enable and reset
C8BCP	3-bit binary counter with clock and parity enable
C8BCR	3-bit binary counter with clock enable and reset

NAME	DESCRIPTION
Counters (con't.)	
C8BCRD	3-bit binary counter with clock enable and reset direct
C8JCR	4-bit Johnson counter with clock enable and reset
C8BCPRD	8-bit binary counter with clock, parity enable and reset direct
C10BCRD	4-bit BCD counter with clock enable and reset direct
C10BCPRD	4-bit BCD counter with clock, parity enable, and reset direct
74_160	4-bit BCD counter with clock, parity enable, and reset direct
C10BPRD	4-bit BCD counter with clock, parity enable, and reset direct
C10JCR	5-bit Johnson counter with clock enable and reset
C12JCR	6-bit Johnson counter with clock enable and reset
C16BARD	4-bit binary ripple counter with clock and reset direct
C16BCRD	4-bit binary counter with clock enable and reset direct
C16BCPR	4-bit binary counter with clock, parity enable, and reset
C16BCPRD	4-bit binary counter with clock, parity enable, and reset direct
C16BCP	4-bit binary counter with clock, parity enable
74_161	4-bit binary counter with clock and reset direct
74_162	4-bit binary counter with reset
74_163	4-bit binary counter with reset
C16BPRD	4-bit binary up counter with clock, parity enable,and reset direct
C16BUDRD	4-bit binary up-down counter with clock, parity enable, and reset direct
C16JCR	8-bit Johnson counter with clock enable and reset
C256FCRD	8-bit modulo 256 feedback shift register with clock enable and reset direct
C256BCP	8-bit binary counter with clock, parity enable
C256BCPR	8-bit binary counter with clock, parity enable, and reset direct
C256BCR	8-bit binary counter with clock enable and reset
C256BCRD	8-bit binary counter with clock enable and reset direct
Flags:	
C	Critical signal flag
L	Longline signal flag
N	Non-critical signal flag
X	Explicit signal flag
K	K pin for clock
G	G pin for clock
I	Input (C) pin for clock
CLB:	
CLB	CLB primitive
IOB :	
IOB	IOB primitive
Resistors:	
Pullup	Pullup resistor

LCA-MEK01 Logic Cell Array Evaluation Kit

The Monolithic Memories Logic Cell Array is a high-performance CMOS user-programmable gate array. The Monolithic Memories' Logic Cell Array Evaluation Kit is a software package that provides the capability to evaluate the Logic Cell Array for new applications.

Features

- **Design software package for IBM PC-XT, PC-AT or compatible computer**
- **Interactive graphics-oriented designer interface**
- **Simplified definition, placement and connection capability for implementation of complex logic**
- **Boolean equation or Karnaugh map alternatives to specify logic functions**
- **Macro library of 113 standard logic equivalents plus support for user-defined macros**
- **Point-to-point timing calculations for critical path analysis**
- **Automatic checking for connectivity and design consistency**
- **Hardcopy output of logical and physical configuration information**

General

The Evaluation Kit can be used to enter complete designs using a subset of the XACT design editor, including the use of the Monolithic Memories macro library. Critical timing for the design can be evaluated with the timing delay calculator to evaluate the applicability of the Logic Cell Array technology to a particular design.

Functional definition of Configurable Logic Blocks (CLBs), and their internal routing,I/O Block (IOB) definitions, and interconnection are all done within an integrated graphics-oriented system. Interactive placement and automatic routing of logic and I/O elements are accomplished quickly and easily via an easy-to-learn user interface.

Designs are captured with a graphics-oriented design editor, using either a mouse or keyboard entry, driven from command or files. User functions are specified in terms of CLB definitions and interconnections. Standard logic functions from the macro library or user-defined macro capabilities can be utilized to quickly implement complex logic functions. Placement and routing can be edited easily to modify or optimize a design.

Checking of logical connectivity is performed automatically. All unused internal nodes are automatically configured to minimize power dissipation.

Interactive point-to-point timing delay calculation is provided to simplify timing analysis and critical path determination.

The Evaluation Kit includes hardcopy generation to document a design and automatically track design changes.

System Requirements

Minimum System Configuration

IBM PC-XT, PC-AT or compatible computer with:

- MS-DOS 2.1 or higher
- 640K Bytes RAM
- 1 Diskette Drive
- 10-MB Hard Disk
- IBM or compatible Color Graphic Adapter and Display
- 1 Serial Interface Port
- Mouse Systems, Microsoft or compatible mouse

Evaluation Kit

Publication #	Rev.	Amendment
10856	A	/0

Issue Date: **January 1988**

Advanced
Micro
Devices

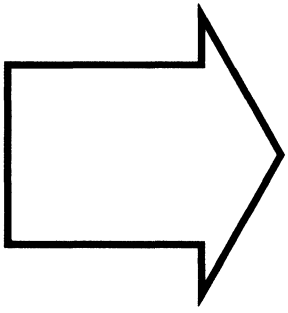


Table of Contents Section 5
Sales Office Listing

ADVANCED MICRO DEVICES' NORTH AMERICAN SALES OFFICES

ALABAMA ..... (205) 882-9122
ARIZONA ..... (602) 242-4400
CALIFORNIA,
Culver City ..... (213) 645-1524
Newport Beach ..... (714) 752-6262
San Diego ..... (619) 560-7030
San Jose ..... (408) 452-0500
Woodland Hills ..... (818) 992-4155
CANADA, Ontario,
Kanata ..... (613) 592-0060
Willowdale ..... (416) 224-5193
COLORADO ..... (303) 741-2900
CONNECTICUT ..... (203) 264-7800
FLORIDA,
Clearwater ..... (813) 530-9971
Ft Lauderdale ..... (305) 776-2001
Melbourne ..... (305) 729-0496
Orlando ..... (305) 859-0831
GEORGIA ..... (404) 449-7920
ILLINOIS,
Chicago ..... (312) 773-4422
Naperville ..... (312) 505-9517
INDIANA ..... (317) 244-7207
KANSAS ..... (913) 451-3115
MARYLAND ..... (301) 796-9310
MASSACHUSETTS ..... (617) 273-3970
MINNESOTA ..... (612) 938-0001
MISSOURI ..... (913) 451-3115
NEW JERSEY ..... (201) 299-0002
NEW YORK,
Liverpool ..... (315) 457-5400
Poughkeepsie ..... (914) 471-8180
Woodbury ..... (516) 364-8020
NORTH CAROLINA ..... (919) 878-8111
OHIO ..... (614) 891-6455
Columbus ..... (614) 891-6455
Dayton ..... (513) 439-0470
OREGON ..... (503) 245-0080
PENNSYLVANIA,
Allentown ..... (215) 398-8006
Willow Grove ..... (609) 662-2900
TEXAS,
Austin ..... (512) 346-7830
Dallas ..... (214) 934-9099
Houston ..... (713) 785-9001
WASHINGTON ..... (206) 455-3600
WISCONSIN ..... (414) 792-0590

ADVANCED MICRO DEVICES' INTERNATIONAL SALES OFFICES

BELGIUM,
Bruxelles ..... TEL ..... (02) 771 91 42
FAX ..... (02) 762 37 12
TLX 61028
FRANCE,
Paris ..... TEL ..... (1) 49-75-10-10
FAX ..... (1) 49-75-10-13
TLX 263282
WEST GERMANY,
Hannover area ..... TEL ..... (0511) 736085
FAX ..... (0511) 721254
TLX 922850
München ..... TEL ..... (089) 41 14-0
FAX ..... (089) 406490
TLX 523883
Stuttgart ..... TEL ..... (0711) 62 33 77
FAX ..... (0711) 625187
TLX 721882
HONG KONG ..... TEL 852-5-8654525
FAX 852-5-8654335
TLX ... 67955AMDAPHX
ITALY, Milano ..... TEL ..... (02) 3390541
(02) 3533241
FAX ..... (02) 3498000
TLX 315286
JAPAN,
Kanagawa ..... TEL 462-47-2911
FAX 462-47-1729
Tokyo ..... TEL ..... (03) 345-8241
FAX ..... (03) 342-5196
TLX ... J24064AMDTKOJ
Osaka ..... TEL 06-243-3250
FAX 06-243-3253
KOREA, Seoul ..... TEL 82-2-784-7598
FAX 82-2-784-8014
LATIN AMERICA,
Ft. Lauderdale ..... TEL ..... (305) 484-8600
FAX ..... (305) 485-9736
TLX .. 5109554261 AMDFTL
NORWAY,
Hovik ..... TEL ..... (02) 537810
FAX ..... (02) 591959
TLX 79079
SINGAPORE ..... TEL 65-2257544
FAX 65-2246113
TLX ..... RS55650 MMI RS
SWEDEN, Stockholm ..... TEL ..... (08) 733 03 50
FAX ..... (08) 733 22 85
TLX 11602
TAIWAN ..... TLX 886-2-7122066
FAX 886-2-7122017
UNITED KINGDOM,
Manchester area ..... TEL ..... (0925) 828008
FAX ..... (0925) 827693
TLX 628524
London area ..... TEL ..... (04862) 22121
FAX ..... (0483) 756196
TLX 859103

NORTH AMERICAN REPRESENTATIVES

CALIFORNIA
I² INC ..... OEM (408) 988-3400
DISTI (408) 498-6868
CANADA
Burnaby, B.C.
DAVETEK MARKETING ..... (604) 430-3680
Calgary, Alberta
DAVETEK MARKETING ..... (604) 430-3680
Kanata, Ontario
VITEL ELECTRONICS ..... (613) 592-0090
Mississauga, Ontario
VITAL ELECTRONICS ..... (416) 676-9720
Quebec
VITEL ELECTRONICS ..... (514) 636-5951
IDAHO
INTERMOUNTAIN TECH MKGT ..... (208) 888-6071
INDIANA
ELECTRONIC MARKETING
CONSULTANTS, INC. ..... (317) 253-1668
IOWA
LORENZ SALES ..... (319) 377-4666
KANSAS
LORENZ SALES ..... (913) 384-6556
KENTUCKY
ELECTRONIC MARKETING
CONSULTANTS, INC. ..... (317) 253-1668
MICHIGAN
SAI MARKETING CORP ..... (313) 750-1922
MISSOURI
LORENZ SALES ..... (314) 997-4558
NEBRASKA
LORENZ SALES ..... (402) 475-4660
NEW MEXICO
THORSON DESERT STATES ..... (505) 293-8555
NEW YORK
NYCOM, INC ..... (315) 437-8343
OHIO
Centerville
DOLFUSS ROOT & CO ..... (513) 433-6776
Columbus
DOLFUSS ROOT & CO ..... (614) 885-4844
Strongsville
DOLFUSS ROOT & CO ..... (216) 238-0300
PENNSYLVANIA
DOLFUSS ROOT & CO ..... (412) 221-4420
UTAH
R² MARKETING ..... (801) 595-0631

ADVANCED MICRO DEVICES 901 Thompson Pl., P.O. Box 3453, Sunnyvale, CA 94088, USA
TEL: (408) 732-2400 • TWX: 910-339-9280 • TELEX: 34-6306 • TOLL FREE: (800) 538-8450
APPLICATIONS HOTLINE TOLL FREE: (800) 222-9323 • (408) 749-5703

A&W—B— 50M—6/88—0

Advanced Micro Devices

901 Thompson Place
P.O. Box 3453
Sunnyvale
California 94088-3000
(408) 732-2400
TELEX: 34-6306
TOLL FREE (800) 538-8450
APPLICATIONS HOTLINE
(800) 222-9323
(408) 749-5703

Printed in USA

10869A/0