Table of Contents

The Programmable Gate Array

Using Programmable Gate Arrays

Programmable Gate Array Design Examples

Programmable Gate Array Development Systems

Appendices

Contributors

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

Xilinx, Inc. does not assume any liability arising out of the application or use of any product described herein; nor does it convey any license under its patent, copyright or maskwork rights or any rights of others. Xilinx, Inc. reserves the right to make changes, at any time, in order to improve reliability, function or design and to supply the best product possible. Xilinx, Inc. cannot assume responsibility for the use of any circuitry described other than circuitry entirely embodied in their product. No other circuit patent licenses are implied.

Xilinx, Inc. cannot assume responsibility for any circuits shown or represent that they are free from patent infringement or of any other third party right.

Xilinx, Inc. assumes no obligation to correct any errors contained herein or to advise any user of this text of any correction if such be made.

Printed in U.S.A.

About the Company...

Xilinx was founded in February 1984 to develop a family of CMOS user-programmable gate arrays and associated development systems. The development of a general-purpose user-programmable logic device with an array architecture was the result of a number of technical breakthroughs, many of which have resulted in patent applications.

Due to its density and to the convenience of user programmability, the Logic Cell™ Array represents an important new alternative in the ASIC market. The company continues to concentrate its resources exclusively on expanding its growing family of Logic Processors™ and associated development systems.

XILINX

The Programmable Gate Array Company

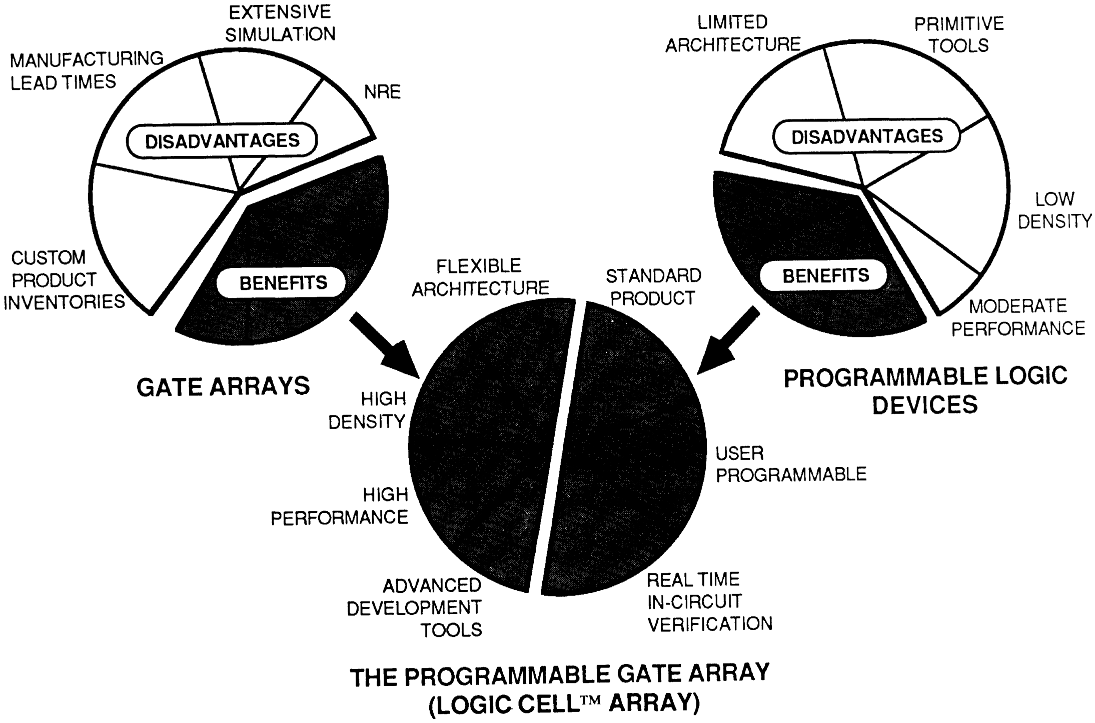


Table of Contents

The Programmable Gate Array

[illegible]

Programmable Gate Array Design [illegible]

Programmable Gate Array Development [illegible]

[illegible]

Introduction to Programmable Gate Arrays

Requirements for improved product features, including lower cost, higher performance, reduced power consumption, smaller size and increased reliability are motivating manufacturers of electronic systems to use high-density VLSI circuits.

The standard product ICs that have best exploited advances in VLSI have been microprocessors and memories. Density improvements in these product families have outpaced other digital integrated circuits, widening the technology gap between them and other logic devices. To achieve comparable densities for their proprietary functions, designers of digital equipment have been forced to consider 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. Figure 1 illustrates the tradeoffs of density and development time for several digital logic alternatives.

Standard SSI/MSI logic devices provide a great deal of flexibility, are well understood by most logic designers, and are readily available. However, they offer less density than other alternatives, and consume more power. These parts typically are manufactured in maturing technologies, with limited opportunity for further cost reductions.

The Programmable Logic Device (PLD) category includes a number of competing alternatives, all based on a programmable AND-OR plane architecture. The PLDs AND-OR plane architecture is most efficient for applications up to a few hundred usable gates. Bipolar PLDs are programmed by opening fuse links. CMOS PLDs can be one-time programmable, electrically programmabale (EPLDs), or electrically erasable (EEPLDs). These Programmable Logic Devices are often used in place of five to ten SSI/MSI devices. Since PLDs are user programmable, this gain in density can be achieved with only a small increase in design time and little schedule risk. Designs can be developed and devices programmed for a particular application in a matter of days. PLDs are best suited for state machines and decoders. Their architecture provides efficient multiple variable decoding and high performance for functions that are readily expressed as a sum-of-products. Architectural restrictions limit their application for general logic replacement, consolidation of miscellaneous "glue" and control functions, or complex processing tasks.

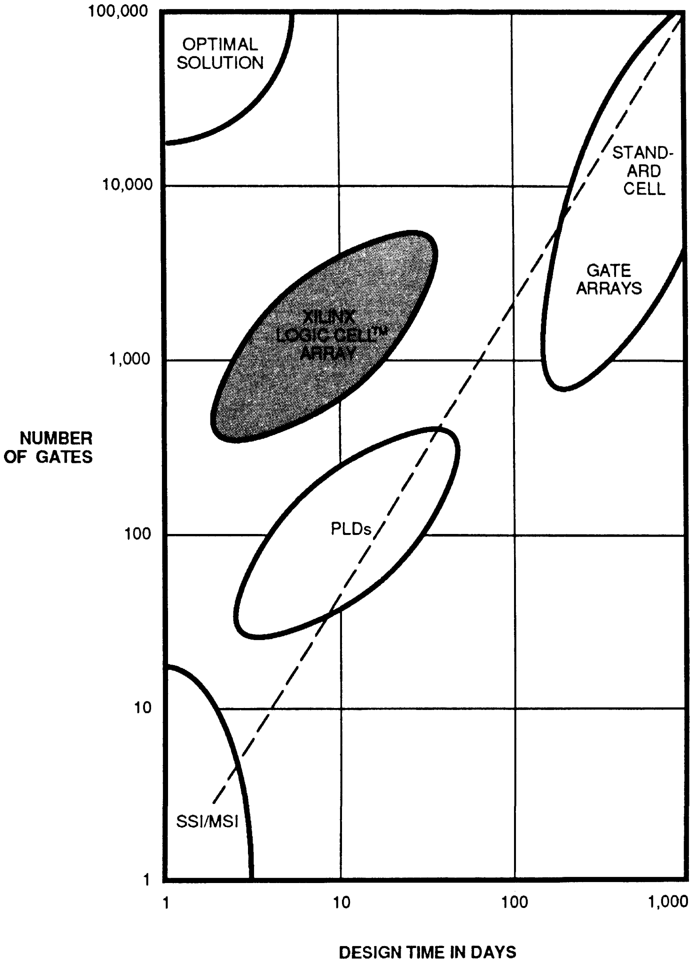
Factory programmable ASIC devices, including gate arrays, standard cells, and compiled silicon, provide logic densities up to 100,000 equivalent logic gates and are sufficiently flexible for most digital logic functions. After design completion and verification, factory programmed ASICs typically require two to four months for prototype fabrication and a similar period for the first production quantities. Because of their high design costs and limited production flexibility, factory-programmed ASICs are most economical in very high volume applications. The logistics of verifying a workable design, testing ICs and coordinating production demand require substantial attention and resources from the equipment manufacturer.

In the diagram, the upper left corner represents the best solution. The traditional tradeoffs between density and development time are illustrated by the dotted diagonal line in the diagram. As indicated, a new digital logic technology, the Logic Cell™ Array (LCA), offers improvements in both dimensions. This user-programmable gate array provides the system designer the usable density of gate arrays and the short development times and low risk of standard logic circuits. It combines the design and production benefits of a standard product with the system benefits of reliability, power savings, space savings, and lower production costs of ASIC devices.

ARCHITECTURE

The user benefits of the Logic Cell Array are derived from its general-purpose array architecture. This architecture is based on a number of technical breakthroughs, many of which have resulted in patent disclosures.

The Logic Cell Array architecture is similar to that of a gate array, with an interior matrix of logic blocks and a surrounding ring of I/O interface blocks. User-programmable interconnection resources are used to create logic networks from these elements. In the Logic Cell Array, the functions of the logic and I/O blocks and the routing of interconnect networks are defined by a configuration program stored in an internal memory. Unlike conventional gate arrays, the Logic Cell Array requires no custom factory fabrication. Each device is identical until programmed by the user. The configuration program is loaded automatically from an EPROM



0010027 1

Figure 1. Logic Technology Tradeoffs

or programmed by a processor whenever the device is powered up, or upon command while the system is operating. Since the configuration program can be copyrighted, designs that employ Logic Cell Arrays can be protected from unauthorized copying under the same legal precedents that have been used effectively to protect microprocessor-based systems.

PROGRAMMABLE LOGIC DEVICES

In PLD architectures, inputs to the AND/OR planes are driven directly by dedicated input pins of the device and some user-selectable input/output pins or feedback paths. Outputs are driven directly from sum-of-products logic outputs or from flip-flops. The primary limitations of this 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 utilization seldom 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. Performance of PLDs is fixed for each level of logic. Each path through the AND/OR plane exhibits the same delay, typically 25–45 ns.

GATE ARRAYS

Array architectures provide flexible resources, both for I/O functions and logic structures. With a gate array, user logic is typically implemented by interconnecting two-input NAND gates into more complex functions using mask-programmed metal segments. Factory processing implements the metal interconnections required for each user configuration. Generation of larger arrays can be accomplished through straightforward extensions of the I/O, logic building blocks, and interconnect resources, much like extending the capacity of a memory device. Gate arrays offer usable densities of 25,000 gates or more. Utilization factors of 80–90% are possible because of the flexibility and regularity of the architecture. Gate array performance is dependent on the placement and interconnection of the elements that make up a logic network. In a gate array characterized by 2 ns gate delays, frequently used functions may have a total delay of 15 ns or more because of the levels of gating and the interconnection required to implement them.

LOGIC CELL ARRAY

Logic Cell Arrays share the gate array architecture's flexibility and ease of extension to higher densities. The function of the LCA's configurable logic blocks and I/O blocks and their interconnection are controlled by a configuration program stored in an on-chip memory. Distributed memory cells are adjacent to the logic, I/O, or interconnect element they control. Interconnect resources exist in the channels between the rows and columns of logic blocks and between the logic blocks and the I/O blocks. Through straightforward extensions of the array architecture, the initial 1200 gate LCA has been extended to an 1800 gate version. Further extensions of the LCA architecture will increase the number of usable gates to over 8000.

Like other standard IC components, Logic Cell Arrays permit the selection of higher speed parts from the natural distribution that results from the semiconductor manufacturing process. This permits the choice of the most cost-effective speed grade for a particular application. Logic Cell Array performance is determined by the fixed delays for logic and storage elements plus interconnect delays. During design, the timing calculation software in the development system can quickly display worst case timing. In general, Logic Cell Array performance is specified by the maximum toggle rate for a logic block storage element 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.

PROCESS

Over the last five years, the most pronounced trend in semiconductor manufacturing processes has been the shift toward CMOS. This has been especially true for ASICs. The advantages of advanced CMOS processes include both high speed and low power consumption.

Logic Cell Arrays are fabricated through a partnership with Seiko-Epson, by means of an advanced twin-well, double-layer metal CMOS process. Two metal layers are essential for array architectures because of the need to propagate logic signals in both horizontal and vertical directions with minimum delays. The LCA manufacturing process is very similar to that used for high-speed memories. As a result it can exploit the photolithography and wafer diameter advances in memory process technology which result in ever higher density and performance at ever decreasing costs.

QUALITY

As quality consciousness has grown among semiconductor users, awareness of the importance of testing has increased. Microprocessors, memories, and other standard products are designed for testability and are tested exhaustively with carefully developed test programs. The testing of most application specific ICs is less comprehensive, due to limitations of design and test program development. With respect to testing, the Logic Cell Array is like other standard products. It has been designed with 100% testability as a requirement and each device is comprehensively tested during the manufacturing process. This is accomplished without any participation by the user in the definition of test programs or the generation of test vectors.

RELIABILITY

The manufacturing process used for the Logic Cell Array is based on a process developed for high performance CMOS static memories. Extensive work on this process to insure the highest quality memory devices has provided the same benefits to the Logic Cell Array. Data collected over millions of Logic Cell Array device hours confirm the reliability of the design and the process. Compared with other logic devices, the Logic Cell Array exhibits extremely low power dissipation. This translates to lower operating temperatures and, correspondingly higher reliability. Packaging materials for the Logic Cell Array have been selected to match closely the thermal coefficient of expansion of the silicon. This minimizes thermal stresses and further improves reliability.

The memory cell used to store the LCA configuration program has been designed to be particularly robust. This memory is written only during device configuration and its static output controls logic elements in the array. Since 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 state. In addition tests involving bombardment with high levels of alpha radiation verify that the storage cell is not disturbed by alpha particles.

DEVELOPMENT SYSTEMS

The development system for the Logic Cell Array is similar in capabilities and usage to development systems for microprocessors.

After the initial design information is entered into the XACT™ development system, it is checked for consistency and obvious errors. The design is then translated into a program for the LCA, which can be stored in EPROM devices or in some other media as needed by the final system. For debugging, the configuration is loaded into the LCA memory and in-circuit emulation is used to verify correct operation. Development system support for the Logic Cell Array includes complete facilities for design entry and design verification. The XACT development system includes a basic configuration and several options to enhance designer productivity. Features of the system include:

- A consistent user-friendly environment under which all the development software and options are available
- Graphic driven design entry
- Schematic entry
- Interactive timing calculations
- Macro library support, both for standard Xilinx 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 netlist extraction
- Logic and timing simulation (P-SILOS™)
- In-circuit emulation for multiple devices

The XACT system operates on an IBM® PC/XT™, PC/AT™ or compatible system. Color graphics is required as well as 640 K bytes of internal RAM and a mouse. A full system also requires a single parallel port and two serial ports. Support for additional schematic editors and other design workstation platforms are being developed.

Design Entry

The first step in designing with the Logic Cell Array is to partition a design into logic blocks and I/O blocks, based on the capabilities of each of these resources. After partitioning, two alternatives are available for design entry. In the first, the XACT editor is used to enter the design directly. Design elements can be configured logic blocks and I/O blocks or system macros. The XACT editor allows the individual elements of either I/O or logic blocks to be configured directly, either through equations or Karnaugh maps. A macro can be selected to automatically configure a block or group of blocks for a specified function. Alternatively, the design can be entered in schematic form using the XACT design library with a supported schematic entry system.

Placement and routing of the individual logic and I/O blocks may be performed interactively using the interconnect portion of the XACT editor. Alternatively, the optional Automatic Place and Route package can be used to place logic elements from a file created either by XACT or a schematic package and automatically route the logic networks. The output files produced by the schematic editor and by the auto place/route program are compatible with XACT, so that it can be used interactively for design optimization.

Timing

Timing for critical logic paths can be determined to check design performance. Delays associated with the routing of a particular signal path are displayed automatically. This represents the total worst-case delay from the source block for that signal to the destination currently indicated by the mouse. In addition to displaying timing for individual networks, XACT can produce a listing showing timing for all logic networks in a design.

Simulation

After a design has been entered into XACT, a logic network can be extracted to create a netlist and timing model file for use with the optional P-SILOS logic and timing simulator. Because the XACT system automatically generates the logic description for the simulator, the user need only supply the input stimuli. Simulator output can be printed in a tabular format that shows the state of each selected node whenever any of the nodes changes. These data can be used to determine the relative timing of signals under worst-case conditions. The timing models from which the relative timing data are extracted are based on the implementation of the design and worst-case temperature, voltage and process conditions. Timing waveforms also can be generated automatically.

In-Circuit Emulation

The ultimate verification of a design is its correct operation in the final system in which it will be used. This is the function of the optional XACTOR™ in-circuit emulator. Completed designs that have been converted into configuration programs are loaded directly into operating devices connected to the user's system. The system consists of software and hardware which is attached to the host PC system.

The XACTOR in-circuit emulator allows the user to emulate up to four devices simultaneously, with several design variations for each device. Emulation is accomplished by selectively isolating and controlling the pins of the Logic Cell Array which are associated with programming and overall device control. Isolation from the user's target system is accomplished with electronic switches controlled from the host system.

The emulator control software allows the user to program devices on command from either the development system or the target system. The system can also read and display the states of the internal logic block storage elements and I/O block inputs. Additional capabilities include reporting asynchronous events which occur in the target system and support for daisy chain programming of Logic Cell Arrays.

XC2064
XC2018
Logic Cell™ Array

Table of Contents

XILINX

XC2064
XC2018
Logic Cell™ Array

Product Specification

FEATURES

- Fully user-programmable:
  - I/O functions
  - Digital logic functions
  - Interconnections
- General-purpose array architecture
- Complete user control of design cycle
- Compatible arrays with logic cell complexity equivalent to 1200 and 1800 usable gates
- Standard product availability
- 100% factory-tested
- Selectable configuration modes
- Low-power, CMOS, static memory technology
- Performance equivalent to TTL SSI/MSI
- TTL or CMOS input thresholds
- Complete development system support
  - XACT Design Editor
  - Schematic Entry
  - XACTOR In Circuit Emulator
  - Macro Library
  - Timing Calculator
  - Logic and Timing Simulator
  - Auto Place / Route

DESCRIPTION

The Logic Cell™ Array (LCA) is a high density CMOS integrated circuit. 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 networks 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)
XC2064	1200	64	58	12038
XC2018	1800	100	74	17878

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.

ARCHITECTURE

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.

Configuration Memory

The configuration of the Xilinx 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, which is covered by a pending patent application, 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 rewritten.

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.

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 electro-static 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

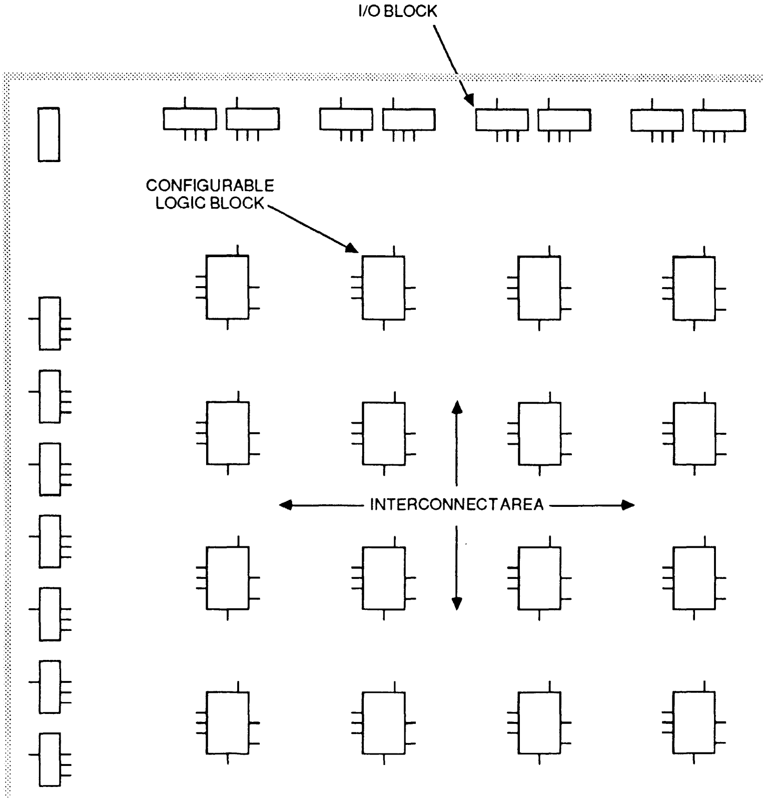


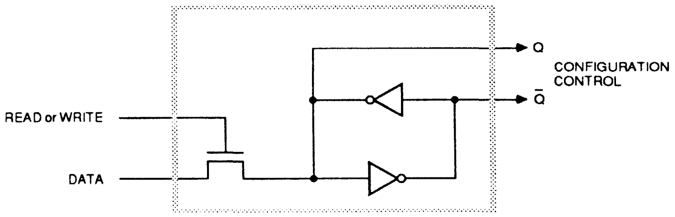
Figure 1. Logic Cell Array Structure

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 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 .

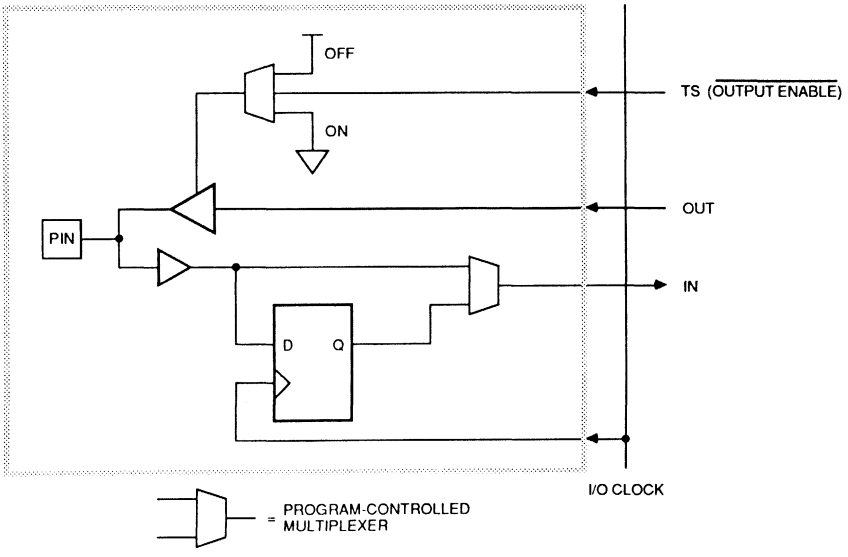
Configurable Logic Block

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 XC2064 has 64 such



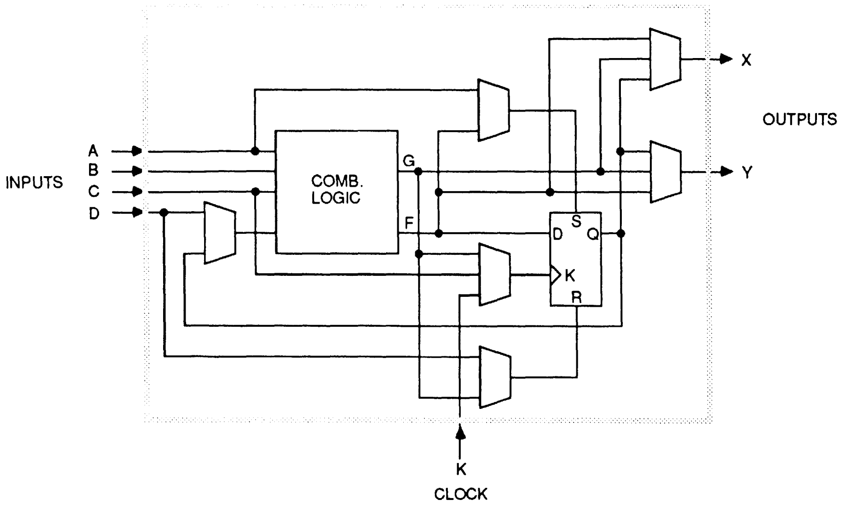
0010003 2

Figure 2. Configuration Memory Cell


OFF
ON
TS ($\overline{\text{OUTPUT ENABLE}}$)
OUT
PIN
IN
D
Q
I/O CLOCK
= PROGRAM-CONTROLLED MULTIPLEXER

0010003 3

Figure 3. I/O Block



0010003 4

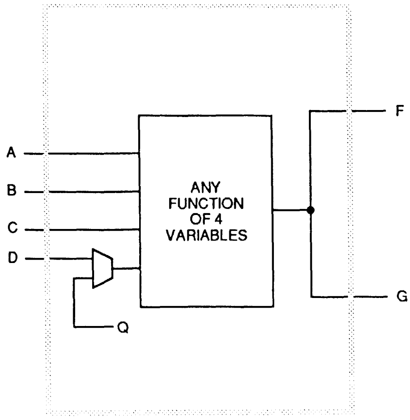
Figure 4. Configurable Logic Block

blocks arranged in an 8-row by 8-column matrix. The XC-2018 has 100 logic blocks arranged in a 10 by 10 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.

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.

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



OPTION 1

1 FUNCTION OF 4 VARIABLES

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

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.

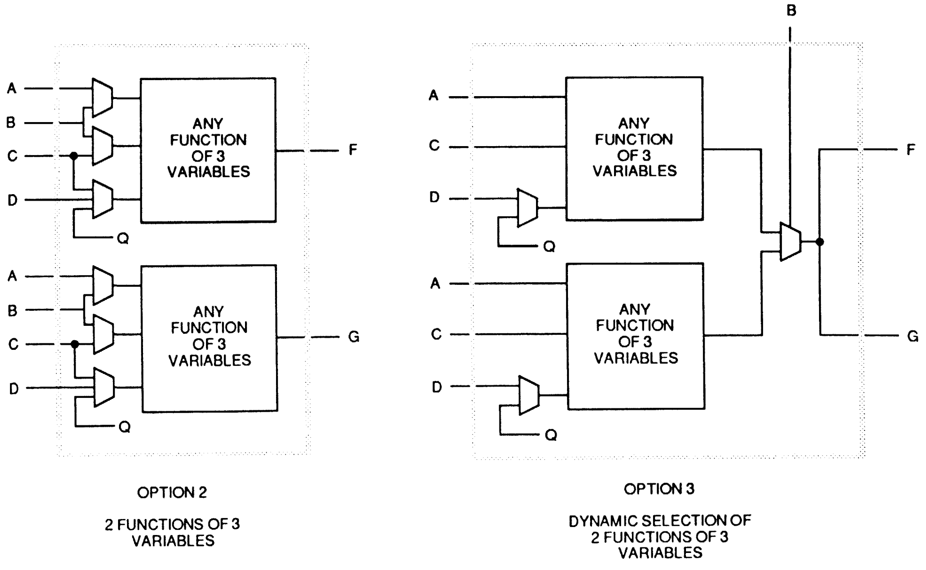
The storage element data input is supplied from the function F output of the combinatorial logic. Asynchronous SET and RESET controls are provided for each storage element. The user may enable these controls independently and select their source. They are active high inputs and the asynchronous reset is dominant. The storage elements are reset by the active-low chip 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 interconnection 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



A
B
C
D
ANY FUNCTION OF 3 VARIABLES
F
Q
A
B
C
D
ANY FUNCTION OF 3 VARIABLES
G
Q
OPTION 2
2 FUNCTIONS OF 3 VARIABLES
B
A
C
D
ANY FUNCTION OF 3 VARIABLES
Q
F
A
C
D
ANY FUNCTION OF 3 VARIABLES
Q
G
OPTION 3
DYNAMIC SELECTION OF 2 FUNCTIONS OF 3 VARIABLES

0010003 5

Figure 5. CLB Combinatorial Logic Options

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.

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 unidirectional (as are block outputs) they are usable *only* for input connection. 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.

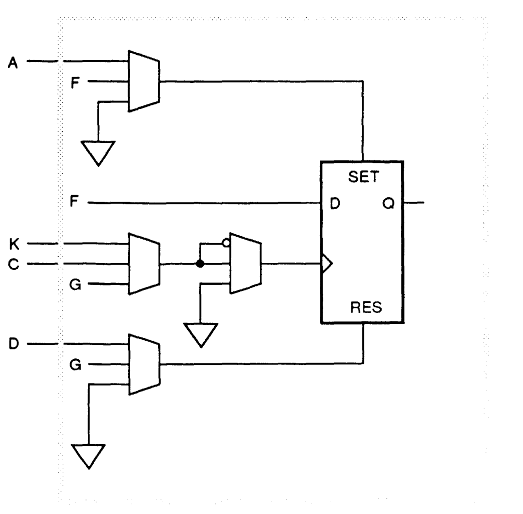


Figure 6. CLB Storage Elememt

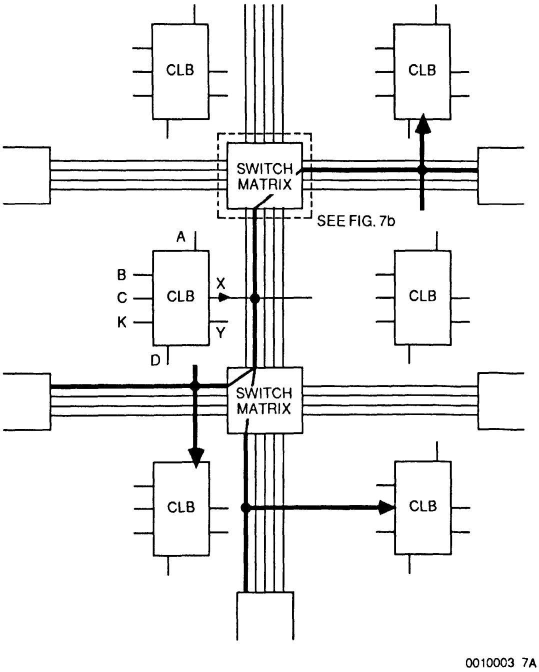


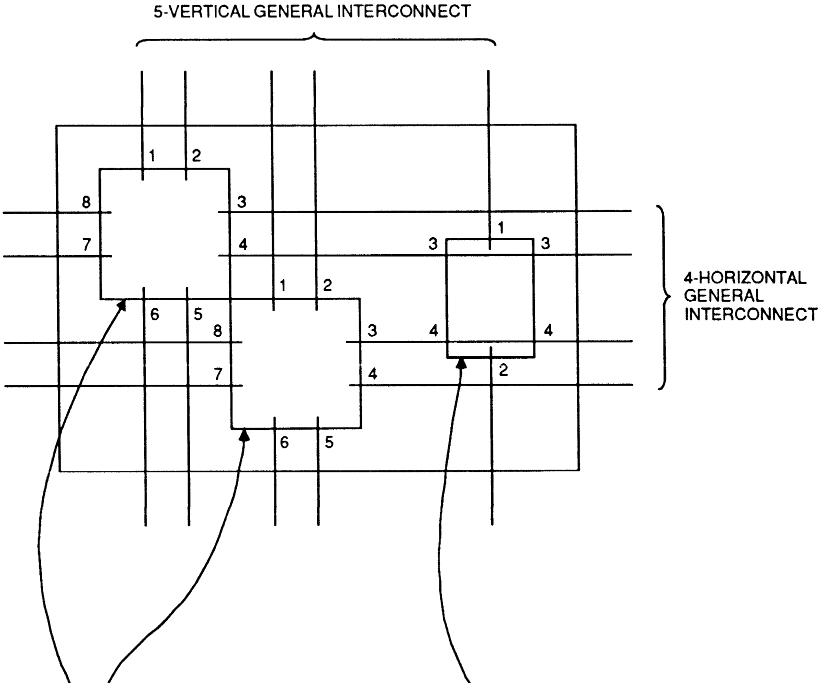
Figure 7a. General-Purpose Interconnect

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.

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


5-VERTICAL GENERAL INTERCONNECT
1
2
8
3
7
4
1
2
6
5
8
3
7
4
6
5
1
3
3
4
4
2
4-HORIZONTAL GENERAL INTERCONNECT

FROM

TO	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	■

FROM

TO	1	2	3	4
1	■	1	1	1
2	1	■	1	1
3	1	1	■	0
4	1	1	0	■

1 = VALID CONNECTION
0 = INVALID CONNECTION

0010003 7B

Figure 7b. Interconnection Switching Matrix

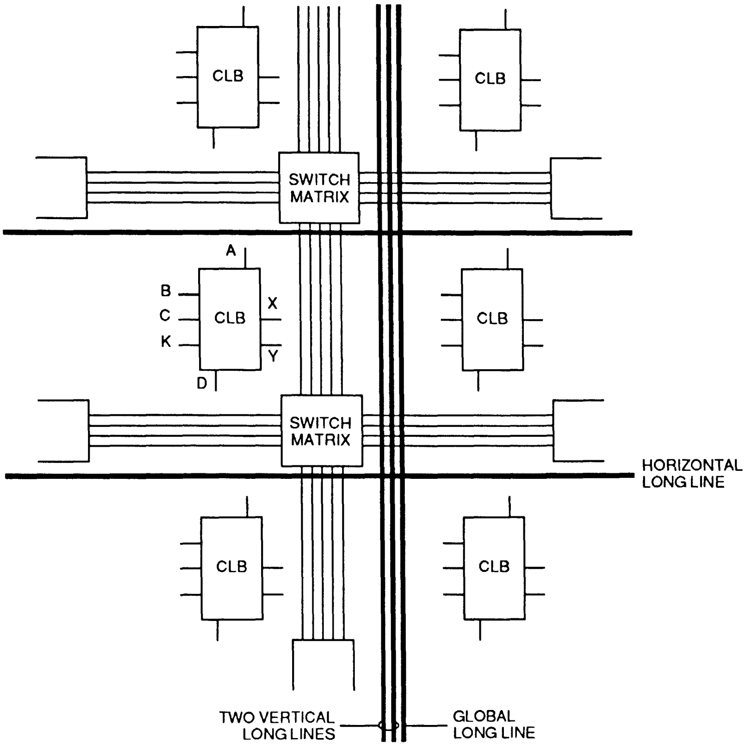
logic blocks. Alternatively, these long lines can be driven by a logic or I/O block on a column by column basis. This capability provides a common, low-skew clock or control line within each column of logic blocks. Interconnections of these long lines are shown in Figure 8b.

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 Configurable Logic Block, 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.

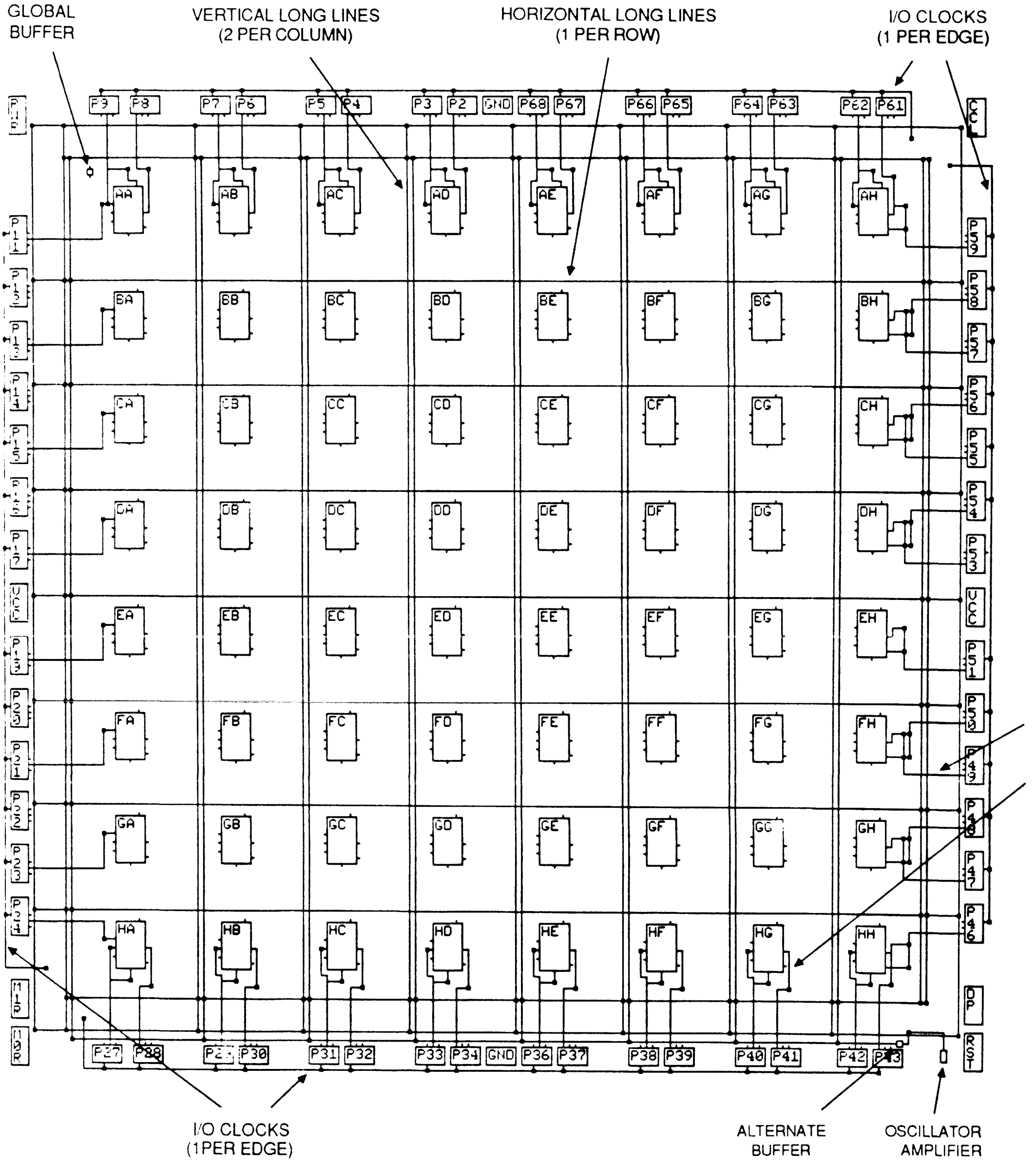
Crystal Oscillator

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, produce the 360-degree



0010003 8A

Figure 8a. Long Line Interconnect



0010003 8B

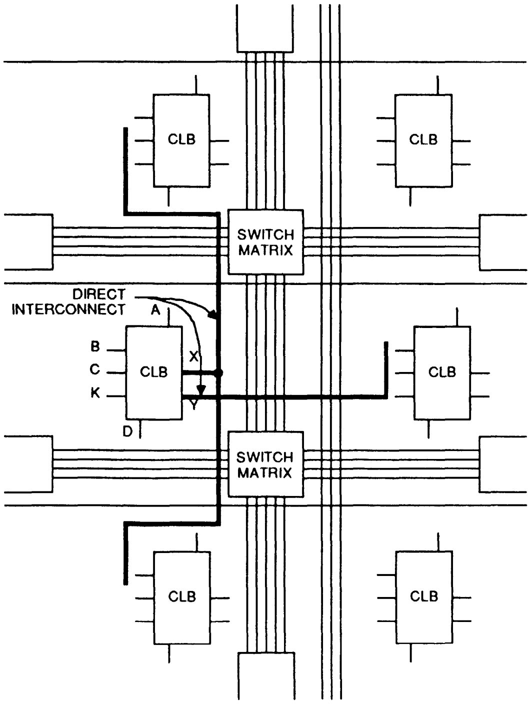
Figure 8b. XC2064 Long Lines, I/O Clocks, I/O Direct Interconnect

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.

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 48 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.



0010003 9

Figure 9. Direct Interconnect

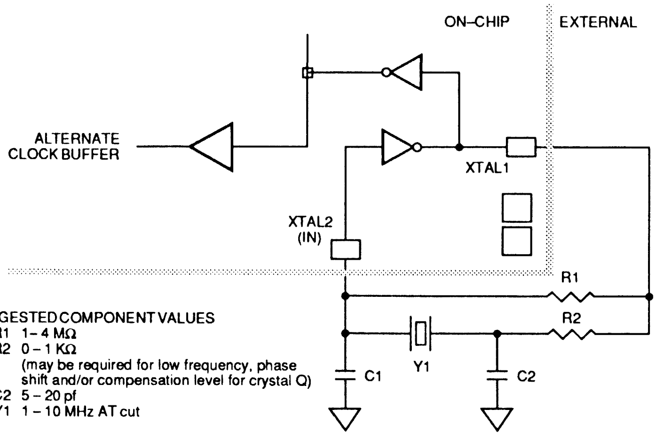
Output buffers capable of driving the specified 4 mA loads under worst-case conditions may be capable of driving 25 to 30 times that current in a best case. Noise can be reduced by minimizing external load capacitance and reducing simultaneous output transitions in the same direction. It may also be 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.

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 2 of the 16 bits. In a 4-input AND gate there will be 2 transitions in 16 states. Typical global clock buffer power is about 3 mW / MHz for the XC2064 and 4mW / MHz for the XC2018. With a "typical" load of three general interconnect segments, each Configurable Logic Block output requires about 0.4 mW / MHz of its output frequency. Graphs of power versus operating frequency are shown in Table 1.

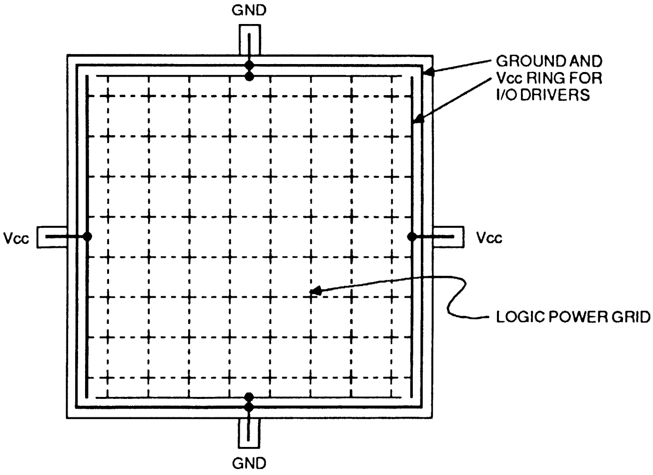


SUGGESTED COMPONENT VALUES
R1 1 - 4 MΩ
R2 0 - 1 KΩ
(may be required for low frequency, phase shift and/or compensation level for crystal Q)
C1, C2 5 - 20 pf
Y1 1 - 10 MHz AT cut

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

0010003 10

Figure 10. Crystal Oscillator



0010003 11

Figure 11. LCA Power Distribution

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. At power-up, all inputs are TTL-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 13 shows the specific data arrangement for the XC2064 device. Future products will use the same data format to maintain compatibility between different devices of the Xilinx product line, but they will have different sizes and numbers of data frames. For the XC2064, configuration requires 12,038 bits for each device. For the XC2018, the configuration of each device requires 17,878 bits. The XC2064 uses 160 configuration data frames and the XC2018 uses 197.

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 / $\overline{\text{PROG}}$ 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 , $\overline{\text{RESET}}$, to other portions of the system. High during configuration (HDC) and low during configuration ($\overline{\text{LDC}}$), are released one CCLK cycle before DONE is asserted. In master mode configurations, it is convenient to use $\overline{\text{LDC}}$ 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, eg. PROM address lines swapped, the LCA will not be ready at the length count and the counter will cycle through an additional 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.

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. Con-

0010003 14

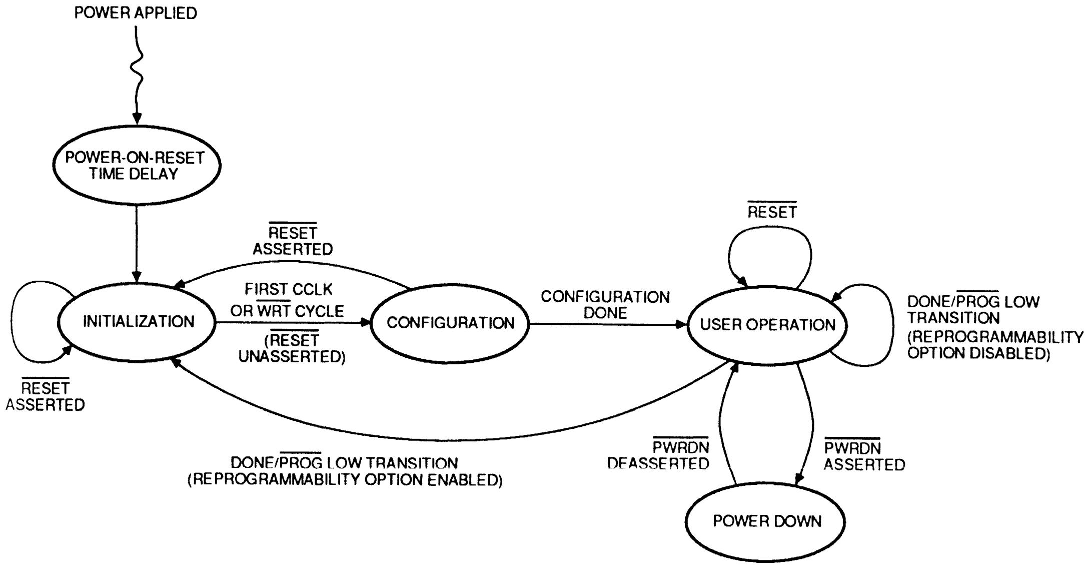
MODE PIN			MODE SELECTED
M0	M1	M2	
0	0	0	MASTER SERIAL
0	0	1	MASTER LOW MODE
0	1	1	MASTER HIGH MODE
1	0	1	PERIPHERAL MODE
1	1	1	SLAVE MODE

MASTER LOW ADDRESSES BEGIN AT 0000 AND INCREMENT
MASTER HIGH ADDRESSES BEGIN AT FFFF AND DECREMENT

Figure 14. Configuration Mode Selection

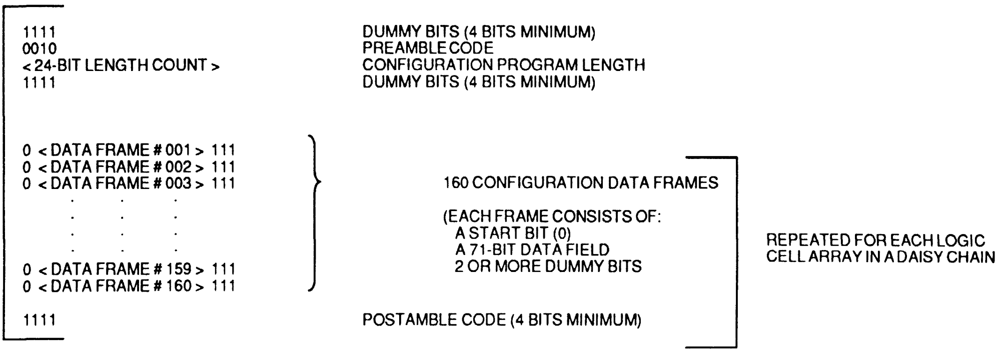
figuration using peripheral or slave modes must be delayed long enough for this initialization to be completed.

The initialization phase may be extended by asserting the active-low external $\overline{\text{RESET}}$. If a configuration has begun, an assertion of $\overline{\text{RESET}}$ will initiate an abort, including an orderly clearing of partially loaded configuration memory bits. After about 3 clock cycles for synchronization, initialization will require about 160 additional cycles of the internal sampling clock (197 for the XC2018) 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 / $\overline{\text{PROG}}$ 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.


POWER APPLIED
POWER-ON-RESET TIME DELAY
$\overline{\text{RESET}}$ ASSERTED
FIRST CCLK OR $\overline{\text{WRT}}$ CYCLE
($\overline{\text{RESET}}$ UNASSERTED)
INITIALIZATION
CONFIGURATION
CONFIGURATION DONE
$\overline{\text{RESET}}$
USER OPERATION
DONE/$\overline{\text{PROG}}$ LOW TRANSITION (REPROGRAMMABILITY OPTION DISABLED)
$\overline{\text{RESET}}$ ASSERTED
DONE/$\overline{\text{PROG}}$ LOW TRANSITION (REPROGRAMMABILITY OPTION ENABLED)
$\overline{\text{PWRDN}}$ DEASSERTED
$\overline{\text{PWRDN}}$ ASSERTED
POWER DOWN

0010003 12

Figure 12. Configuration State Diagram



```
1111                            DUMMY BITS (4 BITS MINIMUM)
0010                            PREAMBLE CODE
< 24-BIT LENGTH COUNT >         CONFIGURATION PROGRAM LENGTH
1111                            DUMMY BITS (4 BITS MINIMUM)

0 < DATA FRAME # 001 > 111  }
0 < DATA FRAME # 002 > 111  }
0 < DATA FRAME # 003 > 111  }        160 CONFIGURATION DATA FRAMES
   .    .    .              }
   .    .    .              }        (EACH FRAME CONSISTS OF:
   .    .    .              }         A START BIT (0)                 REPEATED FOR EACH LOGIC
0 < DATA FRAME # 159 > 111  }         A 71-BIT DATA FIELD             CELL ARRAY IN A DAISY CHAIN
0 < DATA FRAME # 160 > 111  }         2 OR MORE DUMMY BITS

1111                            POSTAMBLE CODE (4 BITS MINIMUM)
```

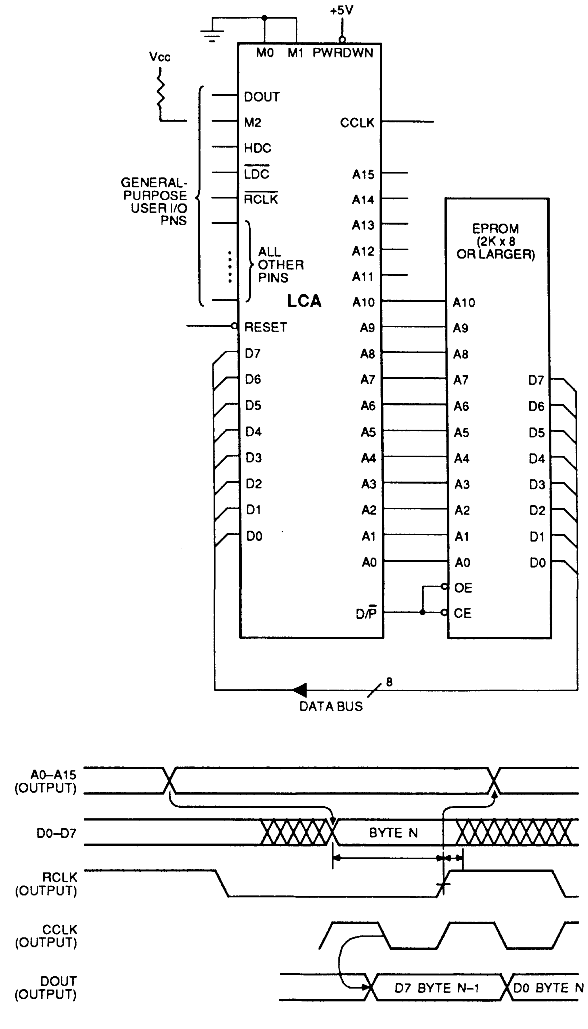
0010003 13

Figure 13. XC2064 Configuration Data Arrangement

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 $\overline{\text{RCLK}}$ (read clock), HDC (high during configuration) and $\overline{\text{LDC}}$ (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



0010003 15A

Figure 15a. Master Low Address Configuration

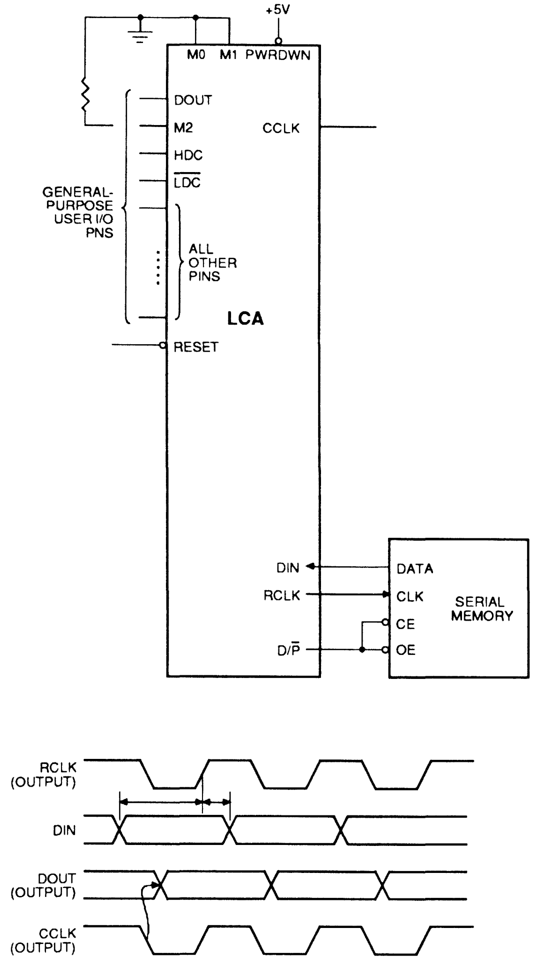
low memory, the Logic Cell Array can load itself from high memory and enable the processor to begin execution once configuration is completed. The DONE / PROG output pin can be used to hold the processor in a Reset state until the Logic Cell Array has completed the configuration process.

The master serial mode uses serial configuration data, synchronized by the rising edge of RCLK, as in Figure 15b.

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 (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 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



0010003 15A1

Figure 15b. Master Serial Mode Configuration

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.

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 of 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.

Daisy Chain

The daisy-chain programming mode is supported by Logic Cell Arrays 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 2 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

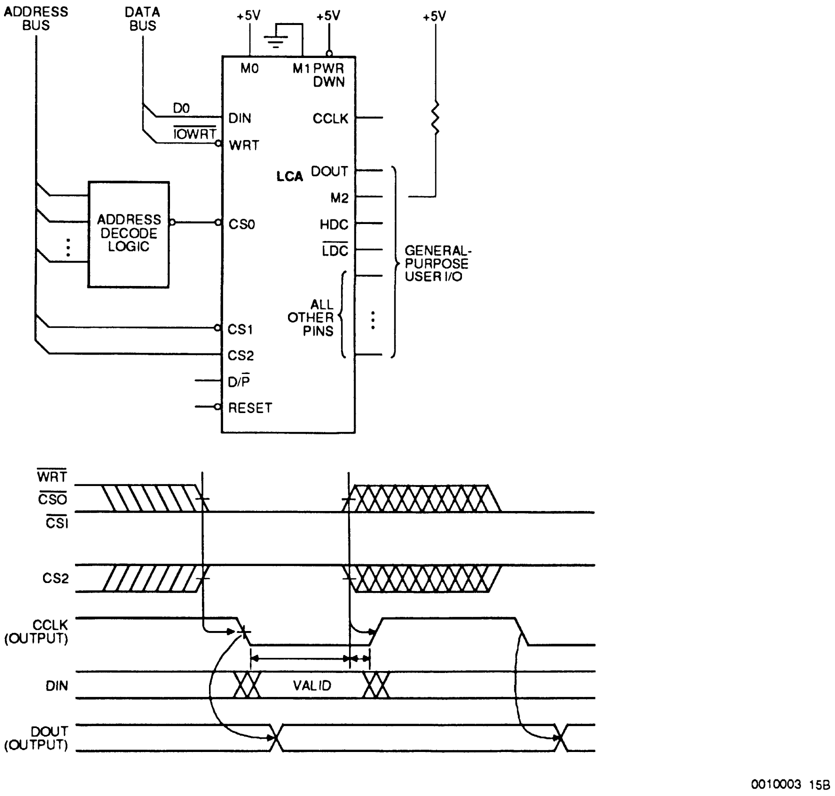


Figure 16. Peripheral Mode Configuration

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 8 times the EPROM address rate, while a peripheral mode device produces CCLK from the chip select and write strobe timing.

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 config-uration is complete to allow time for stabilization before it is connected to the internal circuitry.

Special Configuration Functions

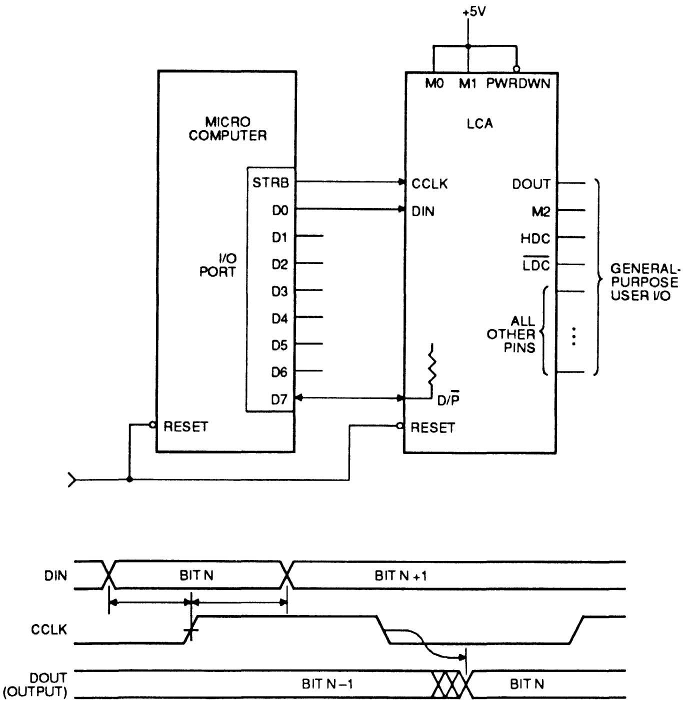
In addition to the normal user logic functions and interconnect, the configuration data include control for several special functions:

- 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 $\overline{\text{PWRDWN}}$ input threshold is an exception; it is always a CMOS level input. The TTL threshold option requires additional power for threshold shifting.



0010003 15C

Figure 17. Slave Mode Configuration

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.

Re-program

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 2–3 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

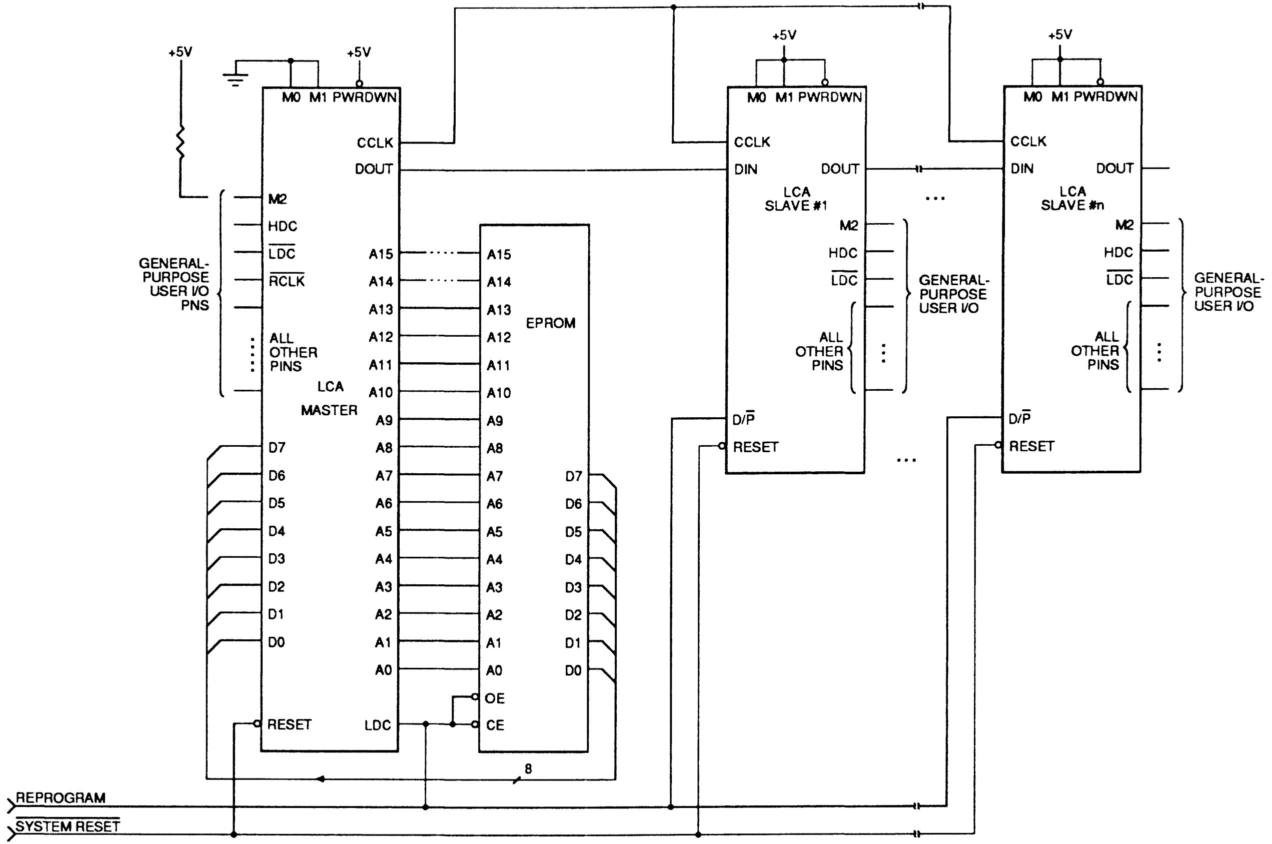
Because the control store of the Logic Cell Array is a CMOS static memory, its cells require only a very low standby current for data retention. In some systems, this low data retention current characteristic facilitates preserving configurations in the event of a primary power loss. The Logic Cell Array 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 volts, the required current is typically on the order of 50 nanoamps. 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 nanoseconds. This short delay provides very good performance under asynchronous clock and data conditions. Short loop delays minimize



0010003 15D

Figure 18. Master Mode with Daisy Chain

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.

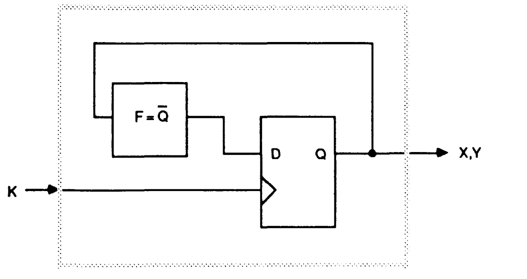


Figure 19. Logic Block Configuration for Toggle Rate Measurement

Logic Block Performance

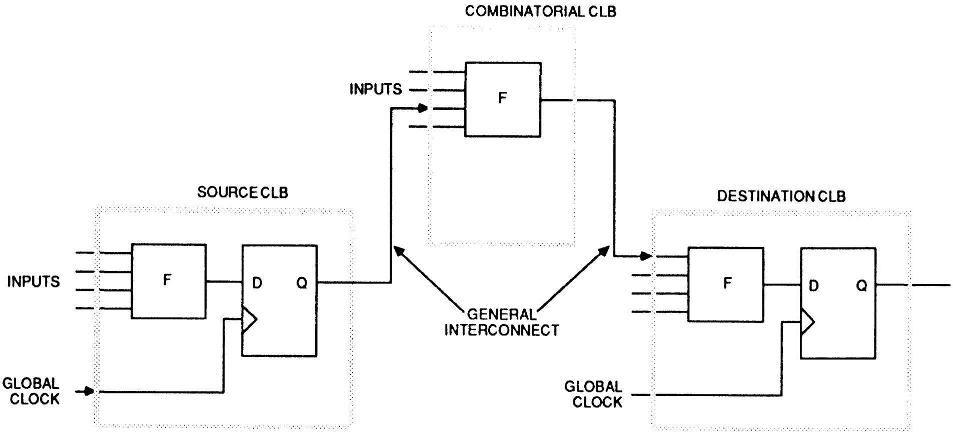
Logic block propagation times are measured from the interconnect point at the input of the combinatorial logic to the output of the block in the interconnect area. Combinatorial performance is independent of logic function because of the table look-up based implementation. Timing is different when the 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 22 and 23.

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.

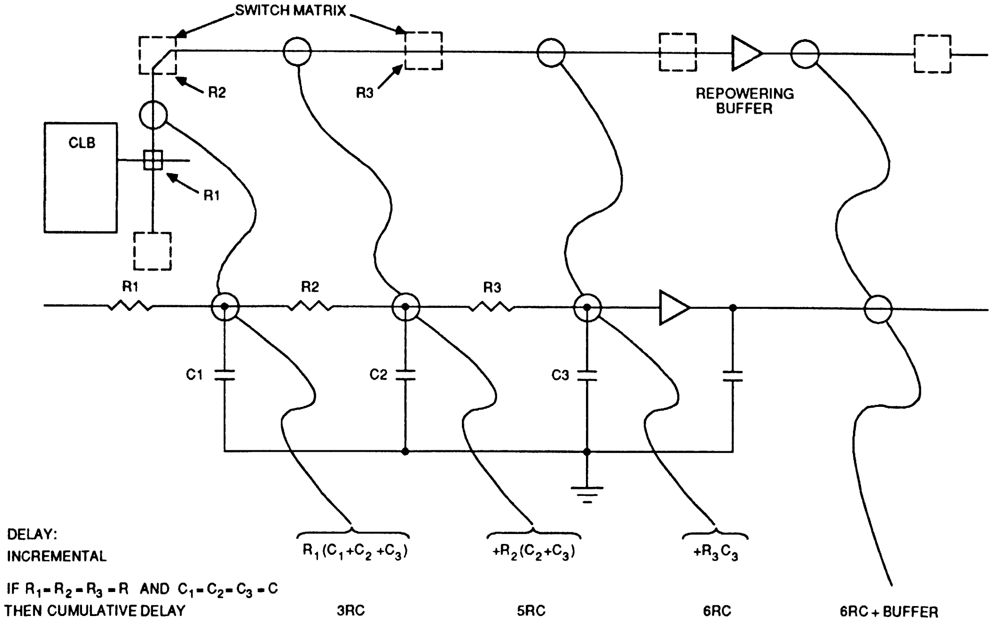
The single metal segment used for long lines exhibits low resistance from end to end, but relatively high capacitance. Signals driven through a programmable switch will have the additional impedance of the switch added to their normal drive impedance.

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 additional 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 21 shows an estimation of this delay.



0010003 16B

Figure 20. Typical Logic Path



0010003 17

Figure 21. Interconnection Delay Example

DEVELOPMENT SYSTEM

To support designers using the Logic Cell Array, Xilinx 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
- Macro library support, both for standard Xilinx 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 DOS 2.1 or higher. Color graphics is required as well as 640K bytes of internal RAM (an Expanded Memory Specification (EMS) card with 256K bytes of memory is required for the XC2018). A complete system requires one parallel I/O port and two serial ports and a mouse.

Designing with XACT

Designing with the Logic Cell Array is similar to using conventional MSI elements or gate array macros. 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 macro 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 macro 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 down loaded 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 Xilinx XACTOR In-Circuit Emulation System directly in the target system and/or the P-Silos™ logic simulator.

PACKAGE PIN DESCRIPTIONS

$\overline{\text{PWRDWN}}$ 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 As *Mode 0,* this input and M1, M2 are sampled before the start of configuration to establish the configuration mode to be used.

RTRIG 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 As *Mode 1*, this input and M0, M2 are sampled before the start of configuration to establish the configuration mode to be used.

$\overline{\text{RDATA}}$ 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 $\overline{\text{RESET}}$. 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.

The DONE open drain output is configurable with or without a pull-up resistor of about 3KΩ. 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.
DONE

$\overline{\text{PROG}}$ 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 circuity 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 circuity 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.

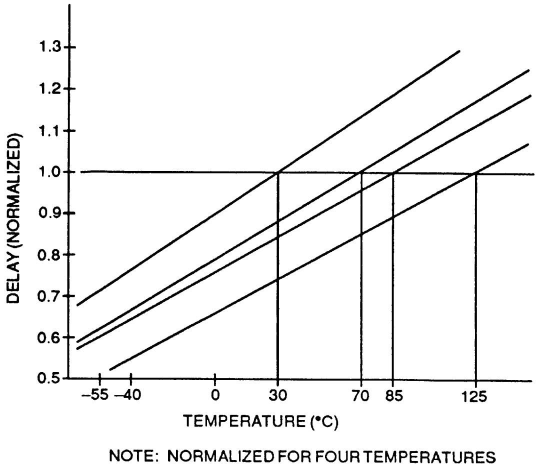
$\overline{\text{CS0}}$, $\overline{\text{CS1}}$
CS2, $\overline{\text{WRT}}$ These 4 inputs represent a set of signals, 3 active low and one active high, which are used in the peripheral mode to control configuration data entry. The assertion of all four generates a LOW CCLK and shifts DOUT data. The 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 8 pins represent the parallel configuration *data* byte for the master mode. After configuration is complete, they are user-programmed I/O.

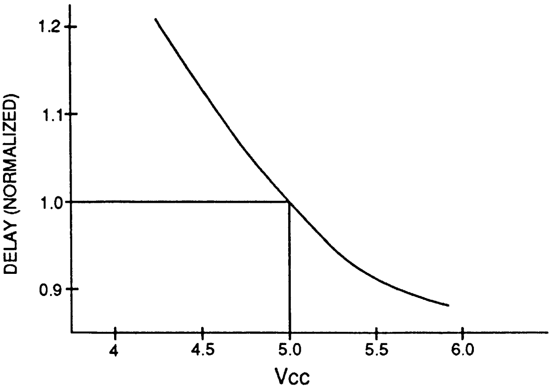
A0–A15 This set of 16 pins present an *address* output for an external configuration memory during master mode. After configuration is complete, they are user-programmed I/O. A12 through A15 are not available in packages with less than 68 pins.

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.



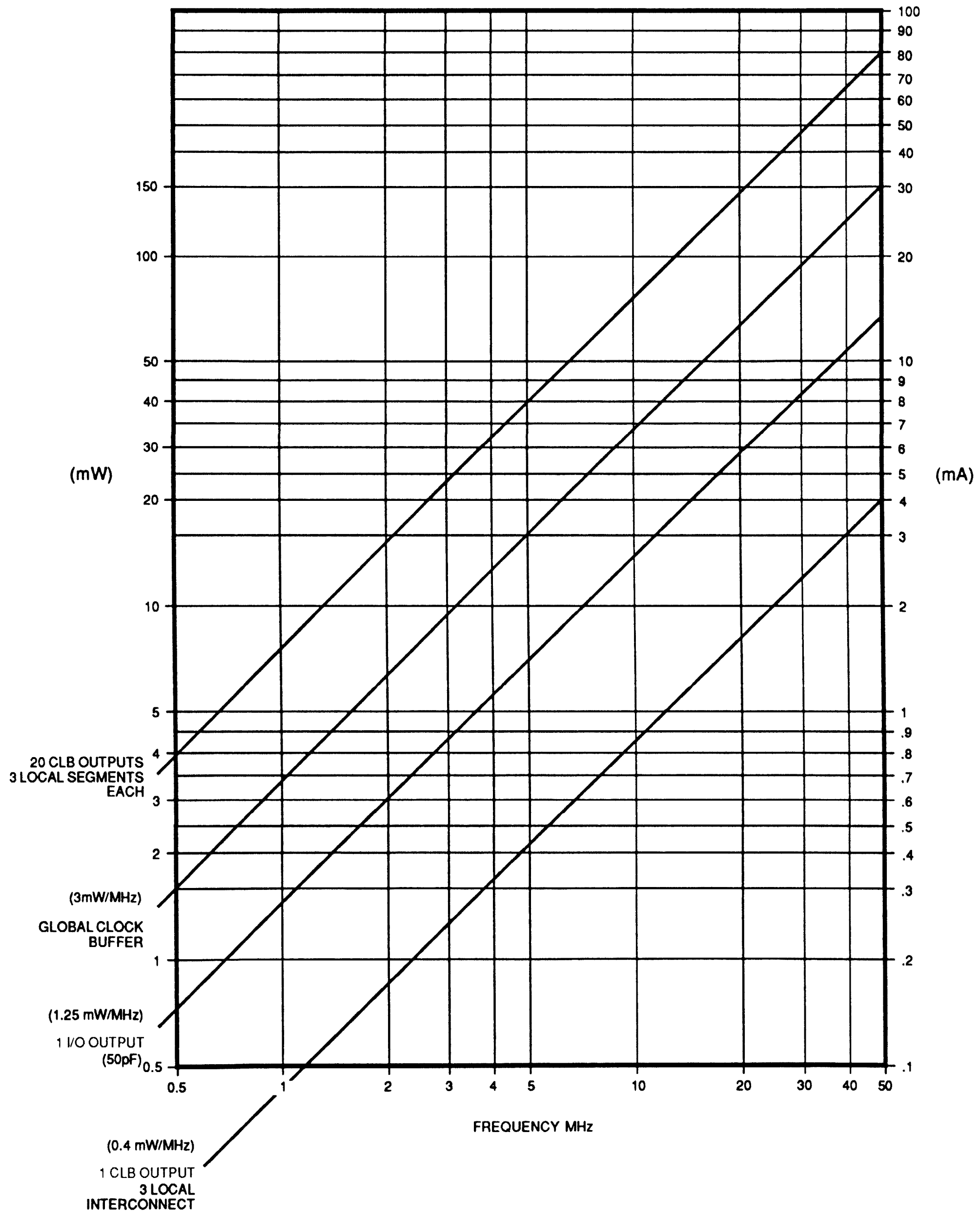
0010003 17A

Figure 22. Delay *vs.* Temperature



0010003 17B

Figure 23. Delay *vs.* Power Supply



0010003 19

Table 1. Typical LCA Power Consumption By Element

48 DIP	68 PLCC	68 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>>				
	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 (LOW)	M0 (HIGH)	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					
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					
	49	H11			D5 (I)		
35	50	G10		$\overline{\text{CSO}}$ (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 20–50 KΩ INTERNAL PULL-UP DURING CONFIGURATION

Table 2a. XC2064 Pin Assignments

0010003 20

68 PLCC	68 PGA	84 PLCC	84 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>>				
		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 (LOW)	M0 (HIGH)	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					
35	K6	43	J6	GND				
		44	J7	<<HIGH>>				I/O
36	L6	45	L7					
37	K7	46	K7					
38	L7	47	L6					
		48	L8					
39	K8	49	K8					
40	L8	50	L9					
41	K9	51	L10			D7 (I)		
42	L9	52	K9			D6 (I)		
43	L10	53	L11					XTL2 OR I/O
44	K10	54	K10	$\overline{\text{RESET}}$ (I)				
45	K11	55	J10	DONE (O)				$\overline{\text{PROG}}$ (I)
46	J10	56	K11	<<HIGH>>				XTL1 OR I/O
47	J11	57	J11					I/O
48	H10	58	H10					
		59	H11					
49	H11	60	F10			D5 (I)		
		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	<<HIGH>>				I/O
54	E10	66	E11		CS2 (I)	D2 (I)		
		67	E10					
55	E11	68	E9					
		69	D11					
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	<<HIGH>>		A0 (O)		I/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			A4 (O)		
67	B7	81	B6			A14 (O)		
		82	B7					
		83	A7					
68	A7	84	C7			A5 (O)		

0010003 21

<<HIGH>> IS HIGH IMPEDANCE WITH A 20–50 KΩ INTERNAL PULL-UP DURING CONFIGURATION

Table 2b. XC2018 Pin Assignments

PARAMETRICS

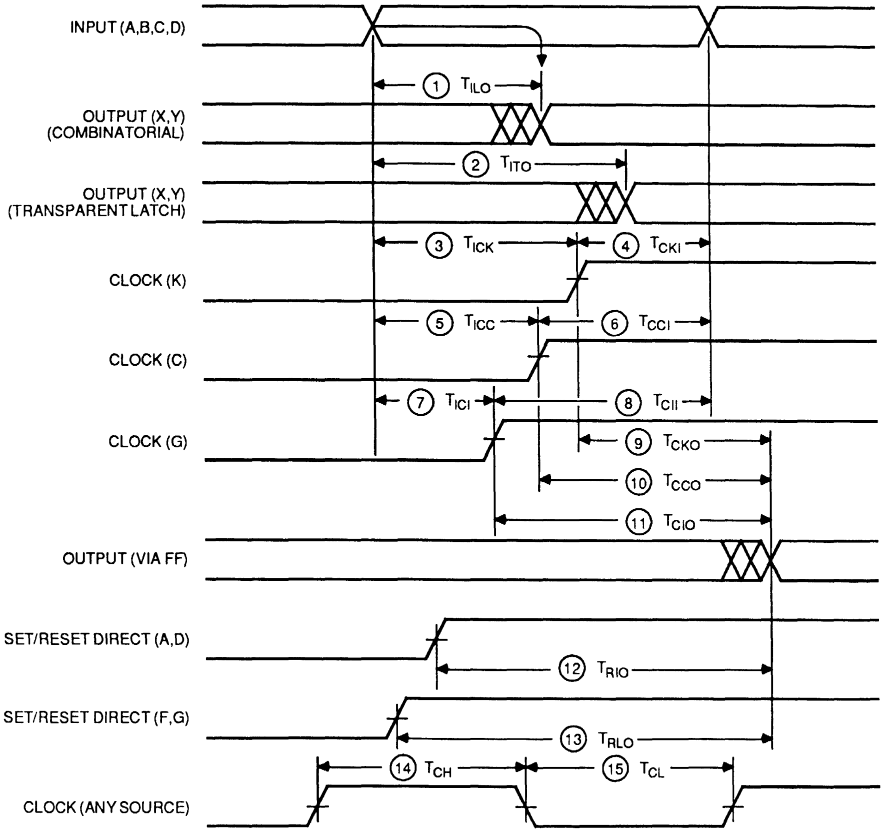
Absolute Maximum Ratings		Units
V_{CC} Supply voltage relative to GND	–0.5 to 7.0	V
V_{IN} Input voltage with respect to GND	–0.5 to V_{CC} + 0.5	V
V_{TS} Voltage applied to three-state output	–0.5 to V_{CC} + 0.5	V
T_{STG} Storage temperature (ambient)	–65 to + 150	° C
T_{SOL} Maximum soldering temperature (10 sec @ 1/16 in.)	+ 260	° 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

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_{CCO}	Quiescent operating power supply current				
	CMOS thresholds (@ V_{CC} = 5.0 V)			5	mA
	TTL thresholds (@ V_{CC} = 5.0 V)			10	mA
I_{CCPD}	Power-down supply current (@ V_{CC} = 5.0 V)			0.5	mA
I_{IL}	Leakage current		–10	+10	µA
C_{IN}	Input capacitance (sample tested)			10	pF

CLB SWITCHING CHARACTERISTICS



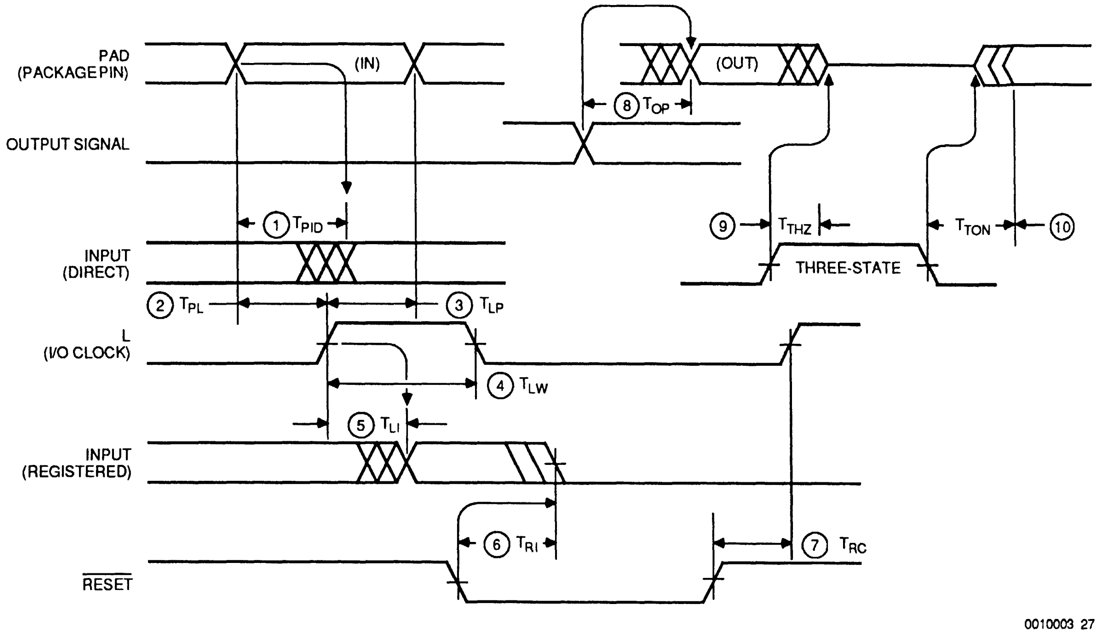
0010003 29

CLB SWITCHING CHARACTERISTICS (Continued)

			Speed Grade	-33		-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	Min	Max	ns
Logic input to Output	Combinatorial	1	T_{ILO}		20		15		10	
	Transparent latch	2	T_{ITO}		25		20		14	
	Additional for Q through F or G to out		T_{QLO}		13		8		6	
K Clock	To output	9	T_{CKO}		20		15		10	
	Logic-input setup	3	T_{ICK}	12		8		7		
	Logic-input hold	4	T_{CKI}	0		0		0		
C Clock	To output	10	T_{CCO}		25		19		13	
	Logic-input setup	5	T_{ICC}	12		9		6		
	Logic-input hold	6	T_{CCI}	6		0		0		
Logic input to G Clock	To output	11	T_{CIO}		37		27		20	
	Logic-input setup	7	T_{ICI}	6		4		3		
	Logic-input hold	8	T_{CII}	9		5		4		
Set/Reset direct	Input A or D to out	12	T_{RIO}		25		22		16	
	Through F or G to out	13	T_{RLO}		37		28		21	
	Master Reset pin to out		T_{MRQ}		35		25		20	
	Separation of set/reset		T_{RS}	17		9		7		
	Set/Reset pulse-width		T_{RPW}	12		9		7		
Flip-flop Toggle rate	Q through F to flip-flop		F_{CLK}	33		50		70		MHz
Clock	Clock high	14	T_{CH}	12		8		7		
	Clock low	15	T_{CL}	12		8		7		

Notes: 1. All switching characteristics apply to all valid combinations of process, temperature and supply.
2. Units are ns unless otherwise specified.

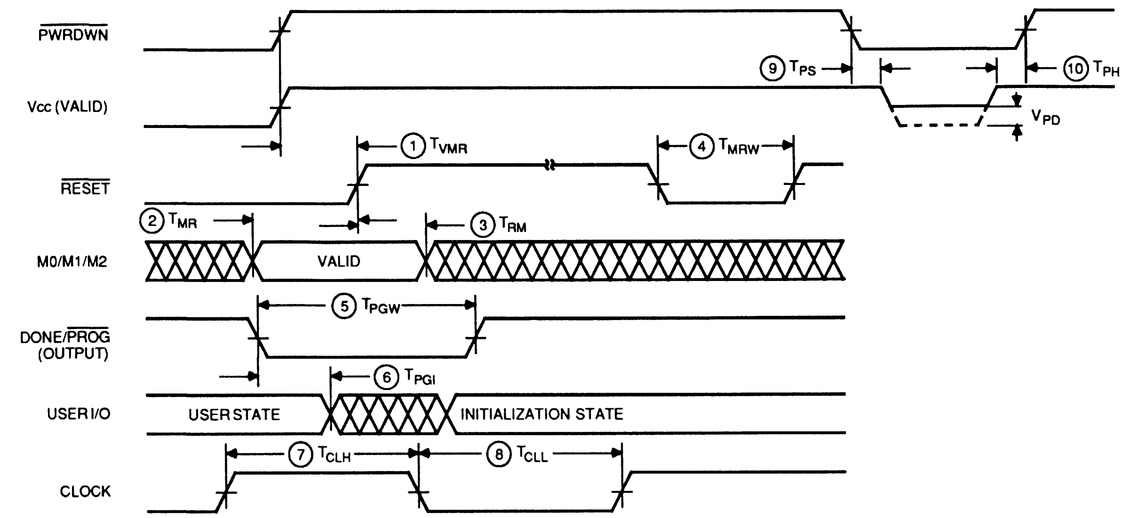
IOB SWITCHING CHARACTERISTICS



				-33		-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	Min	Max	ns
Pad (package pin)	To input (direct)	1	T_{PID}		12		8		6	
I/O Clock	To input (storage)	5	T_{LI}		20		15		11	
	To pad-input setup	2	T_{PL}	12		8		6		
	To pad-input hold	3	T_{LP}	0		0		0		
	Pulse width	4	T_{LW}	12		9		7		
	Frequency				33		50		70	MHz
Output	To pad (output enabled)	8	T_{OP}		15		12		9	
Three-state	To pad begin hi-Z	9	T_{THZ}		25		20		15	
	To pad end hi-Z	10	T_{TON}		25		20		15	
$\overline{\text{RESET}}$	To input (storage)	6	T_{RI}		40		30		25	
	To input clock	7	T_{RC}		35		25		20	

Note: Timing is measured at 0.5 Vcc levels with 50pF output load.

GENERAL LCA SWITCHING CHARACTERISTICS

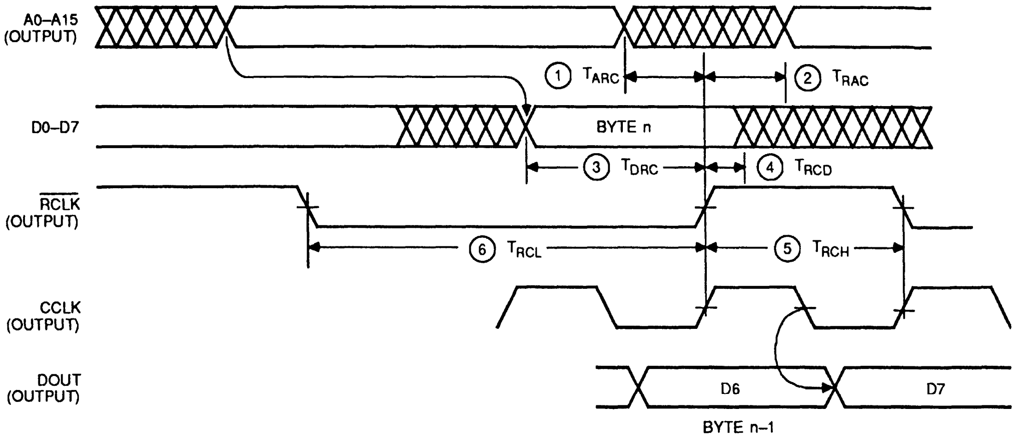


0010003 26C

				-33		-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	Min	Max	ns
$\overline{\text{RESET}}$ (2)	V_{cc} setup (2.0 V)	1	T_{VMR}	150		150		150		
	M2, M1, M0 setup	2	T_{MR}	60		60		60		
	M2, M1, M0 hold	3	T_{RM}	60		60		60		
	Width (low)	4	T_{MRW}	150		150		150		
DONE/$\overline{\text{PROG}}$	Progam width (low)	5	T_{PGW}	6		6		6		µs
	Initialization	6	T_{PGI}		7		7		7	µs
CLOCK	Clock (high)	7	T_{CLH}	12		8		7		
	Clock (low)	8	T_{CLL}	12		8		7		
$\overline{\text{PWR DWN}}$	Setup to V_{CC}	9	T_{PS}	0		0		0		
	Hold from V_{CC}	10	T_{PH}	0		0		0		
	Power Down		V_{PD}	2.0		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 mode.
2. $\overline{\text{RESET}}$ timing relative to power-on and valid mode lines (M0, M1, M2) is relevant only when $\overline{\text{RESET}}$ is used to delay configuration.
3. Minimum CLOCK widths for the auxillary buffer are 1.25 times the T_{CLH}, T_{CLL}.

MASTER MODE PROGRAMMING SWITCHING CHARACTERISTICS

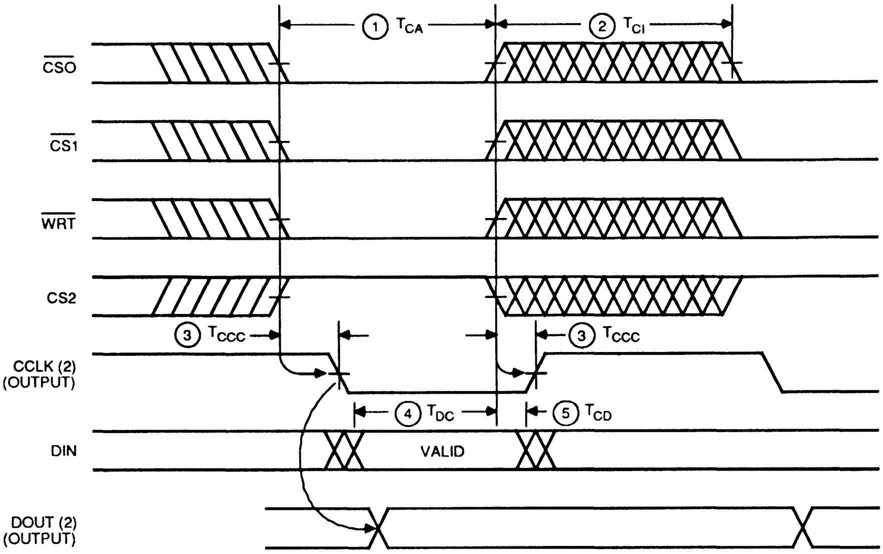


0010003 26A

				-33		-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	Min	Max	ns
RCLK	From address invalid	1	T_{ARC}		0		0		0	
	To address valid	2	T_{RAC}		200		200		200	
	To data setup	3	T_{DRC}	60		60		60		
	To data hold	4	T_{RCD}	0		0		0		
	$\overline{RCLK}$ high	5	T_{RCH}	600		600		600		
	$\overline{RCLK}$ low	6	T_{RCL}	4.0		4.0		4.0		µs

Notes: 1. CCLK and DOUT timing are the same as for slave mode.
2. At power-up, Vcc must rise from 2.0 V to Vcc min. in less than 10 ms.

PERIPHERAL MODE PROGRAMMING SWITCHING CHARACTERISTICS

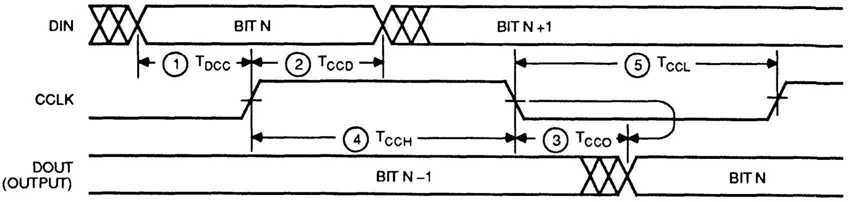


0010003 28B

				-33		-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	Min	Max	ns
Controls[1] ($\overline{CS0}$, $\overline{CS1}$, CS2, $\overline{WRT}$)	Active (last active input to first inactive)	1	T_{CA}	0.25	5.0	0.25	5.0	0.25	5.0	µs
	Inactive (first inactivate input to last active)	2	T_{CI}	0.25		0.25		0.25		µs
	CCLK[2] DIN setup DIN kold	3 4 5	T_{CCC} T_{DC} T_{CD}	 35 5	75	 35 5	75	 35 5	75	

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 Vcc min.

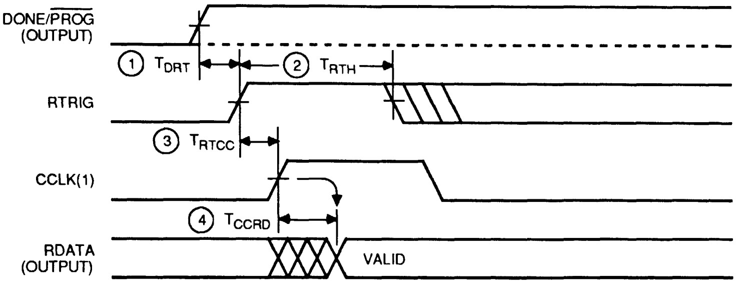
SLAVE MODE PROGRAMMING SWITCHING CHARACTERISTICS



				-33		-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	Min	Max	ns
CCLK	To DOUT	3	T_{CCO}		65		65		65	
	DIN setup	1	T_{DCC}	0		0		0		
	DIN hold	2	T_{CCD}	40			40		40	
	High time	4	T_{CCH}	0.25		0.25		0.25		µs
	Low time	5	T_{CCL}	0.25	5.0	0.25	5.0	0.25	5.0	µs
	Frequency		F_{CC}		2		2		2	MHz

Note: Configuration must be delayed at least 40 ms after Vcc min.

PROGRAM READBACK SWITCHING CHARACTERISTICS



				-33		-50		-70		Units
	Description	Symbol		Min	Max	Min	Max	Min	Max	ns
RTRIG	PROG setup	1	T_{DRT}	300		300		300		
	RTRIG high	2	T_{RTH}	250		250		250		
CCLK	RTRIG setup	3	T_{RTCC}	100		100		100		
	RDATA delay	4	T_{CCRD}		100		100		100	

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).

		48 PIN		68 PIN		84 PIN	
		PLASTIC DIP	CERAMIC DIP	PLASTIC PLCC	CERAMIC PGA	PLASTIC PLCC	CERAMIC PGA
		-PD 48	-CD 48	-PC 68	-PG 68	-PC 84	-PG 84
XC-2064	-33	C	I M	C I	M		
	-50	C		C			
	-70	C		C			
XC-2018	-33			C I		C I	
	-50			C		C	
	-70			C		C	

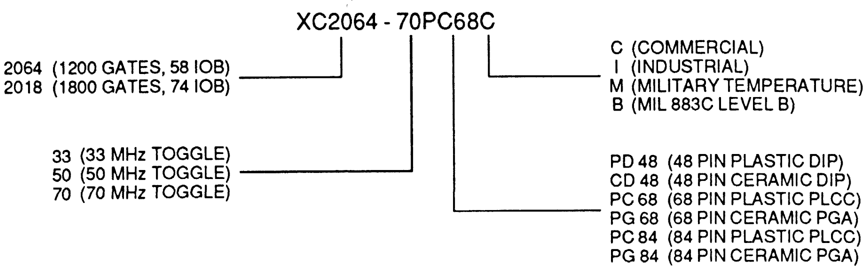
C = COMMERCIAL		0° C TO 70° C
I = INDUSTIAL		-40° C TO 85° C
M = MILITARY		-55° C TO 125° C
B = MILITARY		MIL 883C LEVEL B

0010003 22

Table 5. LCA Package and Temperature Options

Ordering Information

Further information is available from Xilinx franchised distributors or from the nearest Xilinx sales representative. Part numbers are composed as follows:



XC2064 - 70PC68C

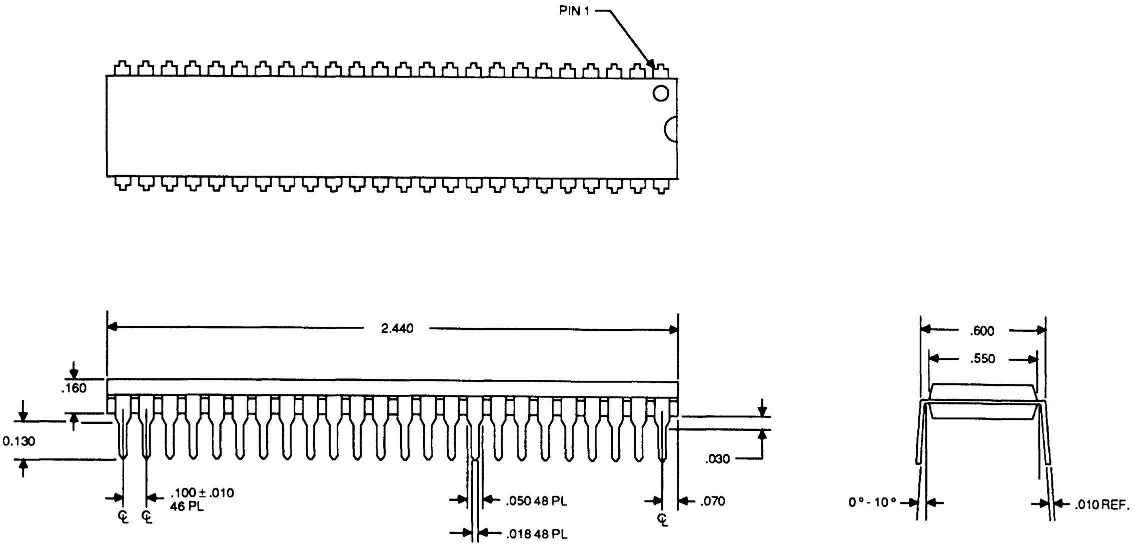
2064 (1200 GATES, 58 IOB)
2018 (1800 GATES, 74 IOB)

33 (33 MHz TOGGLE)
50 (50 MHz TOGGLE)
70 (70 MHz TOGGLE)

C (COMMERCIAL)
I (INDUSTRIAL)
M (MILITARY TEMPERATURE)
B (MIL 883C LEVEL B)

PD 48 (48 PIN PLASTIC DIP)
CD 48 (48 PIN CERAMIC DIP)
PC 68 (68 PIN PLASTIC PLCC)
PG 68 (68 PIN CERAMIC PGA)
PC 84 (84 PIN PLASTIC PLCC)
PG 84 (84 PIN CERAMIC PGA)

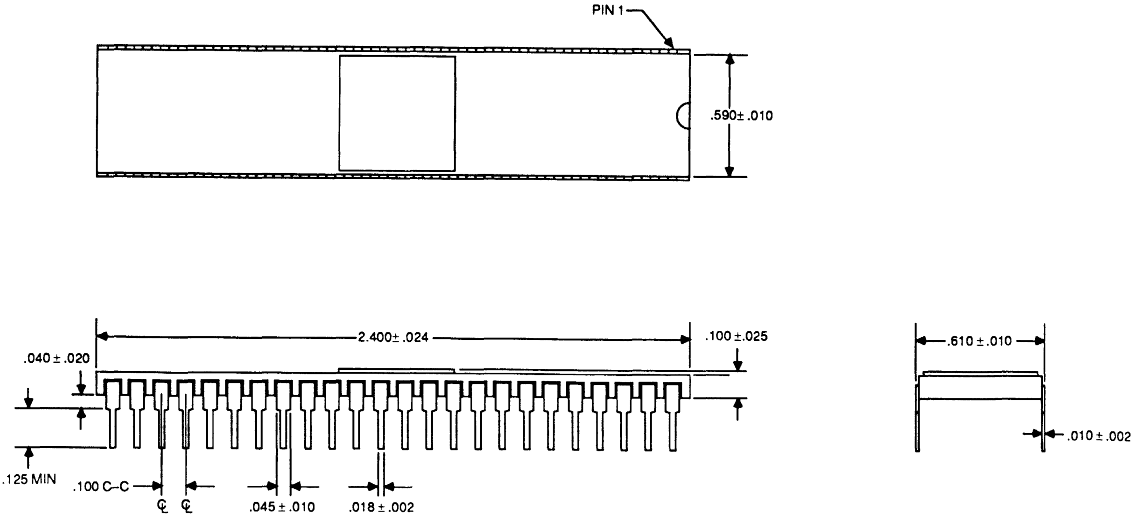
0010003 30



DIMENSIONS IN INCHES
NOT DRAWN TO SCALE

0010003 24

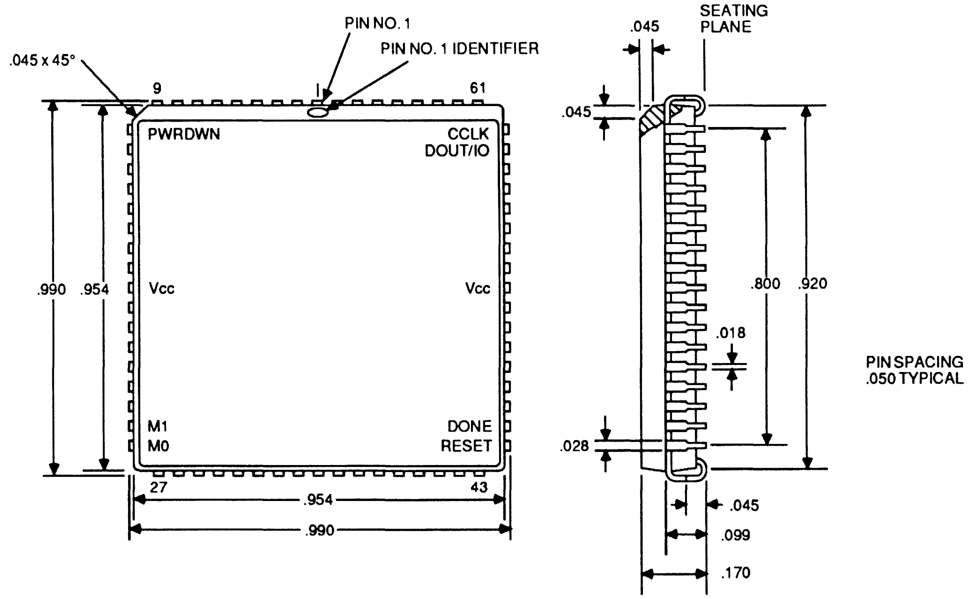
48-Pin Plastic DIP Package



0010003 33

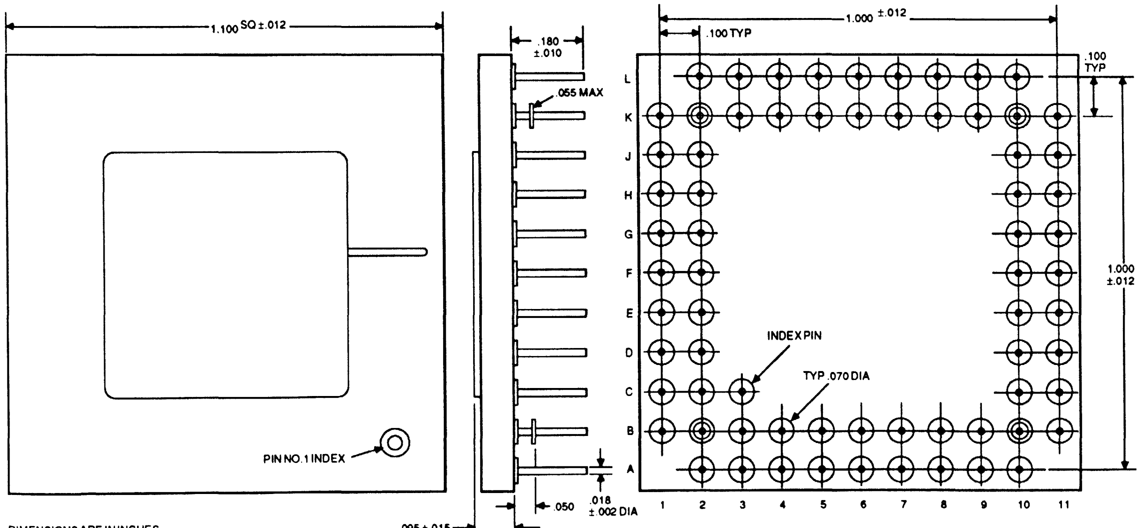
DIMENSIONS IN INCHES
NOT DRAWN TO SCALE

48-Pin Ceramic DIP Package



0010003 31

68-Pin PLCC Package

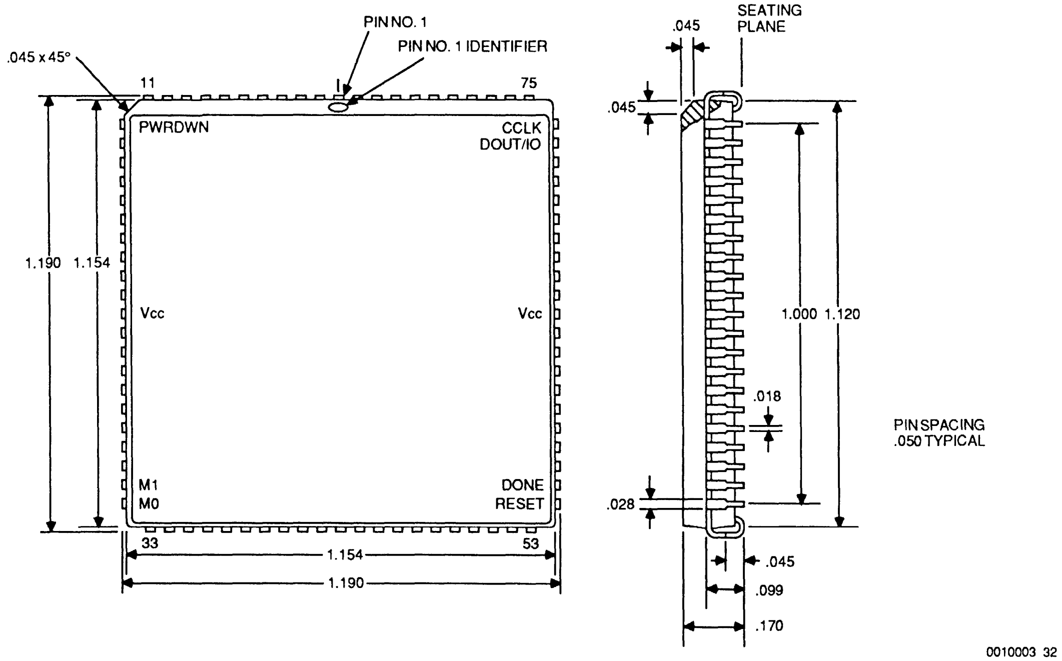


DIMENSIONS ARE IN INCHES

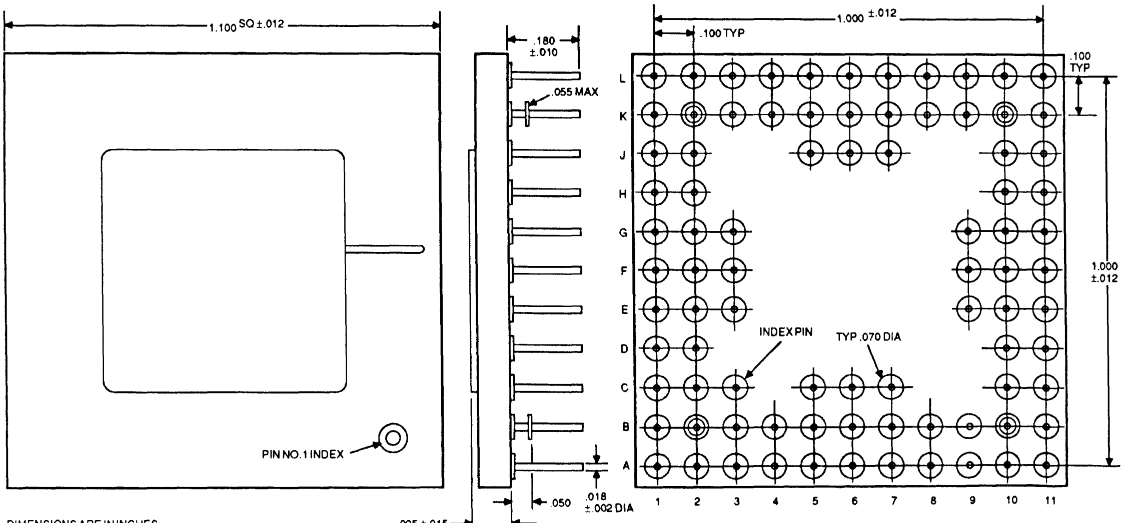
NOTE: INDEX PIN MAY OR MAY NOT BE ELECTRICALLY CONNECTED TO PIN C2.

0010003 25

68-Pin PGA Package



84-Pin PLCC Package



DIMENSIONS ARE IN INCHES

NOTE: INDEX PIN MAY OR MAY NOT BE ELECTRICALLY CONNECTED TO PIN C2.

0010003 34

84-Pin PGA Package

SOCKET INFORMATION

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.

XILINX

XC2064-1 / XC2064-2 Logic Cell™ Array

Product Specification

PARAMETRICS Vcc = 5.0 V ± 10%

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
T_{STG} Storage temperature (ambient)	–65 to + 150	° C
T_{SOL} Maximum soldering temperature (10 sec @ 1/16 in.)	+ 260	° C

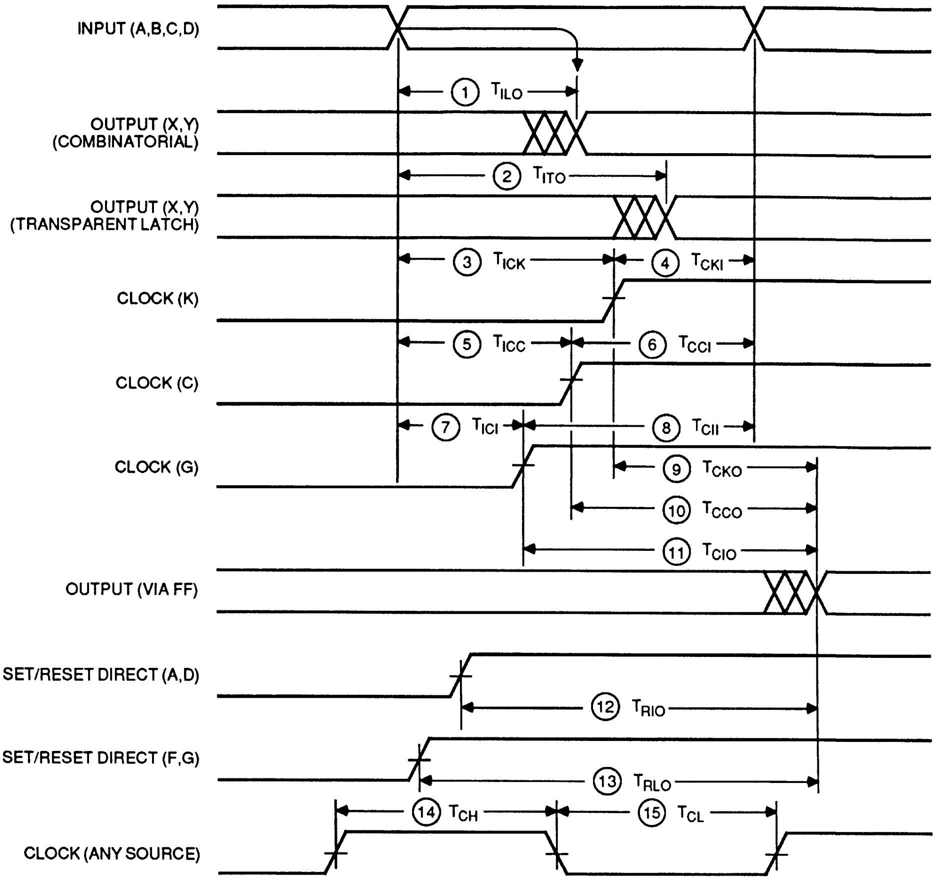
*Note: Stresses beyond those listed under Absolute Maximum Ratings may cause permanent damage to the device. These are stress ratings only and functional operation of the device at these or any other conditions beyond those listed under Recommended Operating Conditions is not implied. Exposure to Absolute 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 0° C to 70° 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

PARAMETRICS (Continued)

Electrical Characteristics Over Operating Conditions			Min	Max	Units
V_{OH}	High-level output voltage (@ I_{OH} = –4.0 ma V_{CC} min)		3.86		V
V_{OL}	Low-level output voltage (@ I_{OL} = 4.0 ma V_{CC} min)			0.32	V
I_{CCO}	Quiescent operating power supply current				
	CMOS thresholds (@ V_{CC} = 5.0 V)			5	mA
	TTL thresholds (@ V_{CC} = 5.0 V)			10	mA
I_{CCPD}	Power-down supply current (@ V_{CC} = 5.0 V)			0.5	mA
I_{IL}	Leakage current		–10	+10	µA
C_{IN}	Input capacitance (sample tested)			10	pF

CLB SWITCHING CHARACTERISTICS



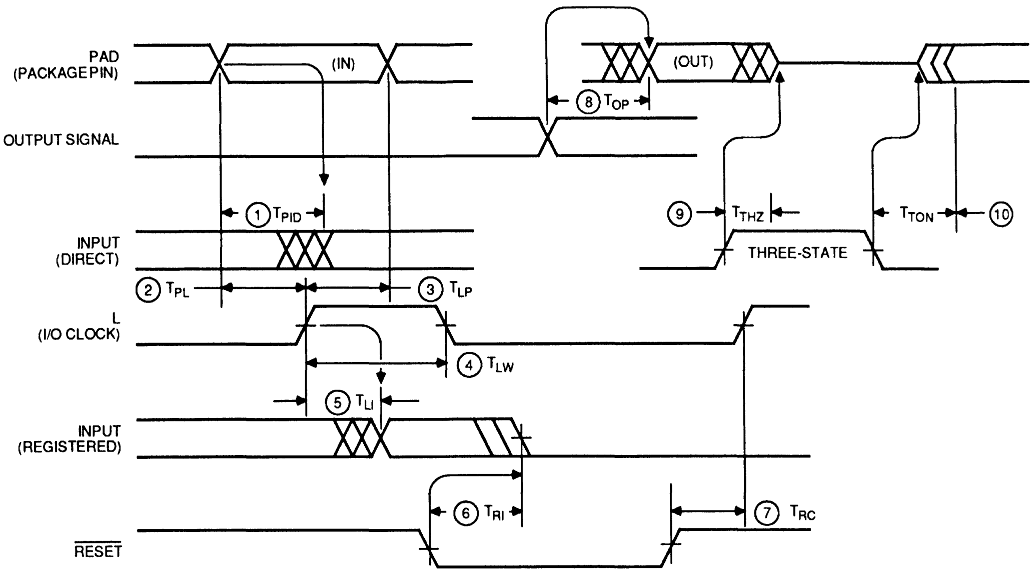
0010003 29

CLB SWITCHING CHARACTERISTICS (Continued)

		Speed Grade		-1		-2		Units
	Description	Symbol		Min	Max	Min	Max	ns
Logic input to Output	Combinatorial	1	T_{ILO}		35		20	
	Transparent latch	2	T_{ITO}		45		25	
	Additional for Q through F or G to out		T_{QLO}		30		13	
K Clock	To output	9	T_{CKO}		35		20	
	Logic-input setup	3	T_{ICK}	22		12		
	Logic-input hold	4	T_{CKI}	0		0		
C Clock	To output	10	T_{CCO}		45		25	
	Logic-input Setup	5	T_{ICC}	18		12		
	Logic-input hold	6	T_{CCI}	10		6		
Logic input to G Clock	To output	11	T_{CIO}		65		37	
	Logic-input Setup	7	T_{ICI}	10		6		
	Logic-input Hold	8	T_{CII}	15		9		
Set/Reset direct	Input A or D to Out	12	T_{RIO}		45		25	
	Through F or G to Out	13	T_{RLO}		65		37	
	Master Reset pin to Out		T_{MRQ}		60		35	
	Separation of Set/Reset		T_{RS}	30		17		
	Set/Reset pulse-width		T_{RPW}	20		12		
Flip-flop Toggle rate	Q through F to flip-flop		F_{CLK}	20		33		MHz
Clock	Clock high	14	T_{CH}	20		12		
	Clock low	15	T_{CL}	20		12		

Notes: 1. All switching characteristics apply to all valid combinations of process, temperature and supply.
2. Units are ns unless otherwise specified.

IOB SWITCHING CHARACTERISTICS

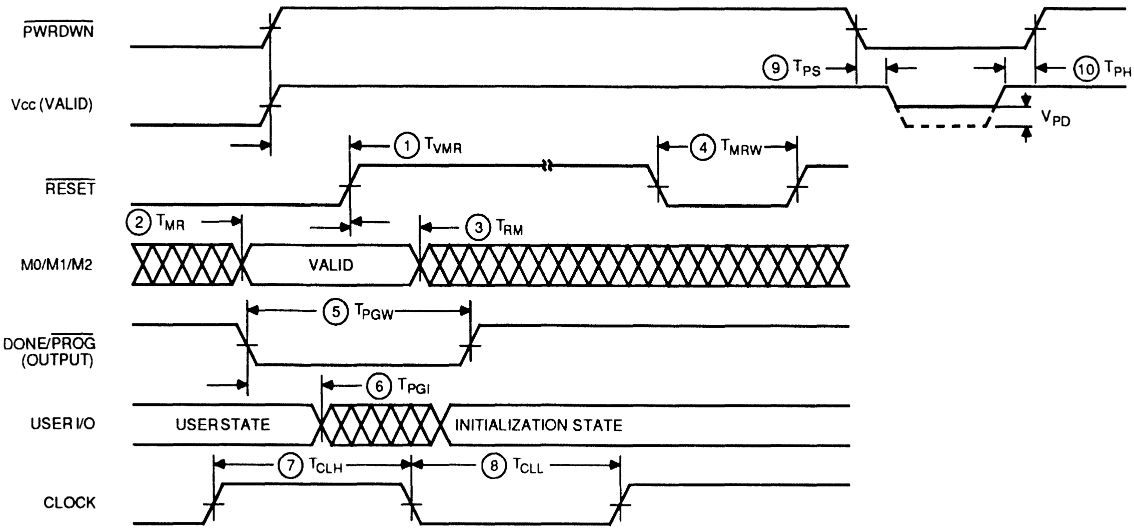


0010003 27

				-1		-2		Units
	Description	Symbol		Min	Max	Min	Max	ns
Pad (package pin)	To input (direct)	1	T_{PID}		20		12	
I/O Clock	To input (storage)	5	T_{LI}		30		20	
	To pad-input setup	2	T_{PL}	20		12		
	To pad-input hold	3	T_{LP}	0		0		
	Pulse width	4	T_{LW}	20		12		
	Frequency				20		33	MHz
Output	To pad (output enabled)	8	T_{OP}		25		15	
Three-state	To pad begin hi-Z	9	T_{THZ}		35		25	
	To pad end hi-Z	10	T_{TON}		40		25	
$\overline{\text{RESET}}$	To input (storage)	6	T_{RI}		50		30	
	To input clock	7	T_{RC}		25		35	

Note: Timing is measured at 0.5 Vcc levels with 50pF output load.

GENERAL LCA SWITCHING CHARACTERISTICS

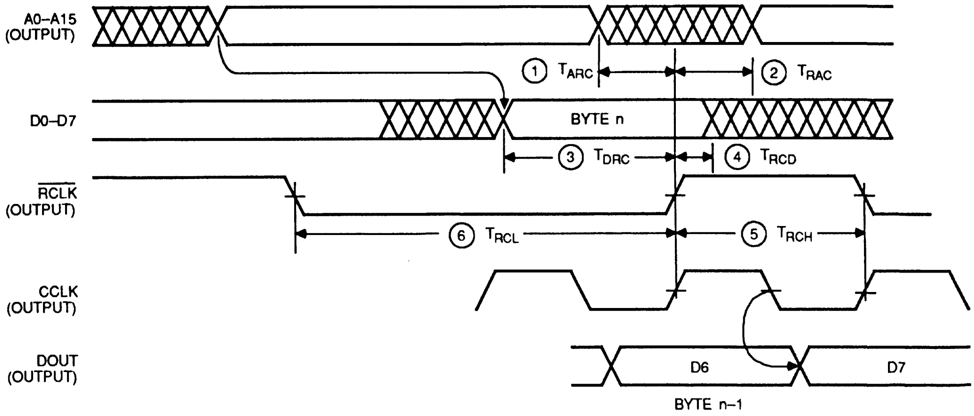


0010003 26c

	Description	Symbol		-1		-2		Units
				Min	Max	Min	Max	ns
$\overline{\text{RESET}}$ (2)	V_{CC} setup (2.0 V) M2, M1, M0 setup M2, M1, M0 hold Width (low)	1 2 3 4	T_{VMR} T_{MR} T_{RM} T_{MRW}	250 100 100 250		150 60 60 150		
DONE/$\overline{\text{PROG}}$	Progam width (low) Initialization	5 6	T_{PGW} T_{PGI}	6	 7	6	 7	µs µs
CLOCK	Clock (high) Clock (low)	7 8	T_{CLH} T_{CLL}	20 20		12 12		
PWR DWN	Setup to V_{CC} Hold from V_{CC} Power Down	9 10	T_{PS} T_{PH} V_{PD}	0 0 2.0		0 0 2.0		 V

Note:
1. V_{CC} must rise from 2.0 Volts to V_{CC} minimum in less than 10 ns for Master Mode.
2. $\overline{\text{RESET}}$ timing relative to power-on and valid mode lines (M0, M1, M2) is relevant only when $\overline{\text{RESET}}$ is used to delay configuration.
3. Minimum CLOCK widths for the Auxillary buffer are 1.25 the T_{CLH}, T_{CLL}.

MASTER MODE SWITCHING CHARACTERISTICS

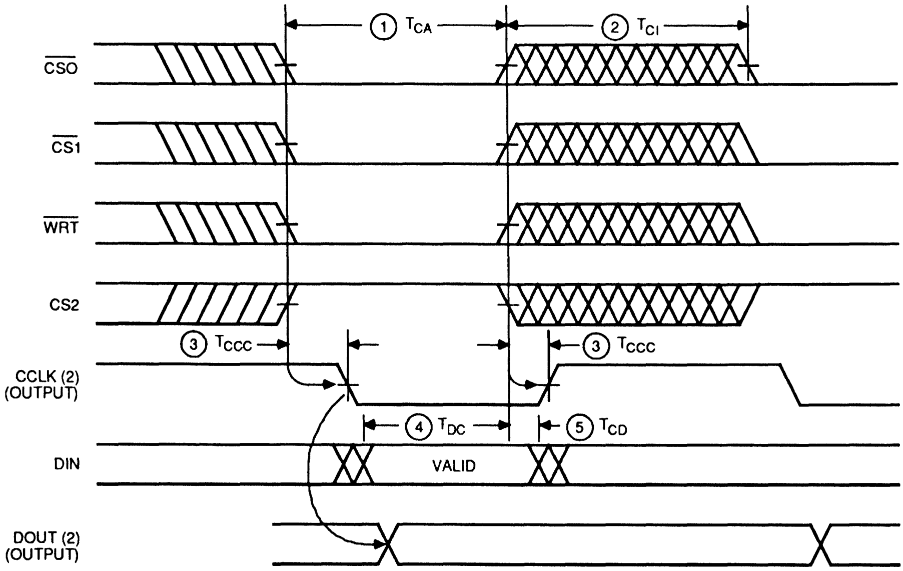


0010003 26A

				-1		-2		Units
	Description	Symbol		Min	Max	Min	Max	ns
$\overline{\text{RCLK}}$	From address invalid	1	T_{ARC}		0		0	
	To address valid	2	T_{RAC}		300		200	
	To data setup	3	T_{DRC}	100		60		
	To data hold	4	T_{RCD}	0		0		
	$\overline{\text{RCLK}}$ High	5	T_{RCH}	600		600		
	$\overline{\text{RCLK}}$ low	6	T_{RCL}	4.0		4.0		µs

Notes: 1. CCLK and DOUT timing are the same as for slave mode.
2. At power-up, Vcc must rise from 2.0 V to Vcc min. in less than 10 ms.

PERIPHERAL MODE PROGRAMMING SWITCHING CHARACTERISTICS

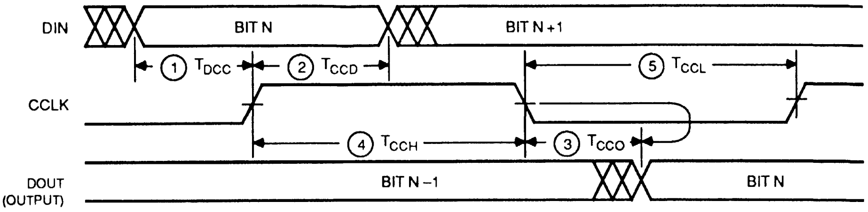


0010003 28B

				-1		-2		Units
	Description	Symbol		Min	Max	Min	Max	ns
Controls[1] ($\overline{CS0}$, $\overline{CS1}$, CS2, $\overline{WRT}$)	Active (last active input to first inactive)	1	T_{CA}	0.3	10.0	0.2	5.0	µs
	Inactive (first inactivate input to last active)	2	T_{CI}	500		300		
	CCLK[2]	3	T_{CCC}		100		75	
	DIN setup	4	T_{DC}	50		35		
	DIN hold	5	T_{CD}	10		5		

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 Vcc min.

SLAVE MODE PROGRAMMING SWITCHING CHARACTERISTICS

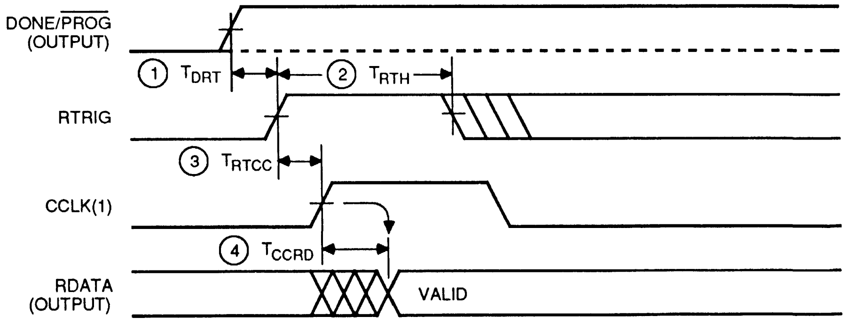


0010003 28A

			-1		-2		Units
	Description	Symbol	Min	Max	Min	Max	ns
CCLK	To DOUT	3 T_{CCO}		100		65	
	DIN Setup	1 T_{DCC}	50		25		
	DIN Hold	2 T_{CCD}	75		40		
	High time	4 T_{CCH}	500		300		
	Low time	5 T_{CCL}	0.3	10.0	0.25	5.0	µs
	Frequency	F_{CC}		1		2	MHz

Note: Configuration must be delayed at least 40 ms after Vcc min.

PROGRAM READBACK SWITCHING CHARACTERISTICS



0010003 26B

				-1		-2		Units
	Description	Symbol		Min	Max	Min	Max	ns
RTRIG	PROG setup	1	T_{DRT}	300		300		
	RTRIG high	2	T_{RTH}	250		250		
CCLK	RTRIG setup	3	T_{RTCC}	100		100		
	RDATA delay	4	T_{CCRD}		100		100	

Notes: 1. CCLK and DOUT timing are the same as for slave mode.
2. DONE/$\overline{\text{PROG}}$ output/input must be HIGH (device programmed) prior to a positive transition of RTRIG (M0).

		48 PIN	68 PIN
		PLASTIC DIP	PLASTIC PLCC
		-PD 48	-PC 68
XC2064	-1	C	C I
	-2	C	C

C = COMMERCIAL 0° C TO 70° C

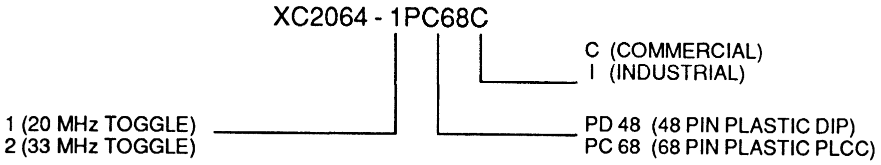
I = INDUSTRIAL − 40° C TO 85° C

0010003 35

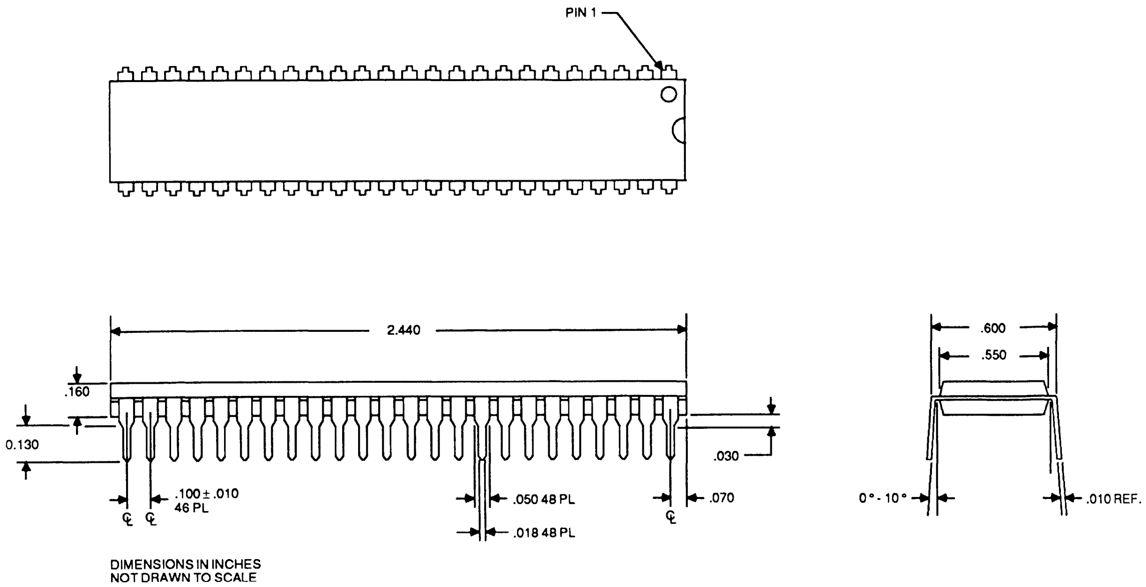
LCA Package and Temperature Options

Ordering Information

Further information is available from Xilinx franchised distributors of from the nearest Xilinx sales representative. Part numbers are composed as follows:

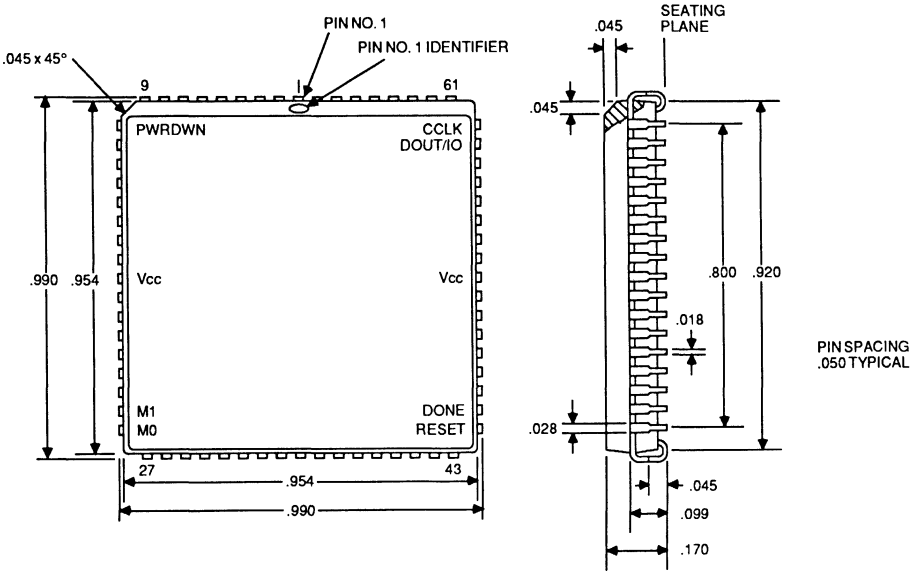


0010003 36



0010003 24

48-Pin Plastic DIP Package



0010003 31

68-Pin Plastic PLCC Package

P/N 0010003 01

Testing and Data Integrity

Xilinx is committed to providing the highest level of quality and reliability for the Logic Cell™ (LCA) Array. Quality is best assured by taking the necessary steps to achieve zero defects. Comprehensive testing confirms that every Logic Cell Array is free from defects and conforms to the data sheet specifications. The memory cell design assures integrity of the configuration program. Careful memory cell design has also minimized the effects of alpha particle emission and electromagnetic radiation on the operation of the Logic Cell Array.

TESTING

As quality consciousness has grown among semiconductor users, awareness of the importance of testability has also increased. Testing for standard components, including memories and microprocessors, is accomplished with carefully developed programs which exhaustively test the function and performance of each part. For reasons explained below, most application specific ICs cannot be comprehensively tested. Without complete testing, defective devices might escape detection and be installed into a system. In the best case, the failure will be detected during system testing at a higher cost. In the worst case, the failure will be detected only after shipment of the system to a customer.

Testing advantages of the Logic Cell Array can be illustrated through comparison with two other application specific ICs: Erasable Programmable Logic Devices (EPLDs) and gate arrays.

EPLDs: In order to test all memory cells and logic paths of programmable logic devices controlled by EPROM memory cells, the part must be programmed with many different patterns. This in turn requires expensive quartz lid packages and many lengthy program/test/erase cycles. To save time and reduce costs, this process is typically abbreviated.

Gate Arrays: Since each part is programmed with metal masks, the part can only be tested with a program tailored to the specific design. This in turn requires that the designer provide sufficient controllability and observability for comprehensive testability. The design schedule must also include time for the development of test vectors and a test program specification. If the gate array user requires a comprehensive test program, then he must perform exhaustive and extensive fault simulation and test grading. This requires substantial amounts of expensive computer time. Additionally, it typically requires a series of time-consuming and expensive iterations in order to reach even 80% fault coverage. The cost of greater coverage is often prohibitive. In production, many gate array vendors either limit the number of vectors allowed or charge for using additional vectors.

The replacement of all storage elements with testable storage elements, known as scan cells, improves testability. Although this technique can reduce the production testing costs, it can add about 30% more circuitry, decrease performance by up to 20%, and increase design time.

Logic Cell Arrays: The testability of the Logic Cell Array is similar to other standard products, including microprocessors and memories. This is the result of the design and the test strategies:

Design strategy:

- Incorporates testablility features because each functional node can be configured and routed to outside pads
- Permits repeated exercise of the part without removing it from the tester because of the short time to load a new configuration program
- Produces a standard product which guarantees that every valid configuration will work.

Test strategy:

- Performs reads and writes of all bits in the configuration memory, as in memory testing
- Uses an efficient parallel testing scheme in which multiple configurable logic blocks are fully tested simultaneously
- Is exhaustive since the circuits in every block are identical

The Logic Cell Array user can better appreciate the Logic Cell Array test procedure by examining each of the testing requirements:

- All of the configuration memory bits must be exercised and then verified. This is performed using readback mode.

- All possible process-related faults, such as short circuits, must be detected. The Logic Cell Array is configured such that every metal line can be driven and observed directly from the input/output pads.
- All testing configurations must provide good controllability and observability. This is possible since all configurable logic blocks can be connected to input/output pads. This makes them easy to control by testing different combinations of inputs and easy to observe by comparing the actual outputs with expected values.

These points bring out an important issue: the Logic Cell Array was carefully designed to achieve 100% fault coverage. With the Xilinx testing strategy, the number of design configurations needed to fully test the Logic Cell Array is minimized and the test fault coverage of the test patterns is maximized. In addition, the user's design time is reduced because the designer does not have to be concerned about testability requirements during the design cycle. The Logic Cell Array concept not only removes the burden of the test program and test vector generation from the user, but also removes the question of fault coverage and eliminates the need for fault grading. The Logic Cell Array is a standard part that guarantees any valid design will work. These issues are critically important in quality-sensitive applications. The designer who uses the Logic Cell Array can build significant added value into his design by providing higher quality levels.

DATA INTEGRITY

Memory Cell Design

An important aspect of the Logic Cell Array's reliability is the robustness of the static memory cells used to store the configuration program.

The basic cell is a single-ended five-transistor memory element (Figure 1). By eliminating a sixth transistor, which would have been used as a pass transistor for the complementary bit line, a higher circuit density is achieved. During normal operation, the outputs of these cells are fixed, since these determine the user configuration. Write and readback times, which have no relation to the device performance during normal operation, will be slower without the extra transistor. In return, the user receives more functionality per unit area.

This explains the basic cell, but how is the Logic Cell Array user assured of high data integrity in a noisy environment? We must consider three different situations: normal operation, a write operation and a read operation. In the normal operation, the data in the basic memory element is not changed. Since the two circularly linked inverters that hold the data are physically adjacent, supply transients result in only small relative differences in voltages. Each inverter is truly a complementary pair of transistors. Therefore, whether the output is high or low, a low impedance path exists to the supply rail, resulting in extremely high noise immunity. Power supply or ground transients of several volts have no effect on stored data.

The transistor driving the bit line has been carefully designed so that whenever the data to be written is opposite the data stored, it can easily override the output of the feedback inverter. The reliability of the write operation is guaranteed within the tolerances of the manufacturing process.

In the read mode, the bit line, which has a significant amount of parasitic capacitance, is precharged to a logic one. The pass transistor is then enabled by driving the word line high. If the stored value is a zero, the line is then discharged to ground. Reliable reading of the memory cell is achieved by reducing the word line high level during reading to a level that insures that the cell will not be disturbed.

Alpha Particle (Soft Error) Sensitivity:

The CMOS static memory cell was designed to be insensitive to alpha particle emissions. To verify that this design goal was achieved, the following tests were performed.

A one microcurie alpha particle source (Americum 241) was placed in direct contact with the top surface of an XC2064 die. This allows the die to capture at least 40% of the emissions from the radiation source. The following sequence of tests was performed:

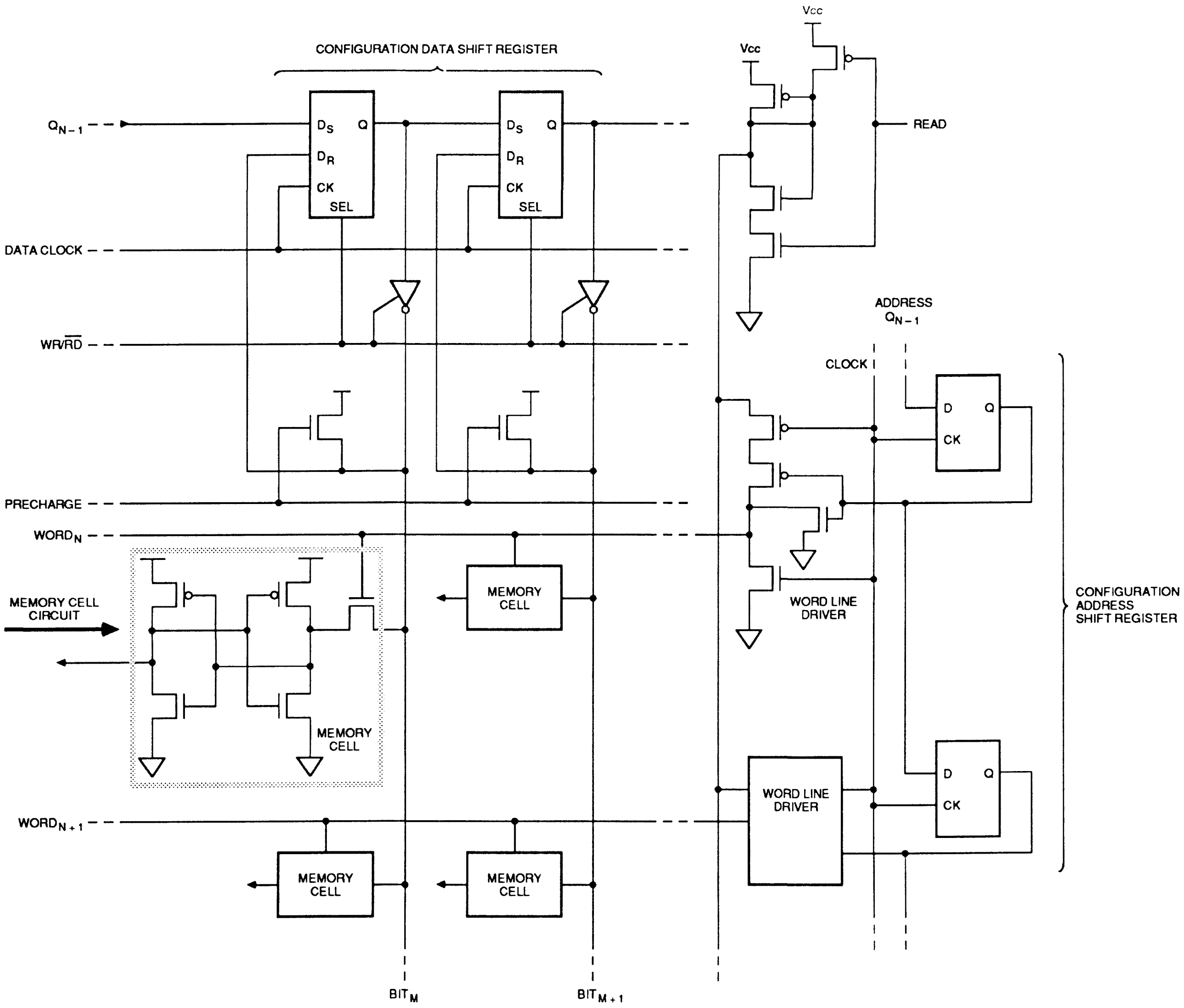
1. A complex pattern containing roughly 50% logic ones was loaded into the XC2064. The operating conditions were 25°C and 5.0 volts.

2. A pause of variable duration was allowed.

3. The entire contents of the XC2064 were read back and compared with the original data.

Validation tests to ensure that the test setup would detect errors were performed before and after the alpha particle tests. The results are as follows:

Test	Time Duration, sec	Readback Time, sec	Total Time Exposed, sec	Number of Errors
1	10	70	80	0
2	120	70	190	0
3	300	70	370	0
4	1500	70	1570	0
Total			2210	



0010019 1

Figure 1. Configuration Memory Cell

Analysis

A one microcurie source emits 3.7×10^4 alpha particles per second. Assuming that 40% of these are captured by the XC2064 during this experiment; this corresponds to 5.3×10^7 alpha particles per hour.

The alpha particle emission rate of the molding compound used by Xilinx is specified to emit fewer than 0.003 alpha particles per square centimeter per hour (alpha particles/cm^2/hr). The surface area of the XC-2064 die is less than 0.5 cm^2, so less than 0.0015 alpha particles per hour will be captured by the XC2064 in normal operation. The error rate acceleration in this test is therefore equal to:

$$\frac{5.3 \times 10^7 \text{ particles/hour}}{0.0015 \text{ particles/hour}} = 3.6 \times 10^{10}$$

The 0.61 hours of test time without error then corresponds to 2.2×10^{10} hours or 2.5 million years of 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 an XC2064 in a ceramic package, this still results in error-free operation for almost a million years.

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 these glasses are used only for the package seal, they present a relatively small emitting cross section to the die (less than 0.1 cm^2 square). A low alpha glass would therefore cause fewer than 0.8 alpha particle hits per hour. The acceleration factor is then 6.6×10^8, which translates to about 46,000 years without an error.

The memory cell of the Xilinx Logic Cell Array has been designed so that soft errors caused by alpha particles can safely be ignored.

Electrostatic Discharge

Electrostatic discharge (ESD) protection for each pad is provided by a circuit that uses forward and reverse biased distributed resistor-diodes (Figure 2). In addition, inherent capacitance integrates any current spikes. This gives sufficient time for the diode and breakdown protections to provide a low impedance path to the power-supply rail. Geometries and doping levels are optimized to provide sufficient ESD protection for both positive and negative discharge pulses.

Latchup

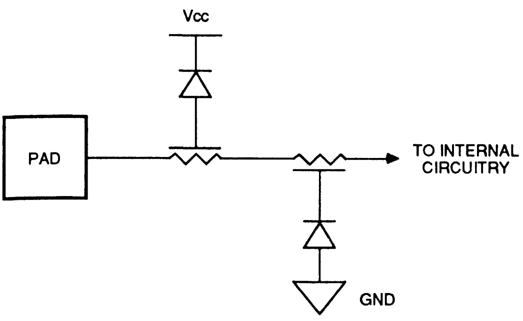
Latchup is a condition in which parasitic bipolar transistors form a positive feedback loop (Figure 3), which quickly reaches current levels that permanently damage the device. Xilinx uses techniques based on doping levels and circuit placement to avoid this phenomenon. The cross section of a typical transistor (Figure 4) shows several features. The beta of each parasitic transistor is minimized by increasing the base width. This is achieved with large physical spacings. The butting contacts effectively short the n^+ and p^+ regions for both wells, which makes the V_{BE} of each parasitic very close to zero. This also makes the parasitic transistors very hard to forward bias. Finally, each well is surrounded by a dummy collector, which forces the V_{CE} of each parasitic almost to zero and creates a structure in which the base width of each parasitic is large, thus making latchup extremely difficult to induce.

Radiation Hardness

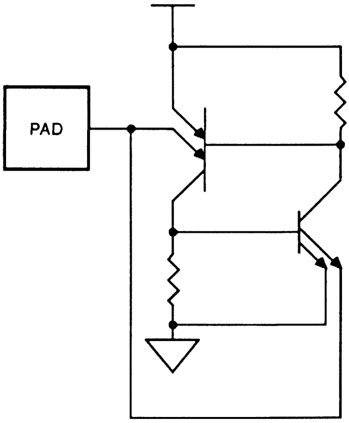
A preliminary estimate of the hardness of the circuit to withstand ionizing radiation ranges from 10,000 to 100,000 rads Si. This estimate was reached from a discussion with Sandia National Labs and is based upon the design and layout parameters of the Logic Cell Array.

High Temperature Performance

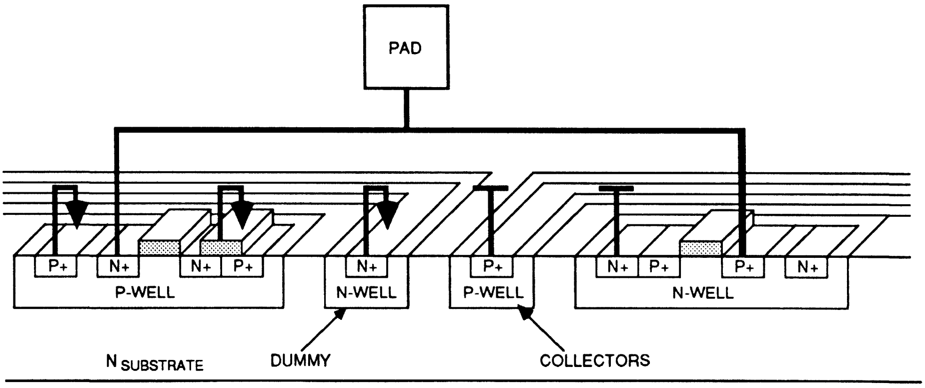
Although Xilinx guarantees parts to perform only within the specifications of the data sheet, extensive high temperature life testing has been been done at 145°C with excellent results.



0010019 2 **Figure 2. Input Protection Circuity**



0010019 3 **Figure 3. SCR Model**



0010019 4

Figure 4. CMOS Input Circuit Layout

Nonhermetic Package Reliability

INTRODUCTION

From its inception, Xilinx has been committed to delivering the highest quality, most reliable programmable gate arrays available. A strong Quality Assurance and Reliability program begins at the initial design stages and is carried through to final shipment. The final proof of our success is in the performance of the Logic Cell™ Array (LCA) in our customers' systems applications. An extensive, on-going reliability-testing program is used to predict the field performance of our devices.

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, predictions of actual field performance under a variety of conditions can be calculated.

This report describes the nature and purpose of the various reliability tests performed on finished devices. Table 2 is the initial summary of the Quality Control reliability testing performed. Updated summaries are available upon request from the Quality Assurance and Reliability Department at Xilinx.

OUTLINE OF TESTING

Qualification testing of nonhermetic devices is performed to demonstrate the reliability of the die used in the device, and the materials and methods used in the assembly of the device. Testing methods are derived from and patterned after the methods specified in MIL-STD-883.

A summary of the reliability demonstration tests used at Xilinx is contained in Table 1.

Referral to the test methods of MIL-STD-883 is not intended to imply that nonhermetic products comply with the requirements of MIL-STD-883. These test methods are recognized industry-wide as stringent tests of reliability and are commonly used for nonmilitary-grade semiconductor devices, as well as for fully compliant military-grade products.

DESCRIPTION OF TESTS

Die Qualification

1. **High Temperature Life** This test is performed 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 is contained in the Group C Quality Conformance Tests. Because of the acceleration factor induced by higher temperatures, data representing a large number of equivalent hours at a normal temperature of 70°C, can be accumulated in a reasonable period of time. Xilinx performs its High Temperature Life test at a higher temperature, i.e., 145°C, than the more common industry practice of 125°C. For comparison, the Reliability Testing Data Summary in Table 2, gives the equivalent testing hours at 125°C.

2. **Biased Moisture Life** This test is performed to evaluate the reliability of the die under conditions of long-term exposure to severe, high-moisture environments which could cause corrosion. Although it clearly stresses the package as well, this test is typically grouped under the die-related tests. The device is operated at maximum-rated voltage, 5.5 VDC, and is exposed to a temperature of 85°C and a relative humidity of 85% throughout the test.

Package Integrity and Assembly Qualification

3. **Unbiased Pressure Pot** This test is performed at a temperature of 121°C and a pressure of 2 atm. 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 nonglassivated metal areas of the die [bonding pads only for Xilinx LCAs], and could possibly cause, under extreme conditions, moisture drive-in and corrosion under the glassivation. Although it is difficult to correlate this test to actual field conditions, it provides a well-established method for relative comparison of plastic

packaging materials and assembly and molding techniques.

4. **Thermal Shock** This test is performed to evaluate the resistance of the package to cracking and resistance of the bonding wires and lead frame to separation or damage. It involves nearly instantaneous change in temperature from –65°C. to +150°C.

5. **Temperature Cycling** This test is performed to evaluate the long-term resistance of the package to damage from alternate exposure to extremes of temperature or to intermittent operation at very low temperatures. The range of temperatures 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.

6. **Salt Atmosphere** This test was originally designed by the US Navy to evauate resistance of military-grade ship board electronics to corrosion from seawater. It is used more generally for nonhermetic industrial and commercial products as a test of corrosion resistance of the package marking and finish.

7. **Resistance to Solvents** This test is performed to evaluate the integrity of the package marking during exposure to a variety of solvents. This is an especially important test, as an increasing number of board-level assemblies are subjected to severe conditions of automated cleaning before system assembly operations occur. This test is performed acording to the methods specified by MIL-STD-883.

8. **Solderability** This test is performed to evaluate the solderability of the leads under conditions of low soldering temperature following exposure to the aging effects of water vapor.

9. **Vibration, Variable-Frequency** This test is performed to evaluate the resistance of the completed assembly to vibrations during storage, shipping, and operation.

TESTING FACILITIES

Xilinx has a growing investment in reliability testing equipment. The company has the complete capability to perform High Temperature Life Tests, Biased Moisture Life Tests, and Unbiased Pressure Pot Tests in its own Reliability Testing Laboratory. Additional equipment is being purchased as required by testing volume. Other tests are being performed by outside testing laboratories with DESC laboratory suitability for each of the test methods they perform.

SUMMARY

The attached testing data shows the actual performance of the Logic Cell Arrays during the initial qualification tests to which they have been subjected. These test results demonstrate the reliability and expected long life inherent in our nonhermetic product line. This series of tests is ongoing as a part of our Quality Conformance Program on nonhermetic devices.

DIE QUALIFICATION

Name of Test	Test Conditions	Lot Tolerance Percent Defective Minimun Sample Size/ Maximum Acceptable Failures
1. High Temperature Life	1000 hr min. equivalent at temperature = 125°C Actual test temperature = 145°C Max. rated operating voltage. Life test circuit equivalent to MIL-STD-883	LTPD = 5, s = 105, c = 2
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 = 105, c = 2

PACKAGE INTEGRITY and ASSEMBLY QUALIFICATION

Name of Test	Test Conditions	Lot Tolerance Percent Defective Minimun Sample Size/ Maximum Acceptable Failures
3. Unbiased Pressure Pot	96 hr min. exposure T = 121°C, P = 2 atm H_2O sat.	LTPD = 5, s = 105, c = 2
4. Thermal Shock	MIL-STD-883, Method 1011, Cond. C –65°C to +150°C 100 cycles	LTPD = 7, s = 75, c = 2
5. Temperature Cycling	MIL-STD-883, Method 1010, Cond. C –65°C to +150°C 200 cycles	LTPD = 5, s = 105, c = 2
6. Salt Atmosphere	MIL-STD-883, Method 1009, Cond. A 24 hrs	s = 25, c = 0
7. Resistance to Solvents	MIL-STD-883, Method 2015	s = 8, c = 0
8. Solderability	MIL-STD-883, Method 2003	s = 15, c = 0
9. Vibration, Variable-Frequency	MIL-STD-883, Method 2007	s = 11, c = 0

Table 1. Reliability Testing Sequence for Nonhermetic Logic Cell Arrays

XILINX XC-2064 Reliability Testing Summary, Initial Lots

Device Type: XC-2064
Die Attach Method: Silver Epoxy
Molding Compound: Nitto MP 150 SG

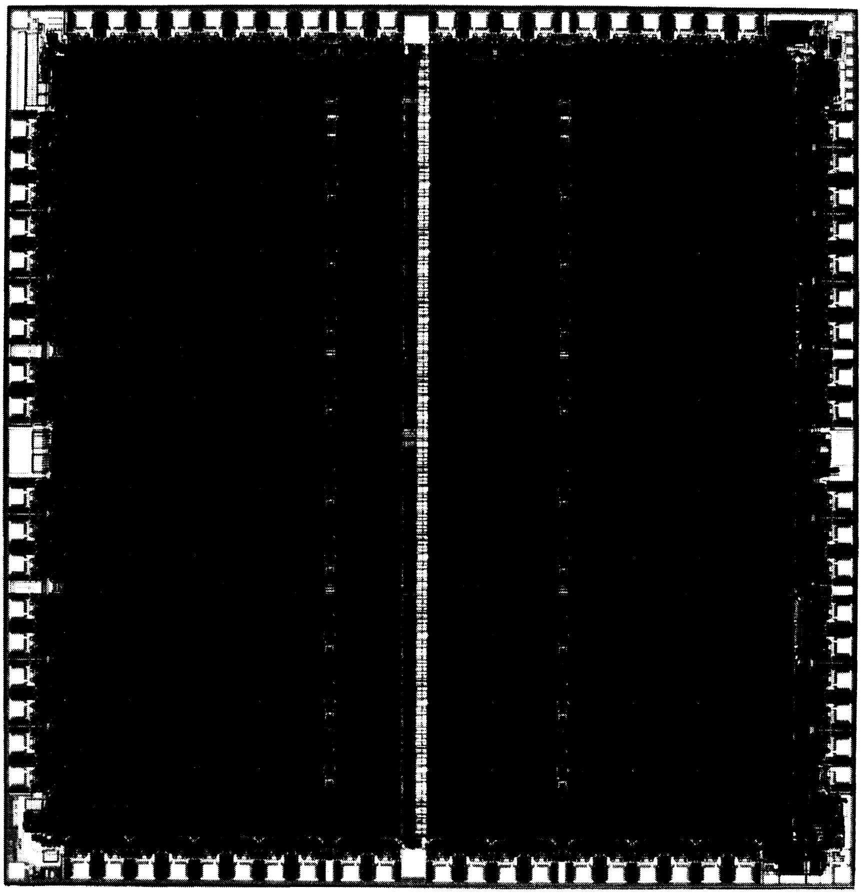
Process/Technology: 2.0 Micron Double Layer Metal CMOS
Package Type: 68 lead PLCC
Date: July 21, 1986

1. High Temperature Life Test T = 145°C	Combined Sample	Failures	Equivalent Mean Hrs/Device at T = 125°C	Equivalent Device/Hrs at T = 125°C	Equivalent Failure Rate in %/1000 hrs at T = 125°C	Equivalent Failure Rate in %/1000 hrs at T = 70°C
	210	1	5545	1164349	0.0859	0.0012
2. Biased Moisture Life Test T = 85°C; RH = 85%	Combined Sample	Failures	Mean Hrs per Device at T = 85°C	Total Device Hrs at T = 85°C		
	210	0	1250	262500		
3. Unbiased Pressure Pot Test +121°C, 2 atm sat. steam	Combined Sample	Failures	Mean Hrs per Device	Total Device Hrs		
	263	1	96	25248		
4. Thermal Shock Test –65°C/+150°C	Combined Sample	Failures	Mean Cycles per Device	Total Device Cycles		
	154	0	1100	169400		
5. Temperature Cycling Test –65°C/+150°C	Combined Sample	Failures	Mean Cycles per Device	Total Device Cycles		
	210	0	1000	210000		
6. Salt Atmosphere Test MIL-STD-883, Method 1009, Cond. A	Combined Sample	Failures	Mean Hrs per Device	Total Device Hrs		
	50	0	24	1200		
7. Resistance to Solvents Test MIL-STD-883, Method 2105	Combined Sample	Failures				
	16	0				
8. Solderability Test MIL-STD-883, Method 2003	Combined Sample	Failures				
	30	0				
9. Vibration, Variable Freq. Test MIL-STD-883, Method 2007	Combined Sample	Failures				
	22	0				

Table 2. Reliability Testing Summary

XILINX

The Programmable Gate Array Company



XILINX

Table of Contents

The Programmable Gate Array

Using Programmable Gate Arrays

Programmable Gate Array Design Examples

Programmable Gate Array Development Systems

Appendices

XILINX

Methods of LCA™ Configuration

Table of Contents

Methods of Configuring the Logic Cell™ Array

INTRODUCTION: A PROGRAMMABLE GATE ARRAY

This application note addresses some of the device and system-related considerations involved in loading Logic Cell Arrays (LCAs) with configuration programs. The configuration techniques covered here apply to the XC2064 in both the 48-pin and the 68-pin packages, and to the XC2018 in the 48 pin, 68 pin and 84 pin packages. Topics covered include:

- Descriptions of each of the device's configuration modes
- Device pin definitions before, during, and after configuration
- Selection of a configuration mode for a given application
- Configuration of multiple LCA devices.

An Overview

In a typical LCA design the systems designer first identifies those areas of the logic schematics which are suitable for implementation in an LCA (or LCAs). Those logic sections are then partitioned into clusters of basic logic elements representing Configurable Logic Blocks (CLBs) and I/O Blocks (IOBs). Using a Personal Computer with the Xilinx XACT™ development system software, the designer creates a design file for each LCA. The design file is then compiled into a configuration program which determines the function that LCA is to perform. Using the XACT development system, the configuration program can also be translated into formatted files compatible with standard EPROM programming equipment. An EPROM may then be programmed to store the LCA configuration program.

LCA Configuration Sequence

The behavior of the LCA is best described in terms of three distinct states: the initialization state, the configuration state, and the user-operation state. After an initial power-up delay, the LCA awakens in the initialization state in which its internal configuration memory is cleared, and all internal user-definable logic is held in a quiescent or idle state. Once this initialization is complete, the LCA checks the input logic level present at the $\overline{\text{RESET}}$ pin. When it detects a valid logic "1" level, the device enters the configuration state during which the configuration program is loaded. The precise method used for loading the program into the LCA (i.e., the configuration mode) depends on the logic levels of the Mode Select pins (M0, M1, and M2). The configuration program is formatted as a serial bitstream, and is loaded into the LCA as though it were a shift register. Although several methods may be used to enter the data (e.g., the configuration modes), the content and format of this bitstream are fixed for a given logic application. The configuration program contains a bit field which indicates its length. When the correct number of bits have been entered, as indicated by this length count, the D/$\overline{\text{P}}$ open drain output pin goes HIGH indicating that configuration is complete. Once the configuration process has begun, it must either be completed or aborted and restarted. Partial configurations are not possible. Further details on the configuration program format are presented in a later section.

Once configuration is complete, the LCA enters the user-operation state and performs the user specified logic functions. During user-operation, the device can be instructed to return to the initialization state and repeat the configuration process. A state diagram illustrating this sequence is shown in Figure 1. This ability to be re-programmed can be disabled by setting the appropriate bit in the bitstream. In this event, the LCA's configuration can be changed only through removing and re-applying power to the device.

During the initialization and configuration states, all user-I/O pins (except those used for configuration purposes) have passive internal pull-up resistors which will cause those pins to go to a HIGH state if not externally overdriven. Upon entering the user-operation state, all user I/O-pins become functional simultaneously according to their user-specified definition.

Configuration Modes

The LCA supports five methods for loading program information. Selection of a configuration mode is accomplished by connecting the two Mode Select Pins, M0 and M1, to either a logic "1" or a logic "0" signal as indicated in Table 1. A third mode select pin, M2, provides for future expansion of configuration options. Unlike pins M0 and M1, the M2 pin becomes available as general purpose user-I/O after configuration is completed. During configuration, pins M0 and M2 have internal pull-up resistors, pin M1 does not. Except for Master Serial mode, pin M2 should not be driven LOW during configuration. If left unconnected it will be pulled HIGH.

Mode Select Pins	M0	M1	M2
Master serial mode	0	0	0
Master low mode	0	0	1
Master high mode	0	1	1
Peripheral mode	1	0	1
Slave mode	1	1	1

Table 1. Configuration Mode Selection

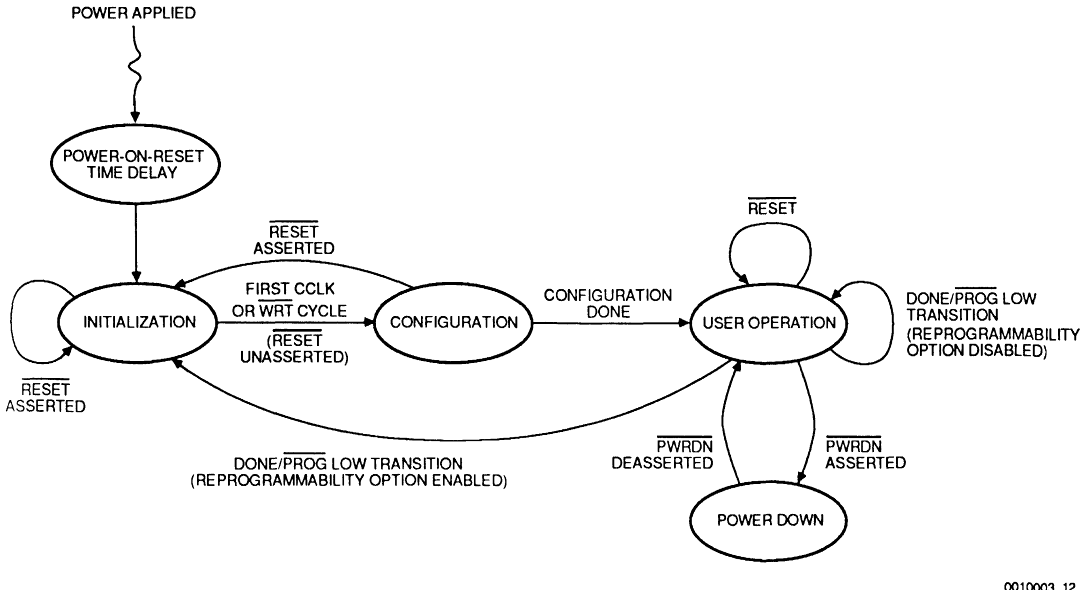
In many applications where the LCA's readback capability will not be used, the mode select pins may be tied directly to ground or to Vcc. Since the M0 and M2 pins are supplied with internal pull-up resistors at the conclusion of configuration, they may also be left unconnected. Since the mode pins are sampled at the conclusion of the initialization state or with the rising edge of RESET if used to delay configuration, their levels need not be maintained once configuration has begun.

Choosing a Configuration Mode

Each configuration mode involves a different set of application design considerations and variations in device pin usage during the configuration process. The choice of a configuration mode depends on the specific system application. Some considerations in choosing a configuration mode include:

- Whether control of the configuration process will be automatic or externally controlled.
  - If externally-controlled (slave or peripheral mode), then
    Via software control
    Via DMA hardware
  - If automatic (master mode), then
    Configuration program shared with microprocessor program code
    Configuration program stored in separate byte wide PROM
    Configuration program stored in a serial memory device.
- The length of time available for configuration,
- If a multiple LCA application, whether to configure them
  - Serially as a "daisy chain,"
  - In parallel
- I/O pin requirements—i.e., will I/O pins used by the target application also be involved in configuration? If so, can pins be assigned to minimize or eliminate external isolation?

The first consideration above should be viewed from a system-design standpoint, since it affects the rest of the



0010003 12

Figure 1. LCA Configuration Diagram

design. In the processor-controlled case, either slave mode or peripheral mode is used to load the configuration program into the LCA one bit at a time. This can be accomplished as part of the system's "bootup" process, or can be done "on-the-fly." This method is the most flexible since the LCA program may be read from PROM or disk or any other source accessible by a processor, but may take longer to complete than the automatic method. The alternative method is to let the LCA program itself automatically using one of the "master" modes of configuration. Here the LCA automatically accesses an external PROM for the configuration program and configures itself in 12 to 24 milliseconds for the XC2064, 17 to 35 milliseconds for the XC2018. Table 2 compares characteristics of the configuration modes. Although master mode uses more I/O pins for configuration than the other modes, those pins become general purpose user-I/O pins again once configuration is complete. These pins can usually be assigned application uses such that no external isolation is required. Figure 2 shows the LCA's pin usage during configuration for each of the configuration modes. The pins used in each mode are also summarized in Table 3.

For some applications the time required to configure the LCA may be a consideration. Although the minimum time required to load the LCA program (approximately 12 ms for the XC2064 and 17 ms for the XC2018) is the same for all configuration modes, processor-driven configuration techniques controlled by software may take longer to complete. The program loading time for master mode, unlike that of the user-driven slave and peripheral modes, is controlled by an internal oscillator. Since the frequency of this internal oscillator is process-dependent, program loading time may extend to twice the minimum.

In applications employing multiple LCAs, special daisy chaining capabilities permit all the LCA programs to be loaded from a single data source. This is described in further detail in the section on "Cascading Multiple LCAs".

In all configuration modes, some of the user's I/O pins are temporarily assigned configuration-related functions. The number of such pins ranges from 5 in the Slave mode to 29 in the Master mode. Once configuration is complete these pins are returned to the user as general-purpose I/O pins. It is up to the designer, however, to guarantee that no signal conflicts occur between the pin's use while in the configuration state and its use while 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, the designer should have no problems in using these dual-function I/O pins. Although signal conflicts are resolvable with external buffers for isolation, careful selection of the pinout assignment can usually eliminate the need for isolation.

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
Number of user I/O pins required	2	6	25	3
Configuration time	Source dependent (Note 1)	Source Dependent (Note 1)	12–24 ms (XC2064) 17–34 ms (XC2018) (Note 3)	12-24 mS (XC2064) 17-34 mS (XC2018) (Note 3)

Notes: 1. The minimum time in any case is approximately 12 ms for the XC 2064 and 17 ms for the XC 2018.
2. Also used by Xilinx's 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 2. Comparison of Configuration Modes

Pin Functions During Configuration

The LCA pins used for configuration are of two types: non-programmable pins dedicated to control functions, and user-programmable pins which are available as general purpose I/O pins once configuration is completed. The six non-programmable pins dedicated to control functions are:

M0,M1	Mode select pins
CCLK	Configuration clock
$\overline{RESET}$	Master reset
D/$\overline{P}$	Done/Program
PWRDWN	Power-down

The user-programmable I/O pins that may be used during configuration are:

M2	Mode select	
DIN	Configuration data In	(Present in all
DOUT	Configuration data out	Configuration
HDC	HIGH during configuration	Modes)
$\overline{LDC}$	LOW during configuration	
$\overline{CS0}$,$\overline{CS1}$,CS2	Chip Selects	
$\overline{WRT}$	Write strobe	(Peripheral mode only)
$\overline{RCLK}$	Read strobe	(Master modes only)
A0–A15	Address bus	(Master parallel only)
D0–D7	Input data bus	

In addition to the dedicated control pins, several user-programmable I/O pins have configuration functions assigned to them regardless of which configuration mode is selected. These pins, as well as the dedicated

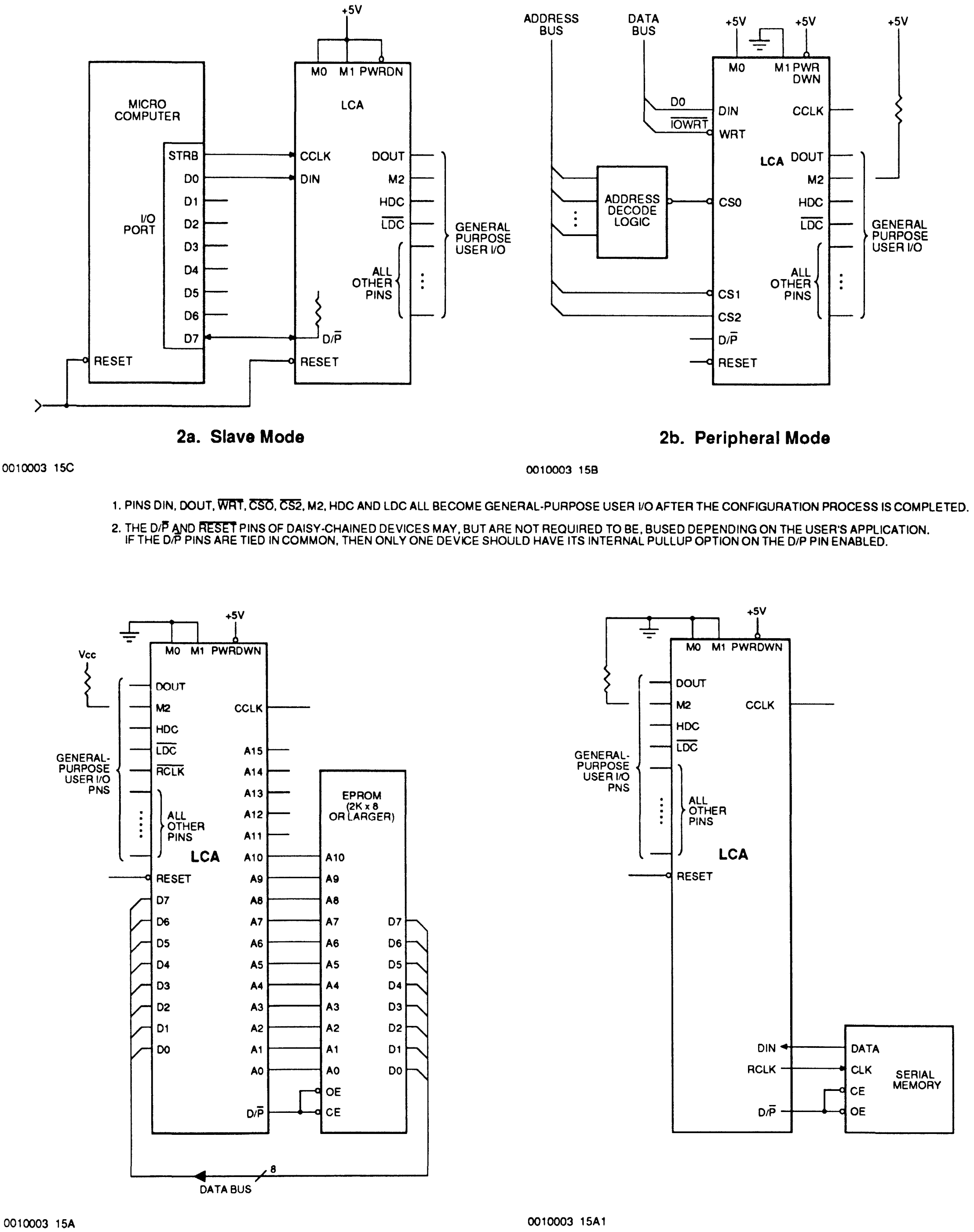
Pin Name	Applicable Config. Mode(s) S	P	MH	ML	MS	Function During Configuration		Function During User Operation	
M0	•	•	•	•	•	Mode select 0	(I)	Readback trigger	(I)
M1	•	•	•	•	•	Mode select 1	(I)	Readback data out	(O)
M2	•	•	•	•	•	Mode select 2	(I)	<User I/O>	
D/$\overline{P}$ (Note 1)	•	•	•	•	•	Indicates when config. process is done	(O)	Initiates/Inhibits Re-configuration	(I)
$\overline{RESET}$ (Note 1)	•	•	•	•	•	Abort/Restart config.	(I)	Master clear of all internal FF's	(I)
CCLK	•	•	•	•	-	Configuration clock (See Notes 1 & 2)		Readback clock	(I)
DIN	•	•	-	-	•	Config data in	(I)	<User I/O> (Note 3)	
DOUT	•	•	•	•	•	Config data out	(O)	<User I/O>	
HDC	•	•	•	•	•	Logic HIGH	(O)	<User I/O>	
$\overline{LDC}$	•	•	•	•	•	Logic LOW	(O)	<User I/O>	
A0–A15	-	-	•	•	-	Address bus	(O)	<User I/O>	
D0–D7	-	-	•	•	-	Data bus	(I)	<User I/O> (Note 3)	
$\overline{RCLK}$	-	-	•	•	•	Read clock	(O)	<User I/O>	
$\overline{WRT}$	-	•	-	-	-	Write strobe	(I)	<User I/O>	
$\overline{CS0}$	-	•	-	-	-	Chip select 0	(I)	<User I/O>	
$\overline{CS1}$	-	•	-	-	-	Chip select 1	(I)	<User I/O>	
CS2	-	•	-	-	-	Chip select 2	(I)	<User I/O>	

Abbreviations:

S = Slave	I = Input
P = Peripheral	O = Output
MH = Master high	
ML = Master low	
MS = Master serial	

Notes:
1. The $\overline{RESET}$, CCLK, and D/$\overline{P}$ 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.

Table 3. Summary of Pins Used For Configuration



2a. Slave Mode

0010003 15C

2b. Peripheral Mode

0010003 15B

1. PINS DIN, DOUT, $\overline{WRT}$, $\overline{CS0}$, $\overline{CS2}$, M2, HDC AND LDC ALL BECOME GENERAL-PURPOSE USER I/O AFTER THE CONFIGURATION PROCESS IS COMPLETED.
2. THE D/$\overline{P}$ AND $\overline{RESET}$ PINS OF DAISY-CHAINED DEVICES MAY, BUT ARE NOT REQUIRED TO BE, BUSED DEPENDING ON THE USER'S APPLICATION. IF THE D/P PINS ARE TIED IN COMMON, THEN ONLY ONE DEVICE SHOULD HAVE ITS INTERNAL PULLUP OPTION ON THE D/P PIN ENABLED.

0010003 15A

2c. Master Parallel Mode

0010003 15A1

2d. Maser Serial Mode

Figure 2. Typical Configuration Circuits

control pins, are described below. Other I/O pins are used in only in one specific configuration mode and are described in the corresponding section.

M0, M1, and M2 are Mode Select input pins used to select which configuration mode the LCA is to use. These pins were described n the previous section on configuration modes. DIN and DOUT are used for the serial data path of a Slave mode daisy chain as well as Master Serial mode.

HDC and LDC are user-I/O pins which are driven by the LCA to constant HIGH and LOW levels respectively during configuration. These two pins are useful in controlling external logic during the initialization and configuration states. They may be used to enable or disable external logic circuits depending on whether that logic is required during or after configuration. 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 complete.

CCLK is a dedicated control pin which serves as a clock input during slave mode configuration, but a clock output in all other configuration modes. As an input CCLK is used during the serial loading of a configuration program. As an output, CCLK serves as a clock source for configuring any slave mode LCAs that may be daisy chained to it. During user 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 and should remain in the HIGH state when not in use. However, it may be driven from a clock source which violates this limit, as long as de-assertion of RESET is used to enable configuration once the clock is normal. The CCLK pin has an internal pull-up resistor which allows an external clock source to be three-stated once configuration is completed.

RCLK performs the function of a read strobe for dynamic memories for master parallel modes. For the master serial mode it is an output used to synchronize the supply of serial data.

The RESET pin is an active LOW master reset input that has different functions depending on the LCA's state. During the initialization state (i.e., after power-up and prior to beginning the configuration process), this pin serves to delay the start of configuration. Once the configuration process has commenced and until it is complete, assertion of RESET will abort the configuration process and return the LCA to the initialization state. Configuration is restarted once initialization is complete and RESET is HIGH. When configuration is completed, the RESET pin changes function and instead becomes a "master reset" control pin that clears all internal flip-flops and latches to the zero state.

The D/P (DONE/PROGRAM) pin is both an input and an open-drain type output with a programmable pull-up resistor option. As an output, it is used to indicate the current configuration status of the LCA. Prior to initial configuration, and during subsequent re-configurations, the LCA holds the D/P pin LOW to indicate that the LCA is not ready for user operation. When D/P goes HIGH, it indicates that configuration has been completed (i.e., "done") and the LCA has entered the user/operation state. Consequently D/P can be used in the system reset logic to ensure that the LCA is configured before reset of the rest of the system is terminated.

Pins configured as LCA outputs become active one clock cycle before D/P goes HIGH. This allows time for any user-I/O signals between LCAs to propagate through and become established prior to use by the target application. Subsequent re-configurations of the LCA can be initiated by applying a logic "0" to the D/P pin with an open-collector type signal source. Once recognized as LOW, the LCA then forces D/P LOW until configuration is complete. Note that by using its internal pull-up resistor option, the D/P pin may be left unconnected, thereby eliminating that pin's need or any external passive components. Since the D/P pin must be held LOW for several microseconds in order to be recognized, it is unlikely to be triggered by noise. The D/P pin must be allowed to go HIGH before it can be used to again initiate reconfiguration. Preventing the D/P pin from going HIGH after configuration may be used as an alternative technique for disabling the LCA from being re-programmed.

The PWRDWN pin is an active LOW input which forces the LCA into a low power state. Vcc may be reduced to 2.0 Volts after PWRDWN is active. Entering the power-down state does not change or modify the configuration information stored in the LCA; it merely causes the device to reduce its overall power requirements by disabling its I/O pins and certain internal logic. Power-down causes the D/P pin to be forced LOW, clears all internal storage elements, and forces all I/O pins to become high impedance. Internally, logic nodes which were driven by inputs to the LCA prior to power-down, are electrically isolated from their pins and forced HIGH. The PWRDWN pin should be left in the inactive (HIGH) state during the initialization and configuration states, and only be asserted while D/P is HIGH. Applications that do not use the power-down feature should tie the PWRDWN pin to Vcc.

Slave Mode M2:M1:M0 = 1:0:0

Configuring the LCA in the Slave mode is the simplest and most efficient method since it involves the fewest number of pins. In this mode, the configuration program is written into the device in a bit-serial fashion. During configuration, each bit in the program is sequentially shifted into the LCA's DIN input with the rising edge of the clock signal applied to the CCLK pin. See Figure 3. Note that in Slave mode, the CCLK pin is an input, not an output as it is in other modes. 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. The Slave mode configuration is especially appropriate in applications where a host processor configures the LCA through an I/O port, since the CCLK and DIN pins can then be driven via I/O instructions and the minimum data setup and hold times easily met. Another use of Slave mode configuration is in multiple LCA applications where the DIN and DOUT pins of several devices can be strung together in a daisy chain arrangement. This arrangement permits several LCAs to share a common source of configuration data.

In addition to the six non-programmable control pins, five programmable pins are used in this configuration mode: M2, DIN, DOUT, HDC, and LDC. These five pins are available as general purpose user I/O pins once configuration is completed. The 53 remaining programmable I/O pins are not used during configuration. See Figure 2a and Table 4.

In daisy chained LCA applications where the first LCA is configured in slave mode with a free-running CCLK source, care should be taken to insure synchronization with other devices in the chain. To accomplish this the designer must insure that RESET is released with the proper setup and hold times relative to CCLK. This guarantees that all LCAs in the daisy chain become operational simultaneously by insuring that they all begin configuration on the same clock cycle. This is easily done by de-asserting RESET with the falling edge of CCLK.

Peripheral Mode M2:M1:M0 = 1:0:1

The peripheral mode allows the configuration program to be written into the LCA by a host processor via the data bus as though it were an ordinary peripheral device. In this configuration mode the LCA may be thought of as a one-bit wide peripheral device, since the configuration program must be written into it one bit at a time. Typically, data bus bit 0 is tied to the DIN pin of the LCA and the data byte shifted between successive write instructions to the LCA. Next to the Slave mode, this mode involves the fewest number of LCA device pins for configuration. See Figure 2b.

As in the Slave mode, the configuration program is written into the device in bit-serial fashion. When the correct number of bits have been written into the LCA, the D/P pin goes HIGH indicating that configuration is complete. 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. During peripheral mode configuration, seven of the LCA's programmable I/O pins function as configuration control pins in addition to the six fixed, non-programmable control pins. Table 5 shows the configuration pins used in this mode. Figure 4 illustrates the timing relationship between the signals on these control pins.

While the DIN and DOUT pins function the same as in

Pin Name	Pin Number PLCC	Pin Number DIP	Pin Type	Value During Configuration	Description
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/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>	Config Data In
DOUT	59	41	Output	<Data>	Config Data Out
HDC	28	20	Output	HIGH	Constant "1" Level
LDC	30	21	Output	LOW	Constant "0" Level

Table 4. Slave Mode Pin Summary

Slave mode, four other pins serve as bus interface controls. Three of these pins—CS0, CS1, and CS2 become chip selects, while the fourth pin becomes the Write Strobe (WRT) input. The WRT pin serves the same function in this mode as CCLK does in Slave mode: a pulse applied to it while the three Chip Selects are asserted causes one bit of the program to be shifted into the DIN input of the LCA. Each write strobe to a peripheral mode LCA also produces a CCLK *output* pulse for purposes of driving the CCLK *inputs* of cascaded LCAs as shown in Figure 7. The three chip selects (two active LOW, one active HIGH) allow the LCA to be mapped to a specific I/O or memory address for configuration purposes. All nine pins are available as general purpose user-programmable I/O pins once configuration is completed. The 49 other programmable I/O pins are not used for peripheral mode configuration.

Master Modes M2:M1:M0 = 1:0:0 (Master Low Mode)
1:1:0 (Master High Mode)
0:0:0 (Master Serial Mode)

In the Master configuration modes, the LCA itself controls the loading of the configuration program automatically. In this mode, the LCA uses on-chip control logic to automatically address an external bytewide memory device (e.g., an EPROM) or uses RCLK to synchronize serial input data providing the configuration program. For the bytewide modes, sixteen of the LCA's I/O pins are used to generate an address bus, and eight other I/O pins to form a unidirectional data bus. Two options for the bytewide master mode exist: the Master Low mode, in which the memory is addressed in ascending sequence beginning at zero, or the Master High mode which uses a descending address sequence starting at hex address FFFF. With this addressing flexibility, the configuration data may share space in a ROM or EPROM used by the system, typically a microprocessor program. Once configuration begins, memory read cycles continue until the correct number of bits have been read, at which point the D/P pin goes HIGH indicating that program loading is completed. Bytes of data read from the external bytewide memory are serialized on-chip, and are independent of physical byte boundaries.

In addition to the sixteen address outputs and eight data bus input pins, several other signals are employed in this configuration mode. One is the RCLK output signal which is active LOW and goes HIGH while the address bus is changing states, allowing the use of "clocked" EPROMs for storing configurations. Other signals are the CCLK and DOUT outputs which are both used to drive cascaded (daisy chained) LCAs as shown in Figure 8. The pins used are summarized in Table 6 and the waveforms are shown in Figure 5.

Although sixteen bits of address are generated in the bytewide Master mode, not all are required to address the bytes needed to configure a single LCA. The extra addressing capacity of the LCA provides for storage of multiple configuration programs in a single EPROM device so that several daisy chained LCAs may be configured from a single source. Figure 8 presents an

Pin Name	Pin Number PLCC	DIP	Pin Type	Value During Configuration	Description
Fixed, Non-programmable Pins					
M0	26	18	Input	HIGH	Mode select
M1	25	17	Input	HIGH	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					
M2	27	19	Input	HIGH	Mode select
DIN	58	40	Input	<Data>	Config data in
DOUT	59	41	Output	<Data>	Config 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

Table 5. Peripheral Mode Pin Summary

example of a master mode LCA tied to a daisy chain of slave mode LCAs. Figure 5 shows the timing for the Master configuration mode.

In order to insure the successful configuration of daisy chained LCAs from a master mode device, the master mode device pauses briefly upon power-up before commencing with the configuration process. This power-up delay, which is substantially longer than the initialization delay for either slave or peripheral mode, allows for variations in LCA response to Vcc rise times and insures that all slave mode LCAs have time to become fully initialized and are ready for configuration data from the Master mode LCA. If longer delays are required to guarantee that all slave devices have been powered, then $\overline{\text{RESET}}$ may be used to hold off the start of configuration.

CONFIGURING MULTIPLE LCAs

Designs using multiple LCAs can reduce configuration overhead by logically concatenating the configuration programs. Using this option, LCAs can be connected together in daisy chains with one data source supplying the configuration program for all devices in the chain. The first LCA in the daisy chain may be configured in any of the configuration modes. Programs for all remaining devices in the chain are loaded using the pin-efficient Slave mode. When cascaded in this way, the LCA devices are programmed one at a time in sequence, starting with the first in the chain. Daisy chains of virtually any length can be configured in this manner.

LCAs are daisy chained together by connecting the DOUT pin of one device to the DIN pin of the next device in the chain. Each slave mode device is supplied with CCLK and program data from the device immediately proceeding it in the chain. Once a given LCA in the daisy chain has received its share of the configuration program, the balance of the program data are passed through to the remaining LCAs in the chain. Data passing through the LCA, from the DIN pin to the DOUT pin, is subject to a one clock cycle re-synchronization delay. Once configuration is complete, both DIN and DOUT are available to the designer as

Bytewide Master Mode Pin Summary

Pin Name	Pin Number PLCC	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 Ccock
$\overline{\text{RESET}}$	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	o	o	Outputs	<Address>	Memory address bus
				A15 A10 A0	
		48 DIP		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	o	o	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 6. Master Mode Pin Summary

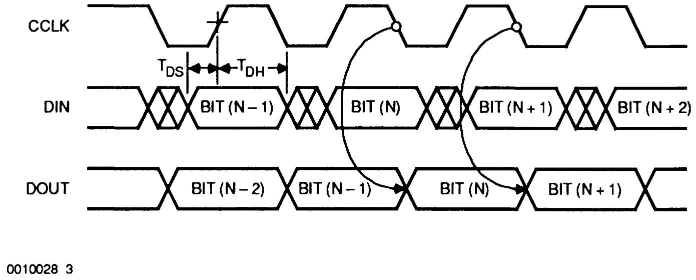


Figure 3. Slave Mode Configuration Timing

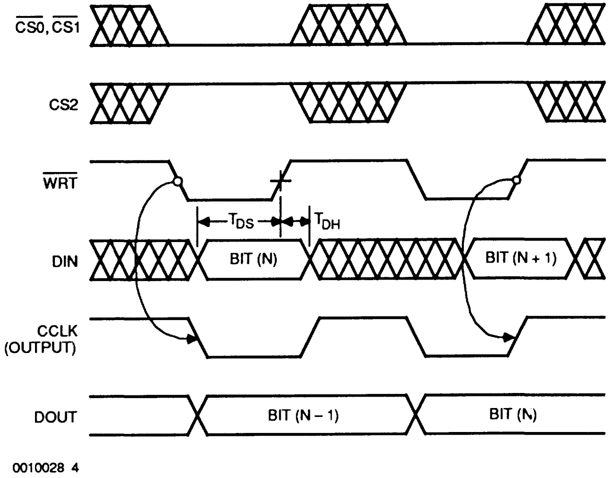


Figure 4. Peripheral Mode Configuration Timing

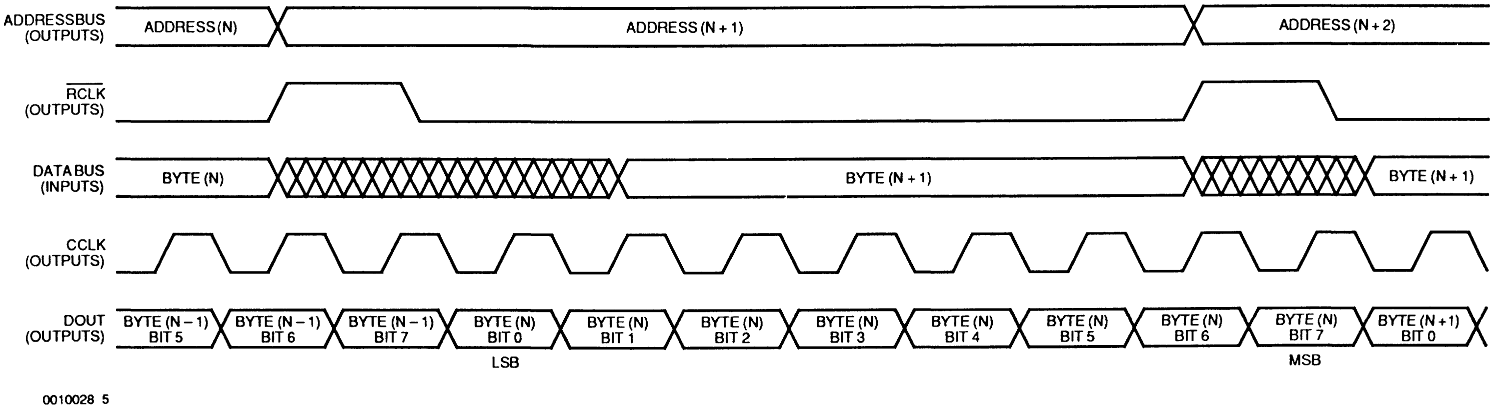


Figure 5. Master Mode Configuration Timing

general purpose I/O. Figures 7 and 8 illustrate how multiple LCAs can be connected together into daisy chains. Figure 6 shows the configuration timing for daisy chained LCAs.

CONFIGURING LCAs IN PARALLEL

In multiple LCA applications, there is a great deal of latitude in designing the configuration logic. The serial daisy chain technique described above is only one method by which multiple LCAs may be programmed. Another possibility which takes advantage of the bit-at-a-time nature of the slave and peripheral configuration modes is the simultaneous configuration of the LCAs in parallel. Multiple LCA devices, as shown in the peripheral mode example in Figure 9, can be simultaneously configured with each write cycle loading one bit into each of the LCAs. The total time required to configure the entire array of LCAs is now reduced to the time required for configuring a single device. Performance can be improved further through the addition of hardware to configure the group of LCAs via DMA transfers. If a processor is available for example, up to eight LCAs in parallel could be configured simultaneously from one program file stored on disk.

DESIGN CONSIDERATION FOR USE OF MULTIPLE-FUNCTION I/O PINS

Once a suitable configuration mode is selected, the designer may turn his attention to assignment of input/output functions to specific I/O pins. Usually this is based on logic block placement within the LCA, common I/O clock constraints, and I/O pin usage during configuration. User-definable I/O pins employed in configuring the LCA may be used by the end application, but require more careful design attention than the other I/O pins. For applications that require most of the programmable I/O pins, it is worthwhile to consider techniques for making efficient use of these dual-function pins.

Good design practice dictates that no logic signal conflicts should occur during either the configuration phase or the user-operation phase. These conflicts may not be obvious, since the directional nature of some of the I/O pins used for configuration change when the LCA completes its configuration and enters the user operation state. The design should guarantee that pins used as outputs during configuration (even though possibly not utilized in a given application) must not conflict with other logic sources also tied to those pins.

An example of an output pin that is easily overlooked is the DOUT pin: during configuration it becomes an output, regardless of whether or not it is used to drive the DIN pin of another LCA. Other examples include the HDC and $\overline{\text{LDC}}$ pins, which are driven HIGH and LOW respectively during configuration. An application design should be able to tolerate activity on these and the other I/O pins used during configuration without presenting a problem to external circuits that may also be tied to these pins. This can usually be accomplished either through careful pinout assignment, or through isolation buffers. The possibilities may be analyzed as three cases:

Case 1. I/O pins used for configuration are dedicated to that function only and are not used during user operation. In this case, no signal conflicts occur, but the number of /O pins available for use by the application is reduced.

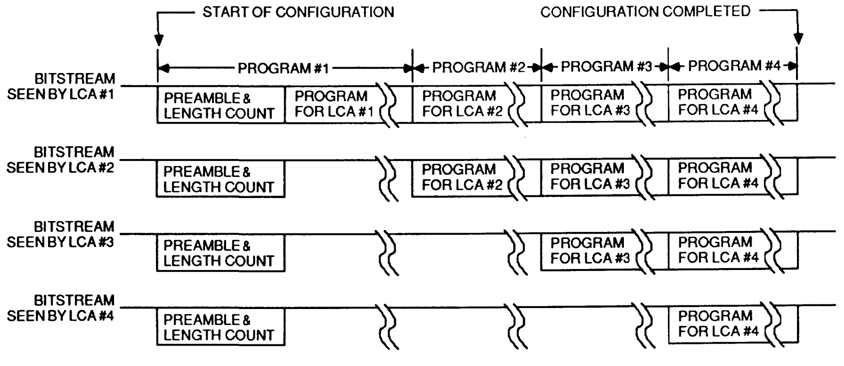
Case 2. I/O pins used for configuration are also used during user operation. However, since the signals are similar in input/output sense and the system suffers no adverse effect from transitions occurring on those pins during configuration, isolation buffers are not required.

Case 3. I/O pins used for configuration are also used during user operation. However, they either conflict in input/output sense, or have signal transitions during configuration which will adversely effect other system logic. Three-state buffers can be employed for this purpose, with perhaps the D/$\overline{\text{P}}$, $\overline{\text{LDC}}$, or HDC pin serving as the enable control for the buffers.

Recognition of the above scenarios in an LCA application, along with careful assignment of I/O functions to actual pins, can significantly reduce or eliminate external logic components in most cases. When faced with conflicts as in Case 3 above, the best approach is to try another pinout assignment which may eliminate the conflict. Isolation buffers should not be necessary in most designs since a typical design will have a number of inputs and outputs which can be assigned to I/O pins used during configuration without any conflicts. For example assigning output functions to pins that are already outputs during configuration (e.g., address outputs in master mode) may obviate the need for buffering those signals. In general, any sharing of similar pin functions during and after configuration may eliminate the need for external buffer logic.

The following are a few examples of how careful pinout assignment can reduce external component count:

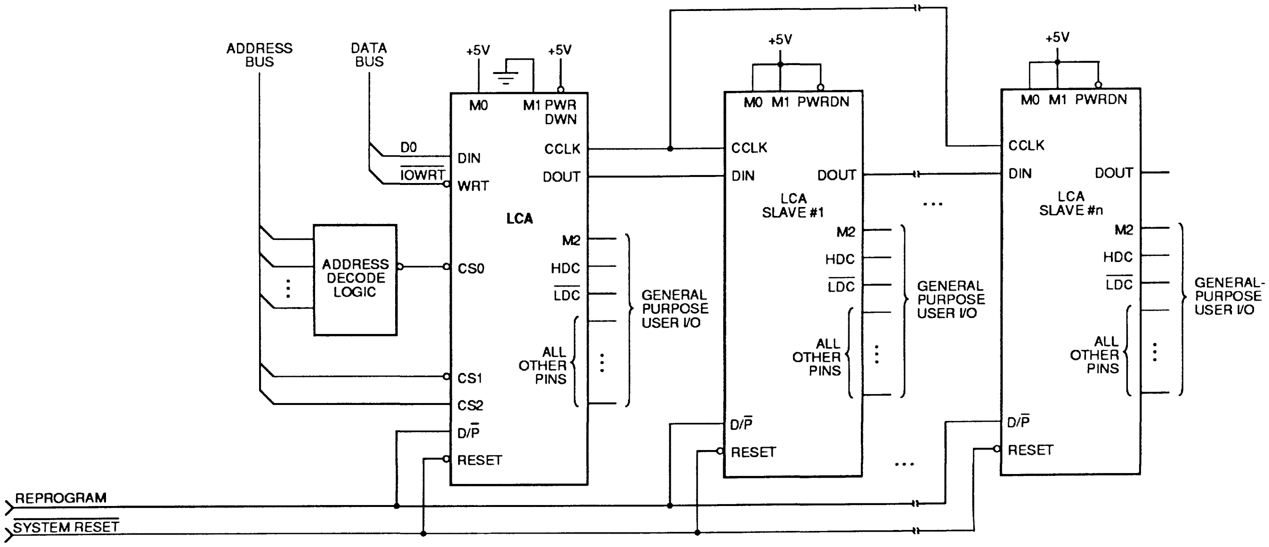
- Applications involving an address or data bus: If the application calls for such a bus and will use the LCA's master mode for configuration, these buses can effectively share pins with the master mode's address and/or data buses.
- Applications in which the LCA interfaces to a CPU bus and is configured in the peripheral mode: Configuration pins such as $\overline{\text{WRT}}$, DIN, and $\overline{\text{CS0}}$, $\overline{\text{CS1}}$



NOTE: HORIZONTAL DIMENSION (TIME) NOT DRAWN TO SCALE.

0010028 7

Figure 6. Timing for Daisy-Chained LCAs (Example Using Four LCAs)



0010028 8

Figure 7. Peripheral Mode LCA With Daisy-Chain

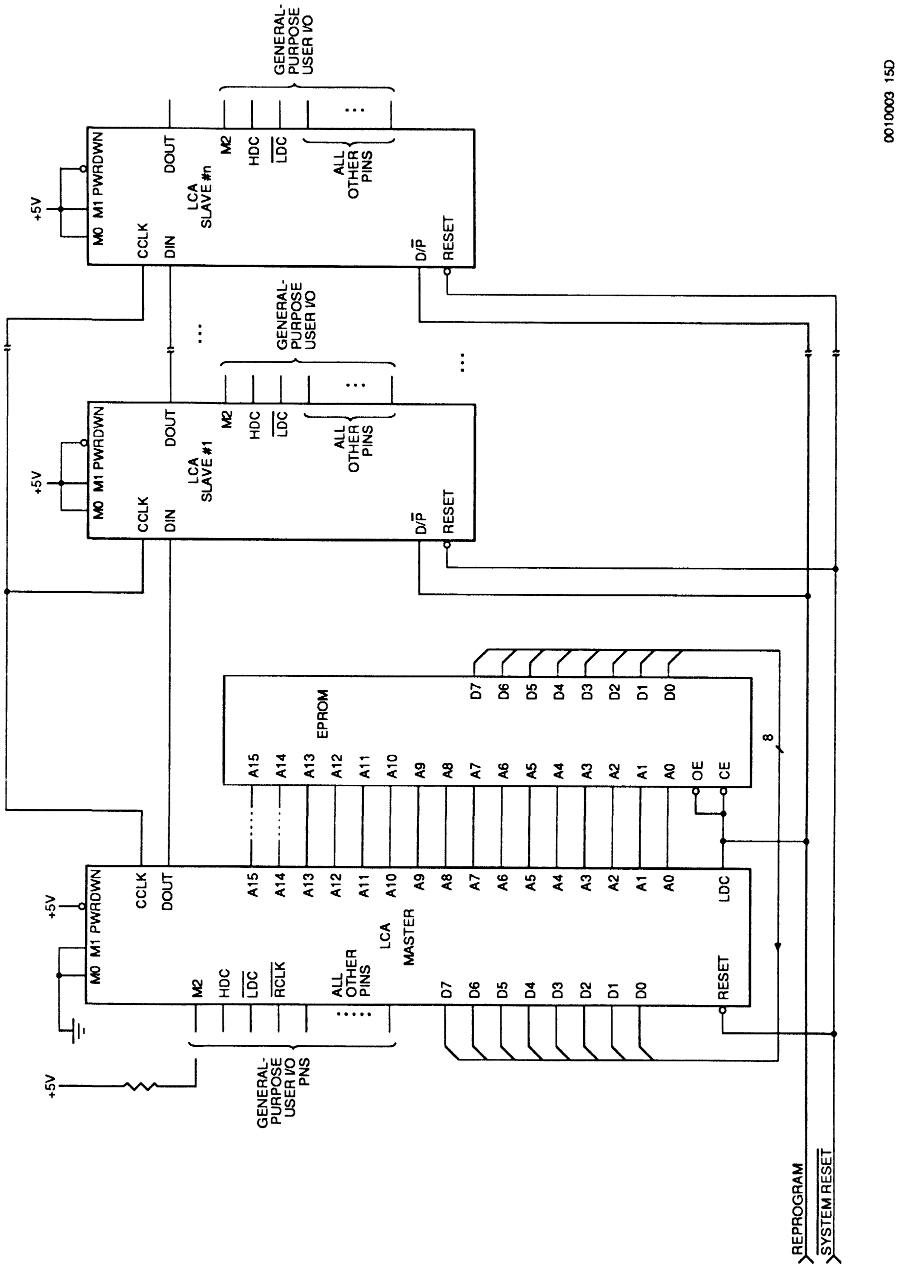


Figure 8. Master Mode LCA With Daisy-Chain

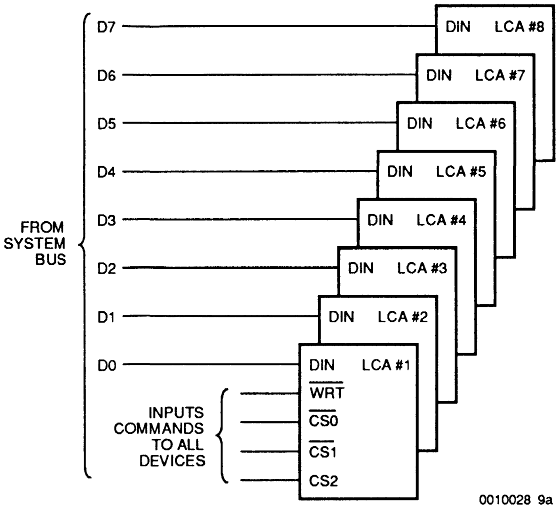


Figure 9a. Parallel LCA Configuration Using Peripheral Mode

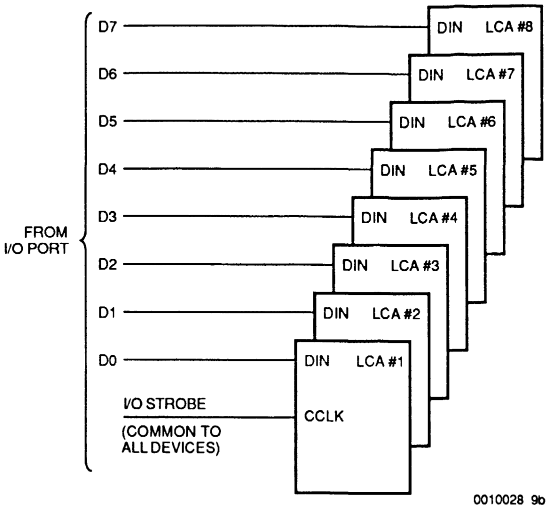


Figure 9b. Parallel LCA Configuration Using Slave Mode

and CS2 can be assigned similar functions in the final application since these pins will be driven from the host system's CPU bus both during and after configuration.

- Applications involving multiple LCAs where at least one signal passes from one device to another: it is most natural to assign that signal to the DIN-to-DOUT connection between those devices.

Unused I/O Pins

An LCA pin programmed as in input and not connected to any external logic is a "floating" input. As with any CMOS device, floating inputs can result in a low impedance current path from Vcc to ground and permanent device damage. Thus, unused LCA pins should be either:

1. Defined as an output and driven with an internal signal, preferably a constant "0" or "1" level.
2. Defined as an input and:
   a. Driven externally with logic, or
   b. Tied to an external pull-up or pull-down resistor, or
   c. Tied to Vcc or Ground

The relative advantages and/or disadvantages in defining unused pins as inputs or outputs will vary from application to application. Some of the considerations include: minimizing power dissipation (both static and dynamic), minimizing component count, risk of electrical damage to the device, and future circuit board modifications, etc. The preferred method of treating unused pins is to:

- Externally leave the pin open or unconnected,
- Internally configure the pin as an output, and
- Drive it with some constant level signal.

This is usually accomplished by selecting a nearby unused CLB output, defining it as either a constant "1" or "0", and tying that signal to all nearby unused IOBs. If internal routing congestion in the area precludes routing this dc signal to the IOB, the next best option is driving the IOB's output pin with one of the nets which is accessible. In this case, a net with the lowest toggle frequency is best since it will result in less power dissipation.

Test points

Another practical use for unused LCA pins is as diagnostic test point outputs. These test points can be very valuable for later monitoring of internal logic nodes which would otherwise be inaccessible.

THE CONFIGURATION PROGRAM

The information required to program the LCA can be viewed as a serial string of bits (i.e., 1's and 0's) that is shifted into the LCA one bit at a time until all the necessary configuration information has been loaded. This is appropriately referred to as the configuration program. The number of bits required to supply all the configuration information for a single LCA depends on depends on the device type and is outlined in Table 7. For applications using multiple LCAs connected as a daisy chain, the program grows for each additional LCA device.

The configuration program begins with several logic "1" bits (termed "dummy bits") followed by a 0010 preamble bit pattern (leftmost bit first). Following the preamble are 24 bits which represent the length count. The magnitude of this length count must equal or exceed a value two less than the total number of clock cycles required to shift in all the bits (including the dummy bits) in the bitstream. Length counts greater than this number up to $(2^{24}-1$ are permissible and merely serve to delay the D/$\overline{\text{P}}$ pin from going HIGH indicating the completion of configuration; all data associated with these additional clocks is ignored. Configuration bitstreams for several LCAs connected in a daisy chain have only a single preamble and length count.

Within the LCA, this count value is held in the length count register and compared to a CCLK cycle counter to determine when the configuration process is complete. When the value of the CCLK cycle counter compares to the value in the length count register, and all required data frames have been entered, configuration is complete and the D/$\overline{\text{P}}$ pin released. Since all devices in the daisy chain start their clock cycle counters simultaneously with the first CCLK cycle to occur after $\overline{\text{RESET}}$ is released, all LCAs in a daisy chain complete configuration and become operational simultaneously.

Configuration Bitstream Format: (Shown In Binary)

	XC2064 LCA	
1111	Dummy bits (4 Bits Minimum)	
0010	Preamble code	
< 24 Bit length count >	Configuration program listing	
1111	Dummy bits (4 Bits Minimum)	
0 < Data frame # 001 > 111		
0 < Data frame # 002 > 111		
0 < Data frame # 003 > 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
· · ·		
· · ·		
0 < Data frame # 159 > 111		
0 < Data frame # 160 > 111		
1111	Postamble code (4 bits minimum)	
	XC2018LCA	
1111	Dummy bits (4 bits minimum)	
0010	preamble code	
< 24 Bit Length Count >	total number of bitstream bits	
1111	dummy bits (4 bits minimum)	
0 < Data frame # 001 > 111		
0 < Data frame # 002 > 111		
0 < Data frame # 003 > 111	197 Configuration data frames	
· · ·		
· · ·	(Each frame consists of: a 0 start bit, a 87-bit data field, and 2 or more dummy bits)	Repeated once for each LCA in the daisy chain
· · ·		
· · ·		
0 < Data frame # 196 > 111		
0 < Data frame # 197 > 111		
1111	Postamble code (4 bits minimum)	

Notes:

1. Data bits (as shown in the table) are shifted into the LCA with the leftmost bit of each line in the table above being entered first. The bit field containing the length count is shifted in 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 LCAs being configured from this one bitstream.

0010028 11

Table 7. Configuration Bitstream Format

48 DIP	68 PLCC	68 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
	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>>				
	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 (LOW)	M0 (HIGH)	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					
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					
	49	H11			D5 (I)		
35	50	G10		$\overline{\text{CSO}}$ (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 20–50 KΩ INTERNAL PULL-UP DURING CONFIGURATION

0010003 20

XC2064 Pin Assignments

68 PLCC	68 PGA	84 PLCC	84 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>>				
		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 (LOW)	M0 (HIGH)	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					
35	K6	43	J6	GND				
		44	J7	<<HIGH>>				I/O
36	L6	45	L7					
37	K7	46	K7					
38	L7	47	L6					
		48	L8					
39	K8	49	K8					
40	L8	50	L9					
41	K9	51	L10			D7 (I)		
42	L9	52	K9			D6 (I)		
43	L10	53	L11					XTL2 OR I/O
44	K10	54	K10	$\overline{\text{RESET}}$ (I)				
45	K11	55	J10	DONE (O)				$\overline{\text{PROG}}$ (I)
46	J10	56	K11	<<HIGH>>				XTL1 OR I/O
47	J11	57	J11					I/O
48	H10	58	H10					
		59	H11					
49	H11	60	F10			D5 (I)		
		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	<<HIGH>>				I/O
54	E10	66	E11		CS2 (I)	D2 (I)		
		67	E10					
55	E11	68	E9					
		69	D11					
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	<<HIGH>>		A0 (O)		I/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			A4 (O)		
67	B7	81	B6			A14 (O)		
		82	B7					
		83	A7					
68	A7	84	C7			A5 (O)		

<<HIGH>> IS HIGH IMPEDANCE WITH A 20–50 KΩ INTERNAL PULL-UP DURING CONFIGURATION

0010003 21

XC2018 Pin Assignments

The value used for the length count is a function of how many LCAs will be configured by this one program. For example, if there are three XC2064 LCAs connected in a daisy chain, then the configuration program would be approximately 36,000 bits in length. This value is approximate since the length count is included only once at the beginning, and since several additional cycles are required to compensate for the re-synchronization delay of the data at each DOUT pin. The precise value of the length count is computed by the XACT development software and automatically entered into the program data file. The preamble and length count bits are sensed by each LCA at its DIN pin and immediately passed on to the next LCA in the daisy chain via the DOUT pin. Afterwards, however, each LCA in turn accepts its portion of the configuration program before passing any subsequent data on to the next device. See Figure 6.

Within the configuration program data are presented in frames which begin with a "start" bit and end with two or more dummy or "stop" bits. Between the start and stop bits of each frame there is a data field which defines the user's logic functions. The last frame is followed by a field of postamble bits.

Notes:

1. Xilinx reserves the right to change the format, organization, and/or length of the program used to configure the LCA.
2. The documentation presented here applies only to the bitstream data generated by the XACT development system for use in EPROMs; the XACTOR In-Circuit emulation uses a somewhat longer version of the configuration program.

XILINX

Ins and Outs of Logic Cell™ Array I/O Blocks

Table of Contents

Ins and Outs of Logic Cell™ Array I/O Blocks

INTRODUCTION

This Application Note describes various uses for Input/Output Blocks within the XC2064 and XC2018 Logic Cell Arrays.

The architecture of the XILINX Logic Cell Array (LCA) provides great design flexibility in using inputs and outputs. Since the Input/Output Blocks (IOBs) in an LCA are not dedicated to any fixed logic, they may also be used for many things beyond simple inputs or outputs. Many designs will not use all of the IOBs available within the LCA. In such cases, the design engineer can build logic structures such as shift registers or Johnson counters in unused IOBs.

The Input/Output Block

All of the Input/Output Blocks (IOBs) of the Xilinx XC2064 and XC2018 Logic Cell Arrays (LCAs) are identical. However, each IOB can be individually configured by the designer 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 do both input and output under three-state control. A schematic of the IOB is shown in Figure 1. The trapezoidal structures are data path selectors or multiplexers. The programming of these data path selectors determines the function performed by the IOB.

Along each edge of the LCA die, the IOBs share a common I/O clock signal which drives the input register. All of the internal registers are reset to a "0" 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.

Logic signals external to the LCA come in through an I/O pad and a non-inverting buffer. The signal is then either

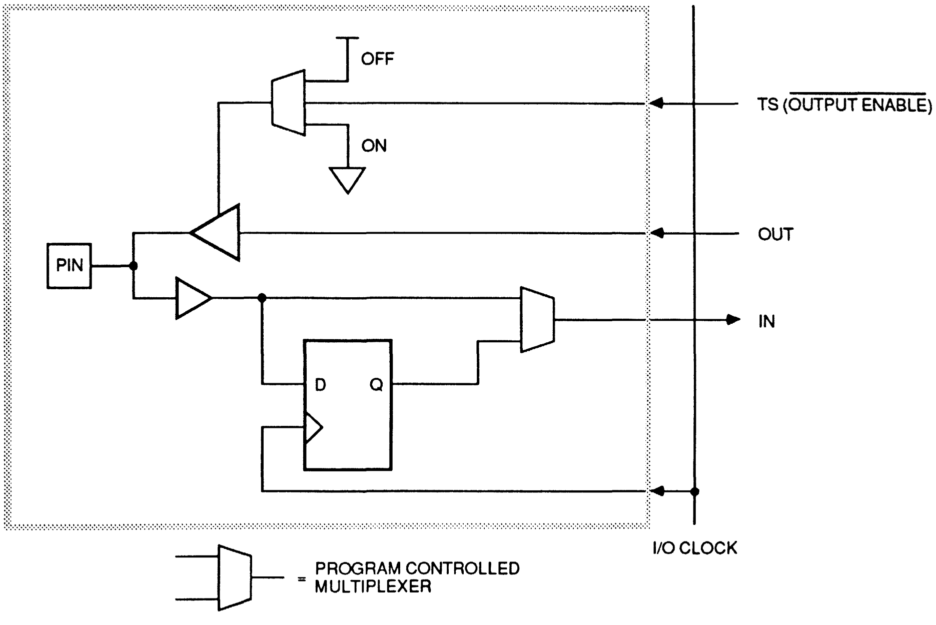


Figure 1. Input/Output Block (IOB). An Input/Output Block (IOB) can be configured as either a direct or registered input, a direct or three-state output, or as a bidirectional data line.

directly propagated or fed into the input register, depending of the configuration of the data path selector. Similarly, output data is driven by a non-inverting buffer. The output buffer is forced into a high-impedance state whenever the three-state 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.

All IOBs can be globally configured to recognize either TTL-level (Vth = 1.4 Volts) or CMOS-level (Vth = 2.2 Volts) input thresholds. This option affects overall device power consumption. Power consumption is lower when CMOS input levels are selected.

Scope and Purpose

The purpose of this application note is to describe some functions available by configuring IOBs in various ways—some obvious, some not so obvious. These structures include:

- Standard Inputs and Outputs
- Open-Collector I/O
- Schmitt-Triggered I/O
- Oscillators
- IOB-based Registers (data, shift, etc.)
- Counters (Johnson, Linear Feedback)

I/O STRUCTURE DESCRIPTIONS

Conventions Used in the I/O Descriptions

For each I/O structure, the following conventions will be maintained. Each structure will be shown in schematic form and the IOB configuration will be described. Configurations available for the input path include:

- (I:PAD)—Direct input from the device pad.
- (I:Q)—Registered input.
- (I:)—No input.

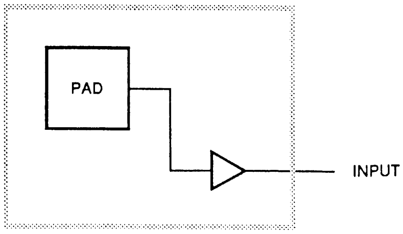
The output buffer may be configured as follows:

- (BUF:ON)–Direct output.
- (BUF:TRI)—Three-state output.
- (BUF:)—No output.

If a macro definition (MACRO) exists for the structure described, it will also be listed. Any special configuration information will be described under "COMMENTS."

STANDARD I/O STRUCTURES

I/O TYPE: Pad Input
MACRO NAME: PIN
SCHEMATIC:



001022 2

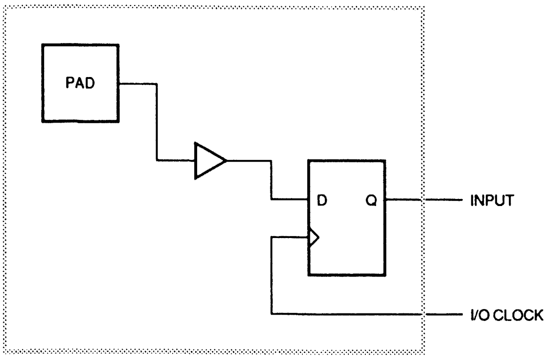
CONFIGURATION:
I:PAD
BUF:

Figure 2. Pad Input

Registered Inputs and Metastability Characteristics

Figure 3 is a schematic of a registered input within the LCA. Logic Cell Arrays are manufactured with a high-speed CMOS process. This allows the I/O Block input registers to achieve flip-flop loop delays of three to five nanoseconds. Short loop delay provides very good performance under asynchronous clock and data transitions. Short loop delays minimize the probability of a metastable condition which can result when the input into the flip-flop is still in transition when the clock is asserted. The short loop delay characteristics of the I/O Blocks allow the device to be effective in synchronizing external signals. Once synchronized in the IOB, the signals can be used internally without further consideration of their relative timing, except as it applies to internal logic and routing path delays. Further information regarding the metastable behavior of registers within an LCA is contained in the XILINX application note, *Metastability Analysis of LCA Flip-Flops.*

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

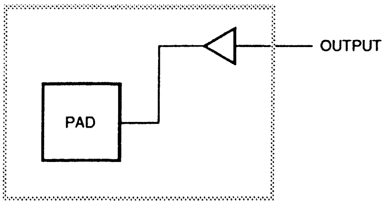


0010028 3

CONFIGURATION:
I:Q
BUF:

Figure 3. Pad Input With Storage Register

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

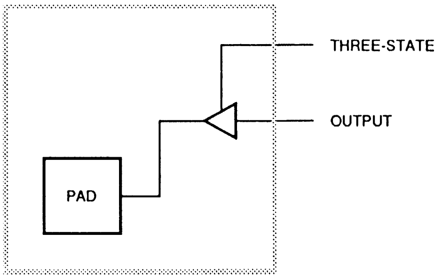


0010022 4

CONFIGURATION:
I:
BUF:ON

Figure 4. Pad Output

I/O TYPE: Pad Output with Three-State Control
MACRO NAME: POUTZ
SCHEMATIC:

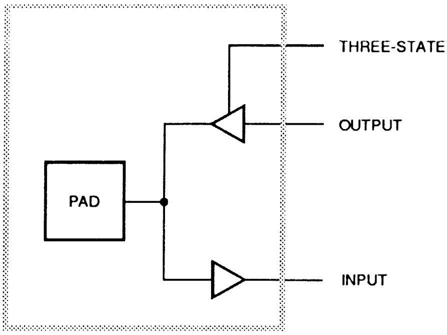


0010022 5

CONFIGURATION:
I:
BUF:TRI

Figure 5. Pad Output With Three-State Control

I/O TYPE: Pad Input/Output (bidirectional data line)
MACRO NAME: PIO
SCHEMATIC:

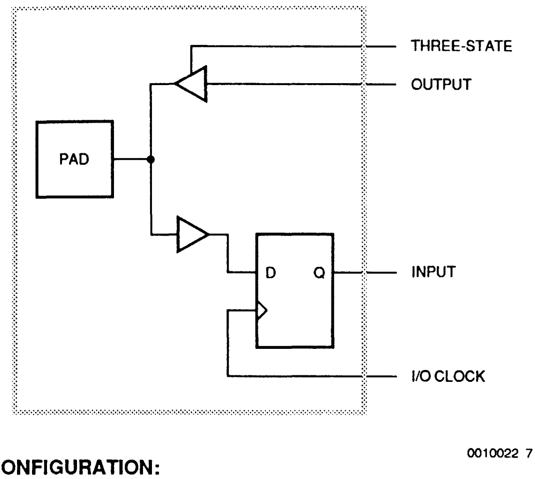


0010022 6

CONFIGURATION:
I:PAD
BUF:TRI

Figure 6. Pad Input/Output (Bidirectional)

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



CONFIGURATION:
I:Q
BUF:TRI

Figure 7. Pad Input/Output With Input Storage

"OPEN-COLLECTOR" I/O STRUCTURES

Overview

"Open-collector" outputs can be used for a variety of functions including "wired" AND and OR structures. For MOS devices like the Logic Cell Array, a more accurate terminology is open-drain outputs, since an MOS transistor has no collector.

To build an open-drain output structure in an LCA, both the output and the three-state control lines are tied together. 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 will also be high impedance, allowing that particular signal line to "float." Connecting this signal line to Vcc through a resistor will pull this line up for an active HIGH output. However, for active LOW signals, the three-state control line is driven LOW. This turns on the output buffer and allows the LOW signal to propagate directly to the I/O pad.

Open-Drain Structures and Routing

When designing with open-drain structures, the designer 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.

Since a routed signal may take longer to reach an IOB's three-state control line than its output line, the pad may be driven for a short period of time during a LOW to HIGH transition. This situation could occur if the output line (O) starts to go HIGH before the three-state control line (T). Depending on how much routing delay there is between the output (O) and three-state (T) lines, the PAD output could start to go HIGH and then be driven into high-impedance. Excessive routing delay differences between the output (O) and the three-state control line (TS) may cause a brief output glitch as shown in Figure 8. Careful design will prevent this.

This situation is not a problem in most designs. The actual routing delay difference between the (T) and (O) terminals of an IOB can be checked using the timing calculator included in the XACT™ Development System.

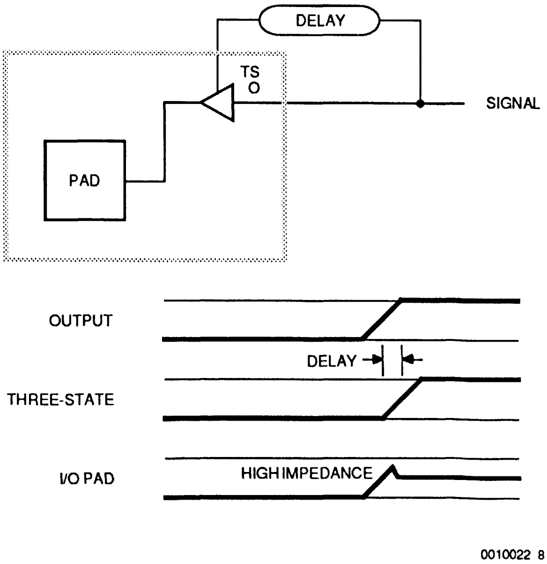
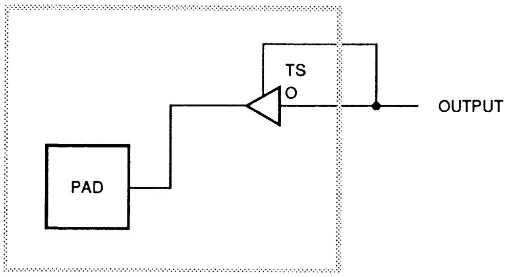


Figure 8. Brief Output Glitch Caused by Three-State Routing Delay

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

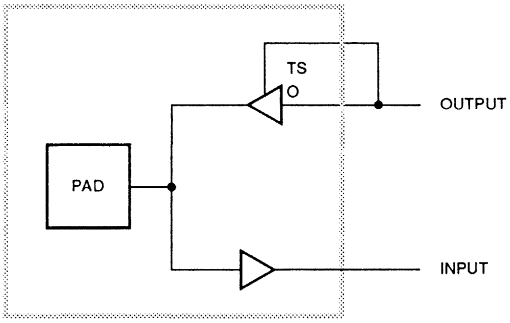


0010022 9

CONFIGURATION:
I:
BUF:TRI

Figure 9. Pad Output With "Open-Collector"

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

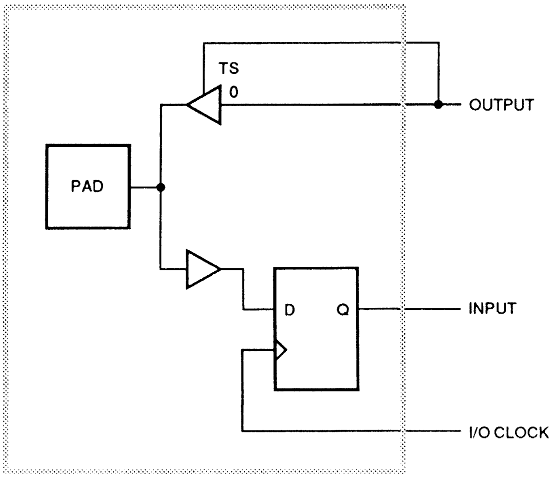


0010022 10

CONFIGURATION:
I:PAD
BUF:TRI

Figure 10. Pad Input/Output With "Open-Collector"

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



0010022 11

CONFIGURATION:
I:Q
BUF:TRI

Figure 11. "Open-Collector" Pad Output With Storage

"Wired" AND and "Wired" OR Structures

The open-drain capability of Input/Output Blocks allows a designer to build "wired" AND and "wired" OR structures. The AND and OR implementations are essentially the same. The only difference between the forms is the type of logic used. "Wired" AND structures are used in positive-logic implementations, while "wired" OR structures are used in negative-logic implementations.

Figure 12 shows a typical "wired" AND or "wired" OR structure. All of the 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 would be turned on and that LOW signal would propagate to the common line. This, in turn, would pull 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. The "wired" AND logic is shown in Equation 1.

IOB1 • IOB2 • IOB3 • ... • IOBn = TRUE [1]

A "wired" OR structure is similar 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 in Equation 2 structure is merely a "DeMorganized" inversion of Equation 1.

IOB1 + IOB2 + IOB3 + ... + IOBn = FALSE [2]

A typical application of a "wired" OR structure is an active low common interrupt line. If any peripheral requests an interrupt, the common interrupt line is pulled low, signalling the processor of the request. A "wired"-AND or "wired"-OR function can be built from any number of open-collector outputs.

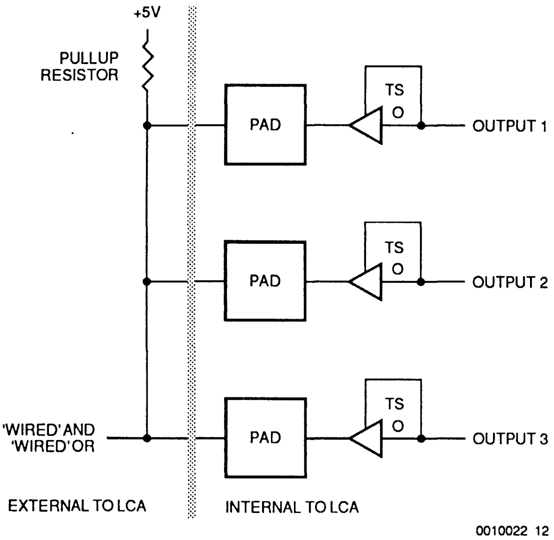


Figure 12. "Wired" AND or "Wired" OR Function

Multiplexers from "Open Collector" outputs

Another structure which can be built using open drain IOBs is an n-bit multiplexer, as shown in Figure 13. All of the PAD outputs are tied together outside of the package on a common line which becomes the multiplexer output. Each IOB in the example is configured as an output with three-state control (Macro = POUTZ). The output line (O) of each IOB becomes an input for the multiplexer. A signal is selected by driving the corresponding three-state control line low (T = 0). The selected signal then propagates to the common output line. The three-state control lines can be driven with a Configurable Logic Block.

CAUTION: The designer must avoid output contentions on the common output line.

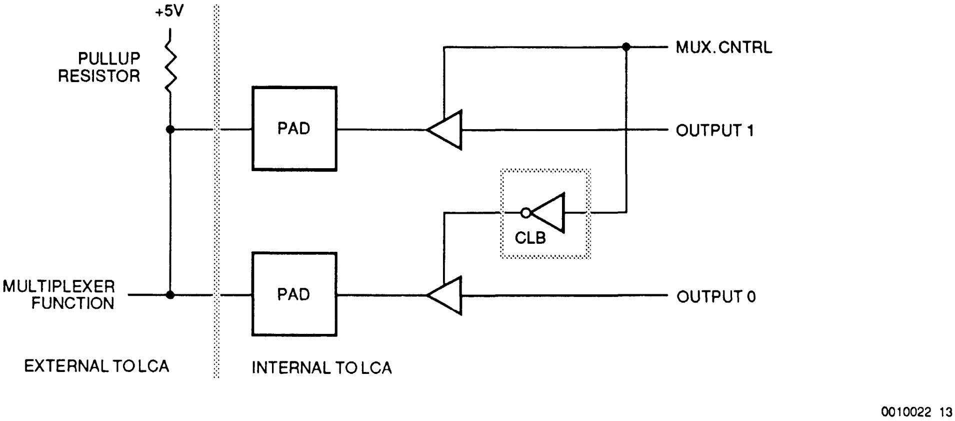


Figure 13. "Open Collector" Multiplexer Function. A multiplexer can be built from open-collector outputs. A specific output signal is selected by enabling the output buffer for that signal (TS = 0).

SCHMITT-TRIGGERED I/O STRUCTURES[1]

Overview

The Schmitt trigger has numerous applications in digital designs. Two of the most common are shown in Figure 14. Schmitt-triggered inputs can filter signal noise due to hysteresis built into the switching characteristics. Schmitt triggers are also useful to generate fast transitions when a slowly changing input function reaches a predetermined level. Again, this effect is due to hysteresis.

Schmitt-triggered inputs can be built in a number of ways within an LCA. Using three IOBs, a CLB, and three resistors, a designer can build Schmitt triggers with selectable voltage hysteresis. If the amount of hysteresis is not critical, then the resource requirements are reduced to only two resistors and two IOBs.

The threshold voltage and the amount of hysteresis for a complete Schmitt trigger are selected using three resistors. The three resistors are separated into two resistor network pairs (R1:R2 and R1:R3) as shown in Figure 15. Each pair forms a voltage divider to set the input voltage level—one to set the HIGH going transition level (Vh) and one to set the LOW going transition level (Vl). The input 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 since both 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.

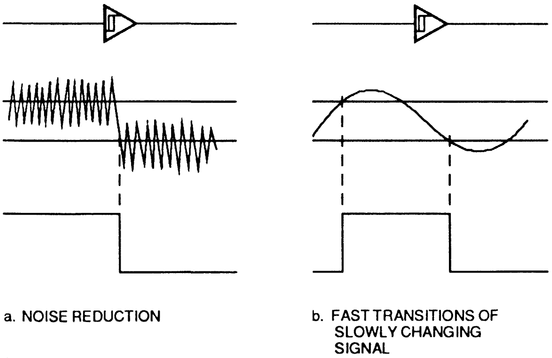


Figure 14. Application of a Schmitt trigger. Example applications of a Schmitt trigger include a.) A noisy input signal cleaned up through a Schmitt trigger. b.) Fast transitions generated from a slowing changing input signal through a Schmitt trigger input.

An inverting Schmitt trigger is similar except that the sense of the logic is inverted inside the LCA.

Theory of operation

Assume that the input voltage is near ground. The output voltage of IOB2 is at Vol, which pulls resistor R2 toward ground. There is then no potential difference across R2. The output buffer of IOB3 is high-impedance, since its three-state control pin is HIGH. Resistor R3 is effectively removed from the circuit. 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 will eventually reach its switching threshold.

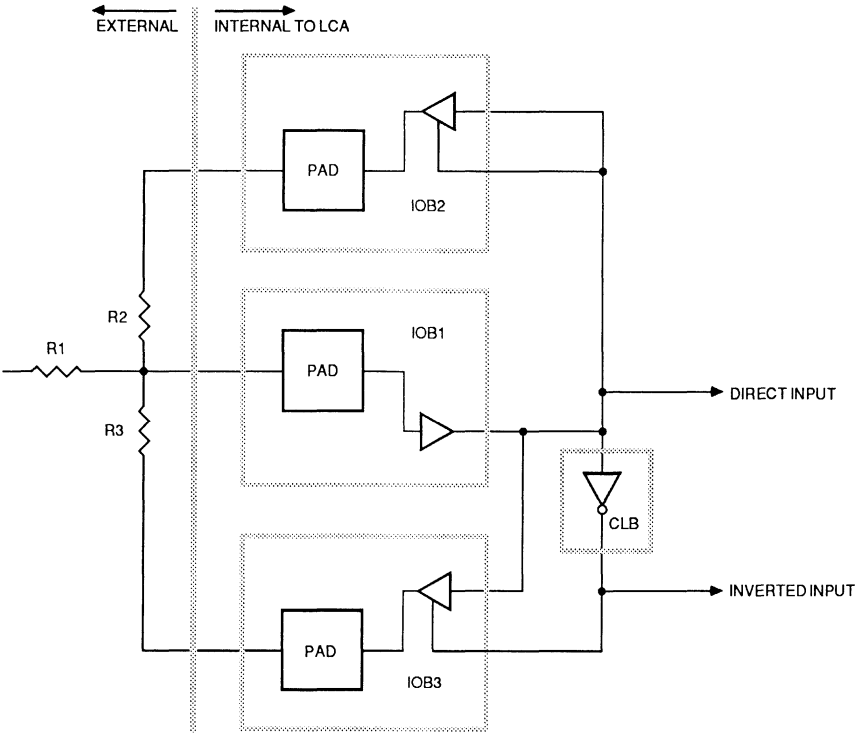
Once the threshold is crossed, IOB1 goes HIGH, driving the output of IOB2 into high-impedance (IOB2 TS=1) and enabling the output buffer of IOB3 (IOB3 TS=0). Now at Voh, IOB3 pulls the input of IOB1 high through resistor R3. In this state, resistor R2 is effectively removed from the circuit, since IOB2 is high-impedance.

This structure will remain in its present state even if the input voltage fluctuates. If the input voltage fluctuates to the opposite hysteresis limit, the Schmitt trigger will go to the opposite state. In other words, the Schmitt trigger will stay HIGH until the input to IOB1 drops below the LOW going hysteresis limit and *vice versa*.

If the values of the hysteresis are not critical, the Schmitt trigger can be reduced to only two IOBs and two resistors as shown in Figure 16. However, the range of Vh and Vl are very limited. The IOB configured as an output pulls the input HIGH or LOW, depending on the transition direction.

If a selectable Schmitt trigger is required only for a single transition direction (HIGH going LOW, or LOW going HIGH), then the circuit shown in Figure 15 can be further simplified to those shown in Figure 17a and 17b. Note, however, that a single CLB is required to invert the sense of input signal to enable or disable the output buffer for IOB2 (the IOB configured as a three-state output).

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



0010022 15

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

COMMENTS: Resistors pairs R1:R2 and R1:R3 form two voltage dividers which set the HIGH-going and LOW-going input hysteresis. Resistors R1 and R2 set the HIGH-going hysteresis (Vh) according to Equation 3.

$$Vh = Vth\ [(R1+R2)/R2] - Vol\ (R1/R2) \qquad [3]$$

Resistors R1 and R3 set the LOW-going hysteresis (Vl) according to Equation 4.

$$Vl = Vth\ [(R1+R3)/R3] - Voh\ (R1/R3) \qquad [4]$$

Notes: Vth = input threshold voltage
for CMOS inputs, Vth = 2.2 V
for TTL inputs, Vth = 1.4 V ± supply tolerance

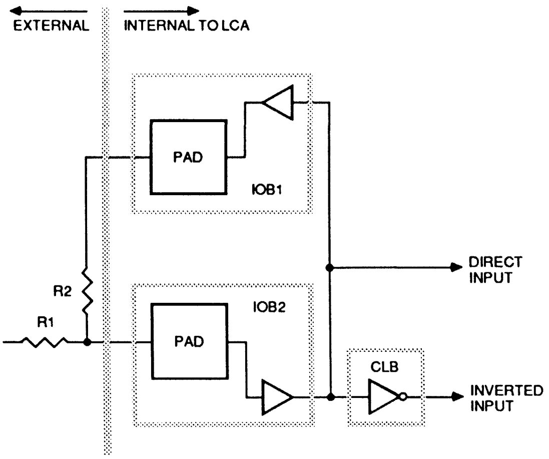
Figure 15. Schmitt-Triggered Input With Selectable Hysteresis.

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

$VI = [(R1+R2) / R2]\ Vth\ Voh(R1/R2)$ [5]
$Vh = [(R1+R2) / R2]\ Vth$ [6]



0010022 16

Figure 16. Schmitt-Triggered Input With Limited Hysteresis.

OSCILLATORS USING IOBs

Overview

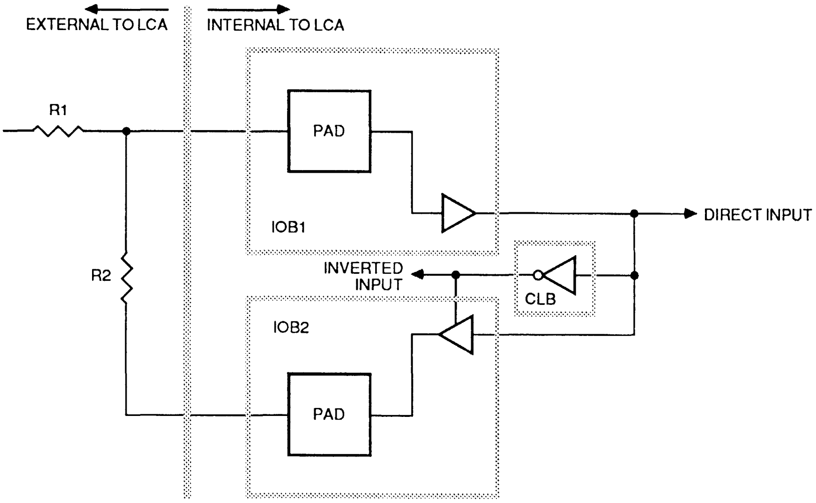
General purpose oscillators can be built using two Input/Output Blocks (IOBs) and a Configurable Logic Block (CLB). The general theory of operation is similar to that described for Schmitt triggers. For the oscillator described below, an oscillating signal is generated by the charging and discharging of two capacitors. The circuit is shown in Figure 18. Capacitor C2 charges to a voltage threshold (on Set) to set a latch. Once the voltage across C2 exceeds the threshold, the SET line causes the "Q" line to go high and starts discharging 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. Once capacitor C1 charges to its threshold, the "Q" output is reset and forced low. Capacitor C1 is now discharged by the IOB named CQ, and capacitor C2 begins charging again. This process repeats, creating a low-frequency resistor-capacitor oscillator.

The designer should consider the routing delay of the three-state control lines within IOBs marked as CQ and CQL in Figure 18. The time period of the oscillator depends on each capacitor being completely discharged during the opposite timing phase. In addition, the 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 (T) may prevent the capacitors from completely discharging.

Any number of these low-frequency oscillators can be used in a design. In most designs, however, only one or two are required. If the oscillator output is used throughout the design to clock the registers within the CLBs, place the oscillator near one of the clock buffers and use the clock buffer. Figure 18 shows a low-frequency oscillator built near the main clock buffer in the upper left-hand corner of the die. A similar low-frequency oscillator may drive the auxiliary clock buffer located in the lower right-hand corner of the die.

The designer should be aware that the low-frequency oscillator circuit will cause an error when using the timing calculator to examine the oscillator. The timing calculator in the XACT Development System detects combinatorial loop conditions and flags them as errors. The oscillator circuit depends on combinatorial loopback for operation and will cause an error message. Such errors messages can be safely ignored if they are detected only in the oscillator circuit.

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

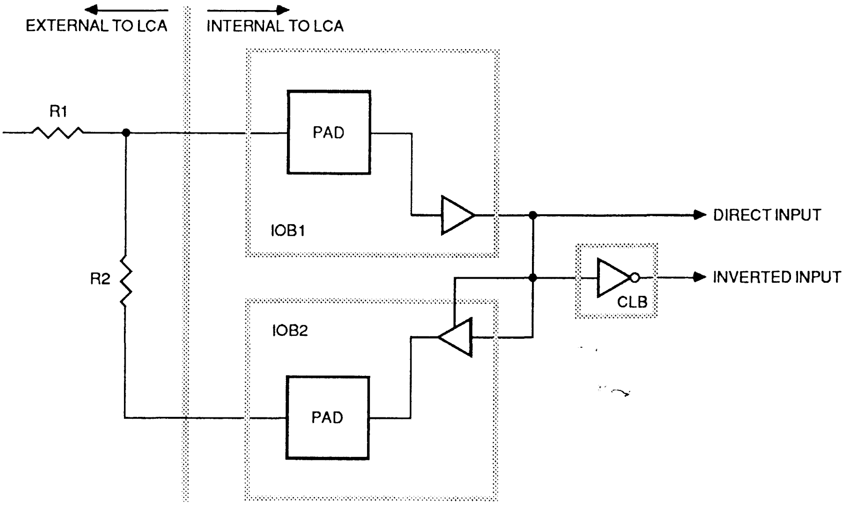


Vh = Vth [7]
Vl = Vth [(R1+R2)/R2] – Voh(R1/R2) [8]

0010022 17a

Figure 17a. Unidirectional Schmitt-Triggered Input HIGH Going LOW

SCHEMATIC:

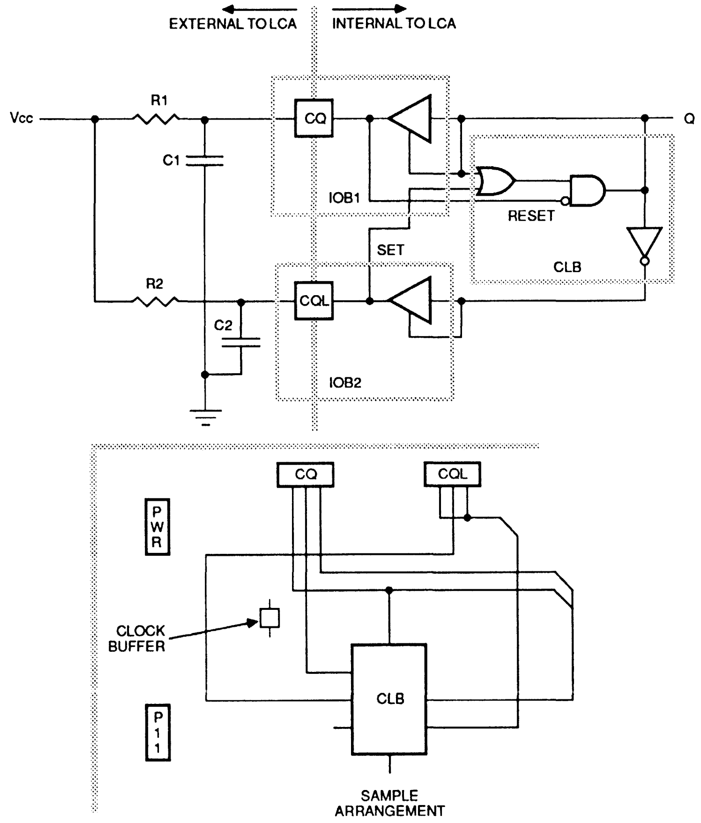


Vh = Vth [(R1+R2)/R2] – Vol(R1/R2) [9]
Vl = Vth [10]

0010022 17b

Figure 17b. Unidirectional Schmitt-Triggered Input LOW Going HIGH

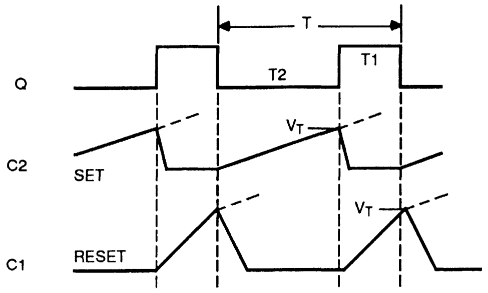
I/O TYPE: Low-Frequency Resistor-Capacitor Oscillator
MACRO NAME: GOSC
SCHEMATIC:



CONFIGURATION:
IOB1
  I:ON
  BUF:TRI
IOB2
  I:ON
  BUF:TRI

0010022 18

Figure 18. Low-Frequency Resistor-Capacitor Oscillator



0010022 19

Figure 19. Low-Frequency Resistor-Capacitor Oscillator Timing Diagram

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: Each capacitor is discharged during opposite timing phase. Capacitors begin charging from GROUND. Effect of three-state routing delay is assumed minimal.

ON-CHIP CRYSTAL OSCILLATOR

Basic Description

Two special user-defined I/O Blocks can be configured to interface directly to the on-chip crystal oscillator located in the lower right-hand corner of the die. The crystal oscillator is associated with the auxiliary clock buffer located near the oscillator. When the interconnect is selected to drive the auxiliary clock buffer two special pins interface directly to the internal high-speed inverting amplifier to form the oscillator. Externally, these pins should be attached to crystal oscillator components as shown in Figure 20. The best way to configure the crystal oscillator is through the MACRO named GXTL.

Even before device configuration is complete, the on-chip oscillator begins operation so that the circuitry can stabilize. The actual internal connection of the oscillator to other circuitry on the chip is delayed until completion of device configuration.

Theory of Operation

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. The series resistor (R2) may be included to add to the amplifier output impedance when needed for phase shift control, crystal resistance matching, and to limit the amplifier input swing to control clipping at large amplitudes.

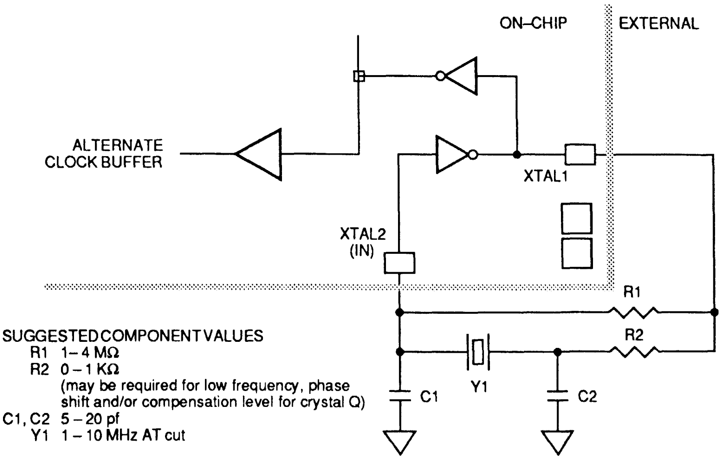
Excess feedback voltage may be adjusted by the ratio of capacitors C2/C1. The amplifier is designed for use in the range from 1 MHz up to one-half the specified CLB toggle frequency. Using the oscillator at frequencies below 1 MHz requires individual characterization with respect to a series resistance. Operation at frequencies above 20 MHz is more involved since it generally requires that the crystal operate in a third overtone mode in which the fundamental frequency must be suppressed by the R-C networks.

REGISTERS IN IOBs

Overview

The previous examples describe the use of IOBs for conventional applications. All involved using either the

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 DIP	33	30
68 PLCC	46	43
68 PGA	J10	L10
84 PLCC	56	53
84 PGA	K11	L11

0010003 10

Figure 20. On-Chip Oscillator Circuit

input, or the output, or both. If an IOB is not required for input or output, the storage element within each IOB can be used to create registers and various types of counters. All of the following designs involve using the output buffer (BUF:ON) fed back into the input register (I:Q). This configuration, which is used with slight modifications in each of the following examples, is shown in Figure 21. The pads of the IOBs involved are not typically connected to anything externally, although they may be if desired.

IOB-based Register Delays

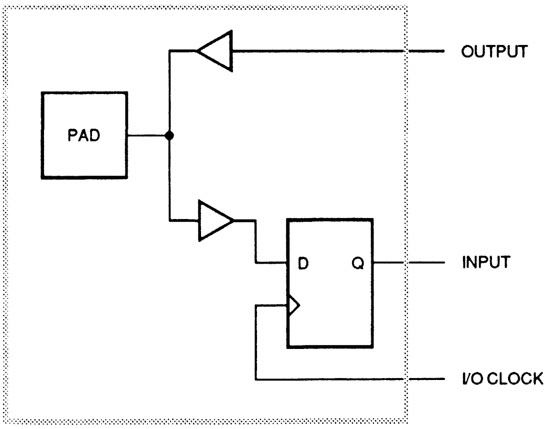
The delays incurred through an IOB-based register depend on the sum of two parameters—the delay through the output buffer and the delay back through the input buffer to the register. While these values are defined in the data sheet for an output load of 50 pF, their values change only slightly for no output capacitance. The delay into an IOB-based register is:

$$T(IOB\text{-}reg) = Top + Tpl \qquad [12]$$

where:

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

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



0010022 21

CONFIGURATION:
I:Q
BUF:ON

Figure 21. Pad With Input Storage (IOB-Based Register)

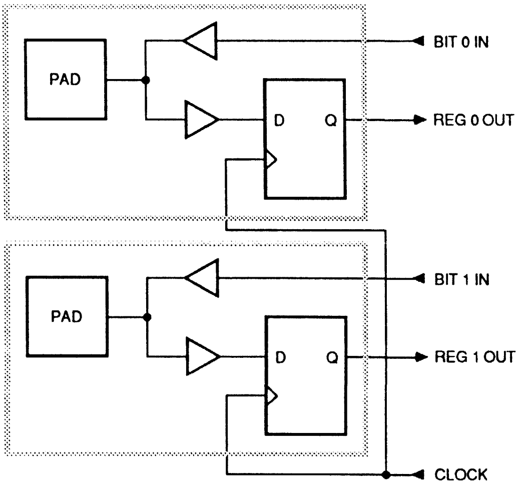
Wide Storage Registers

Wide storage registers can be built from the basic structure described in Figure 21. For example, Figure 22 shows the construction of a n-bit storage register built from IOBs. Wide storage registers are ideal for IOBs, since the I/O clock feeding an IOB is common to all IOBs along each edge of the die.

Read/Write Registers

Another variation of the basic IOB-based register is a simple read/write register. This structure allows data to be written into registers within the LCA from an external device and also read back. Figure 23 shows the structure of a read/write register. In this example, the

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



0010022 22

EXPANDABLE TO N BITS

CONFIGURATION:
All IOBs
I:Q
BUF:ON

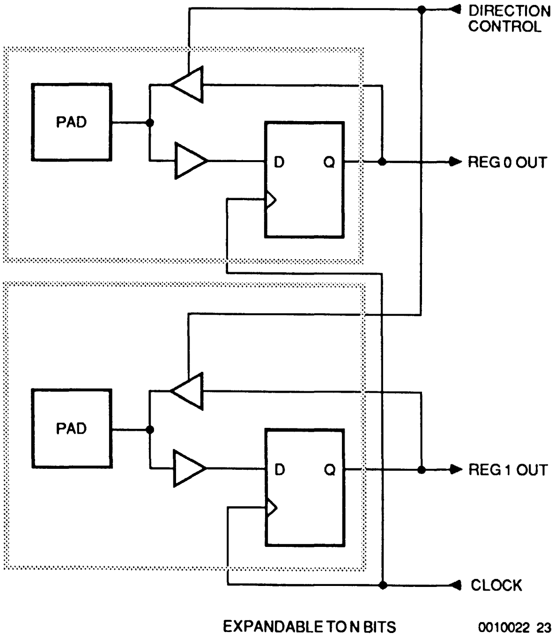
COMMENTS: The I/O clock into each IOB is common to all IOBs along each edge of the die. For best resource utilization, group the storage elements along one edge of the die.

Figure 22. N-Bit Storage Register (IOB-Based Register)

input (I) and output (O) stubs of the IOB are connected together. The three-state control line (T) controls the direction of data flow (T = LOW for a read operation by the external device, T = HIGH for a write operation to the LCA). Typically, the read/write control line (three-state control) also originates external to the LCA, and would come in through an additional I/O block.

The designer should be aware that the input register data can be read from the LCA from the read/write register but data cannot be written to the LCA. Writing the register from inside the LCA 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:



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 utilization, group the storage elements along one edge of the die.

The three-state control line controls the direction of data flow (T = LOW for read, T = HIGH for write). This control line will typically originate off chip and come in through an additional IOB.

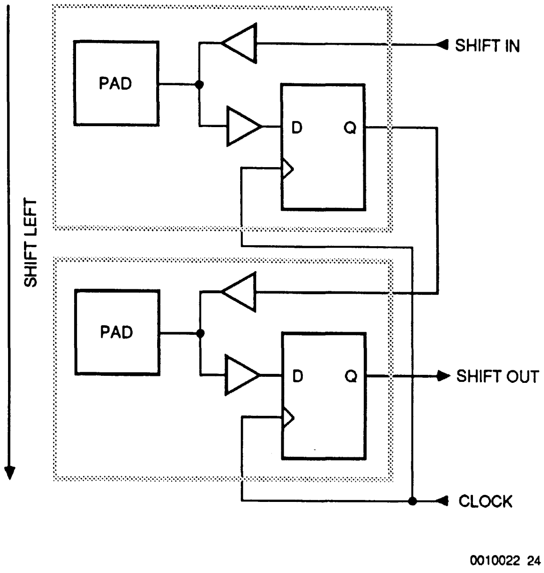
Figure 23. N-Bit Read/Write Storage Register

Shift Registers

Shift registers are also easily constructed with IOBs by feeding the input (I) of one IOB to the output (O) of the next IOB. The figures below describe two shift registers—one that shifts to the left as shown in Figure 24, and one that shifts to the right as shown in Figure 25. The shift direction of the register depends on how the inputs and outputs of the IOBs are connected.

Since the I/O clock line of an IOB is common to all IOBs along each edge of the die, the register is implemented with IOBs along one edge of the die.

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 input (I) of a given IOB goes to the output (O) of the IOB on the left (shift left).

Figure 24. Shift Left Register (IOB-Based Register)

Johnson Counters

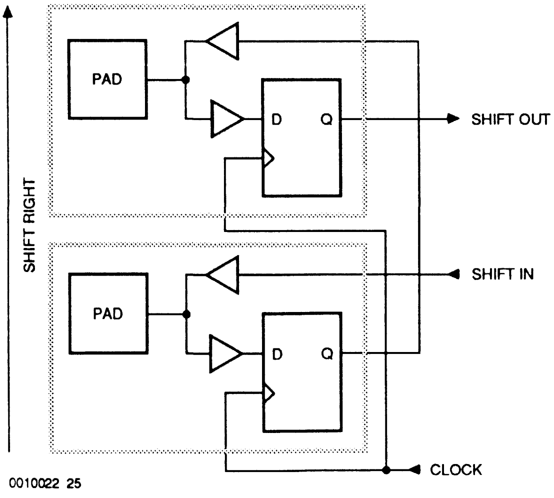
An n-bit Johnson counter will count to 2n states as opposed to standard binary counters which count to 2^n possible states. Johnson counters have a variety of possible uses in a digital design, including low modulo counters and glitch-free decoders.

In IOB implementations, Johnson counters can be thought of as special shift registers. Only one bit changes during a state transition, as shown in Table 1 for a three-bit Johnson counter.

000
100
110
111
011
001
000

Table 1. Transitions of a Three-Bit Johnson Counter

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 input (I) of a given IOB goes to the output (O) of the IOB on the right (shift right).

Figure 25. Shift Right Register (IOB-Based Register)

A Johnson counter built from unused IOBs requires at least one Configurable Logic Block (CLB) to perform an inversion. The Johnson counter is automatically reset to an all zeroes state upon configuration or on a RESET pulse.

The application note *Counter Examples* contains more information on using Johnson counters.

Glitchless Johnson Decoder

A glitch-free decoder can be built using IOBs and CLBs. The decoder will be glitch-free since 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 ANDing just two of the appropriate counter bits. Counters of various modulo and duty-cycle can be extracted as well. For example, Figure 27 shows the schematic implementation of a Johnson counter decoder with various two-input decode states.

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 with feedback of appropriate bits back to the first bit. An LFSR requires some logic function in the feedback path, usually a XOR (exclusive-OR) function.

LFSRs have numerous applications. One example is described in the application note *A UART Design Example.* In the UART example, LFSRs are used to implement the encryption and decryption functions. The application note *Counter Examples* contains more information on using Linear Feedback Shift Registers in general applications. This application note addresses using LFSRs in IOBs.

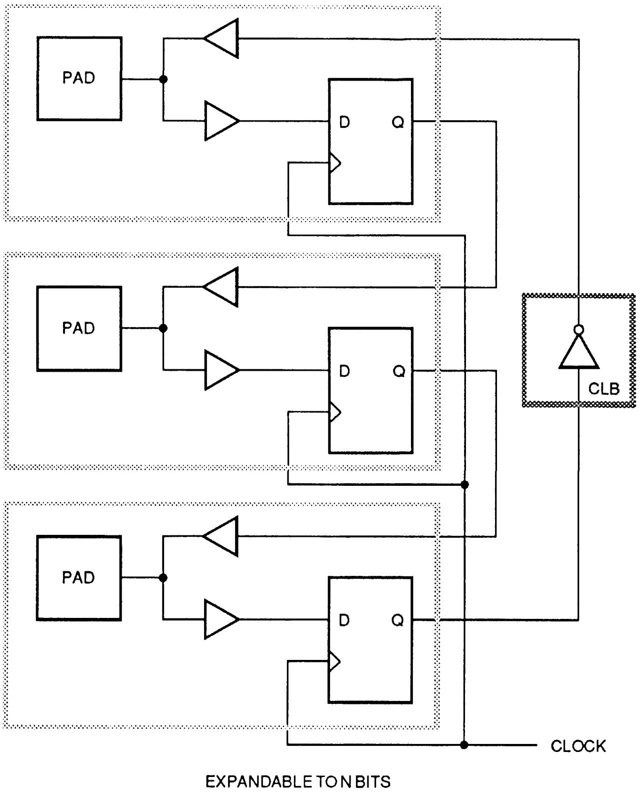
Figure 28 shows the schematic for a three-bit LFSR which implements a modulo 5 (divide by five) counter. An n-bit LFSR counter can produce a pseudorandom sequence of up to 2^n-1 unique states. By adding logic to the feedback path, the LFSR counter can be forced to skip any number of states (from one to 2^n-1). By forcing the counter to skip m states, a LFSR counter can implement any modulo as described in Equation 13.

$$\text{MODULO} = (2^n-1) - m \qquad [13]$$

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

Figure 29 shows the counting sequence for a three-bit LFSR counter with exclusive-NOR (XNOR). All of the possible "skip" paths are indicated. The "stuck" state is also shown.

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



0010022 26

CONFIGURATION:
All IOBs
I:Q
BUF:ON

Figure 26. N-Bit Johnson Counter (IOB-Based)

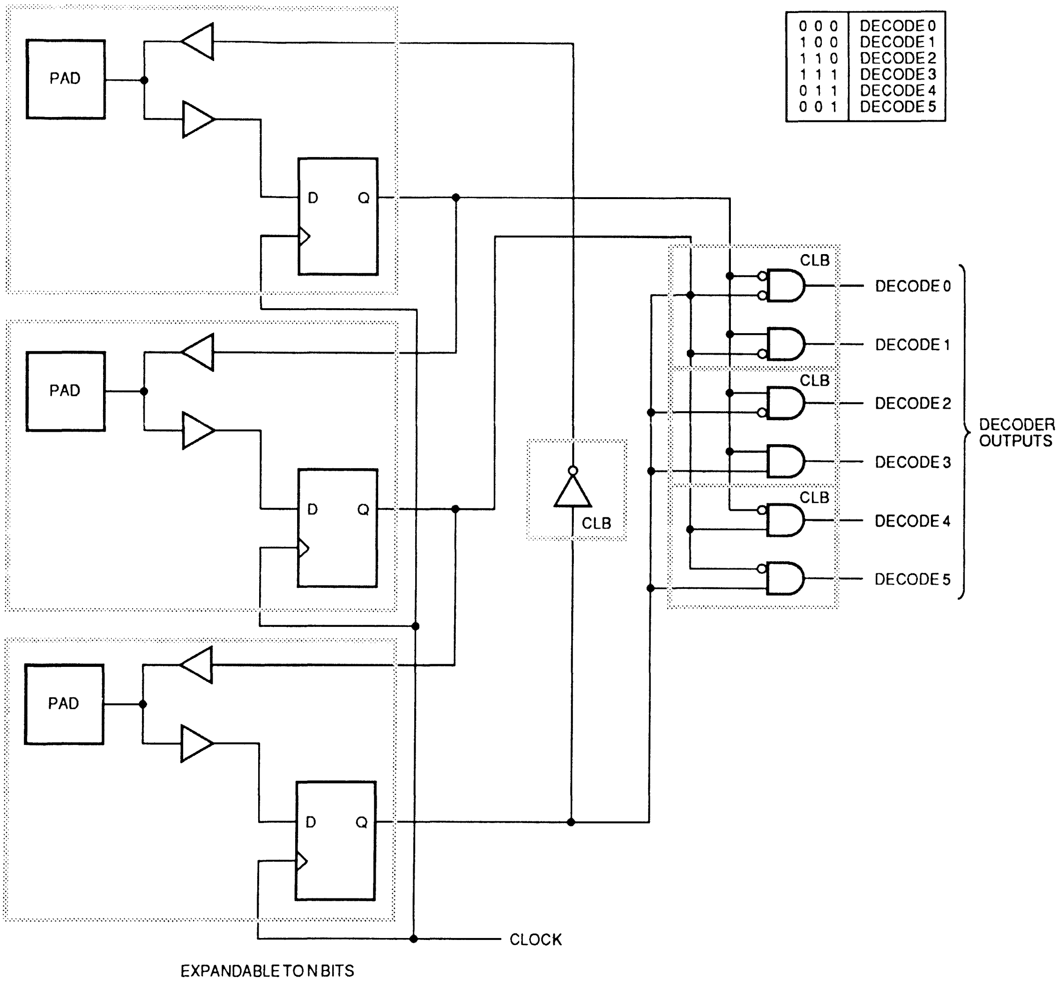
In the counting sequence, note that there are two counter states where only the first bits differ (for example, locate the states 101 and 001). By forcing the feedback logic to invert the sense of the feedback into the first bit, the counter can be forced to "skip" all of the states between the two indicated values. This can be accomplished by decoding (ANDing) the state just previous to the state to be skipped. Again using the modulo 5 counter as an example, locate the initial value that will allow the counter to skip two states (i.e. 101). By decoding the state 011 (the state just prior to the initial skip state, 101), the sense of the feedback into the first is inverted. The counter skips from state 101 to state 001 implementing a modulo 5 counter. Using this method and the proper feedback into the register, a counter of any modulo from one to 2^n–1 can be built.

Upon configuration or upon an externally driven $\overline{\text{RESET}}$ signal, all of the storage elements used in the LFSR counter will be reset to zero.

The designer should be careful to avoid the "stuck" state. The "stuck" state is the state missing from the 2^n-1 counting sequence (if the "stuck" state were included, the LFSR counter could have 2^n possible states). This state occurs when the feedback path forces the counter into an ever-repeating single state. As a simple example, assume that a LFSR counter were built with a two-input exclusive-OR feedback path as shown in Figure 30. Upon configuration or external $\overline{\text{RESET}}$, the counter would begin operation in the all zeroes state (000) and would be "stuck" in that state due to the type of feedback used.

An interesting thing occurs when all but 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 Figure 31.

Longer LFSR counters with higher possible modulos and more complex feedback mechanisms can be built but their discussion is well beyond the scope of this application note. However, Table 2 presents some of



0010022 27

Figure 27. Johnson Counter Decoder. Any state of a Johnson counter can be decoded, glitch-free, with only a two-input logic function.

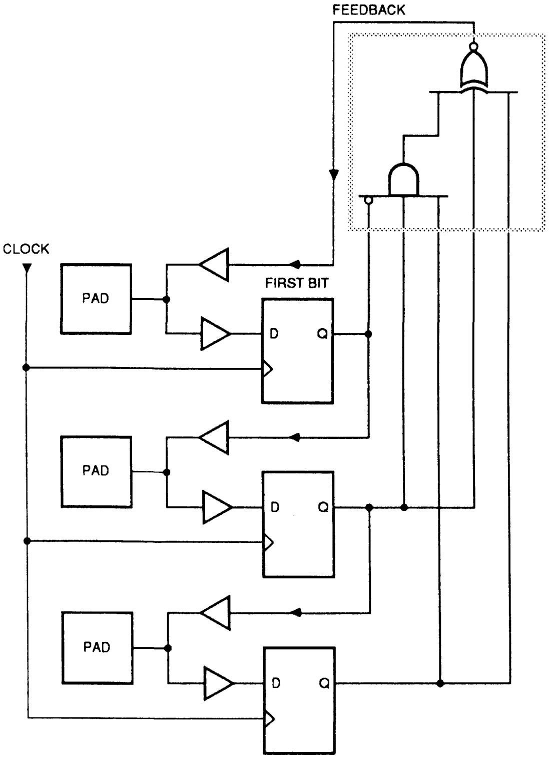
the possible feedback combinations for LFSR counters of three bits to ten bits.

(2^n-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

Table 2. I/O Block

Current Drive			
	4 mA	6mA	8mA
Voh Vol	3.86 V 0.32 V	3.54 V 0.48 V	3.22 V 0.64 V

Table 3. Output Current and Output Voltage Levels for an IOB



00100252 28

Figure 28. Modulo 5 LFSR Counter

Enhanced Output Source/Sink Current

Logic Cell Arrays are specified to have 4 mA worse case source and sink capabilities at Vol = 0.32 Volts and Voh = 3.68 Volts. Increased drive current can be obtained at the cost of decreased voltage margins. For example, Table 3 illustrates the effect on Vol and Voh by increasing the drive current through a single IOB.

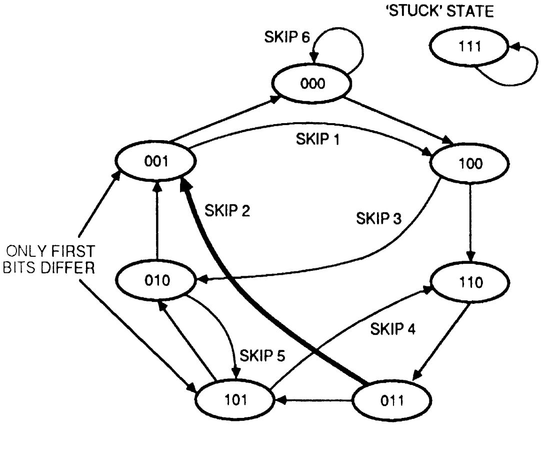
An alternative method of increasing the drive current is to parallel the output drivers of two IOBs. Paralleling two outputs will enable the IOBs to source and sink double the current (worst-case) at no reduction in voltage margins. This method is schematically diagrammed in Figure 32.

One caution to the designer, however. The designer should minimize the difference in routing delay between the two IOBs connected in parallel. Excessive delays may cause output contentions.

SUMMARY

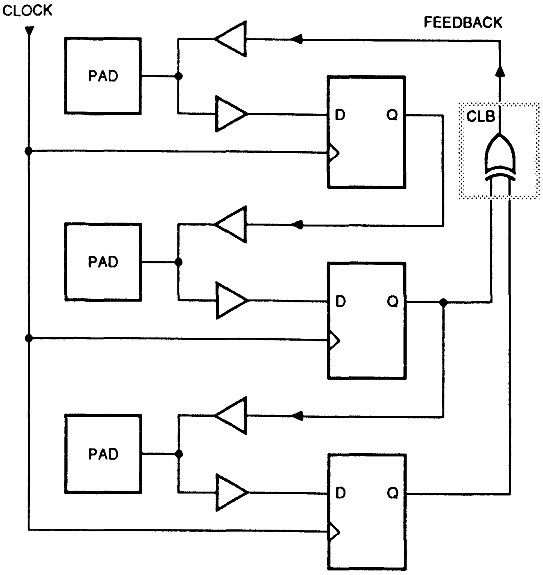
The input and output resources of a Logic Cell Array (LCA) can be used for a variety of logic structures because of the flexibility of the LCA architecture. This application note described how to implement various I/O structures including bidirectional lines, open-drain outputs, and Schmitt-triggered inputs.

Other structures such as oscillators, multiplexers, shift



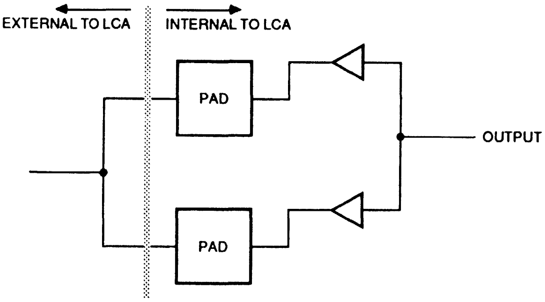
0010022 29

Figure 29. Three-bit LFSR Counting Sequence



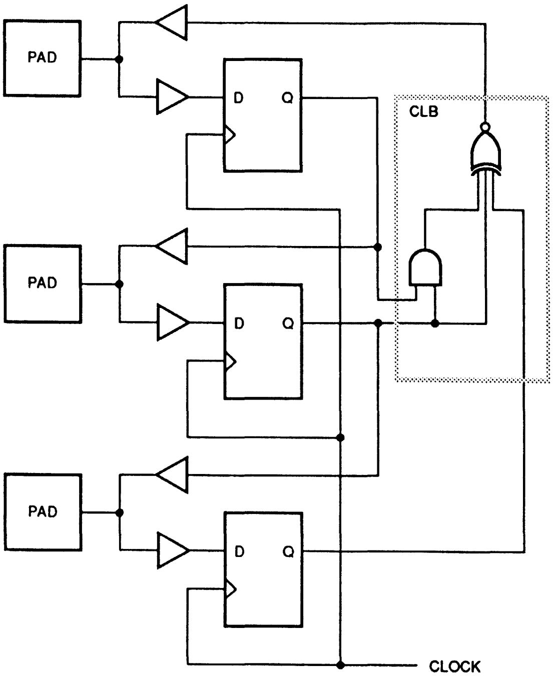
0010022 30

Figure 30. Simple LFSR With "Stuck" State. A simple LFSR counter will be "stuck" in state 000 after configuration since all registers were originally reset.



0010022 32

Figure 32. Parallel Outputs Have Increased Drive Capability



0010022 31

Figure 31. By ANDing all but the last bit of the "stuck" state and using this value in the feedback path, an LFSR can be forced to count to 2^n possible states instead of the 2^n-1 states usually associated with an LFSR counter.

registers, Johnson counters, decoders, and linear feedback shift registers were also described. These resources can only be effectively implemented in flexible, array-type architectures such as found in gate arrays and the Xilinx Logic Cell Array.

TECHNICAL SOURCES

[1] *Schmitt Trigger Using PLS153 and PLS159,* Signetics Programmable Logic Data Manual, 1986. pp. 9-110 to 9-119.

Placement and Routing Optimization

Table of Contents

Placement and Routing Optimization

INTRODUCTION

As with any high density ASIC device, the Logic Cell™ Array offers alternatives in placement and routing which can affect the utilization and performance of the final design. In gate arrays and other factory programmed solutions, these alternatives must be investigated through simulation. The Logic Cell Array allows them to be seen and modified at design time through the capabilities of the XACT™ Development System. Additionally, system performance and function can be easily verified with in-circuit emulation.

Within XACT, several powerful capabilities are included to allow easy "tuning" of the design for performance or resource utilization control. The purpose of this application note is to investigate the methods and XACT-related operations which can be employed by the designer. In addition, this should serve as a primer to help new users to learn skills for effectively using Logic Cell Arrays. The topics addressed include:

- Placement—Physical assignment of logic elements.
- Routing—Utilization of the available resources.
- Delay Calculator—Delay calculation and interpretation.
- Macros— Macro usage and placement and performance considerations.
- Multi-block techniques—Placement and routing consideration when designing with larger collections of blocks.

HOW TO USE THIS APPLICATION NOTE

Users who are not familiar with placement and routing in the Logic Cell Array are advised to read all of this application note. Users who have some level of knowledge about the Logic Cell Array, and in particular have completed some design work, may wish to study only the sections of interest.

PLACEMENT

As with any SSI/MSI circuit board or gate array device, the placement of logic within the Logic Cell Array can be modified to affect resource utilization and performance. Placements and associated routing are interrelated, in that changes in placement can change the routability and consequently performance. The choice of routing to be used will affect the placement choices available. The configurability of the Logic Cell Array provides a great deal of design flexibility. To utilize this flexibility effectively, users need to understand the trade-offs and capabilities of these resources. Several of these resources are discussed here.

Long Lines

Long lines are continuous metal segments which span the width or height of the device to provide minimum-delay long distance signal paths. Although long lines will be used by the automatic router for a general signal when other resources are not available, it is best to direct the use of long lines for specific functions. This insures that they are most efficiently used, based on consideration of their capabilities and interconnection potential. Refer to Figure 1 for the locations of the long lines.

Signals which can most effectively use long lines are generally classified as data distribution or low-skew control. Whether originating at an I/O block or a Configurable Logic Block (CLB), data signals typically have several destinations, each of which uses the data in a different fashion. Following the natural data flow of the device, these signals can be best routed on long lines with one bit per row or column. One consideration in data routing is the direction of the data flow. In the 2064/2018 series of Logic Cell Arrays, internal signals must be unidirectional. For systems which have bidirectional data paths, a pair of long lines in each column can be used to carry input data and output data respectively. This requires that the data input/output pins be located at the top or bottom of the device. Figure 2 shows an eight-bit bidirectional data bus with vertical routing on pairs of long lines.

Control signals such as clocks, reset/set controls, count or shift direction controls, etc. may have critical timing requirements between their source and their multiple destinations. Skew must be controlled to insure that each receiving block performs the desired function at the same time or on the same clock edge. Destination blocks should be arranged in a single column or row if

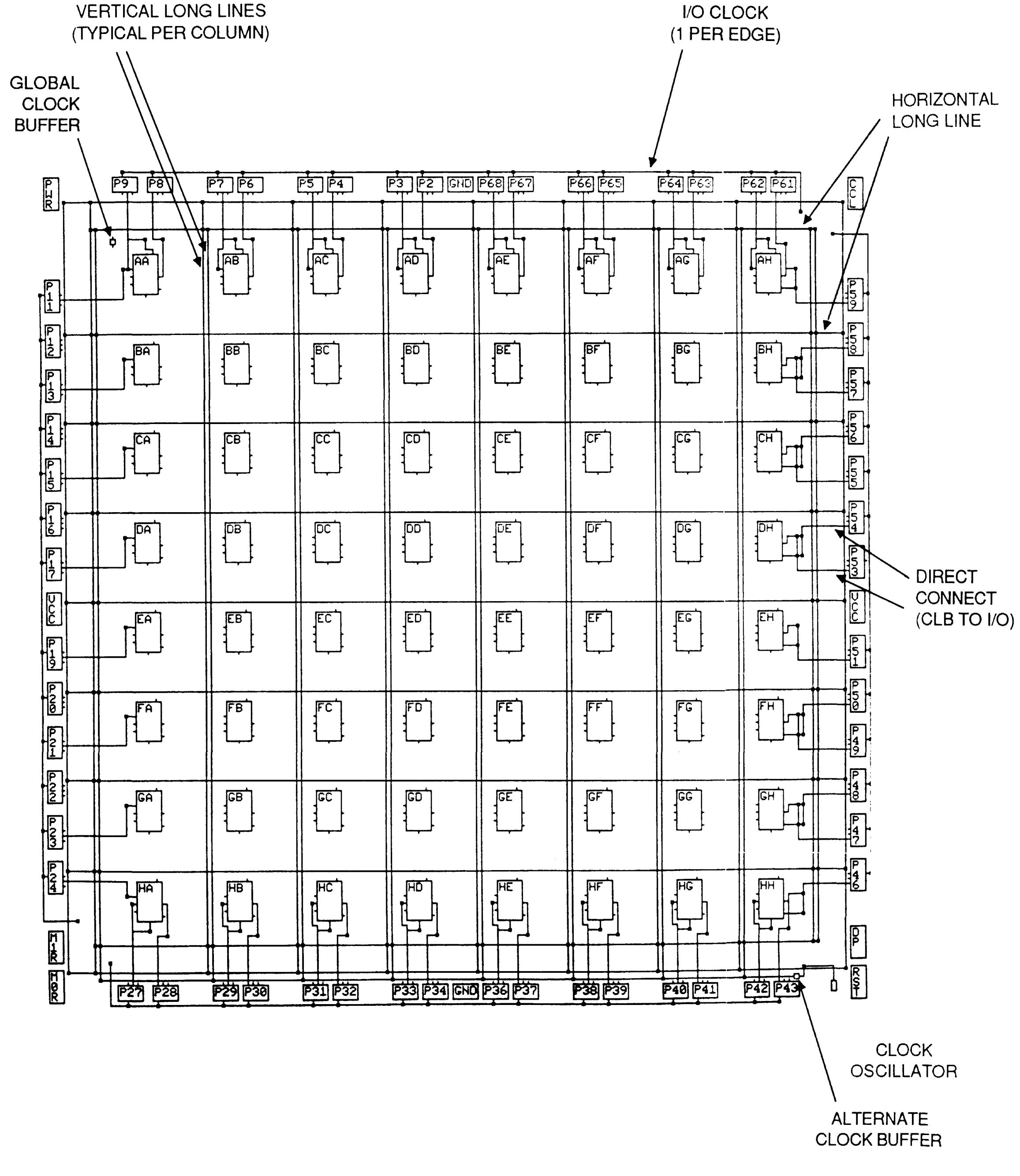


Figure 1. XC2064 LCA Overview

possible and the control function routed onto the appropriate long line. Figure 3 shows two alternative implementations of a reset function generated in a logic block and routed to four destination blocks. The skew reduction associated with the use of the long line can be seen from the accompanying table.

Clock Buffers

The Logic Cell Array has on-chip, special purpose buffers to provide high-fanout, low-skew signal distribution. These buffers are normally used for clock signals, but can be used for any general purpose signal which requires high-fanout or low-skew routing to multiple blocks. Clock buffers are associated with specific long line resources for routing on a column basis. The global buffer (upper left corner) directly drives a long line in each column. The alternate buffer (lower right corner) drives a horizontal line which can be selectively connected to a long line in each column.

In systems which have a single common clock for all the state elements, that clock can be best distributed 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 can use the global buffer and the other may use the alternate buffer, particularly if one clock is derived from the other. When the crystal oscillator is being used, its output drives the

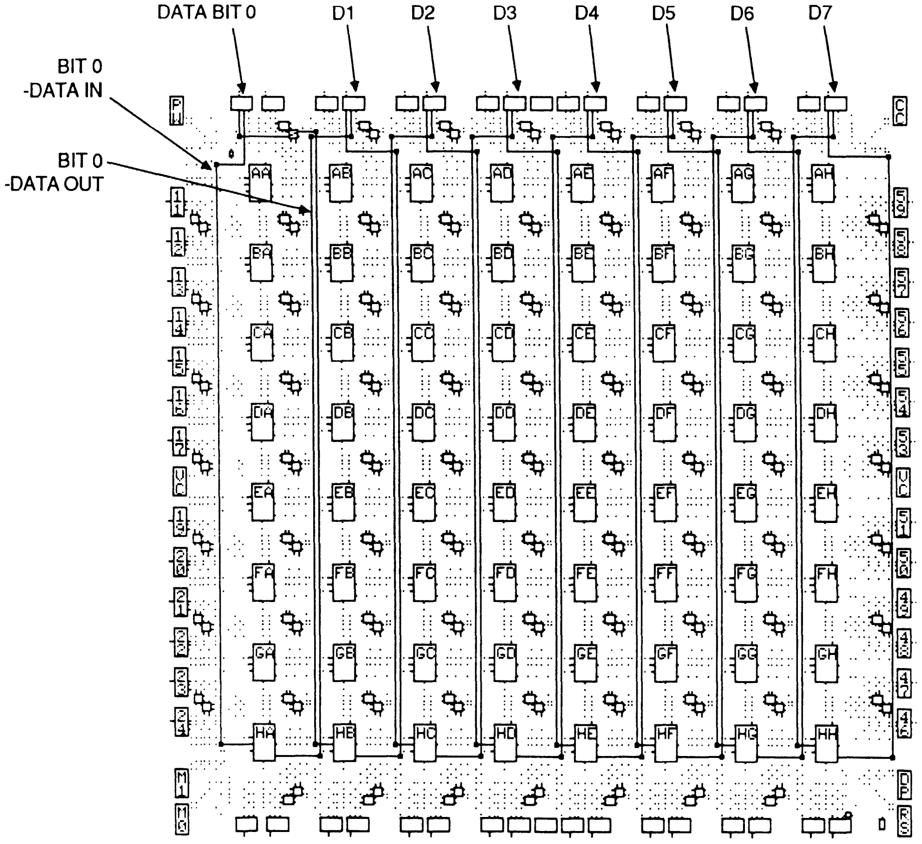


Figure 2. Bidirectional Data Bus Using Long Lines

```
Delay: bidibus8.lca, XACT 1.21

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 ROUTING

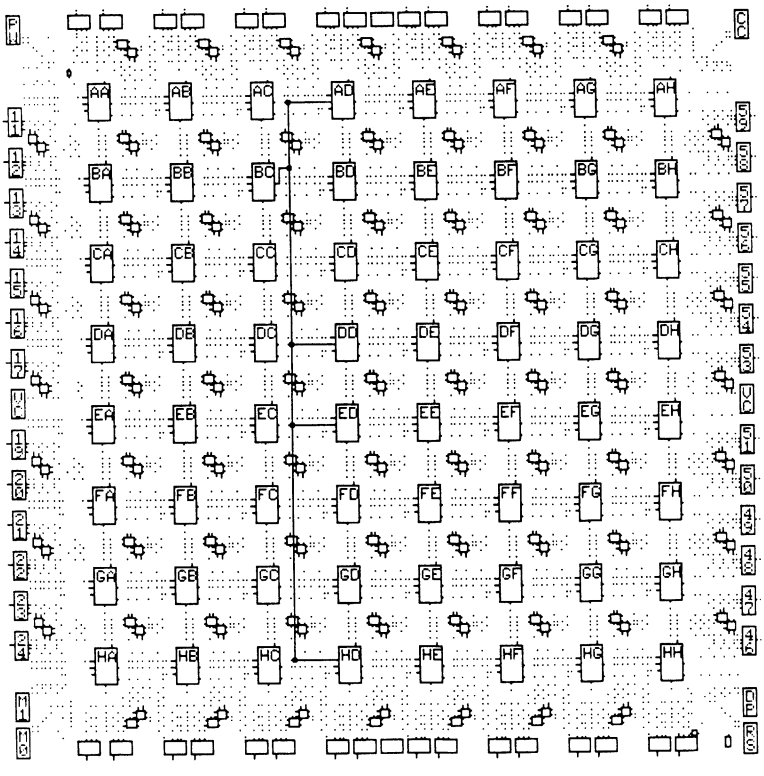


Figure 3a. Signal Routed Via General Interconnect

```
Delay: bidibus8.lca, XACT 1.21

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

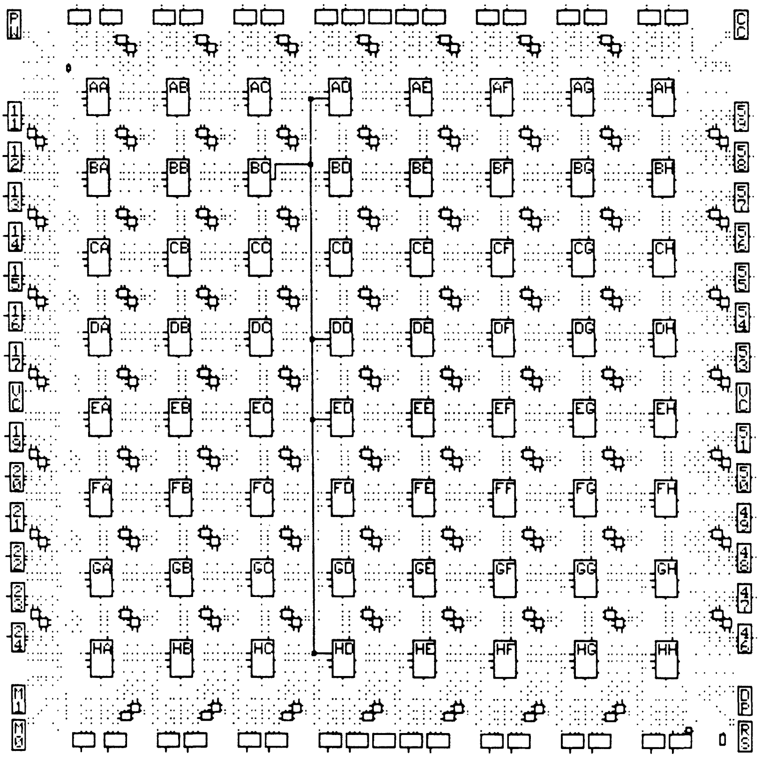
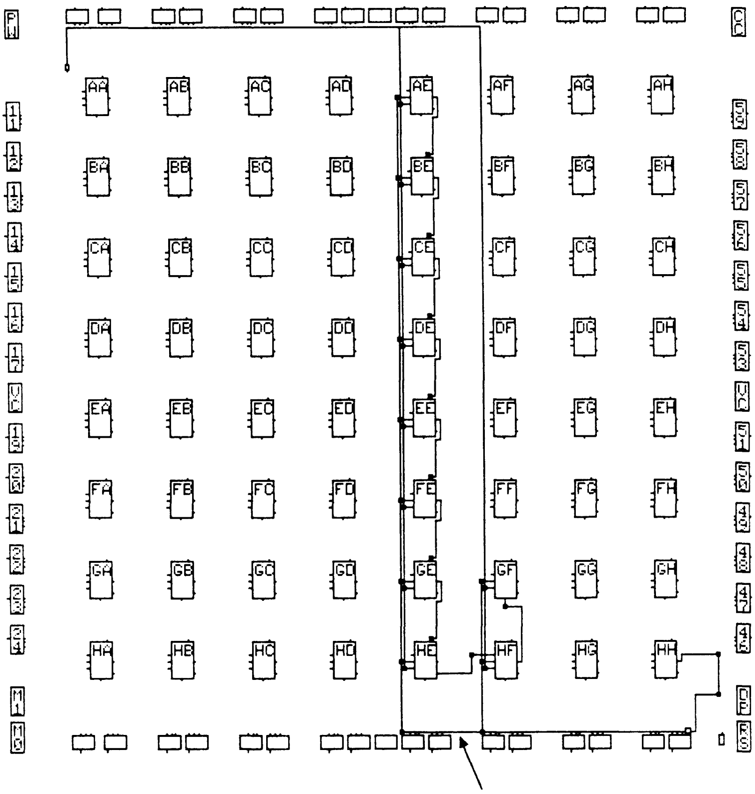


Figure 3b. Signal Routing Via Long Line

alternate buffer directly, and may therefore be the primary clock for the system. In any case, the predominantly used clock should be driven with the global buffer.

For systems which require more than two clocks, the primary clock should be driven with the global buffer. Other clocks are best routed onto vertical long lines, and their respective CLBs arranged in columns adjacent to the long line carrying the appropriate clock signal. These column-oriented clocks can be driven either from an adjacent (to the left) CLB or an I/O block on the edge at either end of the column, using direct connect capability. 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, the former long line should be chosen.

Another use of the buffers is for routing a control signal to many CLBs. By placing the source of the signal near the alternate buffer, a low-delay path can be provided from the source to the buffer, and then out to all of the receiving CLBs or I/O blocks. Figure 4 shows a shift register which has been placed and routed using both buffers. The global buffer is used for the overall shift register clock, while the alternate buffer is used to provide a low-skew control for shift/hold control. If the hold control logic timing is not well controlled, skews between the control signal, as seen by the blocks, could cause a partial shift. Some blocks could get the signal but others may not get it in time to hold relative to the next clock edge. This becomes less of a factor in choosing routing for control signals as the timing



SHIFT/HOLD CONTROL FOR
SHIFT REGISTER

Figure 4. Control Signal Using Alternate Buffer

constraints are relaxed.

Direct Connects

Each CLB and IOB has capabilities to connect directly to other adjacent blocks as shown in Figure 5. A direct connect provides a signal path which has virtually zero delay and does not use any of the general interconnect or long line resources. For maximum performance and minimum routing resource impact, use of direct connects is a primary objective. Because direct connects exist primarily in vertical or left to right directions, blocks which represent stages in a process should be arranged 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-in or direct-out paths.

As an example, consider the circuit of Figure 6. The data byte is to be loaded into the shift register in a parallel fashion and then shifted out with each clock cycle. Figure 7 shows two alternative implementations of this circuit. In implementation 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 interconnect providing many of the signal paths. Implementation B uses direct connects exclusively, providing zero-delay paths from block to block and allowing higher performance. In this design, the worst case delays (shown in Figure 7c) show that the maximum load and shift clock rate is 17.5, 31.3, and 43.5 for the 20, 33 and 50 MHz devices respectively. Note also that the general interconnect in this implementation is available for other uses. The LOAD signal has been routed on a long line driven from a directly-connected I/O block.

In non-synchronous intensive designs, such as commonly found in glue logic replacement, direct connects normally cannot be exploited to the degree shown in this example. However, whenever possible, a

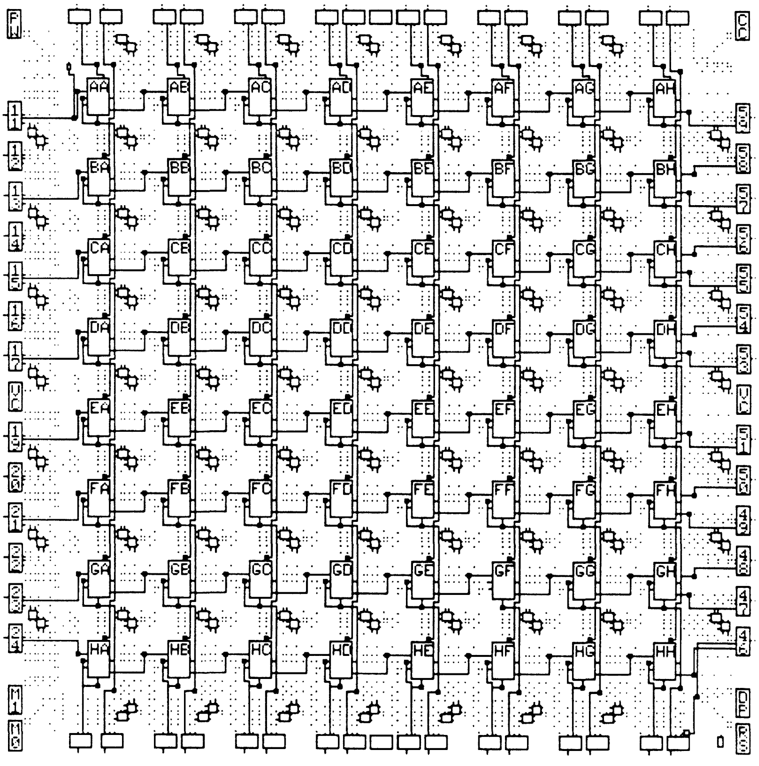


Figure 5. Direct Connect Resources

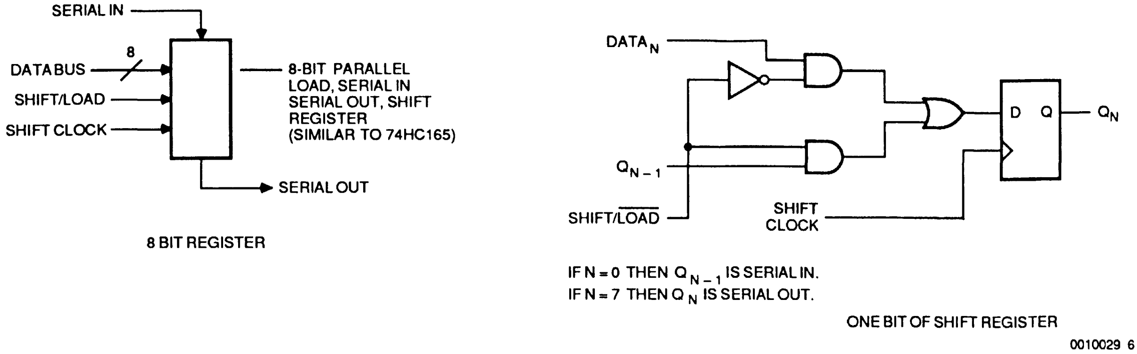
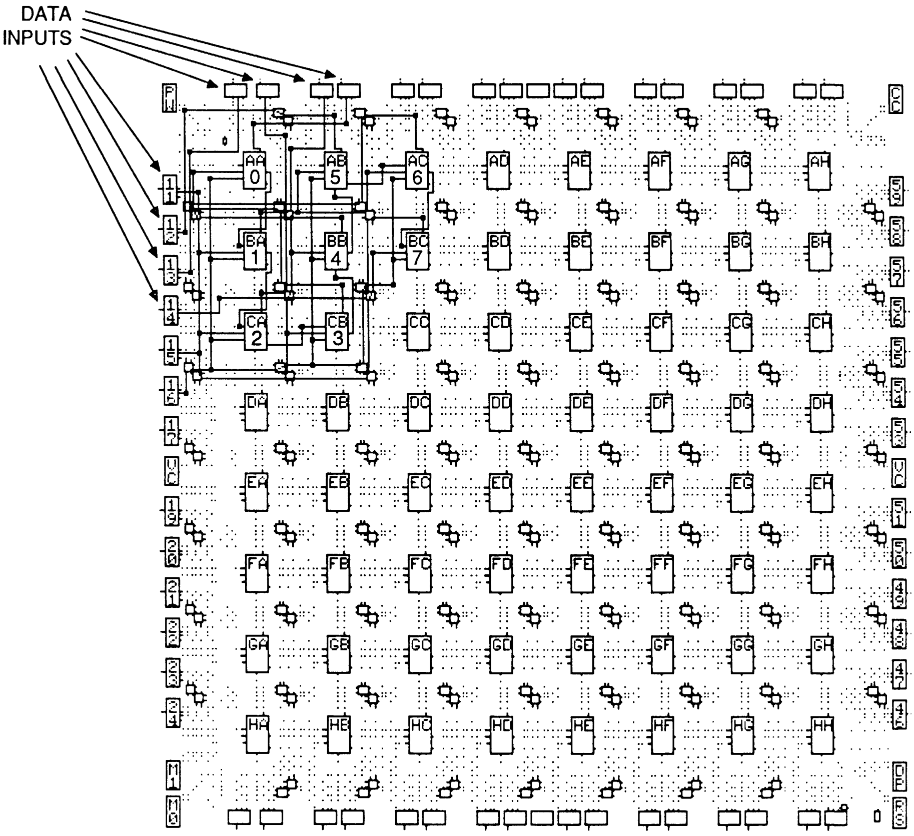

SERIAL IN
8
DATABUS
SHIFT/LOAD
SHIFT CLOCK
8-BIT PARALLEL LOAD, SERIAL IN SERIAL OUT, SHIFT REGISTER (SIMILAR TO 74HC165)
SERIAL OUT
8 BIT REGISTER
$DATA_N$
Q_{N-1}
$\overline{SHIFT/LOAD}$
SHIFT CLOCK
D Q
Q_N
IF N = 0 THEN Q_{N-1} IS SERIAL IN.
IF N = 7 THEN Q_N IS SERIAL OUT.
ONE BIT OF SHIFT REGISTER
0010029 6

Figure 6. 8-Bit Parallel Load Shift Register



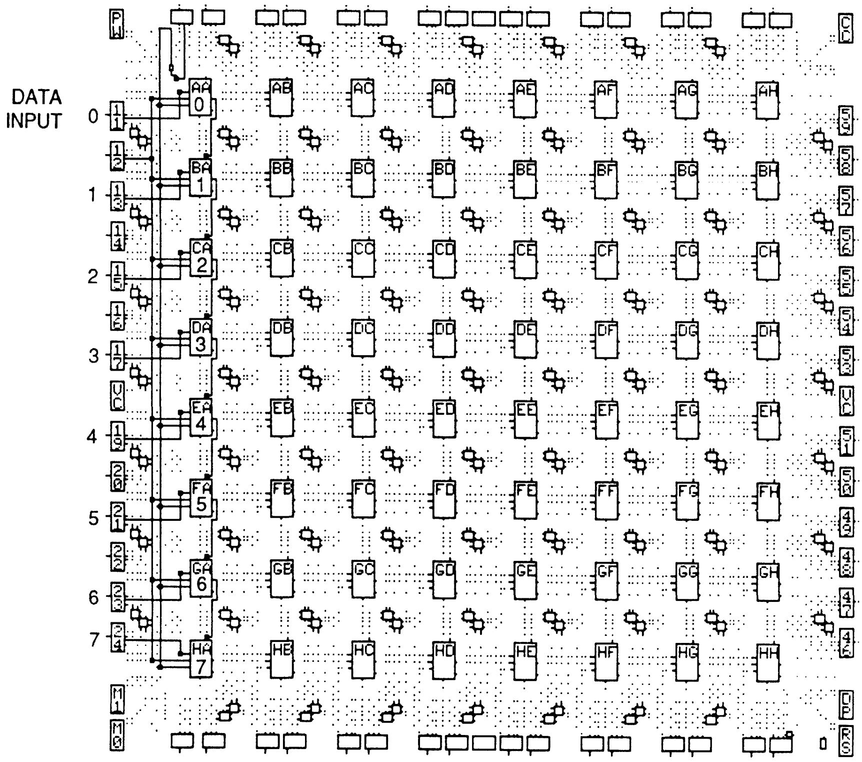
ROUTING CONGESTION CAUSED BY PLACEMENT CHOICE

Figure 7a. 8-Bit Parallel Load Shift Register signal should use direct connect from one block to another in the Logic Cell Array. Direct connect considerations should be a primary factor in block placement. For each connection done with direct connect, a general interconnect resource is released for use in implementing some other function. Experience indicates that extensive use of direct connect can boost the logic utilization of the Logic Cell Array by as much as 30%.

Data Flow

Data flow describes the process of evaluating the sequential nature of logic to be implemented in making placement and routing decisions. An examination of the data flow should provide some insight into which signals are most effectively placed on long lines and direct connects, as well as guidelines in the placement of required logic blocks. In general, data processing in the Logic Cell Array flows most naturally either from left to right or vertically. Flow up and flow down are virtually identical.

To illustrate data flow analysis, consider the block diagram of Figure 8. This example is a dual-ported memory interface used as a high speed serializer, as might be found in a video pixel processing or serial communications application. Both the serial data output device and the microprocessor must have access to the memory. In looking at the data flow, there must be an eight bit path to and from the memory to the microprocessor, as well as an eight bit path from the memory to the serializer. The serializer requires an eight-bit parallel-to-serial data flow. The addresses to the memory are generated internally for the serialization process, and are supplied externally by the microprocessor. They are always outputs to the external memory. In summary, the data flow paths consist of



ALL DIRECT CONNECT USED

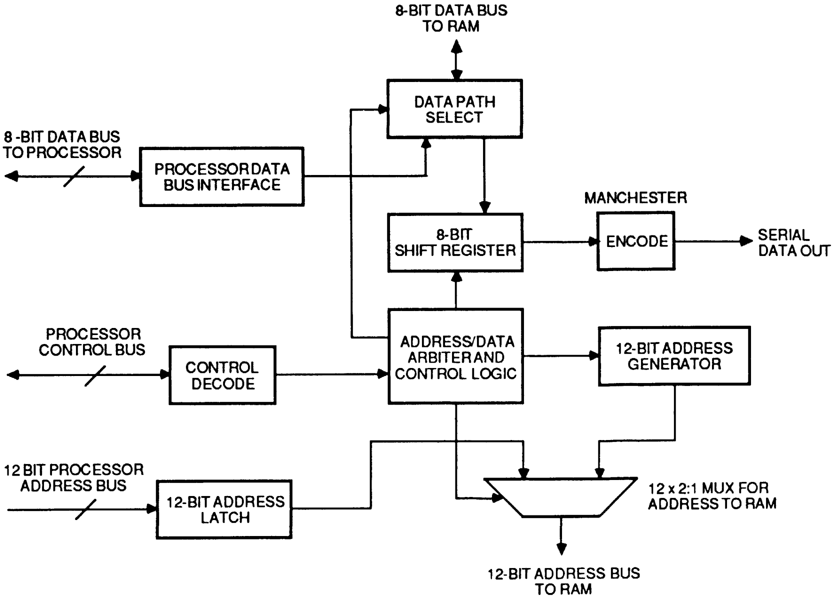
Figure 7a. 8-Bit Parallel Load Shift Register

```
Delays for 8 bit shift register with parallel load

     For -1 (20 MHz) speed device
                :
                :
                :
From: BLK GA          (CLOCK to GA.X )  :   35ns ( 35ns)
Thru: NET S6          (GA.X  to HA.A )  :    0ns ( 35ns)
  To: BLK HA          (HA.A  to SETUP)  :   22ns ( 57ns)*
                :
                :
     For -2 (33 MHz) speed device
                :
                :
                :
From: BLK GA          (CLOCK to GA.X )  :   20ns ( 20ns)
Thru: NET S6          (GA.X  to HA.A )  :    0ns ( 20ns)
  To: BLK HA          (HA.A  to SETUP)  :   12ns ( 32ns)*
                :
                :
     For -50 (50 MHz) speed device
                :
                :
                :
From: BLK GA          (CLOCK to GA.X )  :   15ns ( 15ns)
Thru: NET S6          (GA.X  to HA.A )  :    0ns ( 15ns)
  To: BLK HA          (HA.A  to SETUP)  :    8ns ( 23ns)*
                :
                :
```

* TOTAL CLOCK TO CLOCK DELAY-WORST CASE

Figure 7c. Delay Report For Direct Connect Placement



0010029 8

Figure 8. Serializer Block Diagram

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

In examining these, there is a requirement for a bidirectional data path between the memory and the microprocessor. This same path needs to supply data to the serializer. The serializer may be viewed as a process "perpendicular" to the data flow because it takes parallel data and serializes it. The memory address path is wider, but is unidirectional, with a common "connection" only at the output point. Figure 9 shows a flow analysis of this design example.

Based on the flow analysis of Figure 9, the eight-bit data path should be run vertically to take advantage of direct connect in both up and down directions. The serializer, which could use direct connect in either left to right or vertical orientation, can be placed perpendicular to the vertical data path orientation, and can use the direct left-to-right capabilities. Since it is unidirectional, the address path will be routed with direct connect where possible between CLBs near the edge of the device and adjacent I/O blocks driving the address lines. These general guidelines can be used for selection of routing alternatives, as well as placement of the blocks within the device.

LOGIC BLOCK PLACEMENT

One of the most critical elements in achieving an efficient design with the Logic Cell Array is the proper placement of the logic blocks and I/O blocks. Logic block placement is more critical than I/O block placement because it offers more degrees of freedom and the final CLB placement can dictate most IOB placement. Both performance and routability of the design can be improved by proper placement. Good placement will relieve the designer from solving routing related problems, and will generally result in good initial performance. Fine tuning of a design for ultimate performance may affect final placement, while a good initial placement will serve as a sound basis for achieving design completion with a minimum number of placement and routing iterations. Maximizing the use of the direct connections between blocks is an important goal.

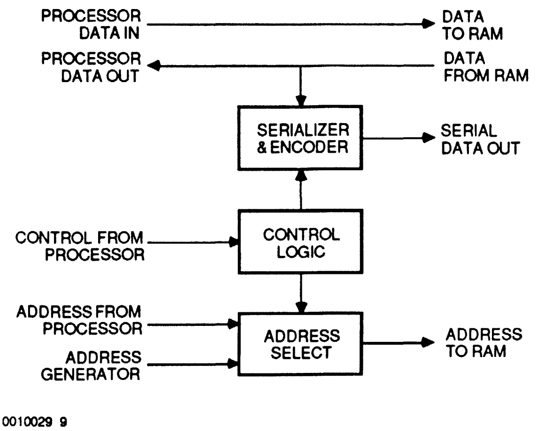


Figure 9. Data Flow Analysis of Serializer

Guidelines for placement of logic blocks can be summarized as:

1. First consider the various functional elements in the design and the shapes that each may take, and their relative interconnection. Try the placement of these functional blocks on a printout of a blank LCA to see how they might fit together. The layout in Figure 10 was obtained using this basic analysis.
2. Inputs and outputs, both internal and external, for each block of logic should be examined. Blocks with a high number of common interconnections should be placed near each other.
3. When considering the relative placements of individual logic blocks and I/O blocks, a key consideration is to utilize the direct connect resources wherever possible.
4. Arrange related groups of logic blocks in rectangular shapes if possible.
5. Place blocks with the greatest number of interconnects to other blocks, both logic and I/O, at the perimeter of rectangular shapes.
6. Use "long" and "thin" shapes only where data is going to flow through the shape to some other logic perpendicular to its long axis.
7. Blocks of control logic or miscellaneous functions which have minimal external I/O are often best placed near the center of the device.
8. Where possible, minimize the number of different clocks in the design, particularly those generated internally. A completely synchronous design with a single clock, using the global clock buffer is ideal.

Many of these recommendations are similar to those applied to layout of printed circuit boards using SSI/MSI devices. The examples in the following section should help illustrate how effective placements can be made.

I/O BLOCK PLACEMENT

I/O block placement is normally dictated by the final placement of the logic blocks which must receive or originate the signals connected to the I/O blocks. However, consideration of placement constraints must be examined, as they can have a significant impact on overall placement and routing. General guidelines for I/O block placement fall into several categories:

1. Locate I/O adjacent to the logic blocks which use the most associated signals.

2. If I/O blocks are being used as busses, special considerations should be made:

   a. Data busses which are to be latched should be located on a single device edge to allow use of the flip-flop in the I/O block, and share the single I/O clock on the edge of the device.

   b. Address busses may be limited to the top of the device if the pins are to be used during configuration as the external EPROM / ROM address lines.

3. Unused I/O blocks can be used as registers for data or shift registers. They must have the I/O clock on that edge of the device available.

4. Care must be exercised in selecting I/O block usage where pins have special functions during configuration. Generally, pins which are inputs during configuration should be used as user inputs,

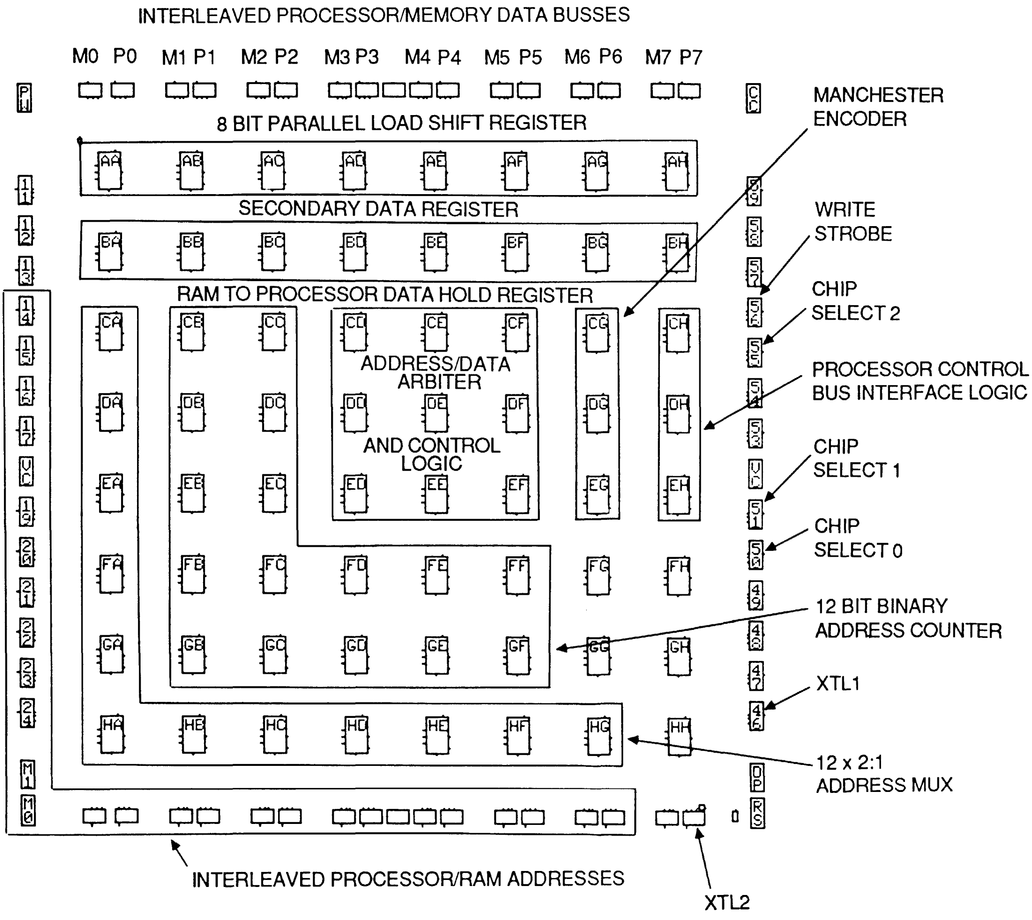


Figure 10. Serializer Placement Plan

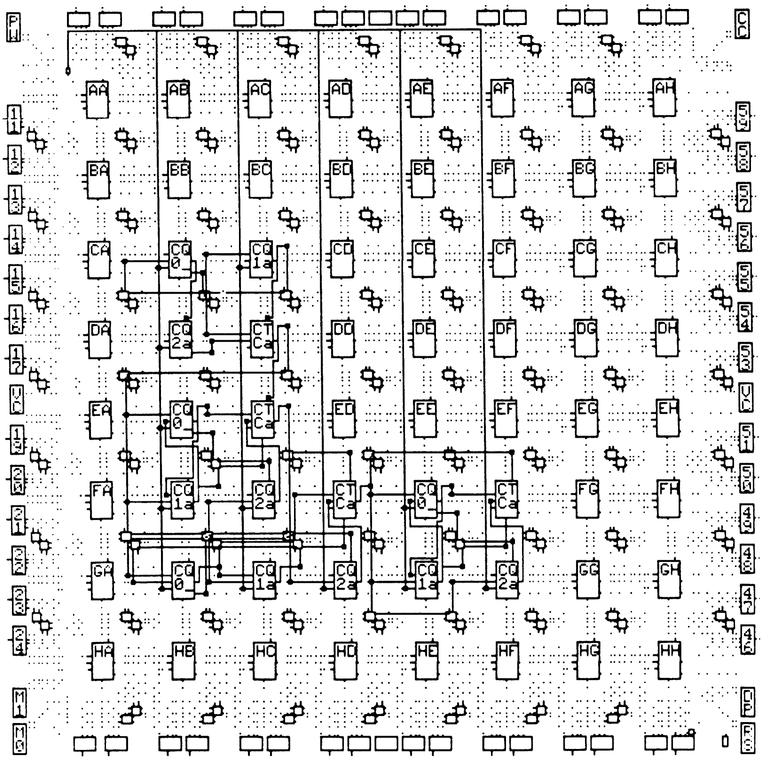
while pins which are outputs during configuration should be used only as user outputs during operation. This generally eliminates contention between configuration use and operation use.

5. If spare I/O blocks are available, multiple I/O connections for a single external signal may greatly improve the routability for that signal. This is particularly true of control inputs which have high internal fanout. Care must be exercised to avoid race conditions or metastability problems when separate inputs are used.

EXAMPLES

The following examples illustrate many of the placement guidelines previously discussed. The sample design used here is the data serializer used for the data flow analysis. Individual elements of the design are used for each topic. Figure 8 shows the overall block diagram, and Figure 10, the LCA layout of the serializer.

1. One method of implementing individual functional blocks is to use macros. The address generator portion of the serializer is shown in Figure 11. This function is a 12-bit binary counter which addresses the external RAM holding the data to be serialized. In generating the counter, macros representing three bits each were used. The placement of the logic blocks in the macro illustrates the desirability of rectangular placements. Figure 12 shows the three-bit macro C8BCR (Counter, modulo 8, Binary sequence, Clock enable and Reset synchronous controls) placed in two ways. Placement A is a linear placement. Placement B uses the recommended placement from the Macro Library reference manual. Notice that in the linear placement, all of the signals must travel the height of the macro to get to the terminal count (CTCname) block. This congests the routing in the columns to the left and right of the column where the macro is located. The rectangular placement shown in B makes the routing more compact and allows additional space for routing



AA AB AC AD AE AF AG AH
BA BB BC BD BE BF BG BH
CA CQ 0 CQ 1a CD CE CF CG CH
DA CQ 2a CT Ca DD DE DF DG DH
EA CQ 0 CT Ca ED EE EF EG EH
FA CQ 1a CQ 2a CT Ca CQ 0 CT Ca FG FH
GA CQ 0 CQ 1a CQ 2a CQ 1a CQ 2a GG GH
HA HB HC HD HE HF HG HH

Figure 11. 12-Bit Address Counter

around the module. In addition, the square structure allows easier placement in a dense design.

2. An example of the "long and thin" approach is the data shift register at the top of the device in Figure 8. Notice how the data comes in from the pins at the top and naturally flows into the blocks of the shift register. If these blocks were placed in a traditional rectangular shape some bits would travel long distances to get to the appropriate shift register block. With this arrangement, the secondary data register in the B row of logic blocks can get the data as it flows through the shift register from the pins.

   The address multiplexer also uses long, thin shapes, with emphasis on the direction of information flow . Figure 13 shows how two different bits of the multiplexer use the direct interconnect paths. Use of direct connect can reduce the congestion in general interconnect and results in an improved placement. The initial placement had each alternate I/O block connected to the memory or processor address bus. After examining the direct interconnect, the position of the processor bus interface I/O relative to the multiplexer block was modified to use direct connect, since it provided an input to each block. Along the bottom, the processor blocks were placed to the left of the memory block for the same bit to take advantage of direct connect for both input and output paths.

3. In some cases, placements may be made which trade off resource utilization for performance. In this design, the primary performance-limiting element will be the speed of the address generator section: the 12 bit counter. In the initial design shown in Figure

PLACEMENT B
LOCAL ROUTING DOES NOT
BLOCK OTHER ROUTES

PLACEMENT A
ROUTING CONGESTION MAKES OTHER
VERTICAL ROUTES DIFICULT

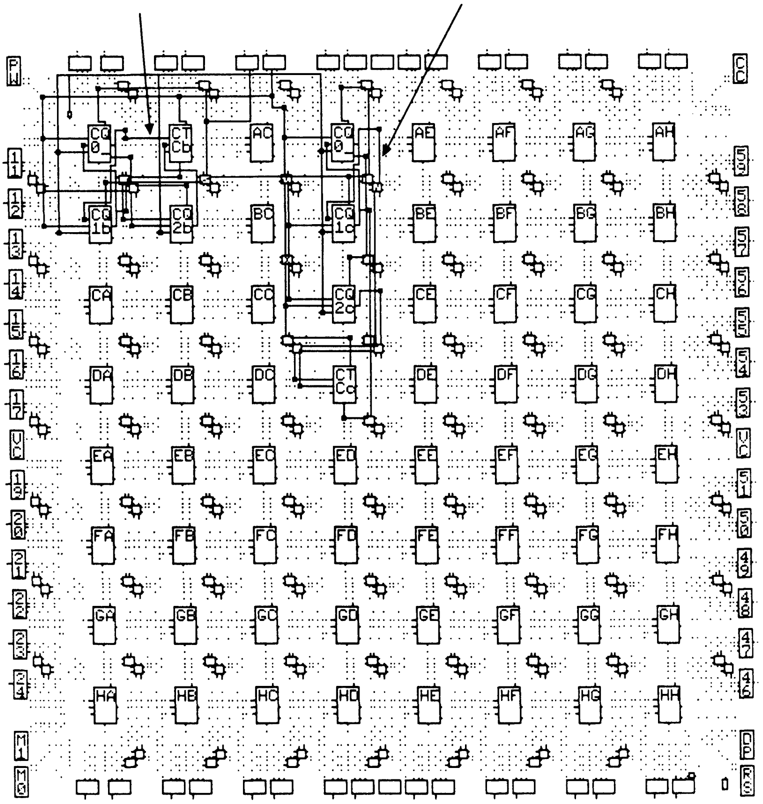


Figure 12. 3-Bit Counter Macro (8BCR) Placement Alternatives

10, macros were used to build the counter. These allow rapid implementation of the function, but may not provide optimum performance. Each three-bit section can operate at high speed because there is only a single logic function required between clock edges. Additional logic block delays are inserted between each three-bit stage, which reduces the overall performance. To obtain higher performance, the toggle condition of each bit in the counter would be generated in the minimum number of logic levels. This would require approximately 18 blocks and much more care in placement and routing. See the application note *Counter Examples* for more information on counter implementations.

MODIFYING PLACEMENT

If initial placement of logic blocks and I/O blocks fails to produce a design which can be readily routed, placement modifications can be made. Some guidelines for modifying the placement include:

1. If congestion exists in the middle of a placement, move the blocks interior to the area to the outside and move the outside blocks inside. This "turn it inside out" concept will normally alleviate congestion except for those cases where the original exterior blocks have a large number of connections to resources outside of the area being examined. Figure 14 shows a block of logic which has interior congestion, and an alternative placement which relieves the congestion.

2. Spreading I/O connections out, rather than clustering them together, will often relieve congestion near the edge of the device. Some I/O intensive applications can benefit from interleaving related I/O blocks as was done in the address bus area in Figure 13.

3. Groups of logic which exhibit routing congestion problems in horizontal directions may be better placed so that the majority of signals are vertically oriented. Remember, there are effectively 10 vertical connections in each column (5 general purpose, 3 long lines and one direct connect to block above and below) and only 6 horizontal connections in each row.

4. Move data register functions out of the logic block area and utilize unused I/O blocks to perform that function. This is particularly effective for function control registers written with an external data bus. Pins adjacent to the data bus input pins can be used for direct data input connections.

ROUTING

Routing resources are comprised of the general purpose interconnects, the long lines and the direct connections from a block to the adjacent blocks. The use of routing resources must balance the partitioning of the logic and the block placement in generating an effective completed design.

Manual Editing

In some designs it will be necessary for the user to interact in the routing process. This may be necessary to; a) relieve congestion to allow a signal to route, b) force use of selected resources for performance or utilization, or c) modify existing routes to "tune" delays for a particular requirement. EDITNET is the XACT command used to perform manual routing of signals or nets.

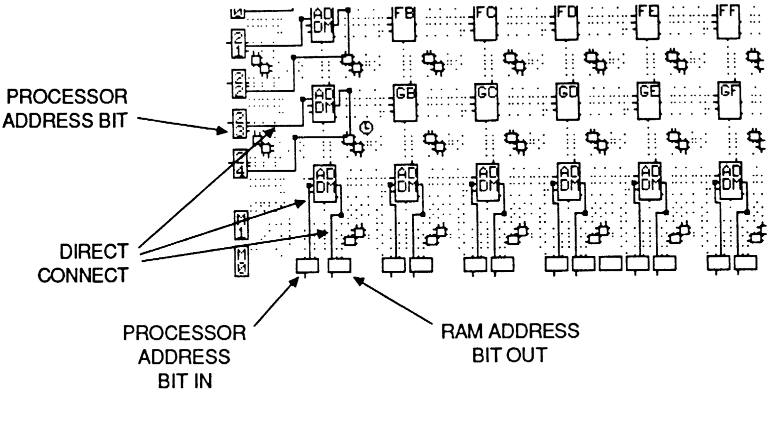


Figure 13. Use of Direct Connect in Address MUX.

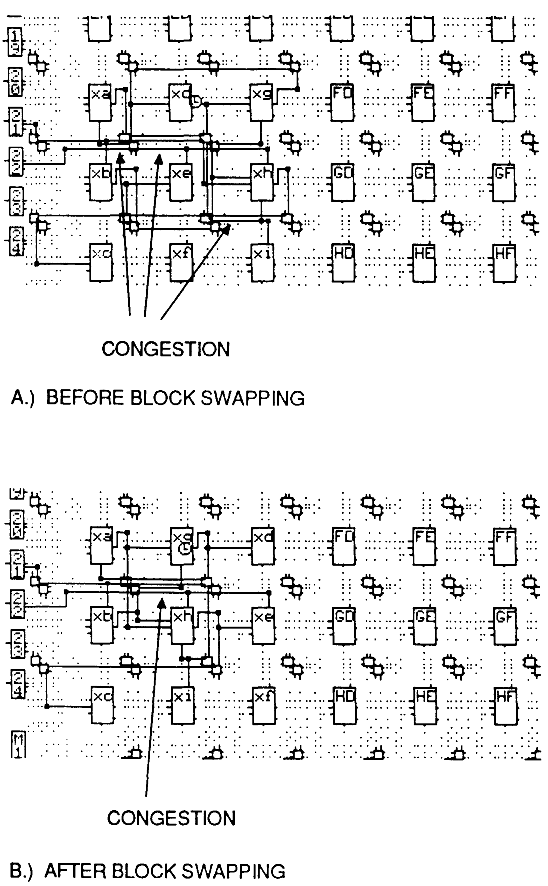
EDITNET allows the user to selectively enable or disable any of the programmable interconnect points (PIPs) on the device. This is done with the following sequence:

1. Select the EDITNET command either with the mouse or enter it from the keyboard.

2. Specify the Net which you wish to manually manipulate. The net must have the source and destination connections on block pins defined.

3. Move the mouse to position the cursor over a programmable interconnect point (PIP) for the desired path.

4. Pushing the select button on the mouse will toggle the selected connection. If it was connected, pushing select will disconnect it, and if it was not connected, pushing select will connect it.



A.) BEFORE BLOCK SWAPPING

B.) AFTER BLOCK SWAPPING

Figure 14. Block Swapping to Relieve Congestion

5. For the switching matrices located where the general interconnect segments meet, a pair of "magic" pins must be selected. Table 1 shows the allowed connections for the various switching matrices. Connections are made or broken by selecting the desired pair of pins. When the second pin is selected, current connections will be broken and unconnected pins will be connected. Figure 15 shows the sequence of operations for editing connections in switch matrices.

6. When all connections have been made, select the DONE option. The XACT system will automatically calculate the delays associated with the interconnections, and make them available for display. The delay from the source of a net to its destination will be shown whenever the cursor is positioned at a destination pin.

When using EDITNET, an error message of "connection shorts pin zz.v" may be displayed. This indicates that a connection would provide a signal to a block pin which has not been assigned to the net being routed. If that pin is to be connected to the net, it must first be assigned to the net using the ADDPIN command.

Although some connections between pairs of switch matrix pins cannot be made directly, it is possible to use a combination of the valid connections to accomplish the desired routing. For example, connection from pin 1 to pin 4 is not valid but can be accomplished by connecting pin 1 to pin 5 and pin 5 to pin 4. This involves an additional switch delay, but may be essential in routing in a congested area.

Some additional routing techniques include:

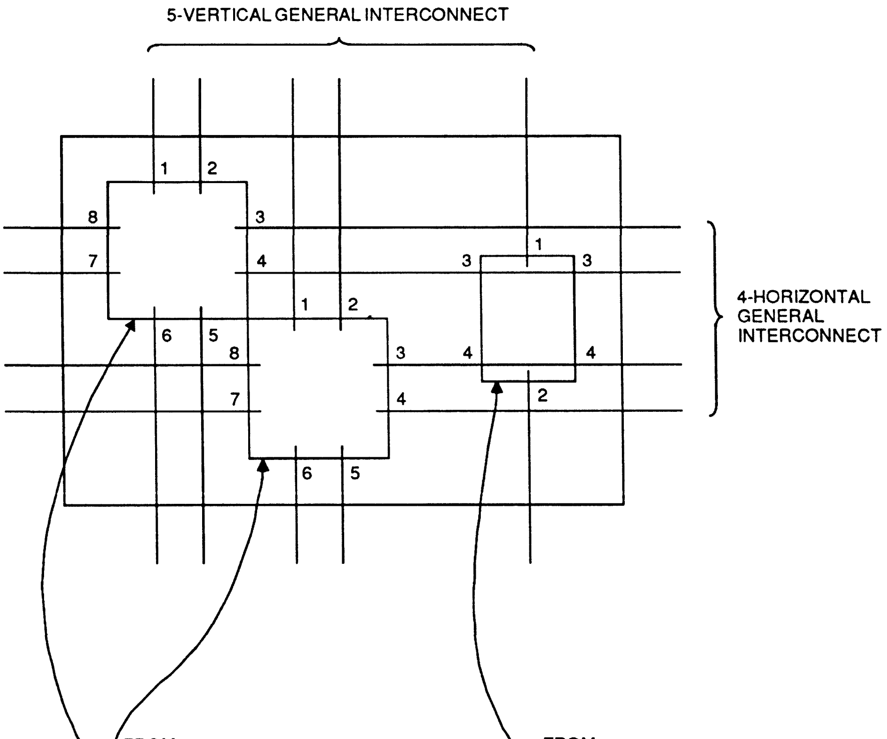
- Do not route through inputs and outputs
- Seed the routing for a net before using auto-routing
- Pre-route selected nets onto long lines
- Route high fanout items first / last

DO NOT ROUTE THROUGH INPUTS AND OUTPUTS

Although the inputs and outputs of the various blocks are shown as lines with multiple connections on them, it is not possible to use them as connections between parallel interconnect segments. Each input or output connection to a pin of a block is uni-directional and only one connection per pin is allowed.

The EDITNET command will allow the user to turn on multiple programmable connections on an input, but only the connection from the driving interconnect segment to the input pin is valid. Any additional connection points which are turned on will not be connected to the driving segment, although they appear to be connected. If the design rule check, DRC, command is executed, nets which have been routed in this way will be flagged as unrouted and their attendant delays will not be calculated. Figure 16 shows an improperly connected net routed through an input switch path.

Outputs of blocks may drive multiple interconnect segments, although it is not generally necessary, but a net not driven by that block may not be interconnected using the output path switches. Figure 16 shows a net improperly connected using the output path switches. In both input and output cases, these connections can


5-VERTICAL GENERAL INTERCONNECT
4-HORIZONTAL GENERAL INTERCONNECT
1
2
3
4
5
6
7
8
1
2
3
4
5
6
7
8
1
2
3
3
4
4

FROM

TO	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	■

FROM

TO	1	2	3	4
1	■	1	1	1
2	1	■	1	1
3	1	1	■	0
4	1	1	0	■

1 = VALID CONNECTION
0 = INVALID CONNECTION

00100003 7b

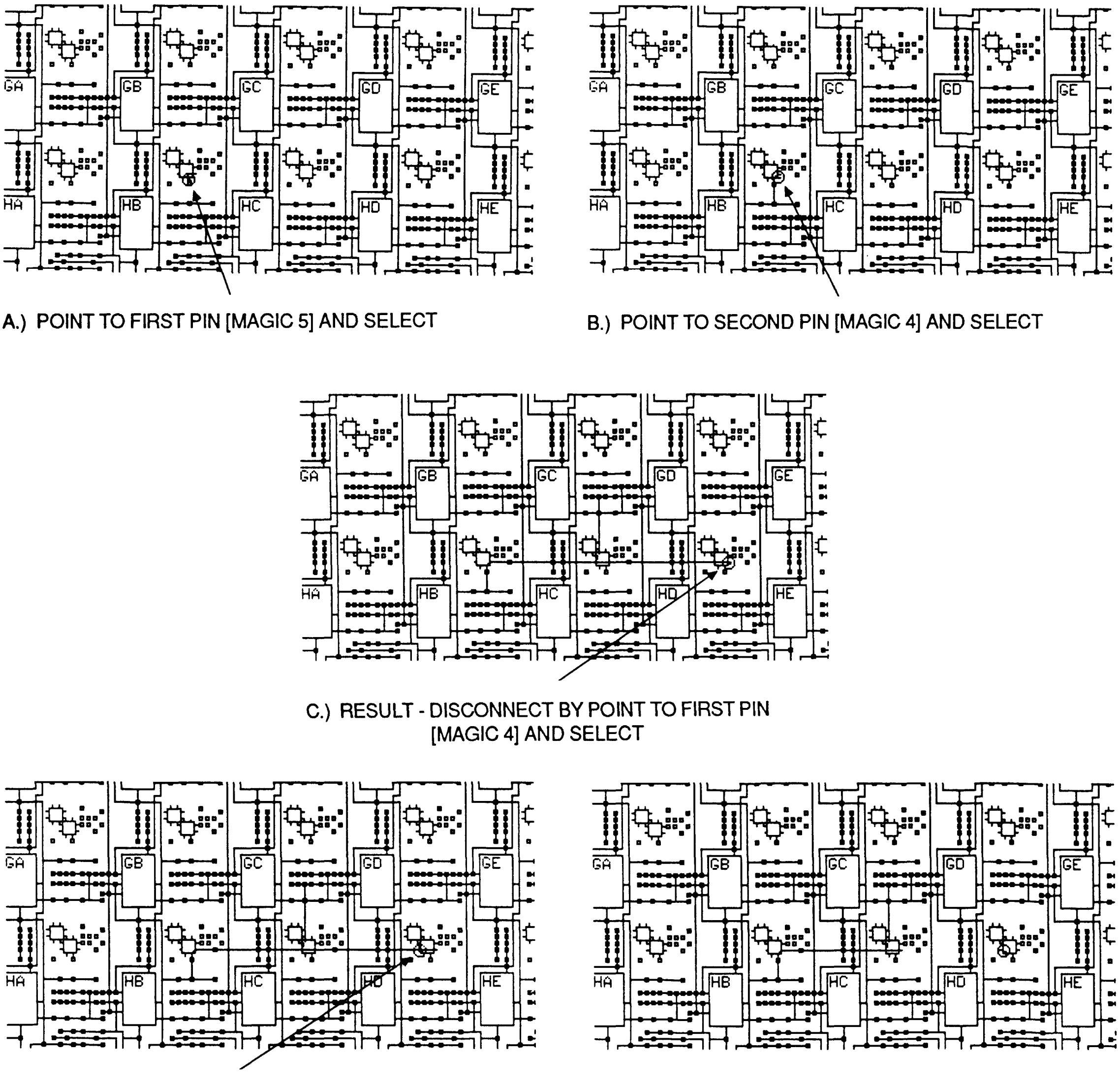
Table 1. Allowed Connections Through Switching Matrices

only be made with the EDITNET command. Caution is advised when using this command to avoid these improper connections.

SEEDING ROUTING

An effective technique for improving the resource utilization of the router is to manually "seed" the routing prior to allowing the router to operate. This seeding may take two forms, depending on the desired effect.

1. If the user chooses to utilize a particular long line resource for a signal path based on delays or general placement, the router typically will not route onto that long line if an alternate path is available. One technique to force a signal onto a long line is to pre-route it onto the long line before actually routing the signal. This is done as follows:



A.) POINT TO FIRST PIN [MAGIC 5] AND SELECT

B.) POINT TO SECOND PIN [MAGIC 4] AND SELECT

C.) RESULT - DISCONNECT BY POINT TO FIRST PIN [MAGIC 4] AND SELECT

D.) POINT TO SECOND PIN [MAGIC 7] AND SELECT

E.) RESULT

Figure 15. Sequence of Operation for Connecting through Switch Matrices

a. If the net is already entered, use UNROUTE or CLEARPIN to deconfigure the routing for each pin on the net. If the net has not been entered, disable the automatic router using the AUTOROUTE OFF command, found in the MISC menu, and define the net using the ADDNET command. This avoids the delays involved in routing each pin and the necessity to unroute them after they are entered.

b. Using EDITNET, choose the net to be routed on the long line and turn on/off the appropriate switches to get the signal from its source block onto the long line.

c. End the EDITNET command by selecting DONE. A warning message will be issued indicating that the net has not been routed.

d. Select the ROUTE command from the NET menu. When prompted, select the net which was manually routed onto the long line. The router will then complete routing of that net.

Figure 17 shows an example of this technique. In some cases where the destination pins are not directly accessible from the long line, the router will still not utilize the selected long line. In these cases it may be necessary to use both techniques 1 and 2 to force use of a long line.

2. In some cases, pins are added to a net throughout the course of the design. With the automatic router enabled throughout this process, each pin will be routed as it is added. The resultant net routing may become contorted and interwoven because the router will route each pin independently. Extreme cases may have loops in the interconnect, or very long delays as the source block becomes more heavily loaded and the routing more degenerate. This may also cause severe congestion in some areas as the routing resources are unnecessarily consumed by the multiple routes. To help relieve this problem, and other similar multi-destination problems, enter the destinations in a sequence which progresses naturally from the source location to the farthest destination. *Remember, the router will route to the first specified destination first, then the second, third and so forth.*

To avoid the necessity of entering destinations in location-specific sequence for large nets, the following may be done.

a. Enter the nets into the design with the router disabled or unroute the net (See 1.a above).

b. When all of the destinations for the high fanout net have been entered, use EDITNET to manually route to the destination which is physically the most distant from the source. If the routing to this pin does not use a long line, ROUTEPIN may be used to accomplish the initial routing.

c. Use the ROUTE command to allow the router to complete routing of the other destinations in the net.

Figure 18 shows the use of this technique for a net with many destinations. Another alternative method is to use a text editor to modify the sequence of destination pin

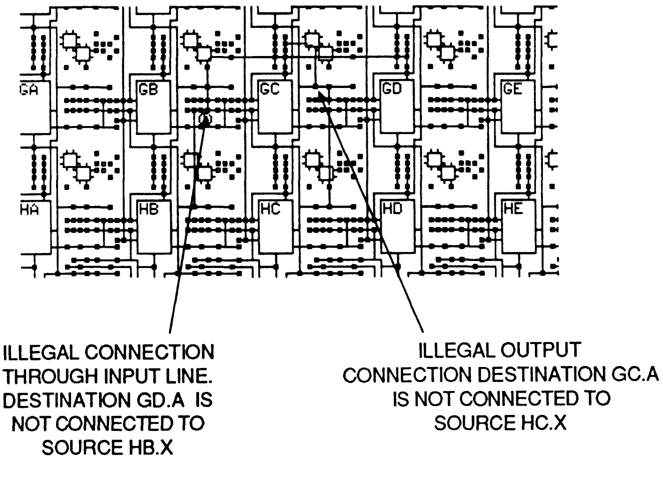


Figure 16. Illegal Connection

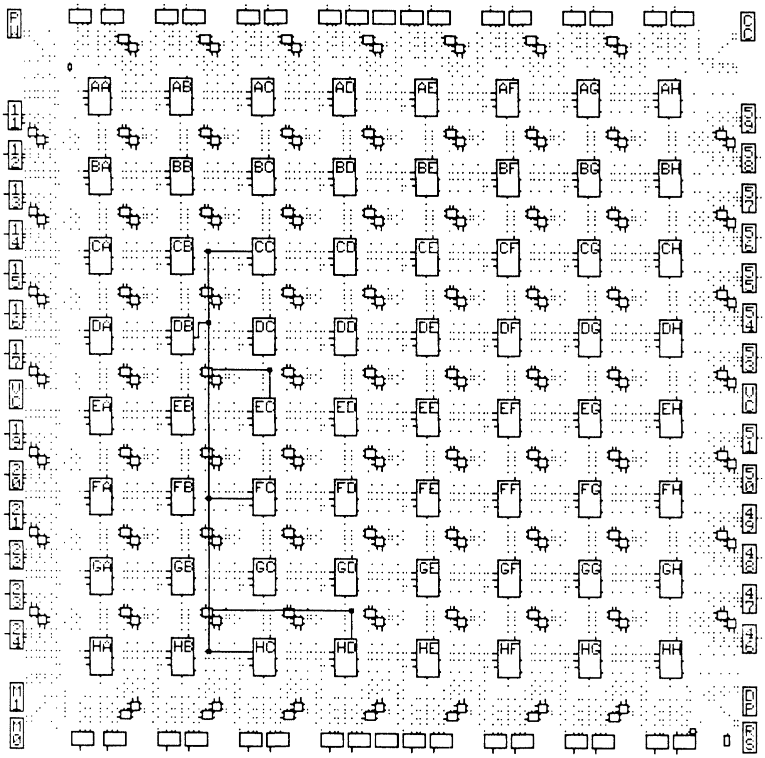


Figure 17a. Output DB.X Routed Via General Interconnect

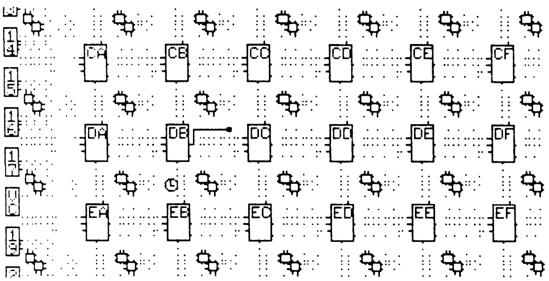


Figure 17b. After UNROUTE Command; EDITNET has Forced Output onto Long Line

specifications in the design file (.LCA) for the design.

ROUTING HIGH FANOUT NETS

When placing and routing designs involving high fanout nets, congestion problems will often occur if the design is routed as it is entered with the auto router enabled. In these instances, alternate placements will probably be needed to complete a good design. The following sequence of steps is suggested.

a. Plan an initial placement on a blank LCA printout with the previous placement tips. Pay particular attention to the use of direct interconnect.

b. Begin entry of the design with the automatic router enabled. *When entering the high fanout net(s), enter only the destination pins. Leave the source undefined, even though you know what it is.* This allows the system to route faster and leaves a less cluttered design.

c. When all of the regular nets have been entered, look at the congested areas. These can be easily identified by counting the used vertical and horizontal general interconnect segments in each column/row. A printout of the complete design with the options SHOW USED enabled may be helpful.

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

e. Generate a new placement based on the congested areas identified above. MOVEBLK and SWAPBLK should be used to move the blocks to new locations. The criteria for the new placement should be to eliminate the congestion as much as possible.

f. Implement the new placement with the automatic router disabled.

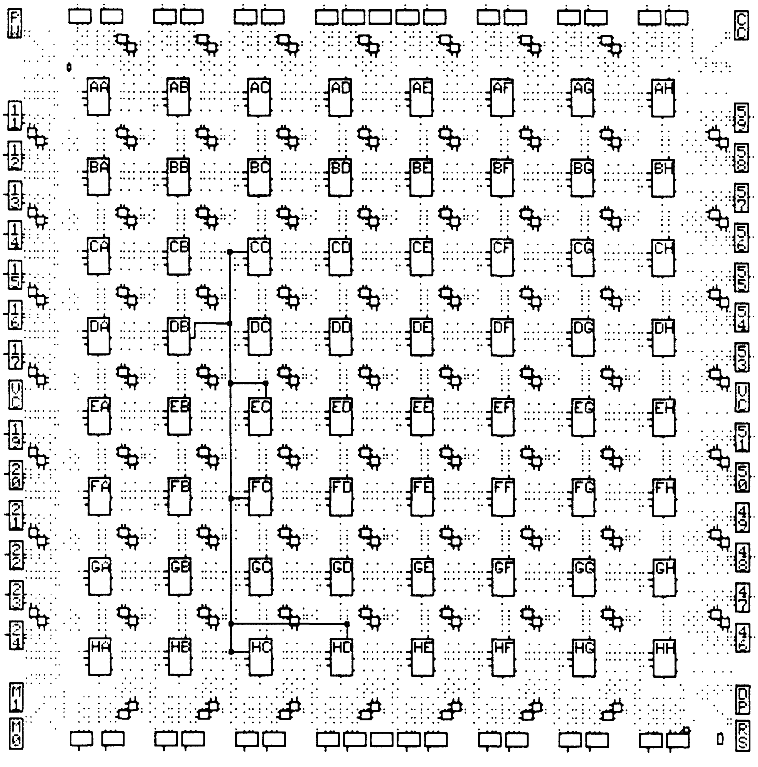


Figure 17c. Result After Route of Signal—Long Line Used

g. Route the high fanout net(s) using techniques 1 and 2 from above. The high fanout net(s) should be optimally routed with this technique. Viewing the design in either large or medium scale allows more of the blocks to be seen at one time to get a good sense of where routes should be placed. Also HILIGHT of the high fanout net will show stubs at each of the required connections, allowing better visualization of their physical relationships.

h. Save the intermediate results as a backup.

i. Route the remaining nets either with a ROUTE * command or by selecting each Net with a ROUTE command.

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

OTHER USEFUL ROUTING FUNCTIONS

There are several other useful routing related functions which should be explored in optimizing designs. These are SWAPSIG, CLEARPIN, and ROUTEPIN.

SWAPSIG

The SWAPSIG command, located in the PIN menu is quite useful when optimizing the routing of a signal to a specific block. In many cases, signals will be better routed to a specific block pin, in spite of the general interchangeability of the pins. Figure 19 shows some typical signal routes where the choice of block pins can be modified to relieve routing congestion. 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. SWAPSIG should always be used as opposed to SWAPPIN when working with pins on a single block since it modifies the internal function to

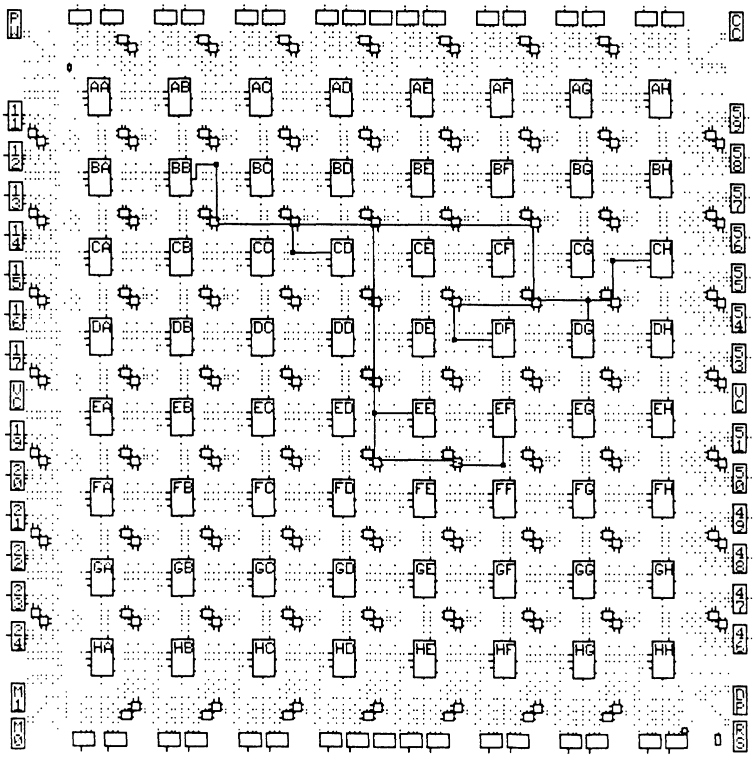


Figure 18a. Routing Without Seeding

match the pin swapping. SWAPPIN is valuable for moving a net connection from one block to another.

Figure 19a shows a net which is routed with the general interconnect. Using SWAPSIG allows the pin assignment of the destination block to be easily "interchanged" to make use of the direct connect. The general segment is then freed for use by other routing. The SWAPSIG command can also be used on block outputs to swap them for use of direct connect, or to allow driving a particular adjacent general interconnect segment. *In the case of outputs, X and Y are completely interchangeable internally, so their selection should be based entirely on their external connection usage.*

Figure 19b shows two pins which have been swapped using SWAPSIG because it allowed more efficient use of the general interconnect. The initial connection to pin C came from a signal running in the adjacent horizontal channel. Pin D came from an adjacent vertical channel. Swapping the signals allows the vertically oriented signal to route directly to C and the horizontal signal to route to D. The internal constraints on the input pins to logic blocks may limit some uses of SWAPSIG. These will be flagged when the command is executed.

CLEARPIN

This command allows the user to de-configure the interconnect for a particular pin on a net. In the process it also removes any spurious interconnect segments from the net. CLEARPIN, located in the PIN menu, is particularly useful when attempting to relieve congestion in an area. It allows interconnect from a single pin on a net to be returned to the available pool of resources. When routing critical or high fanout nets, the freed interconnect can be used for a particular route. The unrouted pin can then be routed either manually or with the ROUTEPIN command.

ROUTEPIN

When manipulating the routing for a portion of a design, pins are often left unrouted. It is possible to route these pins with the ROUTEPIN command for the pin, but

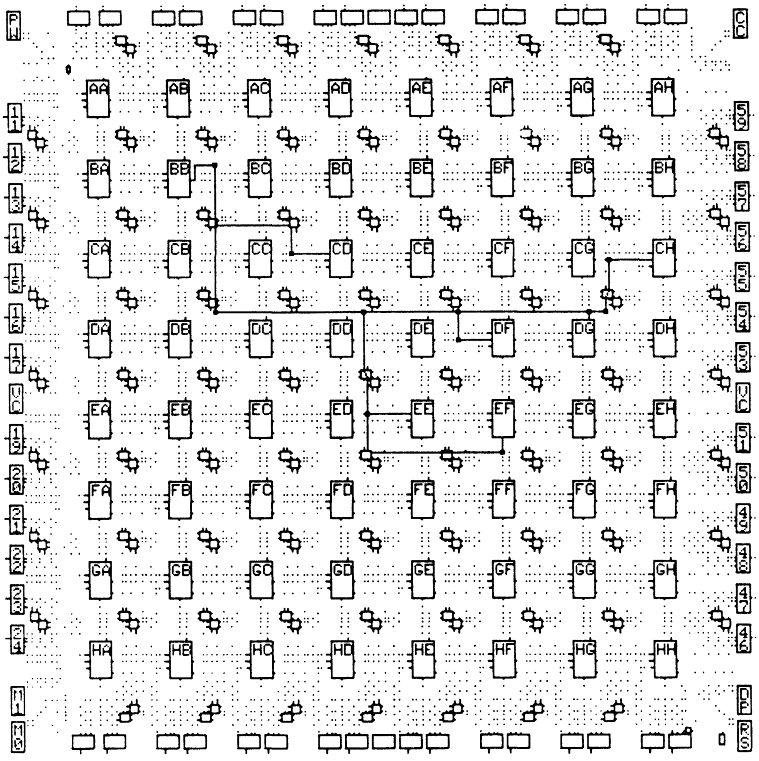


Figure 18b. Routing After Seeding

ROUTEPIN forces the user to select all pins to be routed. ROUTE, on the other hand, routes all of the pins assigned to the net that is selected. Figure 20 shows how this can be more efficient if a large number of unrouted pins are to be handled. ROUTEPIN will, operate faster than ROUTE for a single pin because it is only concerned with a single pin on *any* net. ROUTE will check each pin on the net, and operates on a single net at a time.

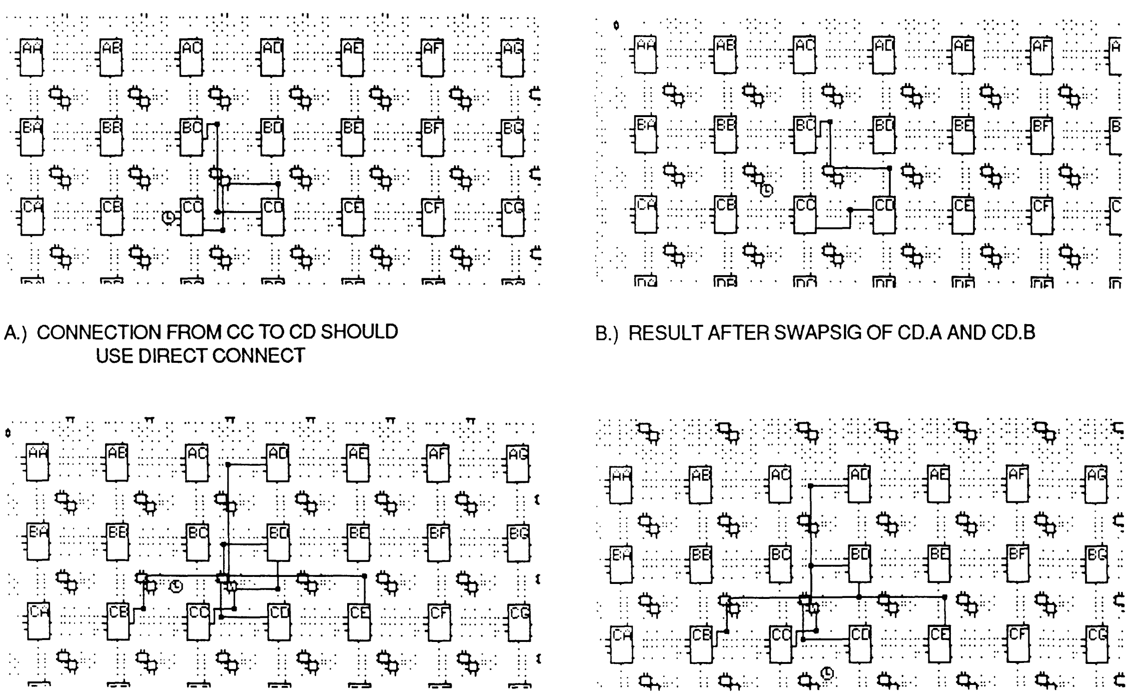
THE TIMING DELAY CALCULATOR

The XACT Development System includes a unique interactive timing delay calculator which allows the designer to see the worst case delays associated with a design, without the need for simulating the design. This is particularly useful in selecting placement and routing alternatives in the process of tuning a design for maximum performance. Delay information is available for the logic blocks, I/O blocks and the interconnect paths.

Logic and I/O block delays are fixed worst case values based on the particular configuration of the block. These delays are characterized from operating devices at worst case conditions and are typically constant for a particular speed grade.

Interconnect delays are more complicated. Each interconnect segment which is used in a signal path represents a distributed R/C delay. Inputs to each logic or I/O block have a small capacitance which can be ignored in comparison to the capacitance associated with the interconnect segments. To correctly calculate the worst case delay for interconnect, the accumulation of these delays must be accounted for. In addition, each transistor switch represents a non-linear impedance which modifies the drive characteristics as viewed by downstream segments. Figure 21 summarizes these delays and the elements included in the model for interconnect delay calculations.

As signals pass through several of these segments and



A.) CONNECTION FROM CC TO CD SHOULD USE DIRECT CONNECT

B.) RESULT AFTER SWAPSIG OF CD.A AND CD.B

C.) CONNECTION CB.X TO BD.C SHOULD BE IN HORIZONTAL CHANNEL AND CC.X TO BD.D IN VERTICAL CHANNEL.

D.) RESULT AFTER SWAPSIG OF BD.D AND BD.C

Figure 19. SWAPSIG Candidates

```
Querynet: PNRFG25B.LCA, XACT 1.21

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

Figure 20. ROUTE and ROUTEPIN Comparison

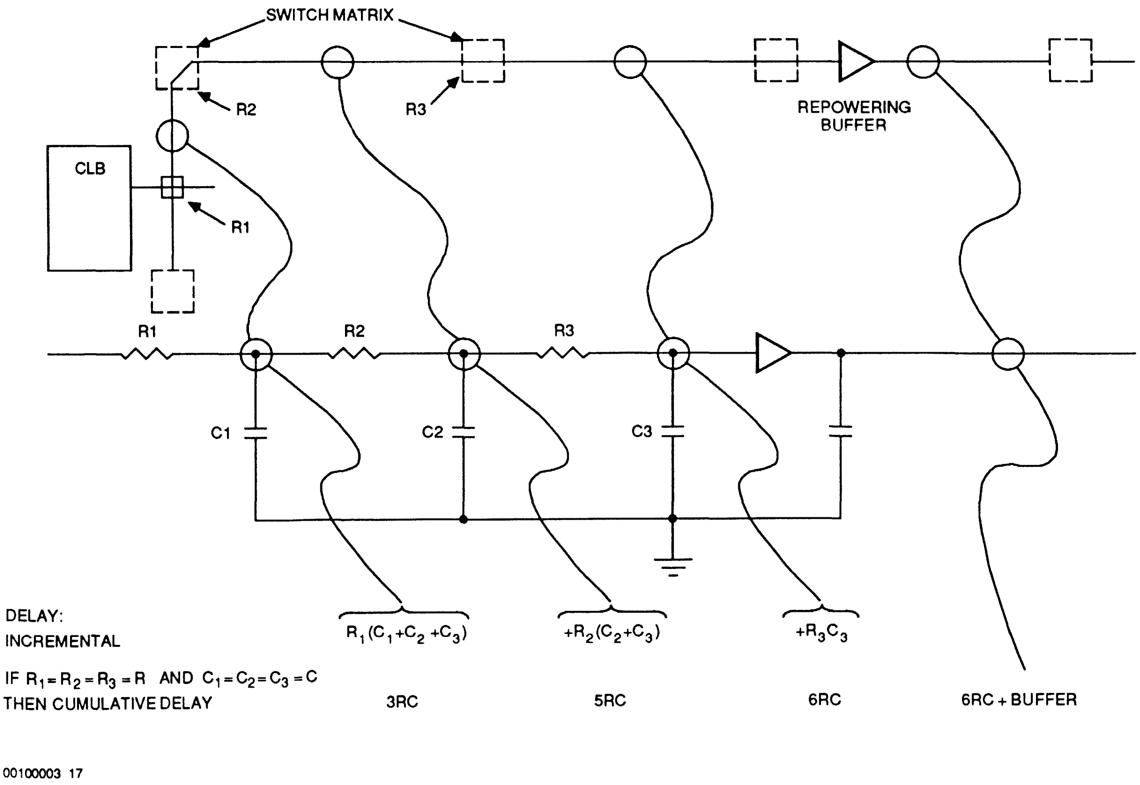


Figure 21. Interconnection Delay Example

switches, their signal quality is degraded. In the general purpose interconnect area, bi-directional buffers are used to re-power the signals after they pass through several segments. Each buffer also represents a delay, but after the buffer, the initial signal quality is restored. These buffer delays are accounted for in the overall delay calculation.

The delay calculator includes all of these elements when calculating interconnect delays. Whenever the cursor in the overall LCA screen is positioned on a destination pin for a net, the worst case delay from the net's source to that destination is displayed on the information line of the display (See Figure 22). As the cursor is positioned on each destination, the appropriate delay is shown.

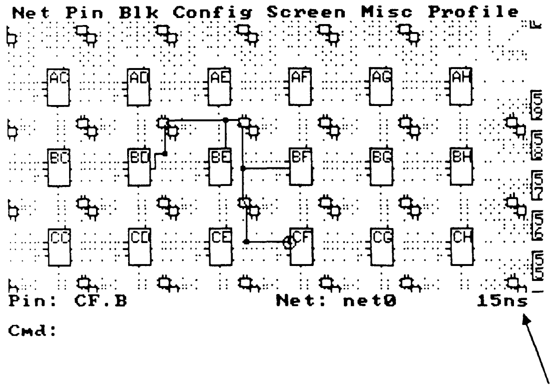
Calculations for delays are performed on a net by net basis, as the complete net configuration must be considered to determine the delay. When nets are being defined, the delay to each point is not available until the source and all of the destinations have been specified. When the DONE option in the net specification process is selected, interconnect delays will be calculated *if* the Net has been routed; this is typical if the automatic router is enabled. For pins which have not been routed, a delay of ? is displayed.

When manual routing is being performed with EDITNET or any of the other techniques, interconnect delay calculations will not be performed until a) the DONE option has been selected and b) a destination pin has actually been connected to the source pin. If a net is subsequently modified by addition of other pins or interconnect, a new net delay calculation will be performed and the new timing information will be available.

Interconnect delay information is available interactively (on the information line of the display). It may also be obtained in text reports, either to the screen or in printed form. Figure 23 shows a sample delay report printed by selecting REPORT DELAY and specifying the desired FROM and TO options. Delay information is also included in printed or screen information obtained with the QUERYNET command.

In a clocked system, delay calculations are made from clock-edge to clock-edge. Since it has no knowledge of the dynamic operation of the system, the delay calculator can only consider the elements which logically are connected from one clocked device to the next clocked device; latch or flip-flop. Simulation is required 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, the worst case clock frequency can be easily obtained: worst case frequency = 1 / (clock-to-clock delay). In Figure 23, the worst case clock-to-clock delay for Net17 and Net18 is calculated as 98 ns. This circuit could be clocked at 10 MHz worst case.

In the Logic Cell Array family multiple speed grades are available. The delay calculator has information which allows it to calculate all of the delays for a design, assuming different speed grades. The speed grade selection is made by selecting the SPEED command from the MISC menu. The currently available speed grades for the selected device will be displayed and the user selects the desired one. The delay calculator then re-computes all of the delays in the device and makes them available, either for display on the screen or in the



Pin: CF.B Net: net0 15ns

Cmd:

DELAY FROM SOURCE TO DESTINATION
POINTED TO BY CURSOR

Figure 22. Delay Calculator Result On-screen

reports available for the design.

WHAT DOES TILDE MEAN ?

Because of the nature of the pass transistors used to perform the interconnection of 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, these two factors can significantly degrade the predictability of the delay for a particular path. The bidirectional buffers used to re-power the signals in the general interconnect will normally alleviate most of these conditions, if they are in the signal path. Because of the presence of manual editing capabilities and the router's ability to route signals with remaining resources, some paths with significant signal degradation may be created in a design.

Signal paths which have degraded signals will be flagged by the delay calculator with a tilde (~) preceding the calculated delay. These "degenerate" nets may be the result of one or more factors:

a. A general interconnect segment and its associated signal are driving a long line. Because of their "long" nature, long lines represent a high capacitance. This high capacitance affects the signal quality, particularly when driven by a general interconnect segment and not the direct source of a signal.

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

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

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

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

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

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

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

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

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

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

WORST CASE
CLOCK TO CLOCK
PATH = 10 mHz CLOCK

Figure 23. Printed Output From Delay Calculator

b. A long line is driving a general interconnect segment or group of segments. In general, the long lines will greatly decrease the effective drive capability of the source of the signal. When driving general interconnect segments, the interconnect switch impedance and long line combine to create a problem.

Degradation of signal quality as indicated by the tilde preceding the calculated delay 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 increases. As an example, consider a delay indicated as ~50. This indicates:

a. Falling signals (1 to 0 transitions) will occur more rapidly than indicated. For this case, the falling transition may propagate in 35 to 40 ns, worst-case.

b. Rising signals (0 to 1 transitions) will occur in more time than indicated. For this case, the rising transitions may require 70 ns or more, worst-case.

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

When delays are displayed with the tilde, caution must be exercised by the designer. If these signals are timing critical, it is highly recommended that they be re-routed to eliminate the tilde indication. In some cases such as static control, the actual delays are not critical and the tilde may be safely ignored.

In other cases, the difference between rising and falling delays may be compensated for 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, reset when "1", then the critical timing edge is the rising edge. Analysis indicates that the rising edge will be slower than the falling edge, so a re-definition of the signal to be low true, reset on "0", will take advantage of the quicker propagation time for falling signal transitions. This may improve the overall capability of the system by eliminating potential metastability or partial counter reset problems that might otherwise occur.

Paralleling Block Outputs

A technique which can be employed for high fanout signals which cannot use either of the clock buffers is to use multiple block outputs to drive a paralleled network. This provides higher initial drive capability and lower path impedence to the various destinations. Figure 24 shows a circuit where a block output drives many other block inputs. Both the global and alternate clock buffers are being used to supply clock signals to many different blocks.

To provide additional drive capability, the signal inside the block drives both the X and Y ouputs. Each output is routed directly onto a long line, and those lines are then connected into a grid to provide low impedence to the many different destinations. This arrangement of multiple outputs and paths provides significant improvement over a single output and path, but at the expense of using more interconnect resources.

Analysis of Intermediate Timing

In a circuit which has a long path, it may be valuable to measure or predict the intermediate delays as part of making decisions about placement and routing alternatives. One method of either seeing the delay calculator results or measuring delay differences along a path is through temporary I/O block connections. Figure 24 shows two I/O blocks which are temporarily defined along the path. The delay calculator can be used to see the total delay to each block. The differences can then be used either to analyze the results of routing changes, or to determine timing skew related issues. If the In-Circuit Emulator is being used, these I/O blocks may be temporarily defined as outputs and the timing differences measured directly.

EXAMPLES

Figure 25 shows an example where a tilde indicates a potential problem because the net has been routed through several general interconnect segments prior to driving a long line. The timing delay calculator number shown in the lower right corner of the screen shows ~35ns. After modifying the routing, as shown in Figure 25 B, the delay has been decreased to 28 ns and the tilde is no longer indicated.

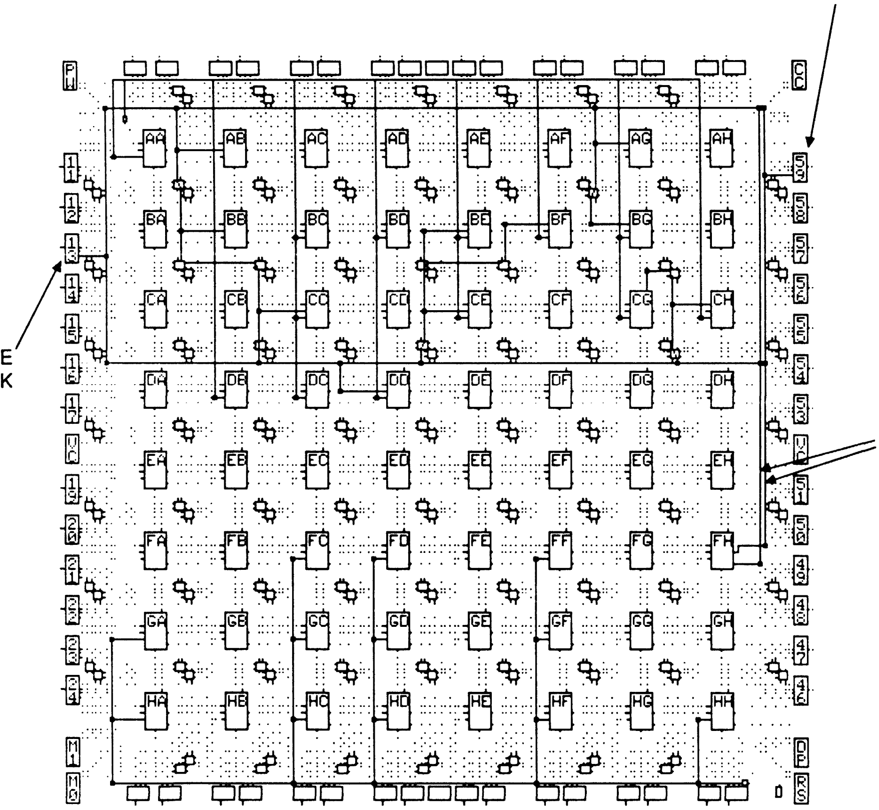
Figure 26 shows an example where the tilde indication can be safely ignored. The net shown is a static input used for function selection by several blocks. In this case, the delay and signal quality is not of concern because the signal is not changing. This might be the case for switch-type inputs or other human interface signals. The only concern with long delay signals of this type is that any blocks which use that signal will latch it correctly after it has made a transition.

CONCLUSION

Appropriate use of the XACT system capabilities gives the user powerful control over all of the aspects of the system design. Simple designs can quite often be entered directly without significant attention to the details of placement and routing. Only when complex designs or ones with stringent performance constraints are to be implemented do the issues of placement and routing need special attention. The techniques discussed here should guide a user in implementing a complex design with minimum effort.

Future products for designing with Logic Cell Arrays will offer improved methods of design entry and will provide greater isolation of the user from the details of the device. Regardless of the sophistication of these development systems enhancements, there will always be a requirement for interactive design optimization, either for performance or resource utilization. The XACT Development System combines simplicity of operation with capabilities to quickly optimize a design.

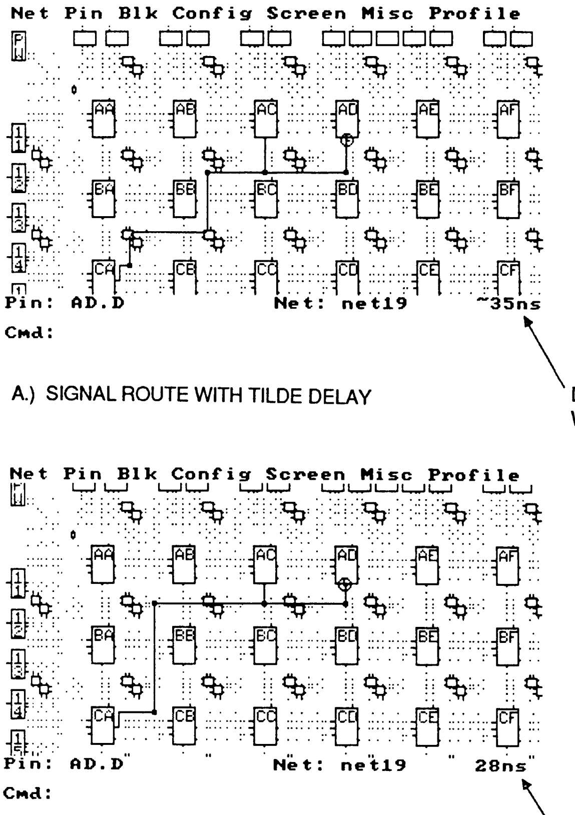
I/O BLOCK TO
ALLOW INTERMEDIATE
DELAY CALCULATION
OR MEASUREMENT



INTERMEDIATE
I/O BLOCK

PARALLELED
LONG LINES

Figure 24. Paralleled Drivers and Long Line to Provide Higher Fanout Signal Source



```
Pin: AD.D          Net: net19     ~35ns
Cmd:
```

A.) SIGNAL ROUTE WITH TILDE DELAY

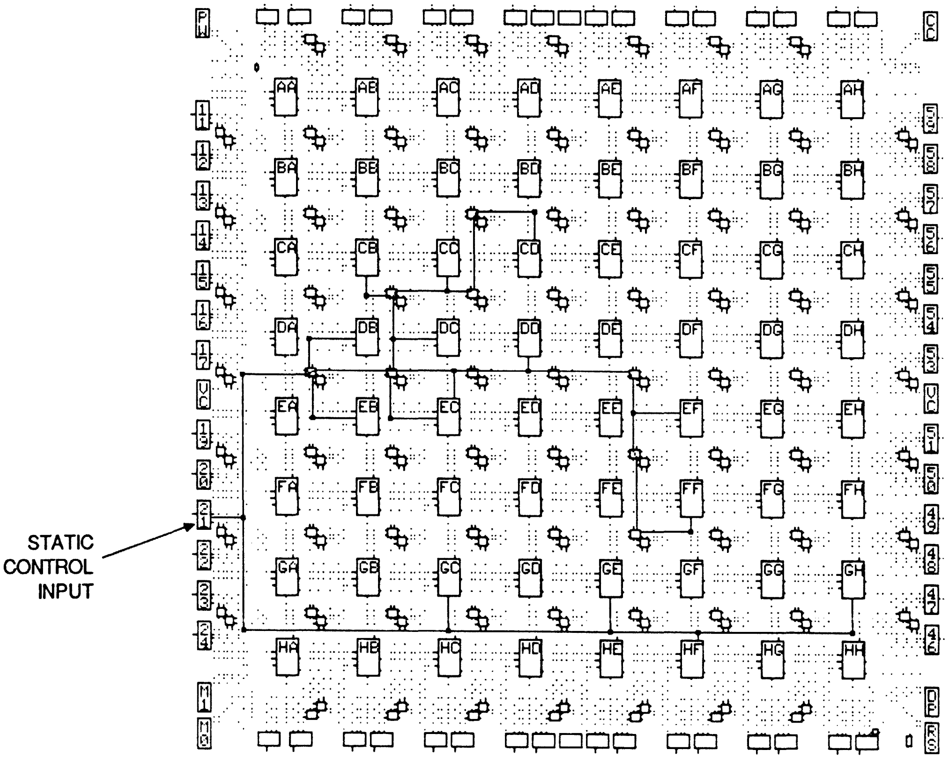
DELAY
WITH ~

```
Pin: AD.D          Net: net19      28ns
Cmd:
```

B.) SIGNAL RE-ROUTED TO ELIMINATE
TILDE IN DELAY

CORRECTED
DELAY

Figure 25. Routing With Tilde on Delay Value



LONG DELAY CAN BE IGNORED BECAUSE DYNAMIC PERFORMANCE OF SIGNAL IS NOT OF INTEREST

Figure 26. Signal With Long Delay to Final Destination

A Design Methodology for the Logic Cell™ Array

Table of Contents

A Design Methodology for the Logic Cell™ Array

INTRODUCTION

Each new technology used for digital design offers the designer a new set of characteristics. These include speed, power, integration level, reliability and selection of logic functions. These factors can affect the designer's methodology and logic architecture. The following examples show that digital design and architecture for Logic Cell™ Arrays (LCA) is similar to that of conventional TTL SSI/MSI or Gate Arrays. The designer using the Logic Cell Array has additional design flexibility, since he is not constrained by limitations such as four-bit or eight-bit increments, a specific set of inputs and outputs, or a specific combination of logic functions. The core of the CMOS Logic Cell Array integrated circuit is an array of user-programmable logic elements called Configurable Logic Blocks (CLBs). User-programmable interconnections implement the required logic networks. Input/Output interfaces are implemented with individually programmable Input/Output Blocks (IOBs). With these facilities, the designer is free to tailor the logic as required and is not confined to standard product devices or gate array library elements. The Xilinx XACT™ Development System provides a macro library of common logic elements to help the designer implement a design. Unique functions and user-defined macros are also available. The IBM PC™-based development system provides the user with graphic design entry and design verification capabilities. Translation of schematic capture to a design file and automatic placement and routing of the design are also available. The development system is also used to generate the configuration program for the device. The LCA can automatically load its program from an external EPROM or be initialized as a peripheral by a microprocessor at power-up.

Compared with other standard product alternatives, the Xilinx Logic Cell Array provides the designer a higher level of integration. Benefits include increased performance and reliability, reduced printed circuit board space, lower power requirements, shorter design times, and smaller component inventories. The logic capacity in one LCA which would typically require from 40–100 SSI/MSI packages to implement. Using the gate array convention of "gate" as a 2-input NAND function, the Logic Cell Array family presently provides logic capacity up to 1800 gates. A single LCA solution can reduce the total package pin count from hundreds of SSI/MSI pins to 48 to 84 Logic Cell Array pins. The devices are available in 48, 68 or 84 pin packages, providing up to 74 pins which the designer may program as logic input, output, or bi-directional package pins. The user-programmable nature of the LCA affords the user a single fully-tested inventory item which may be used in multiple products.

The Configurable Logic Block

A description of the Configurable Logic Block structure will help in understanding its logic capabilities. Each Configurable Logic Block has four logic inputs and two logic outputs. It includes a combinatorial function portion and a "storage element" portion, which may be configured as a transparent latch or an edge triggered flip-flop. The development system provides the method of design entry and verification. The development system translates designs into a configuration program which defines logic look-up tables and multiplexer paths within the Configurable Logic Block. Interconnection between blocks is accomplished by a two-layer grid of metal segments, joined at intersections by switching matrices of program-controlled pass transistors. Additional pass transistors provide connection of the metal interconnections to the block inputs and outputs.

Figure 1 shows a single memory cell controlling a simple two-to-one multiplexer made of two pass transistors. Combining eight readable memory cells and a controlled eight-to-one multiplexer tree, as shown in Figure 2, creates a circuit which is capable of generating any logic function of the three input variables: A, B and C. 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 which can implement any function of four variables. This is the basis of the combinatorial portion of the Configurable Logic Block. The Configurable Logic Block includes programmable multiplexers for the input variables A, B, C, D and Q and a selection of outputs to form a single function of four of the variables or two functions of three variables each. The combinatorial portion of the configurable logic block is shown in Figure 3. When a four-variable function is implemented, the same selection of either input variable D or the Q of the storage element is made in both halves of the look-up table, and the single result produces both F and G outputs. For implementing two functions of three variables, each of the D vs. Q choice is selected independently for the functions F and G. Each function may then use any three of the five available variables: A, B, C, D or Q of the CLB storage element. A third type of function may be implemented by using the input variable B to select between the two three-variable combinatorial functions. This results in a compound function which may involve some combinations of all five variables.

The programmable features of the Configurable Logic Block storage element are shown in Figure 4. The storage element may be left unused, or programmed as a level-transparent latch or as an edge-triggered flip-flop. Its data input is supplied from the combinatorial function F. The invertible flip-flop clock, or latch enable, may be selected from one of three sources, on a block-by-block basis. Each CLB storage element has an active high asynchronous set and reset available. Reset is dominant over set and the active low chip input, $\overline{\text{RESET}}$, clears all storage elements.

The Input/Output Block

As shown in Figure 5, the Input/Output Blocks have the capability of providing a direct or registered input to the chip. The positive edge clock for the register function is common along each die edge. The chip configuration

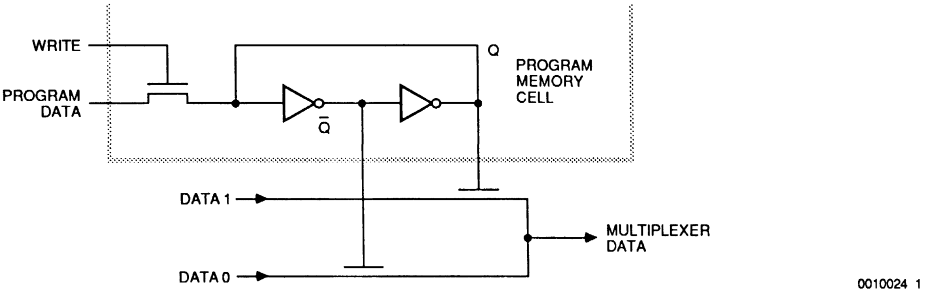


Figure 1. Memory Cell Multiplexer (Mux) Control

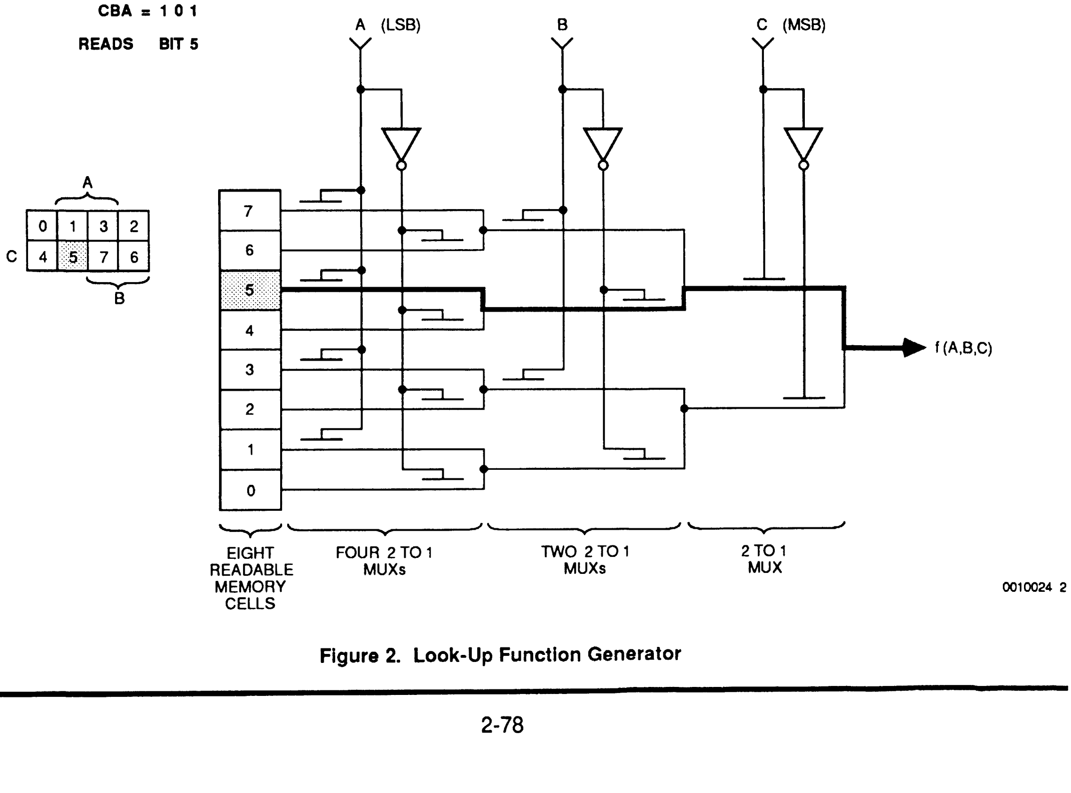
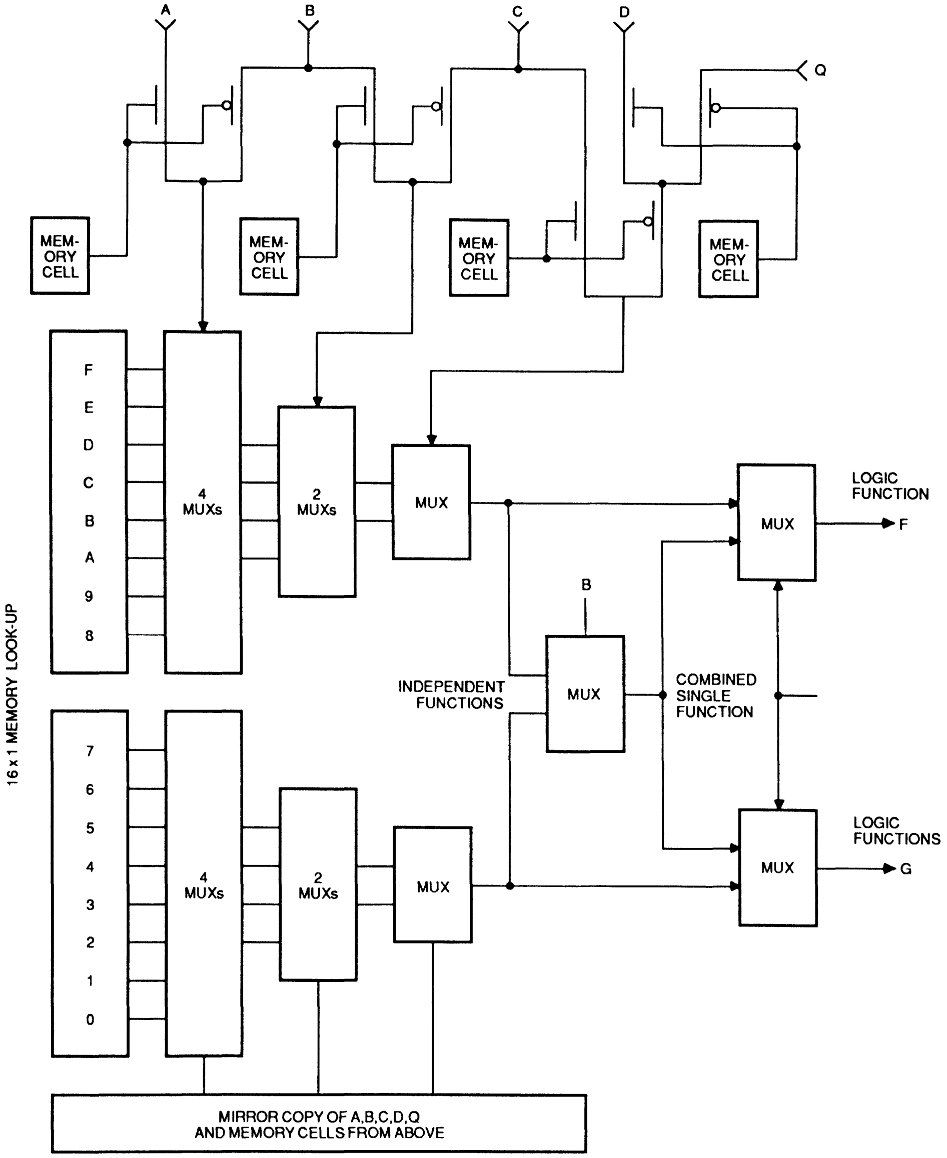


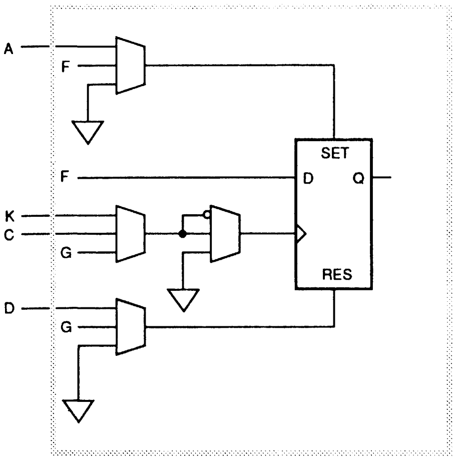
Figure 2. Look-Up Function Generator



0010024 3

Figure 3. Combinatorial Function Generation

process, as well as the active-low chip, $\overline{\text{RESET}}$, clears the storage elements. Each Input/Output Block includes an input/output buffer which may be enabled continuously to implement an output pin, disabled continuously to implement an input or unused pin, or enabled by logic signals to implement an I/O or bus pin.



0010003 6

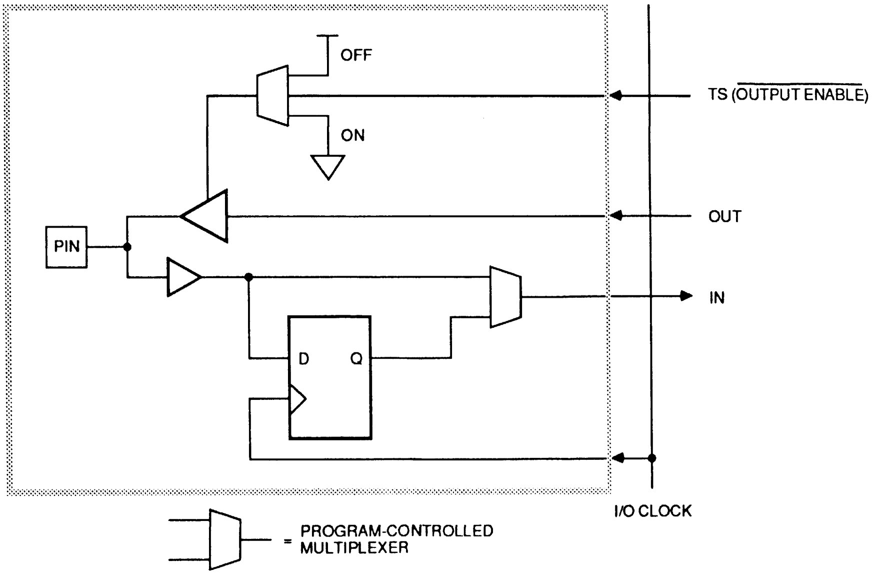
Figure 4. CLB Storage Element

COMBINATORIAL FUNCTIONS

Basic Gate Functions

Logic implemented in TTL devices often uses NAND gates to implement a DeMorganized NAND, an OR of low-level inputs. The function is schematically represented as a NAND due to the package type. The resulting schematic is often confusing when the symbols used do not represent the function performed. The mix of inverters may complicate interpretation of the logic. Descrepancies between the logic symbol and the logic function are particularly confusing to users who were not the original designers. With the availability of complex logic functions in the configurable logic block, there are often several ways to visualize a function. The CLB allows the user to choose between equivalent ways of representing it, both schematically and in equation terms. Karnaugh maps, truth tables and Boolean equations are all supported by the XACT Editor. The ability of the CLB to accommodate either sense of input variables and to generate either sense of an output allows the elimination of extraneous inverters. In most cases, it is practical to route only active high signals avoiding the duplication of routing both true and complement signals.

A typical four-variable combinatorial function is shown in Figure 6a as a logic diagram, a Boolean equation and a


OFF
TS ($\overline{\text{OUTPUT ENABLE}}$)
ON
OUT
PIN
IN
D
Q
I/O CLOCK
= PROGRAM-CONTROLLED MULTIPLEXER

0010003 3

Figure 5. I/O Block

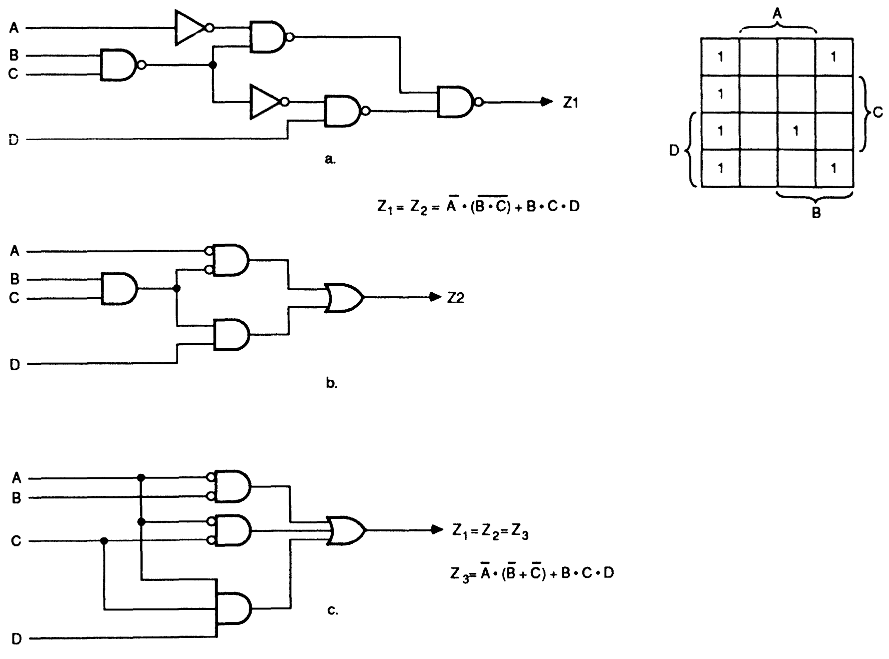
Karnaugh map. An equivalent form of the function is shown in Figures 6b and 6c. The active-low inputs have replaced the inverters of the conventional representation, and the output symbol is an OR. Xilinx software supporting schematic capture can perform this logic conversion and combinatorial gate grouping as a part of its translation to an LCA design file.

The ADDER of Figure 7 is an example of using two combinatorial functions of three variables. The SUM and CARRY functions would usually be grouped in the same CLB by virtue of their common input variables. The four-input exclusive-OR gates of Figure 8a and 8b are an example of a common logic function which is not an obvious four-variable unit. It is a modulo 2 add without carry. COMPARE is a similar function which is typically thought of as a two input function. Figure 9 shows a CLB implementation of a "dual compare" function which compares two bits from each of two sources.

The design element of four input variables may be expanded by using multiple CLB levels. One CLB driven by four others can make the sixteen-variable function of Figure 10. Selection of decodes to use common terms in several functions can allow those CLBs to be shared. A related technique can be used to encode the results of a pair of three-input two-output Configurable Logic Blocks. One of three output codes can be used to indicate which of three selected input conditions exists. The combinations of the two outputs of the CLB may represent four conditions: 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. Figure 11 shows two encoded results, each a function of three inputs. Each CLB responds with the selected code when its inputs match its portion of the desired minterm. A high output indicates that both codes match the same selected value. This yields a sum of three six-variable products.

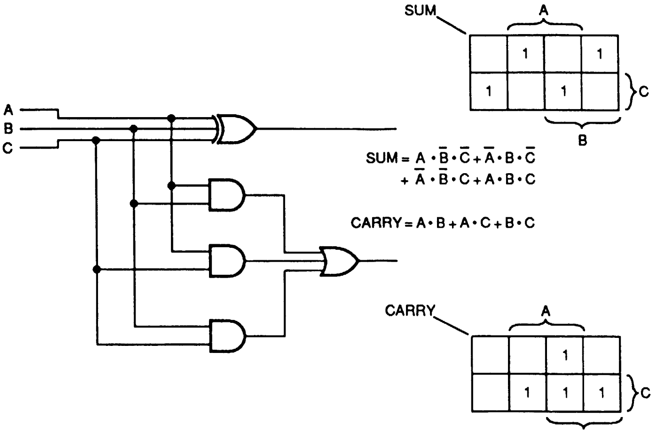
Decoders and Multiplexers

Decoders illustrate several points of practical design with CLBs. If a design does not require some portion of a conventional logic function, that portion need not be implemented. Since in this case the design does not use all of the decodes of a set of input variables, the unused decodes can be omitted. In an output intensive function, using each CLB to implement two functions of three shared variables can be more efficient than implementing one function per CLB. An example is


A
B
C
D
Z1
a.
$Z_1 = Z_2 = \bar{A}\cdot(\overline{B\cdot C})+B\cdot C\cdot D$
A
B
C
D
Z2
b.
A
B
C
D
$Z_1 = Z_2 = Z_3$
$Z_3 = \bar{A}\cdot(\bar{B}+\bar{C})+B\cdot C\cdot D$
c.
A
1
1
1
1
1
1
1
C
D
B

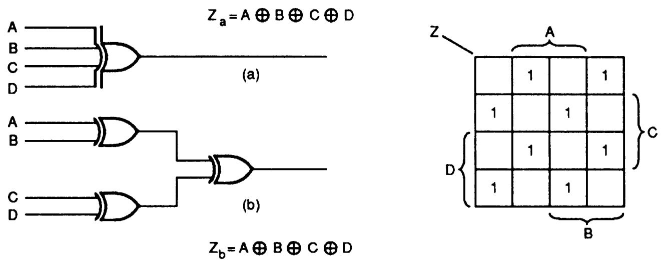
0010024 6

Figure 6. Alternate Representations of Same Function


SUM
A
1 1
1 1
C
B
A
B
C
$SUM = A \cdot \overline{B} \cdot \overline{C} + \overline{A} \cdot B \cdot \overline{C} + \overline{A} \cdot \overline{B} \cdot C + A \cdot B \cdot C$
$CARRY = A \cdot B + A \cdot C + B \cdot C$
CARRY
A
1
1 1 1
C

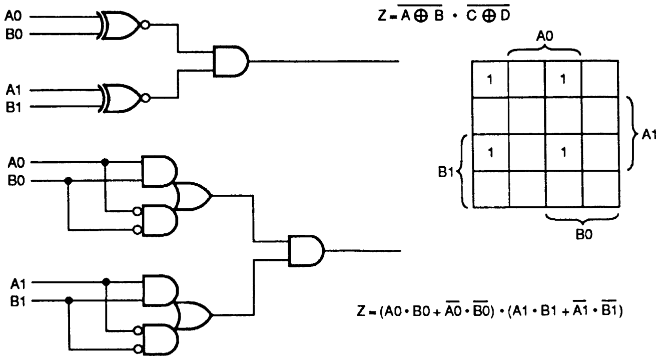
0010024 7

Figure 7. One Bit Adder With Carry In


$Z_a = A \oplus B \oplus C \oplus D$
A
B
C
D
(a)
A
B
C
D
(b)
$Z_b = A \oplus B \oplus C \oplus D$
Z
A
1 1
1 1
1 1
1 1
C
D
B

0010024 8

Figure 8. Four-Input Exclusive-Or


A0
B0
A1
B1
A0
B0
A1
B1
$Z = \overline{A \oplus B} \cdot \overline{C \oplus D}$
A0
1 1
1 1
A1
B1
B0
$Z = (A0 \cdot B0 + \overline{A0} \cdot \overline{B0}) \cdot (A1 \cdot B1 + \overline{A1} \cdot \overline{B1})$

0010024 9

Figure 9. Dual Compare CLB

shown in Figure 12. The "preselect" enabling gates implemented in this figure are an example of a common term of a wider input function. In order to improve system speed the variables which are stable earlier may be used as inputs to the first level of logic. They may propagate while the more time-critical inputs of the design drive the shorter propagation path in order to improve system speed.

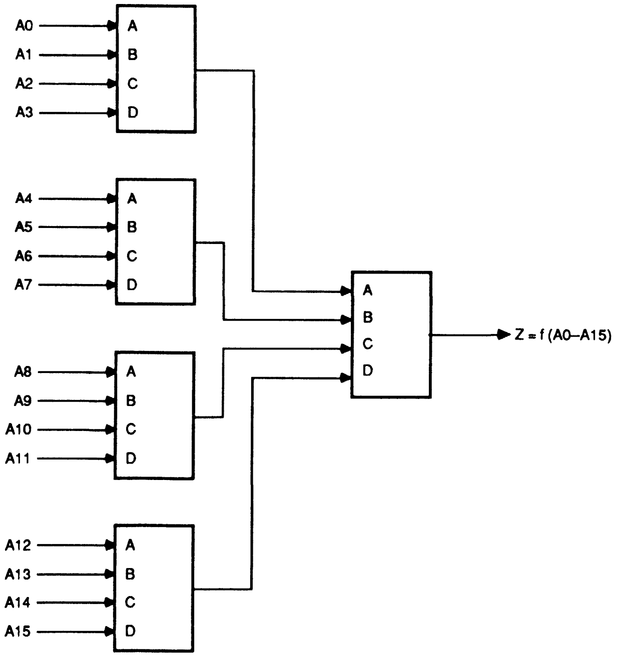
Wide multiplexer functions can be built from a tree of 2-to-1 multiplexers. This leaves the storage element and one input variable of the CLB available for an independent register function, as in Figure 13. In other cases the multiplexer may be the data input to the storage element or may share input variables or may use the output of the storage element. These examples provide a natural grouping of shared functions in a CLB.

A commonly used element in digital systems is a group of registers with sets of enabled output buffers bussed together. The structure shown in Figure 14 is not always recognized as a multiplexer. 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.

TIMING

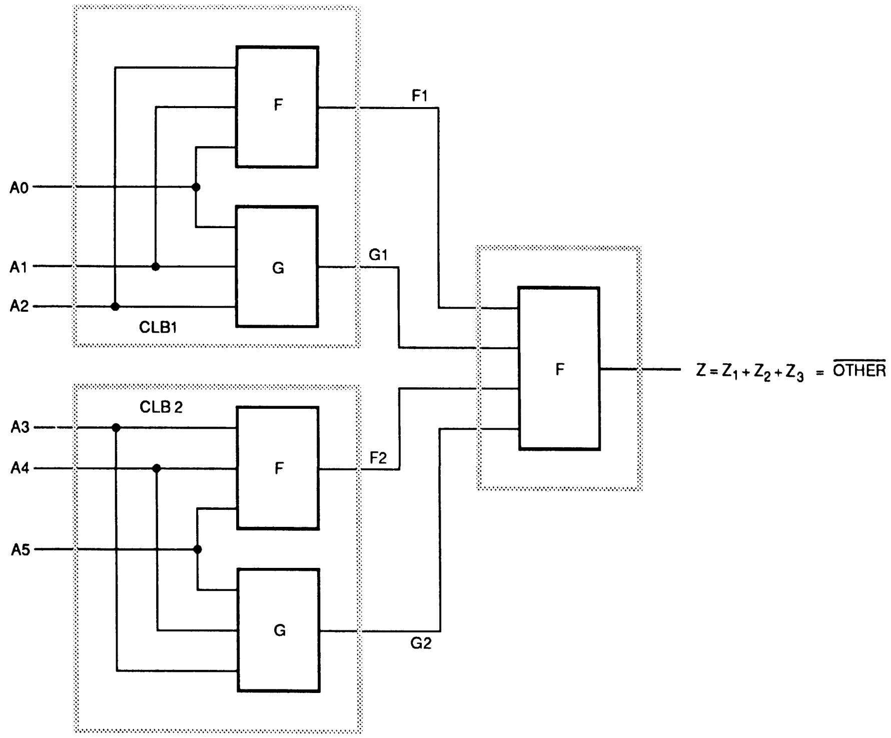
Any Boolean function generated by a Configurable Logic Block requires the same time delay as any other function. The concept of "levels" of logic or gate delay loses its significance with a technology which uses higher-level primltives to perform logic. The primary timing factors involved in design with a Logic Cell Array are the propagation time of a combinatorial Configurable Logic Block, the clock to block output via Q, the input setup time for block input variables of a flip-flop, the delays of input and output pad buffers and the timing of the interconnections. Some of these are represented in Figure 15. Although other switching characteristics are specified in the LCA data sheet, these are the most important factors determining performance.

MSI devices typically have matched internal delay paths and low impedance outputs which are independent of loading. As with CMOS gate arrays, variations in internal signal delays are more significant in the Xilinx Logic Cell Array. In programmable CMOS array architectures, logic delays are more sensitive to output loading than are bipolar devices. With such devices synchronous design



0010024 10

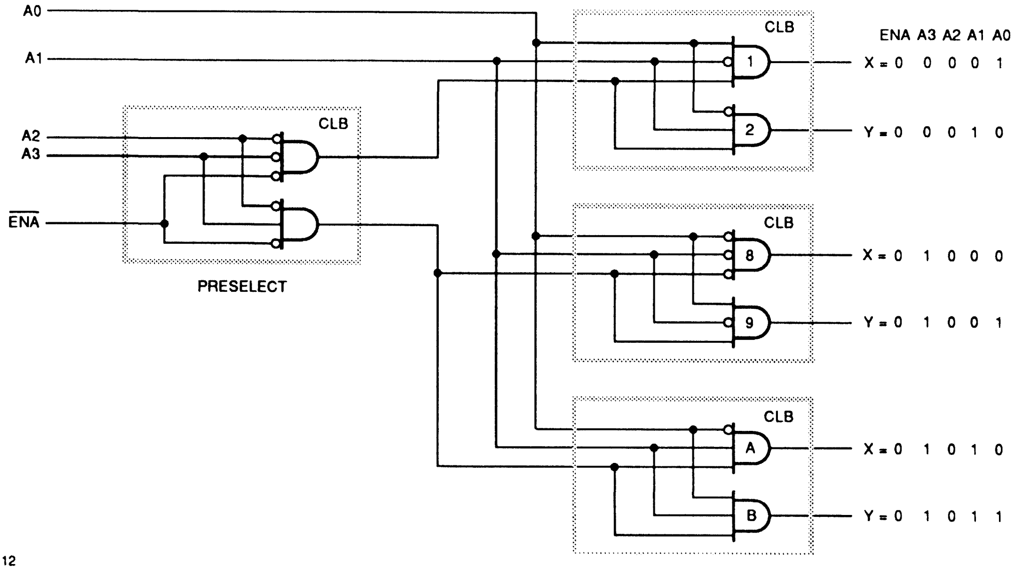
Figure 10. A Function of 16 Variables



	CLB 2		CLB 1			
	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
$\bar{Z}$	OTHER	0 0	OTHER	0 0	$F_2 G_2 \neq F_1\ G_1$	0

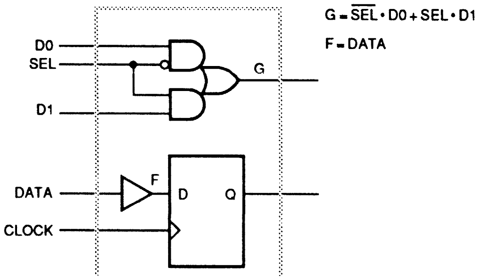
0010024 11

Figure 11. Encoding Partial Results of Six Variables



0010024 12

Figure 12. Decoder with Common Term and Only Required Outputs

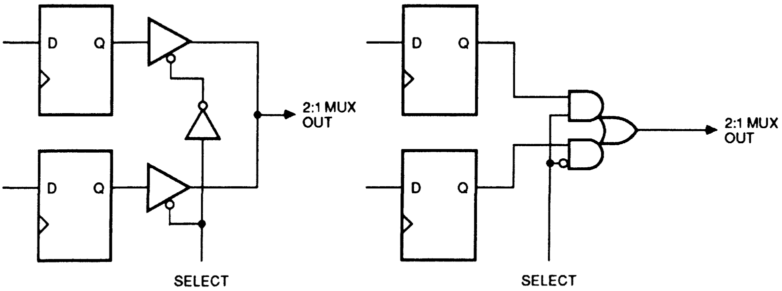


$G = \overline{SEL} \cdot D0 + SEL \cdot D1$

$F = DATA$

0010024 13

Figure 13. CLB Sharing MUX and Register Element



0010024 14

Figure 14. Three-State Function Implements a MUX

minimizes the complexities of signal timing produced by various combinations of delay accumulations. An additional advantage of synchronous design is better control of output timing. The clock distribution resources of the LCA simplify synchronous design. Since any function of the input variables is obtainable, it is simple to include such controls 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, it is possible to generate and use individual CLB clocks, as well as asynchronous set and reset if required by the application.

Latches and Flip-flops

The level transparent form of the storage element is the D-latch. The edge-clocked form is the D flip-flop. For both cases the data input is supplied by the function F and the clock (load enable) is supplied by the .K or .C pin or the function G. This choice and the active sense of the signal is made on a block-by-block basis. The configurability of the Logic Cell Array allows the designer to tailor the storage elements of the CLBs to fit the application. Together with complex combinatorial data functions this allows a wider variety of latches and flip-flops than is found in standard products or gate array cell libraries. Including a RESET variable in the combinatorial input of a flip-flop produces a synchronous reset, as shown in Figure 16a. Use of a combinatorial function of "Q" together with input variables implements a clock enable as shown in Figure 16b. Implementation of a multiplexer as the input of a flip-flop provides a parallel enable as shown in Figure 16c. A flip-flop implementation can have parallel data or reset inputs which do or do not depend on clock enable. Like the J-K flip-flop, an interesting derivative of the set-reset flip-flop is one which will not change for the case of simultaneous set and reset conditions. The set dominant or reset dominant are the other alternatives. The availability of this variety of synchronous set-reset flip-flops provides alternatives for logic implementation that may minimize next-state control conditions.

Registers

Related flip-flops with similar functions form registers. Registers may be grouped into two categories: data registers and shift registers. Data registers are sets of flip-flops with independent parallel input paths and common control. Shift registers are sets of flip-flops with serial data relationships. Both are composed of combinatorial variations of signals supplying the data input of the basic edge-triggered D flip-flop.

Counters

Counters are a simple example of a state machine with a regular sequence. The most familiar counters are the binary weighted sequence, the Johnson (Mobius) counter and the Liner Feedback Shift Register. Johnson counters often offer advantages for counter designs with a modulo of less than 10 or 12. They also lend themselves to simple placement and routing, and the simple combinatorial functions shown in Figure 17 are compatible with maximum clock frequency. Decodes of single or consecutive states are simple and "glitch" free. The initialization of the Logic Cell Array clears all storage elements. However due to the presence of unused states, the Johnson counter might enter an alternate state sequence if there are asynchronous control inputs. As shown in Figure 17, additional input variables from QB and QD in the feedback function can return the count to the proper sequence.

The implementation of a large-modulo CMOS binary-weighted counter presents the designer with a number

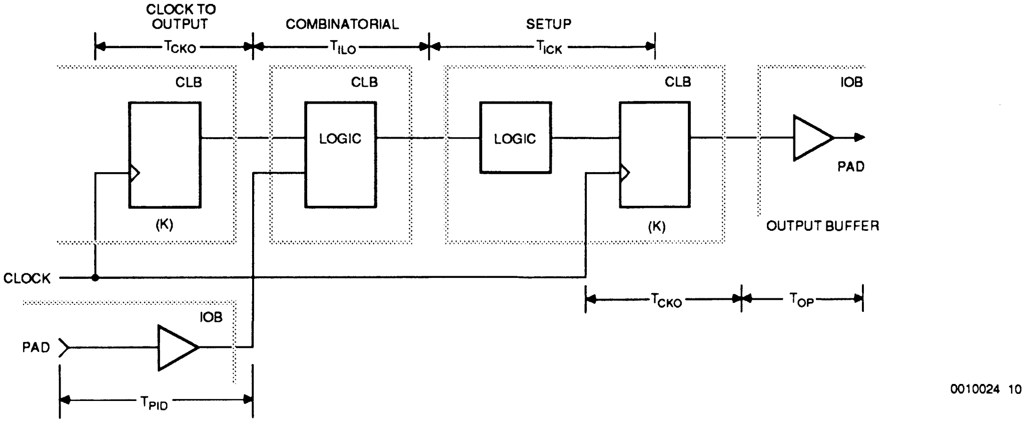


Figure 15. Examples of Speed Factors

of trade-offs. The most effective in terms of resources is a simple ripple counter, but the accumulation of multiple clock-to-Q flip-flop delays can be prohibitive as the outputs ripple for varying times. Figure 18a illustrates a synchronous toggle flip-flop. It changes state synchronously if "T" is HIGH. It is shown with its simplified symbol in Figure 18b. The AND of two inputs to produce the "T" is shown symbolically in Figure 18c. Figure 19 shows a fully synchronous counter solution composed of T flip-flops. Its toggle ripple carry is implemented 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 clock rate is determined by the total propagation time for the carry path from CLKENA to data setup of the last bit. The fully parallel counter of Figure 20a implements each toggle function directly. It will require an n-wide gate for toggle control of each bit of the counter. This design may be extended to 12 bits of counter with a single combinatorial propagation delay between register CLBs, plus one clock-to-Q, one set-up time, and interconnection delay. A compromise can be made by use of block level carries. The 3-bit segment size shown in the lower half of the figure accommodates the three stages and a carry-in within the CMOS 4-input gate limit. The only combinatorial delays involved are those of T3' and T6'. Without a clock enable, input the first section could be 4-bits followed by 3-bit sections. The designer should be observant for design trade-offs and not try to fit a standard solution into all applications. Figure 21 shows another synchronous 8-bit counter with a single level of combinatorial propagation. It illustrates the merging of sequential elements and combinatorial elements of the CLB. Periodic look-ahead carry terms are implemented to make efficient use of variables within the block.

Synchronous Architecture

Efficient LCA-based designs may depart from MSI implementations. MSI elements were designed using another technology and with a goal of creating general-purpose building blocks. Most were designed to fit a set

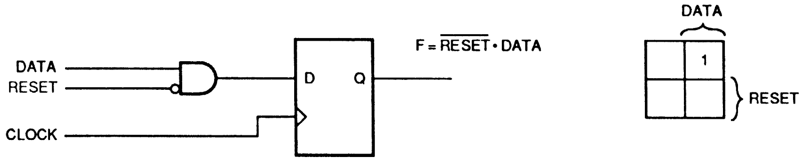


Figure 16a. Synchronous Reset

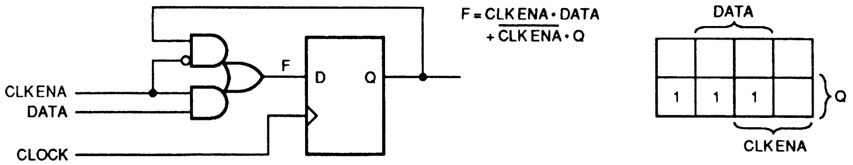
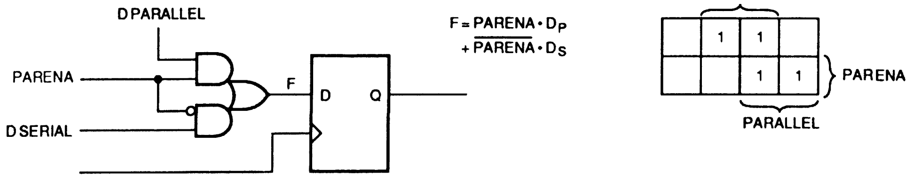


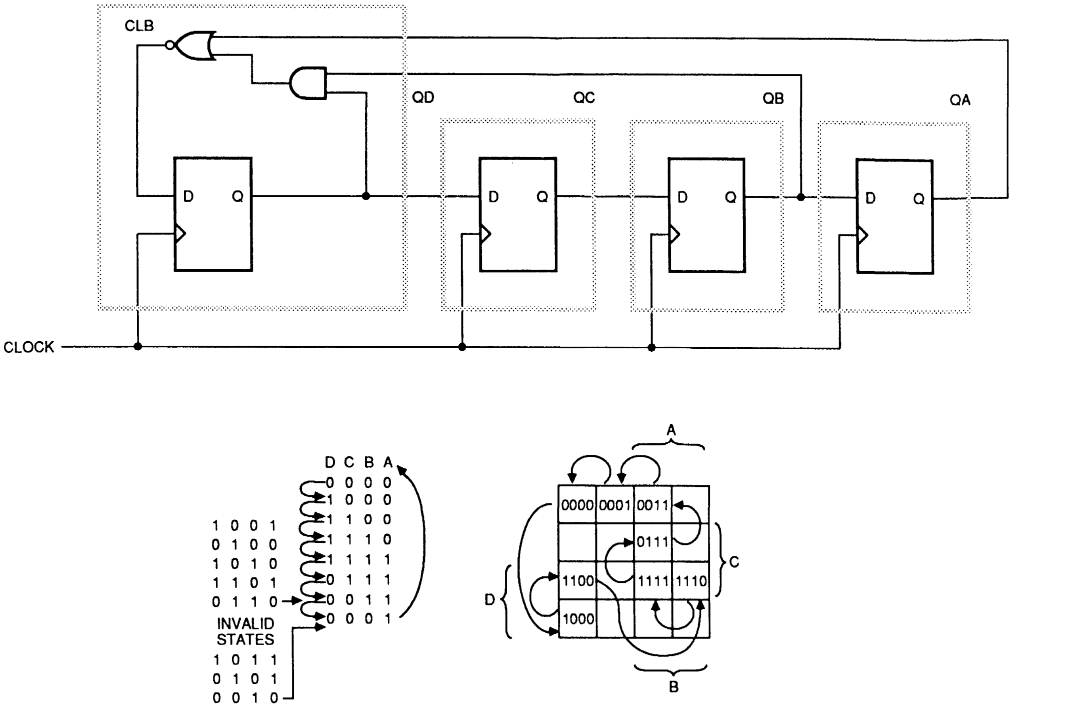
Figure 16b. Synchronous Clock Enable



0010024 16

Figure 16c. Synchronous Parallel Enable

$$F = (\overline{Q_A + Q_B \cdot Q_D})$$
$$= \overline{QA}\,\overline{QB} + \overline{Q}_A \cdot Q_D$$



0010024 17

Figure 17. Divide by 8 Johnson Counter

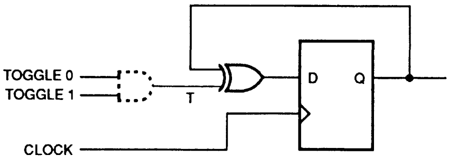


Figure 18a. Diagram of Toggle Flip-Flop

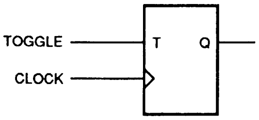


Figure 18b. Symbol of Single-Input Toggle Flip-Flop

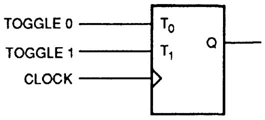


Figure 18c. Symbol of Two-Input Toggle Flip-Flop

0010024 18

of standard package sizes and remaining pin functions were chosen to provide a useful standard product. In an LCA design it is advantageous to implement the specific logic function needed in a way that uses a minimal number of blocks and routing. It is often possible to adapt the logic to minimize the effects of constraints such as availability and function of logic elements and routing resources. This will allow optimization of logic capacity and performance.

Implementation of many designs in LCAs and gate arrays has indicated that the three- and four-variable capability of the Configurable Logic Block is a good balance. Using a conventional logic diagram and grouping combinatorial functions will give an approximate CLB count. In register-intensive designs the number of flip-flops needed determine the logic capacity and related combinatorial functions are merged with the sequential portions.

Figure 22 shows a design example. It incorporates a counter which sets and resets output control bits at specific times in the sequence. Decodes of the desired states with NAND gates drive the asynchronous set and reset inputs of flip-flops. When the counter increments to state D it should asynchronously reset to 0. Since the counter bits in an MSI device are matched, this might operate if it were an MSI device. For implementation of this counter in a gate array or LCA the decodes of the counter states will involve mismatched loading and layout, of various bits of the counter. As a result the decode gates will be likely to produce output spikes which causes erratic operation of the output control flip-flops which use these signals as asynchronous inputs. Although the decode spikes may be too narrow to be noticed during design verification, they might produce erratic output control changes in operation. The decode of the terminal count has the potential for spurious outputs. Even with a valid terminal count decode, a mismatch in counter bit speeds could result in some bits being reset and terminating the reset state decode signal before all bits of the counter are reset. This could leave the counter in an undefined or incorrect state.

The Linear Feedback Shift Register in Figure 23 is an alternative to an asynchronous-reset binary counter. This class of counter 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 design revision also provides timing control of the output by the clock rather than by the accumulation of delay from clock to the counter output, to the state decode, and through the flip-flop to the output as in the 'ripple' imple-

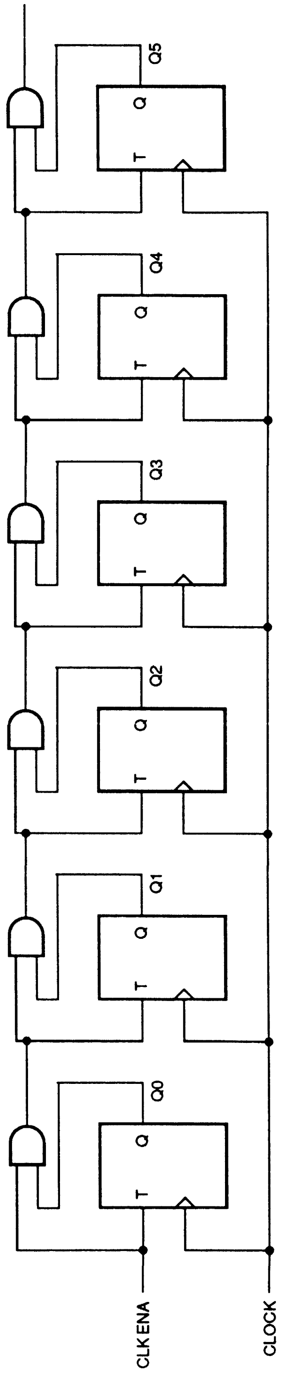


Figure 19. Synchronous Binary Counter with Ripple Carry

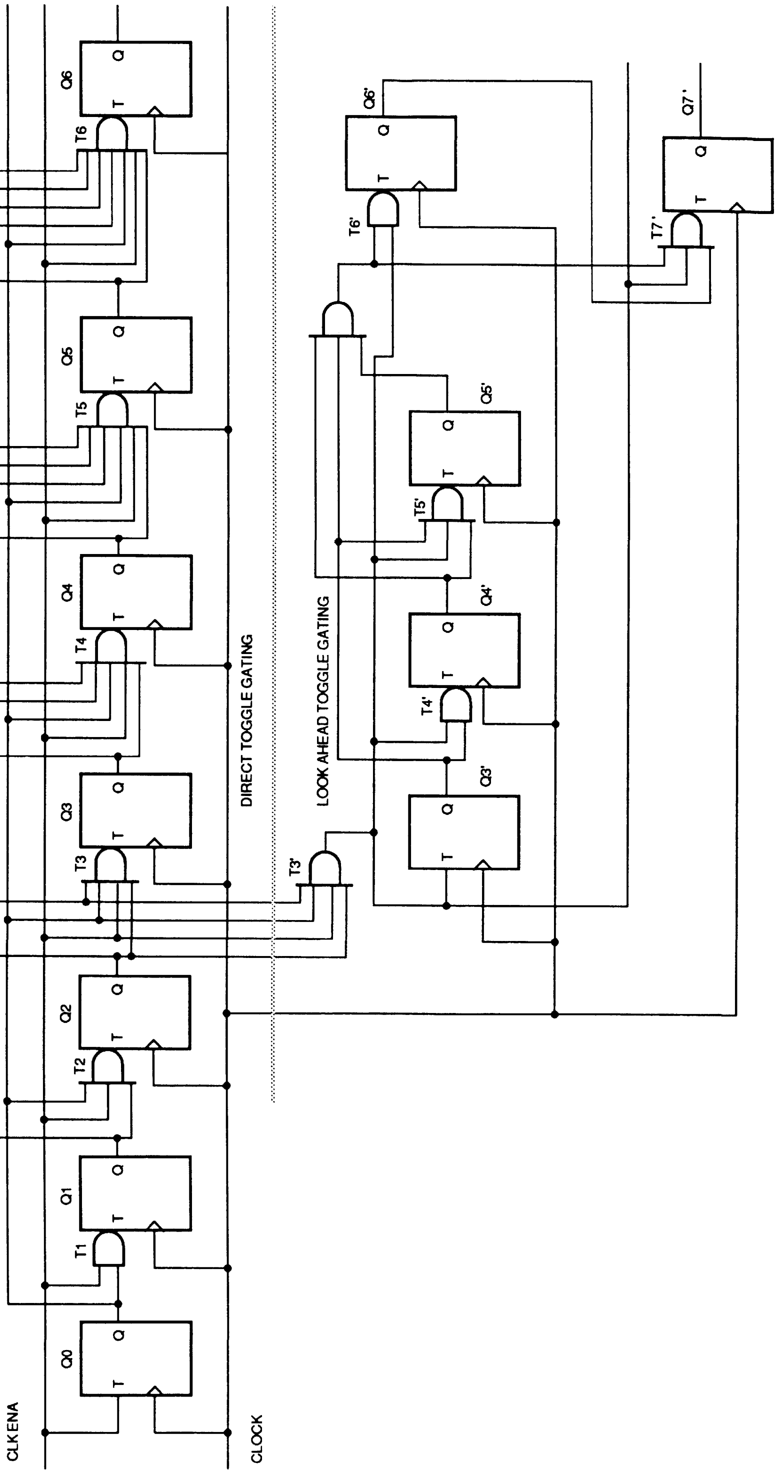


Figure 20. Synchronous Binary Counter

0010024 20

mentation of Figure 22. The use of a fully synchronous counter reset provides more reliable operation and the clock rate may be increased due to the elimination of the terminal count delay path. The implementation of the synchronous reset requires no more resources than the asynchronous reset of the flip-flops.

Clock Skew

The problems of clock skew and asynchronous input signals have accounted for numerous gate array design iterations. Clock skew problems are caused by mismatched delay paths. For example, one flip-flop may be clocked and its lightly loaded new output level may propagate to another flip-flop input, a set-up time before the clock reaches the second flip-flop. The difference in clock timing may occur due to clock gating or unequal routing delays. The clock buffers of the LCA drives a dedicated metal clock distribution network to minimize skew.

Asynchronous Inputs

Asynchronous input signals that affect more than a single flip-flop are another source of design problems. An example is a counter in which an asynchronous parallel load is removed near a clock edge. Various bits of the counter may respond to the clock and change, whereas others retain the previous state. The result is an invalid value in the counter. Another common problem is that of an asynchronous system reset. If the reset signal is removed near a clock-edge time, differing responses by portions of logic may result in invalid states as the logic attempts to begin operation. It is always advisable to synchronize asynchronous signals at their input, as shown in Figure 24.

Asynchronous inputs that require a response to their transition, may be accommodated by a resynchronizer, as shown in Figure 25. The additional input delay may be undesirable in some applications, but it results in a more reliable design when input latency is not a limiting factor. It also acts as a simple noise filter.

MACROS

A macro is a predefined LCA logic function which the user may install in a design. It simplifies the implementation of common logic functions by allowing users to invoke established disk files of executable editor commands easily. Macros in the Xilinx macro library are stored in the \XACT\MACROS directory. Each file has an assigned name which identifies the logic function, and a .MAC extension. A list of macros in the Xilinx library and the order of their parameters can be found in the XACT Quick Reference Card. The XACT Macro Library Manual includes a table of contents indicating macro order in the documentation. In the macro

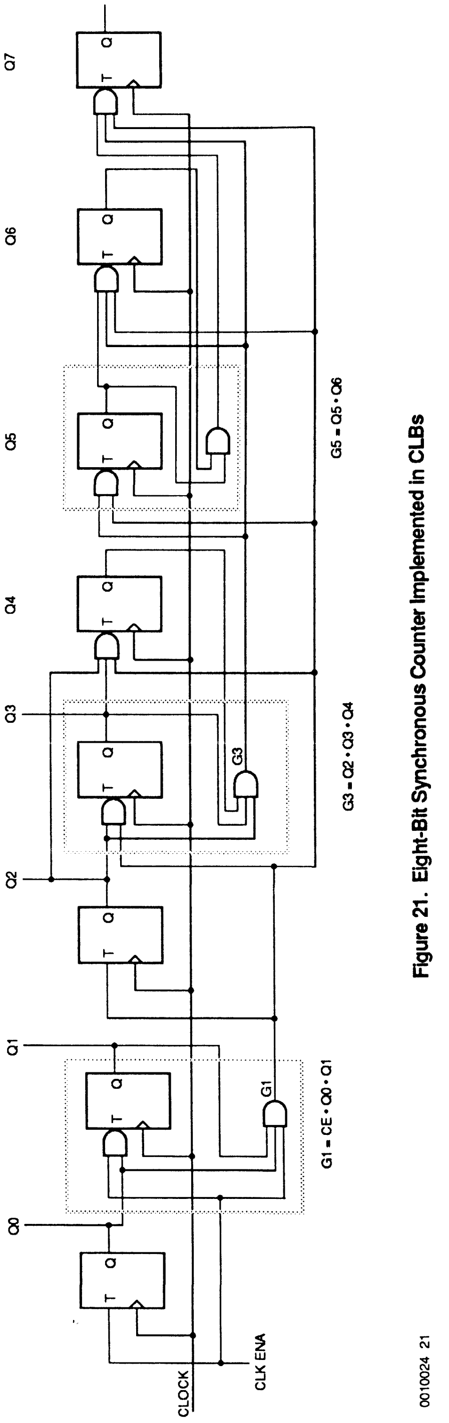
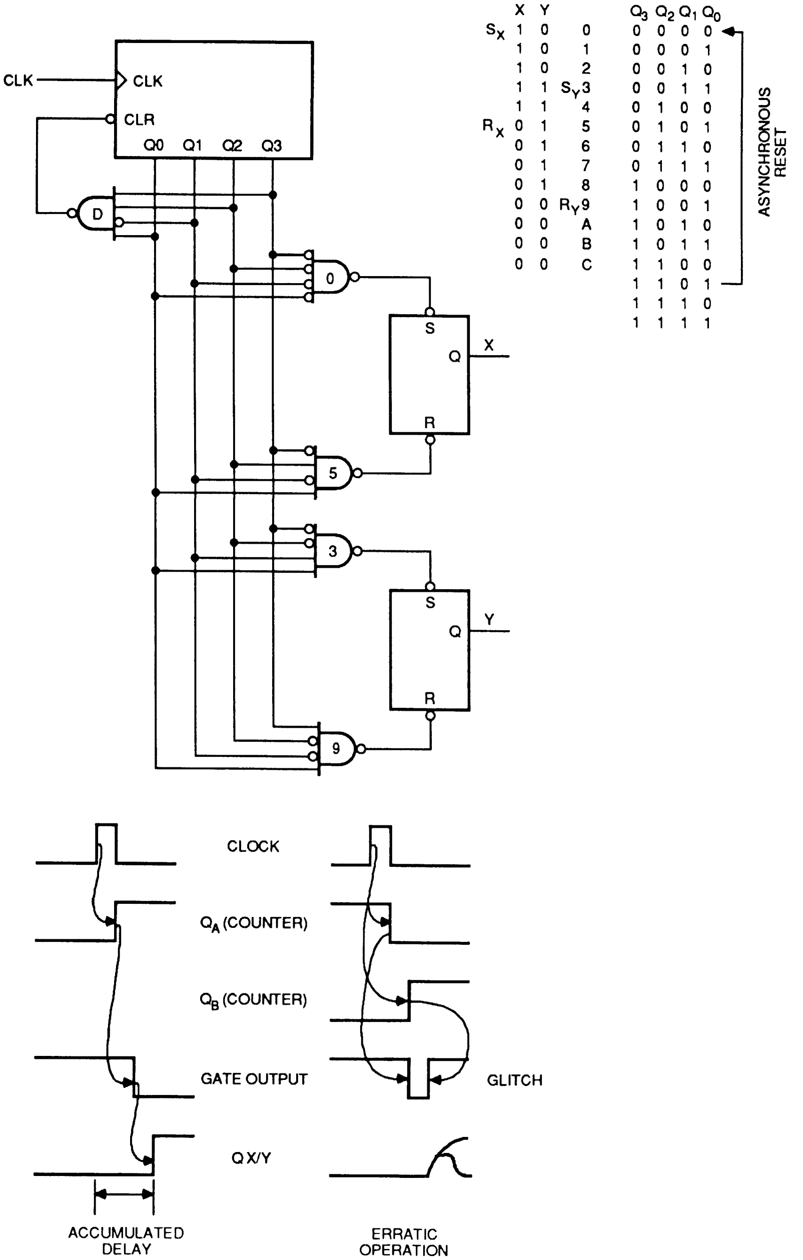


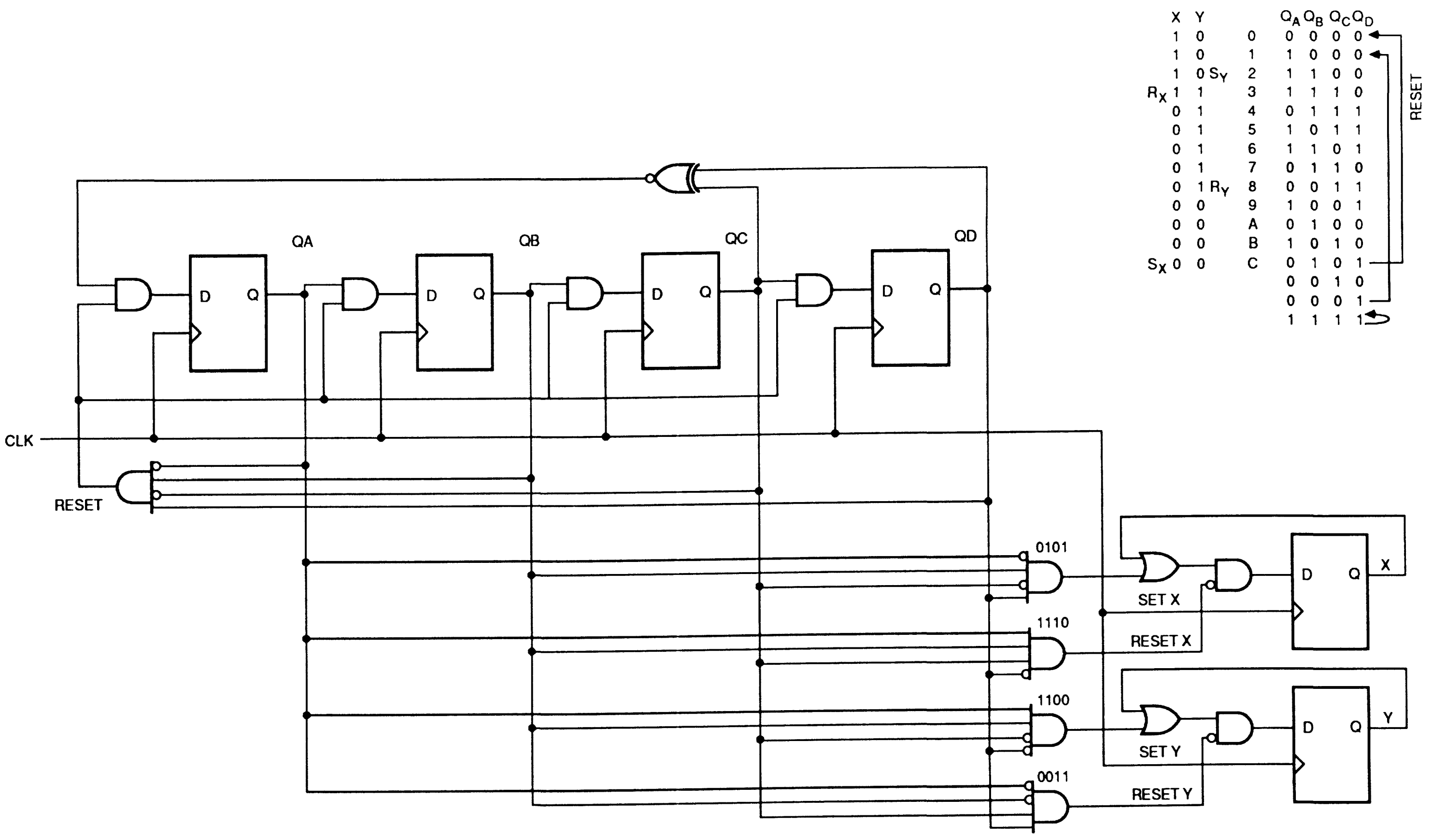
Figure 21. Eight-Bit Synchronous Counter Implemented in CLBs



	X	Y		Q_3	Q_2	Q_1	Q_0
S_X	1	0	0	0	0	0	0
	1	0	1	0	0	0	1
	1	0	2	0	0	1	0
	1	1	S_Y3	0	0	1	1
	1	1	4	0	1	0	0
R_X	0	1	5	0	1	0	1
	0	1	6	0	1	1	0
	0	1	7	0	1	1	1
	0	1	8	1	0	0	0
	0	0	R_Y9	1	0	0	1
	0	0	A	1	0	1	0
	0	0	B	1	0	1	1
	0	0	C	1	1	0	0
				1	1	0	1
				1	1	1	0
				1	1	1	1

0010024 22

Figure 22. Simple "Ripple' State Machines



	X	Y			Q_A	Q_B	Q_C	Q_D
	1	0		0	0	0	0	0
	1	0		1	1	0	0	0
	1	0	S_Y	2	1	1	0	0
R_X	1	1		3	1	1	1	0
	0	1		4	0	1	1	1
	0	1		5	1	0	1	1
	0	1		6	1	1	0	1
	0	1		7	0	1	1	0
	0	1	R_Y	8	0	0	1	1
	0	0		9	1	0	0	1
	0	0		A	0	1	0	0
	0	0		B	1	0	1	0
S_X	0	0		C	0	1	0	1
					0	0	1	0
					0	0	0	1
					1	1	1	1

Figure 23. Synchronous State Machine

0010024 23

documentation, the parameter order is indicated in the syntax statement. Users may create customized macros in their own directories by using the *cutmacro* command.

Execution of a macro involves providing a set of parameters for the executing file. The parameters must be entered in the order required by the macro, and must include such information as an instance name, names of networks providing inputs and block locations. The instance name is used by the macro to compose unique block and net names to distinguish each use of the macro.

An LCA design may be created with edit commands and macro executions, which may in turn be modified with the editor commands. Portions of that design may then be incorporated into a new user macro. For example, it may be desirable to create a one-bit slice of logic which may include several Configurable Logic Blocks. Multiple installations of that macro may be used to implement a logic unit such as a data path. A user-defined macro describing a section of counter may be used to generate a unit of control logic.

In creating macros the designer uses the keyboard or mouse to specify blocks of a design, that are to be included in the macro. The software will assign a parameter for each network needed as an input and each CLB and IOB used. All block names and netnames sourced by the macro blocks will be included in a .MAC file in the current directory. When the macro is invoked the system will prompt for parameters in the order they are needed. The first is always the instance name. This name is used to differentiate one instance of a macro from another instance of the same macro in the same design. The instance name is added as a prefix to the macro's original netnames for all nets driven by blocks included in the macro. This groups net names for Query-Block. The instance name will be added as a suffix to the original block names of all blocks of the macro to allow the first characters of the block names to show in the editor display. All block and net names must be unique. The names A, B, C, D, K, I, O, X and Y are already used for block pin names. AA through last row/column and P1 through highest pin number are used as block names. Some additional names are assigned to configuration and power pins.

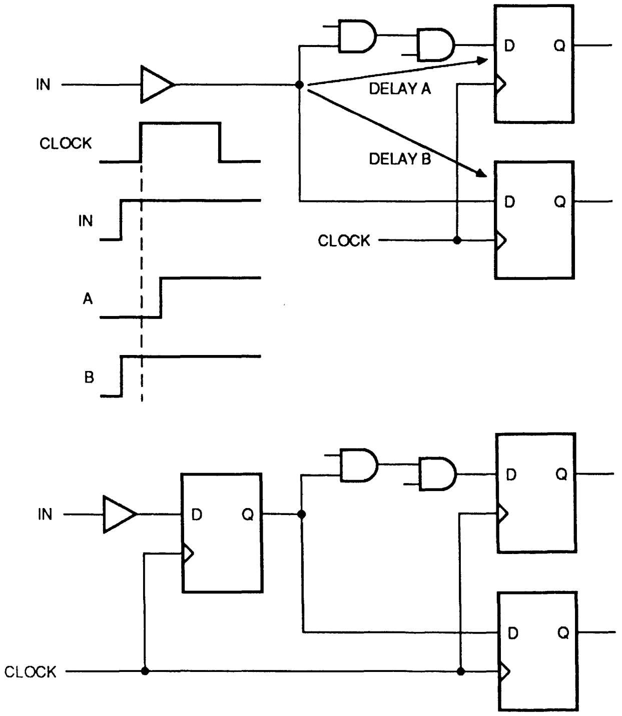


Figure 24. Synchronization of an Asynchronous Input

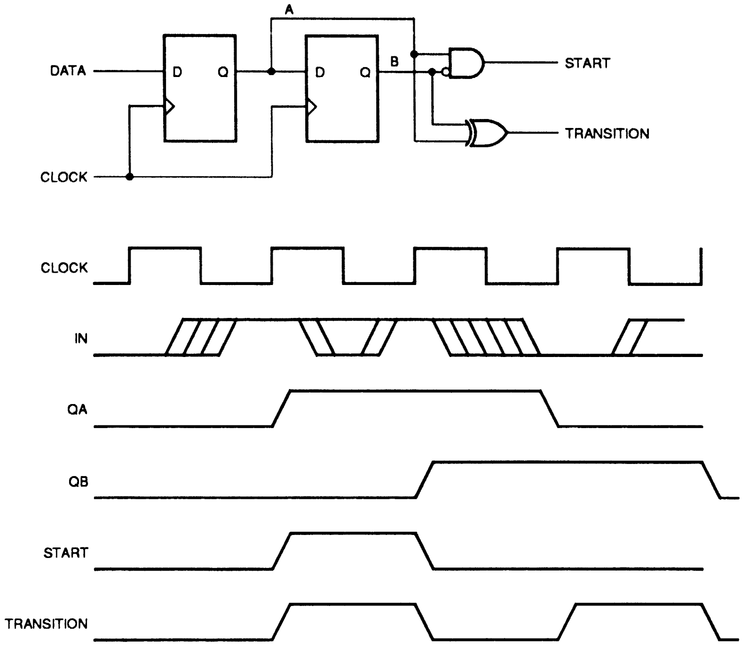
The following is a copy of the macro for the FDR, a simple D-type flip-flop which provides a synchronous reset. The first two lines are comment lines indicating the syntax and parameter order.

```
;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.
```



0010024 25

Figure 25. Data Synchronizer and Filter

The following is a copy of the macro for the GOSC, a simple oscillator which uses two external R-C networks, two input output blocks and one configurable logic block functioning as a set-reset latch.

```
;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 F: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.O %3.T                           | Creates nets of names
Addpin %1Reset %3.I %2.B                                 | 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.
```

The order and concatenation of Xilinx library macros has been chosen for consistency. When a user macro is created the net parameters are chosen in the order that they are encountered by the cutmacro process. The block order is the same as the order selected when created. The user may use a text editor to rearrange parameter order or redefine the concatenation strings. Care should be exercised to maintain a match between all uses of any parameter (%n)

Counter Examples

Table of Contents

XILINX

Counter Examples

INTRODUCTION

Historically, designers used multiple 74-series TTL devices to implement various digital counters. However, in application specific intergrated circuit (ASIC) designs the overall performance and effective utilization of a counter depends on the type of counter used and upon the architecture of the ASIC device employed. The flexible array architecture found in the Xilinx Logic Cell™ Array (LCA) provides the designer with numerous options when implementing counters. Implementing a design on "uncommitted" silicon is different from being forced into an inflexible, fixed architecture. Accordingly, a designer can optimize a counter to meet his specific application needs. Tailoring a counter for an application reduces the amount of wasted silicon resources associated with fixed-architecture logic devices.

Within a Logic Cell Array, a designer can implement various counter types, including

- Binary counters (2^n possible states)
- Johnson counters (2n possible states)
- Linear Feedback Shift Registers (2^n–1 possible states)
- Up/Down counters (typically binary)
- Heterodyne (mixed modulo)

> **Note:** Modulo, when applied to digital counters, refers to the length divider of a non-repeating counter sequence. For example, a counter that produces a terminal count signal every six clock cycles is known as a modulo-6 or a divide-by-6 counter.

By selecting the proper type of counter or even the right mix of counters, a designer maximizes performance and utilization within an LCA design. Choosing the right counter for the application depends on whether the counter application requires

- Binary or non-binary counting sequence, or
- High or low counter modulo, or
- Optimal performance, or
- Optimal register and routing resource efficiency.

Some counter applications require control signals which may include parallel load of data, clock enable, synchronous or asynchronous SET or RESET, UP/DOWN control, or others. Synchronous binary counters are widely-used in digital design but their complexity can degrade their overall speed and effectiveness. The extra routing and logic required to implement a wide toggle lookahead-carry function, for example, quickly consumes the resources available in any logic device (a toggle lookahead-carry should not be confused with the lookahead-carry used in arithmetic logic). For speed, ease of routing, and glitch-free decoding, Johnson counters are often the best solution for applications of modulo ten or less. If a non-binary, pseudorandom counting sequence is acceptable, Linear Feedback Shift Register (LFSR) counters can best implement counts above modulo 32.

The purpose of this application note is to help the designer choose which counter to use for specific applications in the XC2064 and XC2018 Logic Cell Arrays (LCAs). Another goal is to help the designer start thinking about alternatives to 7400-series counters. There are various means of attacking the same counter application. A designer should use a specific type of counter to fill a specific application—not because it is the only counter type available but because it is the best counter for the specific application. The LCA gives the designer added flexibility, and this application note suggest ways to take advantage of this flexibility in counter designs. Descriptions of various counter applications are provided with examples wherever possible.

Resource Efficiency

Binary counters are more register efficient than other counter types since their capability increases as the exponent of the number of registers, as shown in Figure 1. Binary counters have up to 2^n possible states. Sometimes, however, the more complex routing of a binary counter can make it less silicon efficient than a comparable counter of a different type. With only one less state than a binary counter of comparable size (2^n–1), Linear Feedback Shift Registers follow closely behind in counting capability. The ease of routing within an LFSR counter design sometimes outweighs their non-binary, pseudorandom counting sequence. Lowest on the list of register efficiency is the Johnson or Mobius counter. With only 2n possible states, Johnson counters are less resource-efficient for modulos much higher than 12 (six registers). However, extremely high-performance Johnson counters can operate at speeds near the overall toggle frequency of the LCA since the counter logic and routing is simple.

Since each Configurable Logic Block (CLB) within the XC2064 and XC2018 Logic Cell Arrays has a single flip-flop or storage element, there is a direct relation between flip-flop efficiency and CLB efficiency. These terms are practically synonymous when referring to the counter bits required to implement a counter. Since some designs require additional logic beyond the logic associated with the flip-flop inside a CLB, some forms of counters require more CLBs than flip-flops.

BINARY COUNTERS

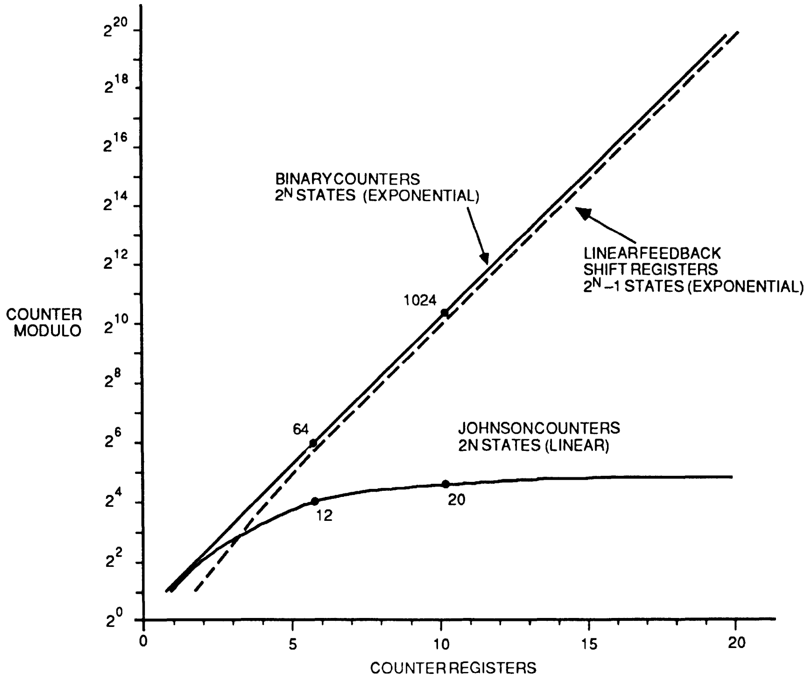
Overview

This section addresses the design and associated topics regarding binary ripple, ripple-carry, and lookahead-carry counters. The performance of a binary counter increases with the amount of lookahead-carry performed at each counter stage. Simple ripple counters have no lookahead-carry and are the lowest-perfomance binary counter. Higher-performance binary counters require additional silicon resources to decode and propagate the carry signals. The ripple-carry and lookahead-carry counters require partial or full decoding of all of the previous counter bits. This requires extra logic and routing resources for the carry signals but buys increased performance.

Binary counters operate differently depending on the amount of lookahead carry. Simple ripple counters operate asynchronously since they do not generate carry signals, whereas ripple-carry and lookahead-carry counters are synchronous. Typically synchronous design is safer since it is more immune to glitches.

Ripple Counters

In simple binary ripple counters, as shown in Figure 2, carry signals are not generated. The output of each counter stage asynchronously clocks the next counter stage. The resource requirements to implement a ripple counter are low—only one CLB per counter bit regardless of counter length. The routing for a binary



0010023 1

Figure 1. Different types of counters require different amounts of registers and resources for the same task. The capability of binary counters increases exponentially with the number of bits while Johnson counters increase only linearly.

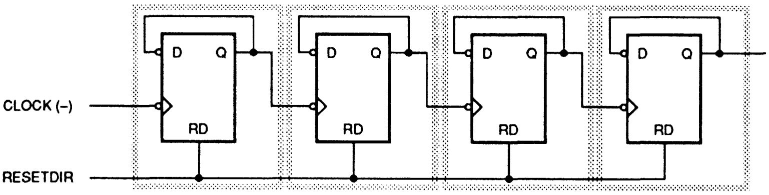
ripple counter is also simple. The penalty for this simplicity is lower performance. Each counter bit degrades the overall performance by a CLB delay time. The overall counter clock period must be greater than or equal to the total delay of all the CLBs used to implement the counter, as shown in Equation 1.

Ripple Counter Clock Period = N • (Clock to Output Delay [1]

where

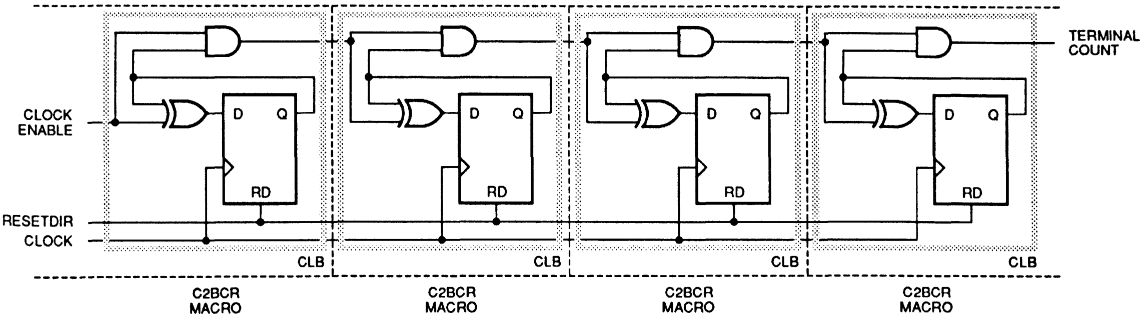
N	= number of ripple counter flip-flops
Clock to Output	= Tcko for K-clock input
	= Tcco for C-clock input
	= Tcio for logic clock input

Binary ripple counters are primarily used only in applications that require optimal device utilization without regard to performance. The increased silicon utilization is gained by eliminating carry signals. However, the flip-flops within a binary ripple counter are asynchronously toggled in operation. There are synchronous forms of this counter that have better performance and require the same resources. This is true because of the flexible array-type architecture of the LCA and because of the capability of a single CLB. An example is shown in Figure 3. This synchronous binary ripple-carry counter still requires only one CLB per bit. Since the ripple-carry includes only the previous counter stage, resource requirements are minimal. Its synchronous design affords more reliable operation.



0010023 2

Figure 2. Binary ripple counters are simple to implement and can be cascaded to nearly any desired length. Their disadvantage stems from their asynchronous operation and their degraded performance with each additional counter bit. They are not recommended for most designs because of their asynchronous operation.



0010023 3

Figure 3. A single-bit ripple-carry counter requires as few resources as a standard ripple counter but has synchronous operation. While highly CLB efficient, this design does have performance below counters with greater amounts of ripple-carry and below all lookahead-carry counters.

Although the single-bit ripple-carry counter requires few resources, it still suffers from lower performance, much like the standard ripple counter. It also exhibits lower peformance with each additional CLB or flip-flop.

Ripple-Carry Counters

The design shown in Figure 3 is just one example of a binary ripple-carry counter. Ripple-counters have been used by designers for quite some time. In the past, designers typically implemented large binary ripple-carry counters with conventional 74-series logic devices. Designers tended to cascade the ripple-carry output of one 4-bit counter segment with the enable of the next 4-bit counter segment to build larger counters. Counter implementations within an LCA do not lend themselves to the "one size fits all" approach of the 74-series devices, since the architectures are different. However, some of the general ideas still apply.

Table 1 shows the wide variety of possible binary ripple-carry counters to eight bits in length. Larger ripple-carry counters are possible but there are far too many permutations to mention them all. The counters shown in the table consist of smaller counter segments, each with its own terminal count and clock enable. Larger counters can be built by cascading these segments together. The terminal count of one segment feeds the clock enable of the next segment and so on. The cascaded connection of the terminal count to clock enable is called a *ripple-carry.*

By cascading various counter segment macros together in the proper order, a designer can construct counters with various performances and control inputs. The table indicates the counter size in bits and in total modulo. The table also indicates whether CLOCK ENABLE (CE) or TERMINAL COUNT (TC) is available for that particular counter. The number of CLB logic levels required to implement the counter function is displayed under "Ripple Levels". The higher the number of ripple levels, the slower the counter will run since each ripple level corresponds to a single CLB block delay.

The actual number of CLBs needed to implement a counter function depends on the number of control signals required. Wheras a simple CLOCK ENABLE (CE) and RESET take minimal resources, a PARALLEL ENABLE (PARENA) typically requires an additional CLB as shown in the table.

DESIGN EXAMPLE 1. Escaping from the TTL "one size fits all" approach. Building an eight-bit ripple-carry counter with 74-series TTL and with an LCA.

Assume that a counter application requires an eight-bit binary counter with reset. Figure 4a shows a common implementation of an eight-bit counter using two 74-161 TTL devices. A designer unfamiliar with the capability of the LCA might choose a similar implementation using two of the 74-161 counter macros available within the XACT™ Development System. However, a direct one-for-one substitution is wasteful. It would require sixteen CLBs to implement the two 74-161 equivalent circuits.

The Xilinx 74-161 macro contains the same functionality as a 74-161 device. Typically however, not all of the resources of a 74-161 are used in a design. This is the case in Figure 4a. Some of the CLB capability has been wasted on circuitry which will not be used for the design (*Note:* The Xilinx FutureNet Schematic Capture Library and Conversion Package reduces the macros to their primitive gate levels and then frees any unused resources).

A more rational approach uses only the required amount of circuitry to implement the design. For example, it is simpler and more resource efficient to build the eight-bit counter with the generic counter macros available within XACT. Table 1 shows some of the possible macro combinations used to build an eight-bit counter.

An equivalent counter built with two C8BC-rd counter macros and one C4BC-rd counter macro is shown in Figure 4b. The eight-bit counter is built with two 3-bit pieces and a single 2-bit piece. This implementation was chosen over using two 4-bit pieces since the former has superior performance. The 4-bit pieces each have two ripple-carry levels each. Therefore the eight-bit counter implemented with 4-bit pieces will have four ripple-carry delay levels overall. A similar implementation based on the two 3-bit pieces, and a 2-bit piece only has three ripple-carry delay levels, and therefore has higher performance.

Building the 8-bit counter by cascading smaller counter macros reduces the number of required CLBs from sixteen to ten while simultaneously increasing performance.

Generally large ripple-carry counters should be built with multiple 3-bit pieces instead of 4-bit pieces since there are fewer ripple-carry delays through a 3-bit piece. The various ripple-carry possibilities are described in Table 1. When given a schematic drawn with 74-series devices, a designer should consider the function required for the application, and not blindly implement a one-for-one 74-series logic substitution.

DESIGN EXAMPLE 2. Avoiding the pitfalls of asynchronous design!

A common practice when designing with 74-series devices is to use the terminal count of a counter as an asynchronous signal to either RESET the counter or to load the counter with a predetermined value. Figure 5a

XILINX XACT COUNTER MACROS

COUNTER SIZE									AVAILABLE SELECTIONS		RIPPLE LEVELS	CLKENA OR CE & RESET	CLKENA & PARENA
2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	(Modulo)	CE	TC		CLBs	CLBs
C4B								(4)	X	X	1	3	3
FT	C4B							(8)	-	X	1	3	4
C4B		FT							X	-	1	4	4
C8B									X	X	1	4	5
FT	C4B		FT					(16)	-	-	1	4	5
FT	C8B								-	X	1	5	6
C8B			FT						X	-	1	5	6
C4B		C4B							X	X	2	4	6
FT	C8B			FT				(32)	-	-	1	6	7
FT	C4B		C4B						-	X	2	5	7
C4B		C4B		FT					X	-	2	5	7
C8B			C4B						X	X	2	6	8
FT	C4B		C4B		FT			(64)	-	-	2	6	8
FT	C8B			C4B					-	X	2	7	9
C8B			C4B		FT				X	-	2	7	9
C8B			C8B						X	X	2	8	10
C4B		C4B		C4B					X	X	3	6	9
FT	C8B			C4B		FT		(128)	-	-	2	8	10
FT	C8B			C8B					-	X	2	9	11
FT	C4B		C4B		C4B				-	X	3	7	10
C8B			C8B			FT			X	-	2	9	11
C4B		C4B		C4B		FT			X	-	3	7	10
C8B			C4B		C4B				X	X	3	8	11
FT	C8B			C8B			FT	(256)	-	-	2	10	12
FT	C4B		C4B		C4B		FT		-	-	3	8	11
FT	C8B			C4B		C4B			-	X	3	9	12
C8B			C4B		C4B		FT		X	-	3	9	12
C8B			C8B			C4B			X	X	4	10	13
C4B		C4B		C4B		C4B			X	X	4	8	12

NOTE: THE ABOVE COMBINATIONS MAY BE CASCADED FOR LARGER MODULO.
CE/CLKENA = CLOCK ENABLE
PARENA = PARALLEL ENABLE
FT = TOGGLE FLIP-FLOP
TC = TERMINAL COUNT

Table 1. Binary ripple-carry counters from two to eight bits.

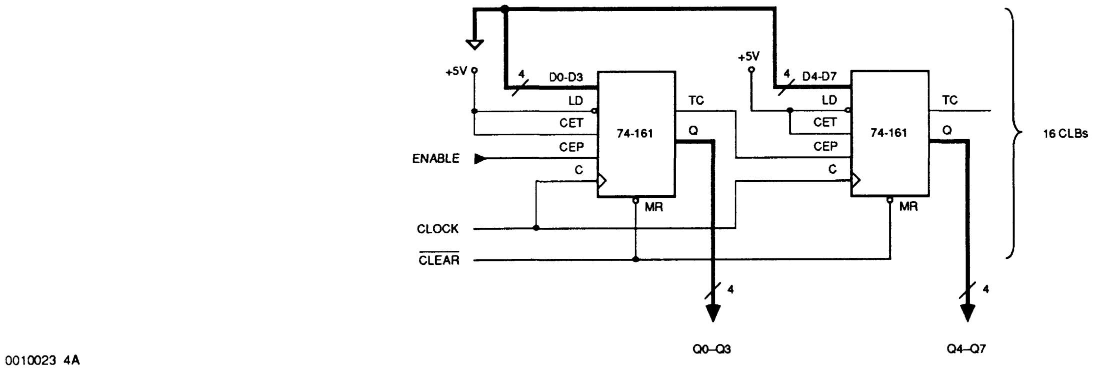


Figure 4a. An eight-bit counter with reset built with two 74-161 TTL devices. Notice that not all of the device connections are used to implement the function required. A direct 74-series substitution proves wasteful. Use only the logic required to implement the required function.

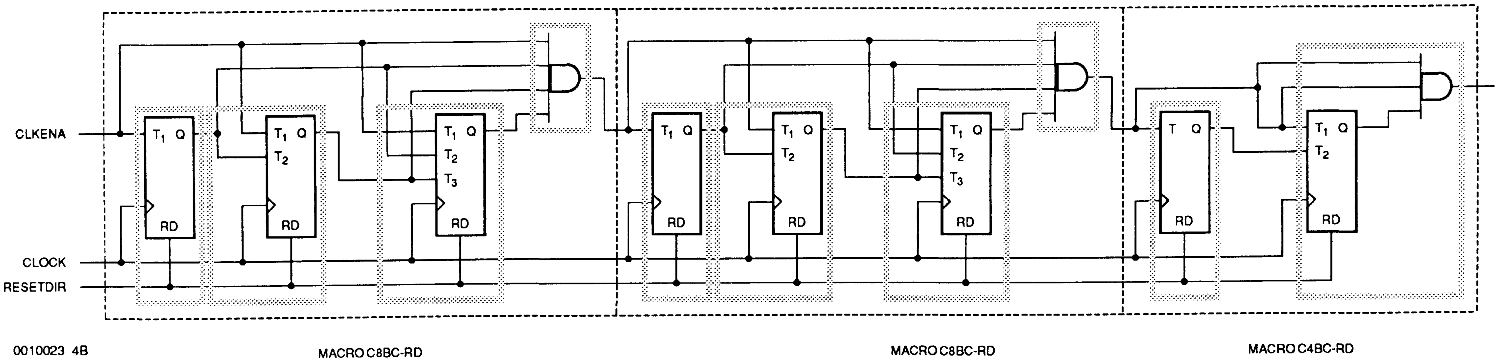
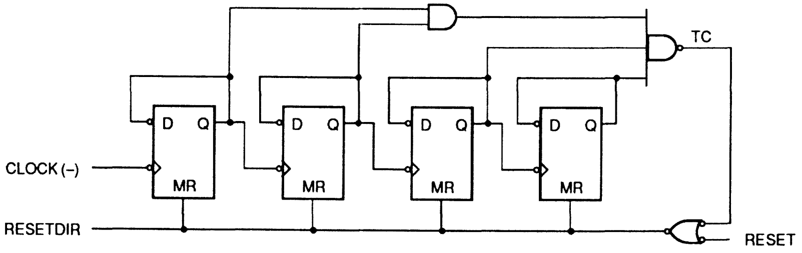
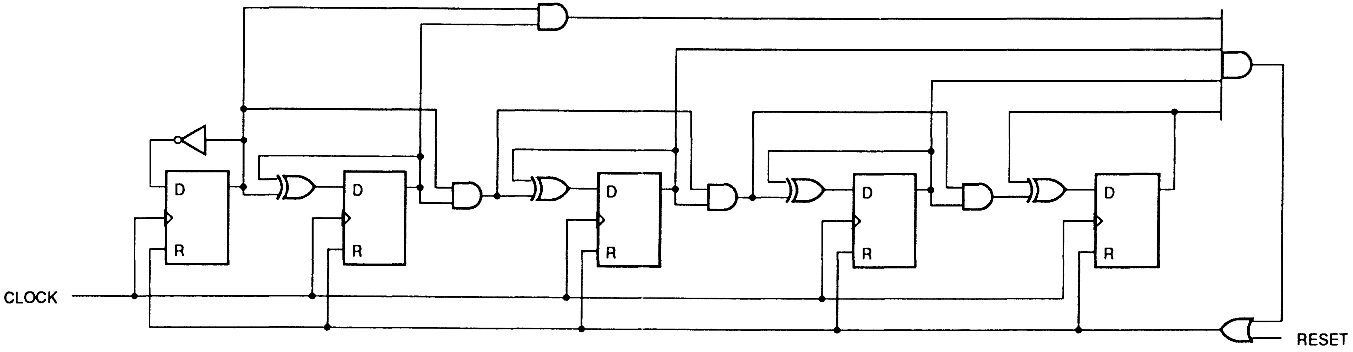


Figure 4b. A better implementation of an eight-bit ripple-carry counter based on various macros from the XACT Development System. When designing with an LCA, an engineer can tailor his logic so that the LCA performs the required function with the minimum amount of logic.



0010023 5A

Figure 5a. The asynchronous design practices of 74-series TTL devices are discouraged. An asynchronous design is less reliable and riskier since it is unforgiving to unknown design conditions.



0010023 5B

Figure 5b. Synchronous design provides more relable operation.

demonstrates how a similar design might be implemented within an LCA although the asynchronous implementation is not recommended. The drawback to this design is that it is potentially unreliable. The terminal count (and RESET signal) is generated by ANDing all of the register outputs. When the counter reaches state 1111, the asynchronous RESET signal is asserted. Not all of the registers will be reset simultaneously because of the delay along the RESET network. Since the RESET signal is formed by the AND of the register outputs, the RESET signal will no longer be asserted and the remaining registers may fail to reset if any one of the registers is reset before the others.

The asynchronous RESET signal will also experience decoding glitches as the various counter bits propagate through their nets to the AND gate. Any glitch on this signal can partially reset the counter.

An improved design appears in Figure 5b. In this example the RESET signal is asserted synchronously. This gives the synchronous RESET signal until the next clock edge to propagate to all of the CLBs and also prevents any of the CLBs from being reset by glitches on the RESET line. Since both asynchronous and synchronous RESET require a single CLB input, no more resources are required to build a completely synchronous design than the more risky asynchronous design. In general, synchronous design practices apply not only to counter design, but to all digital designs.

Lookahead-Carry Counters

There are applications where binary ripple and ripple-carry counters are too slow. In such cases, synchronous binary counters with lookahead-carry are an answer. Building high-performance synchronous binary counters within an LCA is a simple process but, it involves a few special techniques.

High-performance synchronous counters derive their speed by decoding lookahead-carry signals from all of the previous counter stages. There is not a ripple-carry as described earlier—all of the previous counter outputs are decoded to form a single lookahead-carry signal for each counter stage. Combining all of the ripple-carry into a single lookahead-carry reduces overall design delays and increases performance.

Decoding lookahead-carry signals requires additional Configurable Logic Blocks (CLBs) in addition to more routing resources. The additional CLBs allow faster counters to be built. Higher performance is obtained by increasing the number of CLBs required to perform the function (divide and conquer!).

The generalized equation for synchronous binary lookahead-carry counters is given in Equation 2.

$$\begin{aligned} Q0 &= ((CE \bullet \overline{RESET} \bullet \overline{PARENA}) \oplus Q0) \\ &\quad + PARENA \bullet D0 \\ Q1 &= ((CE \bullet \overline{RESET} \bullet \overline{PARENA} \bullet Q0) \oplus Q1) \\ &\quad + PARENA * D1 \\ &\bullet \\ &\bullet \\ &\bullet \\ Qn &= ((CE \bullet \overline{RESET} \bullet \overline{PARENA} \bullet Q0 \bullet Q1 \bullet \ldots \bullet Qn{-}1) \\ &\quad \oplus Qn) + PARENA * Dn \end{aligned} \qquad [2]$$

where

CE	= Clock Enable
RESET	= Register Reset
PARENA	= Parallel Enable
Qn	= Register Value
Dn	= Register Input Value

The complexity of the counter can be reduced by omitting one or more of the control signals (i.e., CE, RESET, etc.). For example, one way to remove RESET is to realize that all of the registers in the LCA are reset after configuration or any time the device $\overline{\text{RESET}}$ line has been asserted LOW.

From the generalized equation shown above, the overall complexity of the design grows with each additional counter bit. Since the CLBs within the XC2064 and XC2018 have up to four and sometimes five inputs, the logic equations for the counter bits need to be partitioned into smaller pieces. Design Example 3 describes the partitioning of a 10-bit counter.

DESIGN EXAMPLE 3. A 10-bit synchronous binary counter with CLOCK ENABLE (CE) and RESET_DIRECT (RESET)

The 10-bit binary counter requires 14 Configurable Logic Blocks (CLBs)—ten blocks to implement the ten counter registers plus four additional blocks to generate the lookahead-carry logic. The clock signal (CLOCK) connects to the global clock buffer (upper left-hand corner of the die) to minimize clocking skew to each of the counter registers. The RESET_DIRECT signal feeds into the reset-direct input to the flip-flop in each CLB (the D-input to the CLB).

The set of logic equations for each CLB used in this counter appears in Equation 3.

```
Bit0 = ClkEna ⊕ Q0                                    [3]
Bit1 = (ClkEna * Q0) ⊕ Q1
Bit2 = (ClkEna * Q0 * Q1) ⊕ Q2
q02_CE = Q0 * Q1 * Q2 * ClkEna      ;lookahead-
                                     carry of bits
                                     0 thru 2
```

```
Bit3 = q02_CE ⊕ Q3
Bit4 = (q02_CE * Q3) ⊕ Q4
Bit5 = (q02_CE * Q3 * Q4) ⊕ Q5
q35 = Q3 * Q4 * Q5              ;lookahead-carry
                                 of bits 3 thru 5
Bit6 = (q02_CE * q35) ⊕ Q6
q36 = Q3 * Q4 * Q5 * Q6         ;lookahead-carry
                                 of bits 3 thru 6
Bit7 = (q02_CE * q36) ⊕ Q7
Bit8 = (q02_CE * q36 * Q7) ⊕ Q8
q78 = Q7 * Q8                   ;lookahead-carry
                                 of bits 7 and 8
Bit9 = (q02_CE * q36 * q78) ⊕ Q9
```

There are two more vertical long lines than horizontal long lines in each routing channel. The counter is placed in a vertical orientation so that both of the high-fan-out lookahead-carry signals (q02_CE and q36) connect to long lines. By routing both the q02_CE and q36 lookahead-carry signals on long lines, the routing-dependent delays are greatly reduced and the counter performance is increased accordingly. The layout for this counter is shown in Figure 6 on an XC2064 die picture. An eight-bit example of a lookahead-carry counter is expressed in schematic form in Figure 8c.

DESIGN EXAMPLE 4. Loading a counter with an initial value.

Preloading a counter with an initial value is desired in some designs. Within an LCA design, this is accomplished with the asynchronous SET and RESET inputs to a CLB or through synchronous SET and RESET inputs. Figure 7 shows a four-bit binary counter which can be set to an initial value of 0100 binary before counting begins. This is accomplished by having three of the CLB registers asynchronously RESET through input-D and one CLB register asynchronously SET through input-A.

Binary Counter Summary

An n-bit binary counter has 2^n possible states. Even within the general category of binary counters, there are various ways to implement a counter function. Typically, highly resource-efficient binary counters have degraded performance, whereas high-performance binary counters require additional resources.

Use of ripple counters is discouraged because of their asynchronous operation. Ripple-carry counters are easily constructed with the macros available within the Xilinx XACT Development System. For high-performance binary counter operation, lookahead-carry counters are recommended.

As an illustrative example, Figure 8 shows three eight-bit counters. The first (Figure 8a) is implemented as a ripple counter, the second (Figure 8b) as a ripple-carry counter and the third (Figure 8c) as a lookahead-carry counter. Note the resource requirements and the relative performance of each implementation.

JOHNSON COUNTERS

Overview

A Johnson (or Mobius) counter can be thought of as a special type of shift register. In a Johnson counter, the last bit of the shift register is inverted and then fed back into the first bit, as shown in Figure 9. Only a single bit changes during a clock transition, as shown in Figure 9. Therefore each state, or contiguous states of a Johnson counter can be decoded, without glitches, using only a two-input AND gate. This structure and counting sequence allows an N-bit Johnson counter to count up to 2n possible states as opposed to the 2^n possible states allowed in a binary counter.

Since the placement and routing of Johnson counters is simple, extremely fast Johnson counters are possible. A well designed Johnson counter can approach the toggle frequency of the LCA (minus any routing delays). One drawback to Johnson counters is that their capability increases only linearly with additional bits as opposed to exponentially like binary counters. Johnson counters become less efficient at modulos greater than ten or twelve. Another potential drawback stems from the invalid counter states possible in a Johnson counter sequence (there are 2^n–2n possible invalid states). Invalid states can be cleared eventually with a small amount of additional logic (see Design Example 5).

Johnson counters can only be effectively implemented in the flexible array architectures like those found in gate arrays and in LCAs. In array architectures, the signals from internal registers can be easily routed to other registers within the device. A Johnson counter can even be built with the storage elements located in the Input/Output Blocks (IOBs) of the LCA (see *The Ins and Outs of Logic Cell Array Input/Output Blocks*). Similar implementations in PLA-type (sum-of-products) architectures are ineffcient since registers are typically hard-wired to outputs.

DESIGN EXAMPLE 5. Building Johnson counters with odd modulos (2n–1).

Odd modulo Johnson counters (2n–1) require few more resources than their even modulo (2n) counterparts. A standard Johnson counter uses only an inverting feedback from the last bit to the first bit. In a modulo 2n–1 Johnson counter, a NOR of the last two

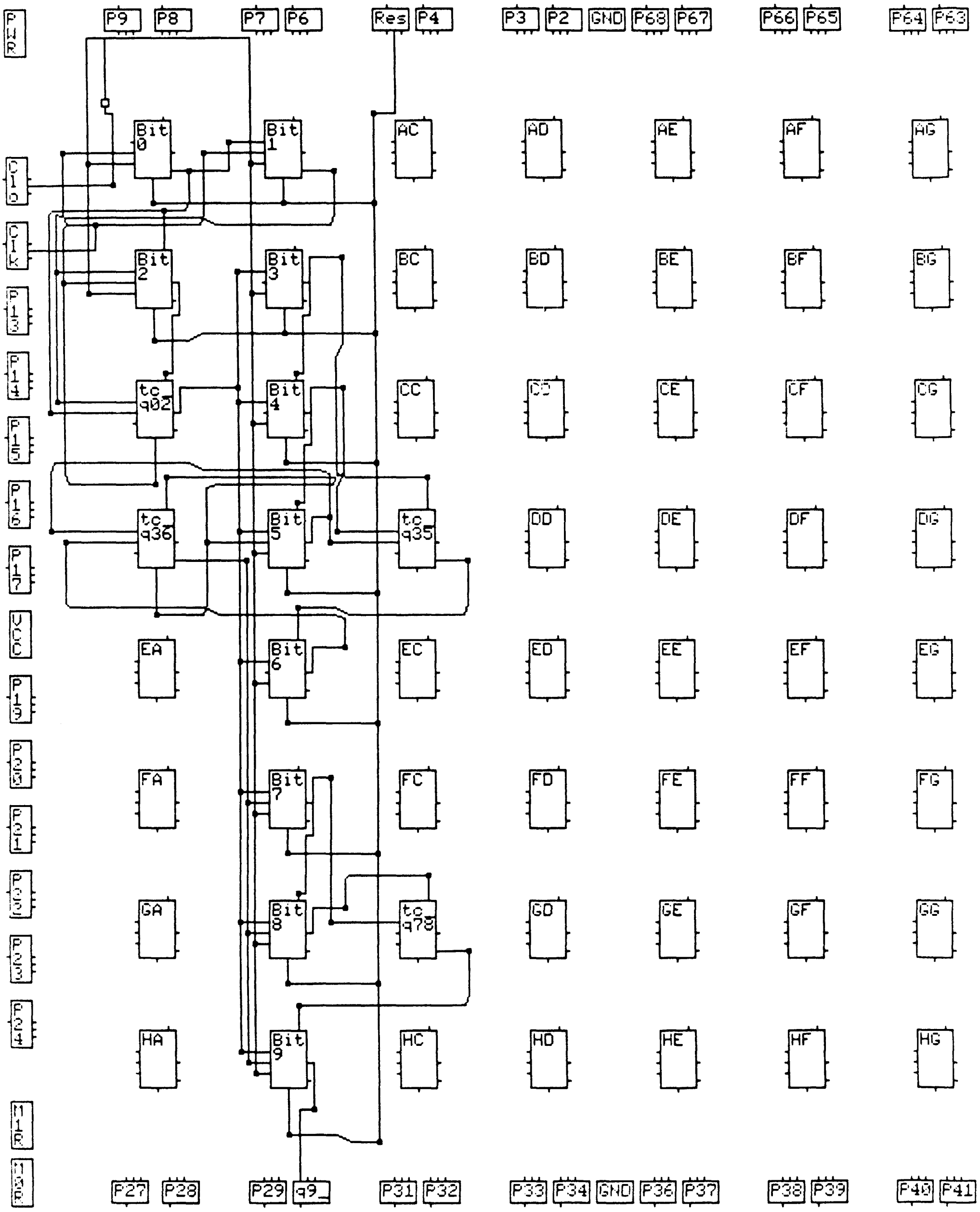


Figure 6. Wide, lookahead-carry counters should have a vertical orientation to make maximum use of the vertical long lines to propagate lookahead-carry signals.

bits fed back into the first bit forces the counter to skip a single state, as shown in Figure 10 for a four-bit Johnson counter. The designer should be aware however, that not all of the counter states can be decoded glitch-free in an odd-modulo Johnson counter. A potential glitch may occur whenever two bits change during one of the clock transitions, as indicated in Figure 10.

The added NOR feedback gives the counter an additional capability. If the Johnson counter ever gets into an "unallowed" state, the NOR feedback eventually forces the counter back into its normal counting sequence, again shown in Figure 10.

DESIGN EXAMPLE 6. Building a fast clock divider using a Johnson counter.

In many designs, counters are used to divide an incoming clock signal to derive lower-frequency clock signals for other portions of the logic. Sometimes this proves difficult if the frequency of lthe incoming clock is extremely high, or if only a binary counter is allowed because of the fixed architecture of the logic device.

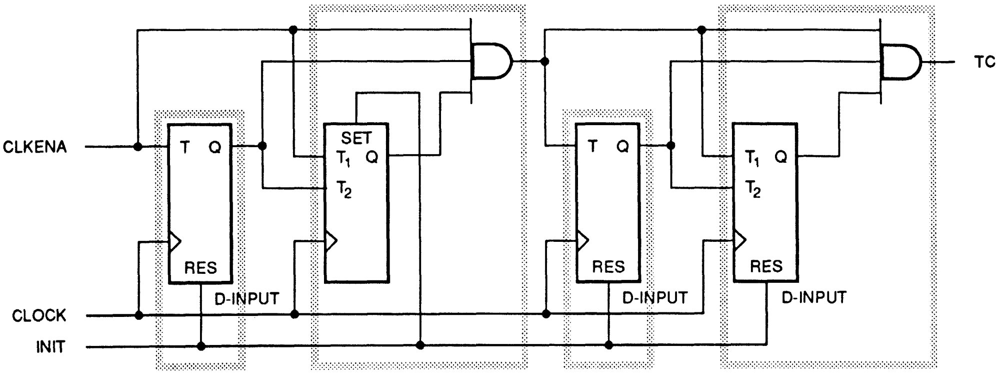
One method of building a clock divider is to use a Johnson counter. The simple placement and routing of Johnson counters makes them extremely high performance, if designed correctly. A well-designed Johnson counter can operate at nearly the maximum toggle frequency of the LCA. Another benefit of a Johnson counter is its ability to generate glitch-free decoded outputs of the counter state by ANDing only two of the register outputs. The circuit shown in Figure 11 demonstrates the use of a Johnson counter to derive a variety of clock frequencies and phases from a single incoming clock source. Typically a clock that is nearly any integral divisor of the input frequency can be derived. Since the counter in Figure 11 is modulo 6, the frequency can be divided by 2, 3, or 6. Duty-cycles are available in 100%/DIVIDER increments (or 16.67% for the three-bit example). Phase shifting can be done in increments of 360 degrees/MODULO or 60 degrees for the three-bit Johnson counter (360/6 = 60).

Table 2 shows some of the various modulos, duty-cycles, and phase shifting that can be derived from the three-bit Johnson counter shown in Figure 11.

DESIGN EXAMPLE 7. Building fast binary counters with a Johnson counter prescaler.

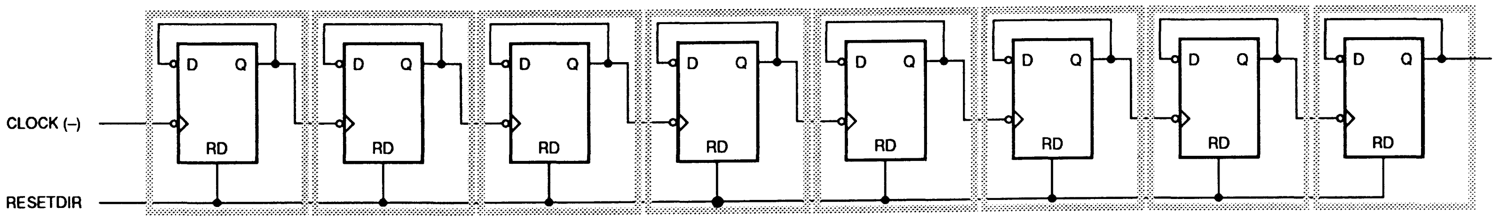
Higher frequencies reduced with Johnson counters can be fed into lower performance counters. For example, instead of requiring a 10-bit synchronous binary counter to operate at near the toggle rate, the same counter can be designed with a two-bit Johnson counter as a two-bit prescaler, and a lower-frequency synchronous binary counter as the most-significant eight-bits. The entire counter can appear to operate at the toggle frequency when, in actuality, only the Johnson counter operates that quickly and the binary counter operates at a quarter of that speed. Figure 12 shows a counter built with both a Johnson counter and a binary ripple-carry counter.

Since binary counters are capable of counting to higher modulos than Johnson counters with the same number of flip-flops, they are more resource efficient for higher modulos. But since Johnson counters can be built with minimal delays, they are higher performance than binary counters. By mixing the two types into a single "hybrid" counter, the designer can optimize both speed and resource utilization.



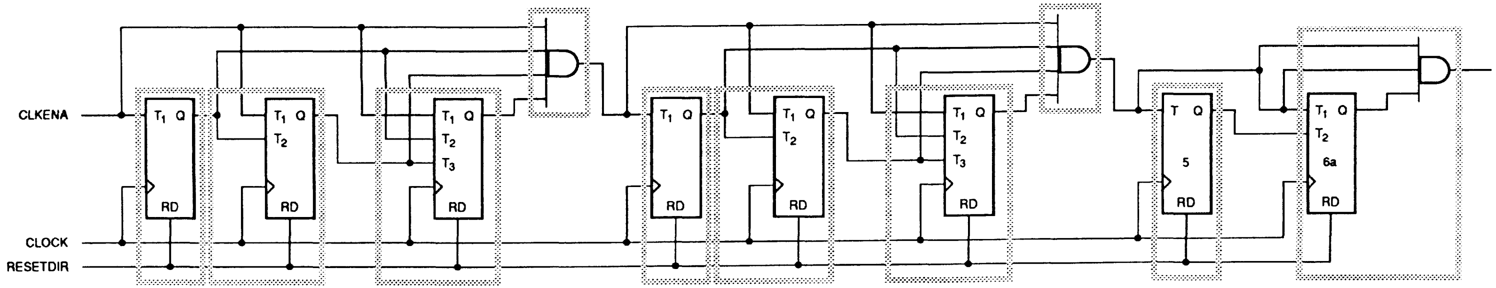
0010023 7

Figure 7. Use the asynchronous SET and RESET pins available within a CLB to preload a counter with an inital value.



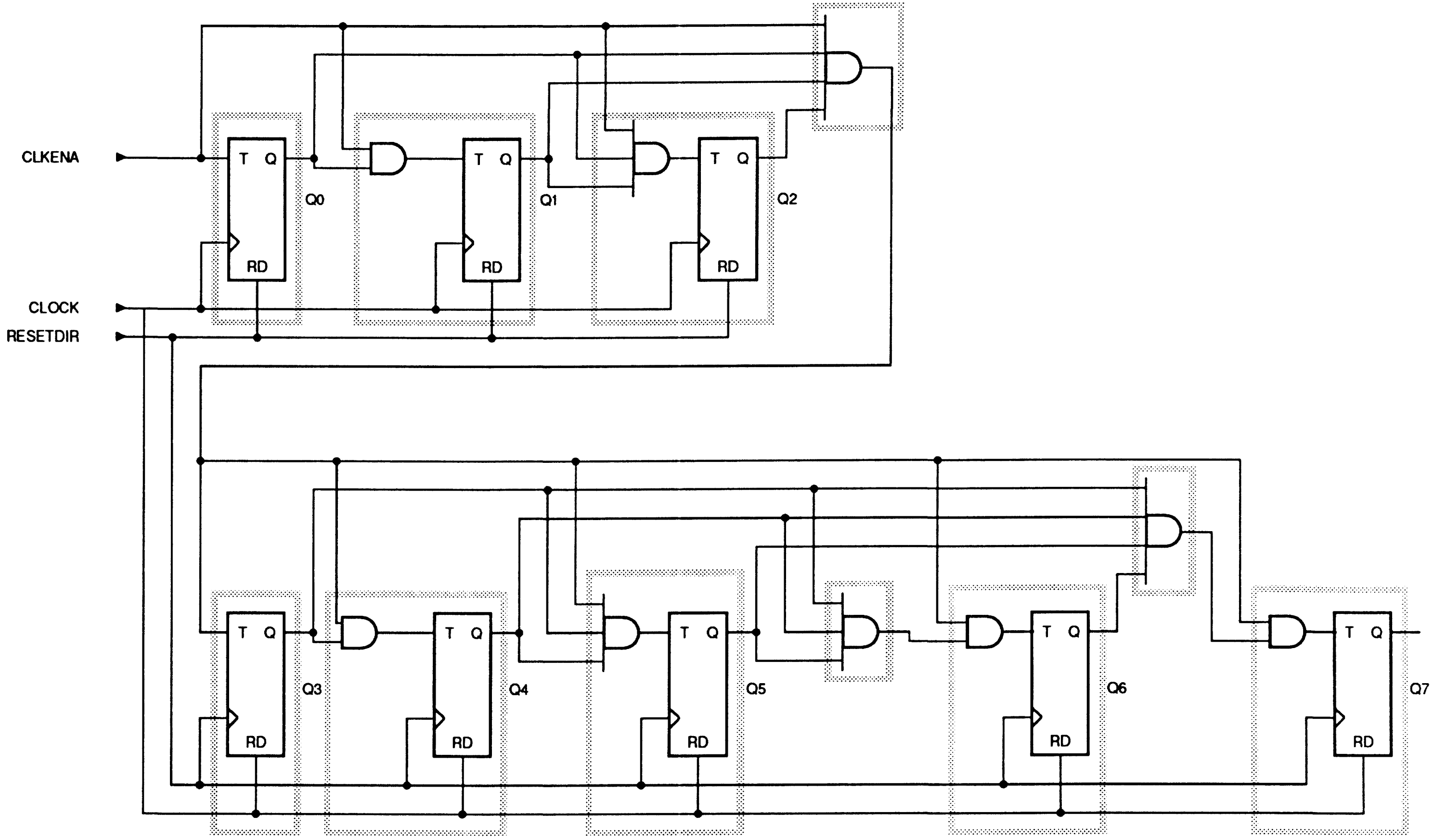
0010023 8A

Figure 8a. Ripple counters are discouraged because they are asychronous.



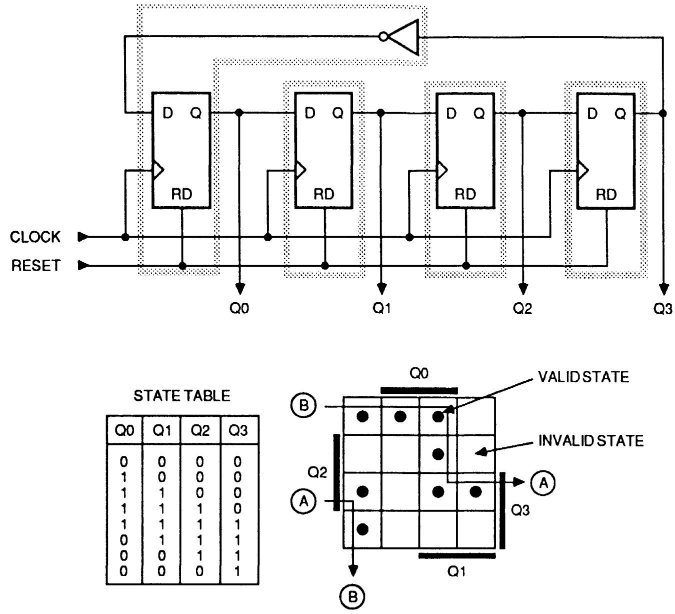
0010023 8B

Figure 8b. Ripple-carry counters are synchronous and are resource efficient. However, the performance of a wide ripple-carry counter will be degraded because of the number of ripple-carry delay levels.



0010023 8C

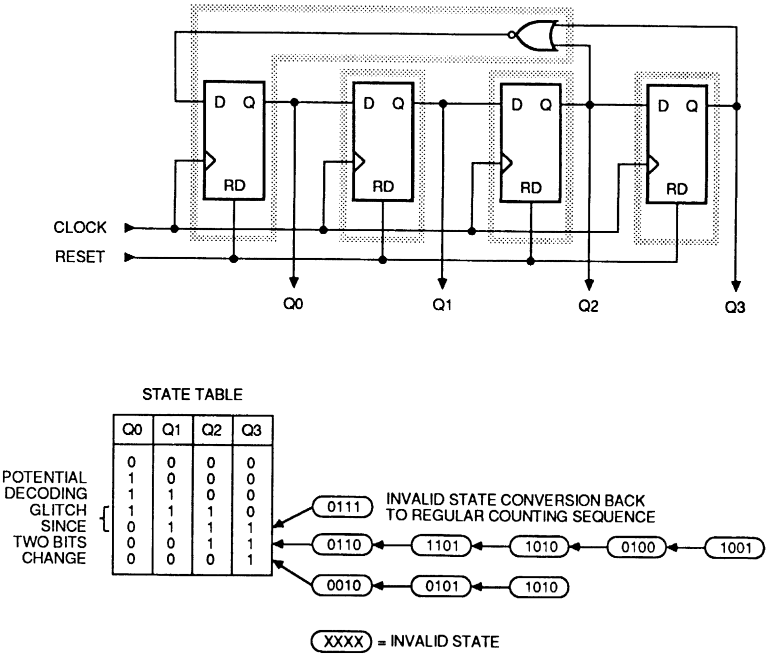
Figure 8c. Lookahead-carry counters are the highest performance binary counters. However, their complexity can make them more difficult to implement if they are exceedingly wide.



Q0	Q1	Q2	Q3
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
1	1	1	1
0	1	1	1
0	0	1	1
0	0	0	1

0010023 9

Figure 9. Johnson (or Mobius) counters can be thought of as a special form of shift register. Only a single bit changes during a clock transition. Johnson counters have 2n possible states.


CLOCK
RESET
D Q
RD
Q0
Q1
Q2
Q3
STATE TABLE
POTENTIAL DECODING GLITCH SINCE TWO BITS CHANGE
0111
INVALID STATE CONVERSION BACK TO REGULAR COUNTING SEQUENCE
0110
1101
1010
0100
1001
0010
0101
1010
XXXX = INVALID STATE

Q0	Q1	Q2	Q3
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0
0	1	1	1
0	0	1	1
0	0	0	1

0010023 10

Figure 10. Johnson counters can have odd modulos (2n–1) by means of adding simple feedback logic. Feedback logic also converts invalid counter states back into valid counter states.

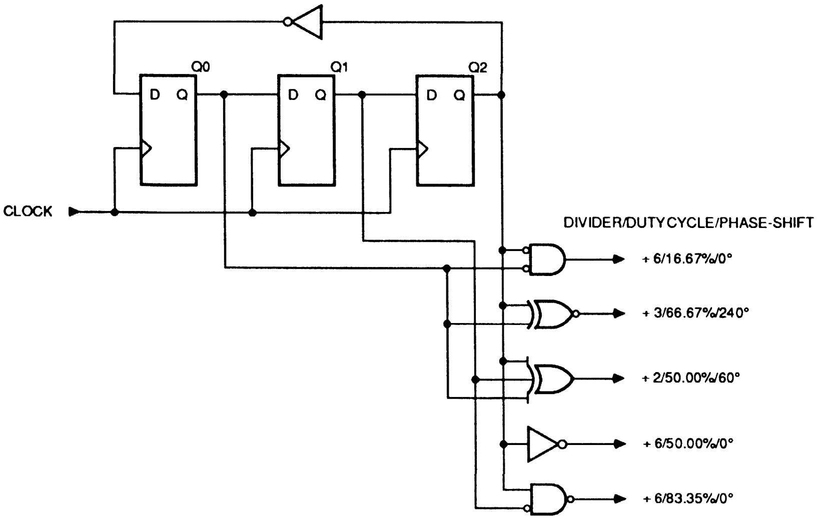


Figure 11. Johnson counters can provide decoded signals of various modulos, duty-cycles, and phase shift from an incoming clock signal.

DIVIDER	DUTY-CYCLE	PHASE-SHIFT	LOGIC EQUATION
2	50.00	0	~(Q0 @ Q1 @ Q2)
2	50.00	60	Q0 @ Q1 @ Q1
3	66.67	0	~(Q1 @ Q2)
3	66.67	60	Q0 @ Q2
3	66.67	240	~(Q0 @ Q2)
3	33.33	0	Q0*Q1*Q2 + ~Q0*~Q1*~Q2
3	33.33	120	Q0*Q1*~Q2 + ~Q0*~Q1*Q2
6	16.67	0	~Q0 * ~Q2
6	16.67	60	Q0 * ~Q2
6	16.67	120	Q0 * Q1
6	16.67	180	Q0 * Q2
6	16.67	240	~Q0 * Q1
6	16.67	300	~Q1 * Q2
6	33.33	0	Q1 * Q2
6	50.00	0	~Q2
6	66.67	0	~(~Q0 * Q2)
6	83.35	0	~(~Q1 * Q2)

Note: ~ = negation
@ = exclusive OR

Table 2. The clocks of various modulos, duty-cycles, and phase-shifts derived from a three-bit Johnson counter.

In some designs, only the terminal count of a counter is used. Binary counters will produce glitches if all of the counter bits are decoded with a large AND gate. Johnson counters, however, are inherently glitch-free for simple AND decoding. Again by mixing the two types, a glitch-free decoded terminal count is produced. This is accomplished by using the Johnson counter outputs as part of the terminal count decoding circuitry, as shown in Figure 12.

LINEAR FEEDBACK SHIFT REGISTER (LFSR) COUNTERS

Linear Feedback Shift Registers (LFSR) counters (also called polynomial, or pseudo-random, or full-cycle counters), are another special type of shift register. Although an LFSR counter is similar in some ways to a Johnson counter, there is one important difference. Johnson counters can count to a maximum of 2n possible states. An LFSR counter, however, has nearly the same capability of a binary counter since it can count to 2^n–1 possible states (the difference between binary and LFSR counters becomes insignifigant at high modulos).

The counting sequence is the major difference between a binary counter and an LFSR counter. The counting sequence of an LFSR counter is non-binary and essentially pseudorandom (the pseudo random behavior of LFSR counters can be used to build encryption and decryption circuits as done in the Xilinx application note, *A UART Design Example).*

LFSRs also have simple placement and routing, much like Johnson counters. Fast LFSRs are also possible, since the primary delay derives from decoding the feedback from the various counter bits. Although not all LFSRs will operate at the same maximum frequency as a Johnson counter, they will have superior performance to binary counters of nearly the same modulo. If the LFSR counts to 2^n–1 instead of some lower modulo, it can operate near the toggle frequency of the device. If the LFSR counts to some lower modulo, the extra required logic lowers the overall performance.

Like shift registers and Johnson counters, an LFSR counter can be built using only the registers in the Input/Output Blocks (IOBs) of the LCA and a few CLBs. This is discussed in detail in the Xilinx application note, *The Ins and Outs of Input/Output Blocks in Logic Cell Arrays.*

An n-bit LFSR counter can produce a pseudorandom sequence of up to 2^n–1 unique states. By adding logic to the feedback path, the LFSR counter can be forced to skip any number of states (from one to 2^n–1). By forcing the counter to skip M states, a LFSR counter can implement any modulo as described in Equation 4.

$$\text{MODULO} = (2^n - 1) - M \qquad [4]$$

where n = number of shift-register bits
M = number of "skipped" states

Figure 13 shows the counting sequence for a three-bit LFSR counter with exclusive-NOR (XNOR) feedback. There are two counter states for which only the first bit differs (for example, locate the states 101 and 001). Inverting the feedback with an XOR or XNOR gate causes the counter to "skip" all of the states between the two indicated values. This can be accomplished by decoding (ANDing) the state just previous to the state to be skipped. Using this method and the proper feedback into the leading bit, a counter of any modulo from one to 2n–1 can be built.

The designer should be careful to avoid the "stuck" state. The "stuck" state is the state missing from the 2^n–1 counting sequence (if the "stuck" state were included, the LFSR counter could have 2^n possible states). This state occurs when the feedback path forces the counter into an ever-repeating single state. As a simple example, assume that a LFSR counter were built with a two-input exclusive-OR feedback path as shown in Figure 14. Upon configuration or external RESET, the counter would begin operation in the all zeroes state (000) and would be "stuck" in that state due to the type of feedback used (0 XOR 0 = 0). Thus, exclusive-NOR (XNOR) feedback is suggested for LCA designs since all register are reset upon configuration (0 XNOR 0 = 1).

An interesting thing occurs when all but the first bit of the "stuck" state is 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.

Wider LFSR counters with higher possible modulos and more complex feedback mechanisms can be built but their analysis is well beyond the scope of this application note. Unfortunately there are no simple rules of determining which bits to use as feedback or which bits to decode to derive a specific skipping pattern. The basic concepts come to digital design via discrete linear algebra. Table 3 presents some of the possible feedback combinations for LFSR counters of three bits to ten bits in length.

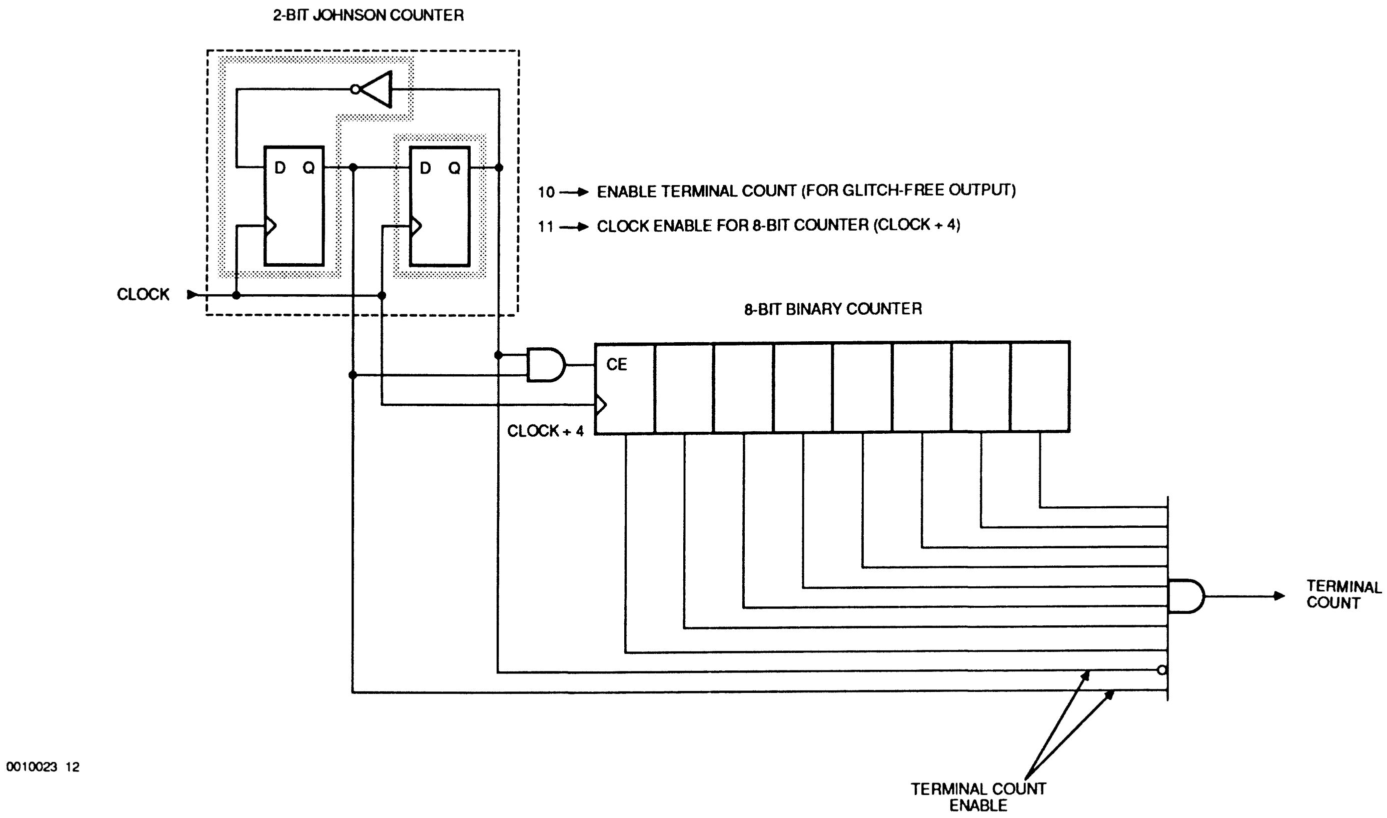
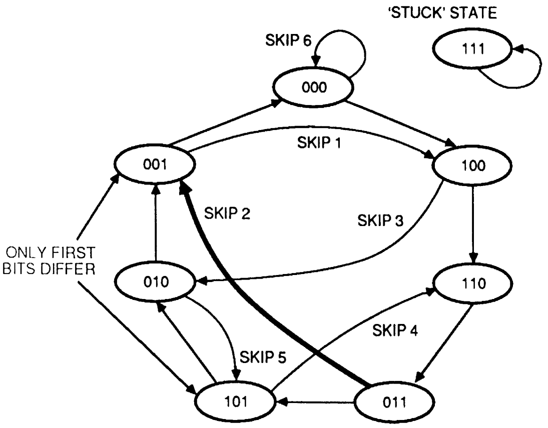
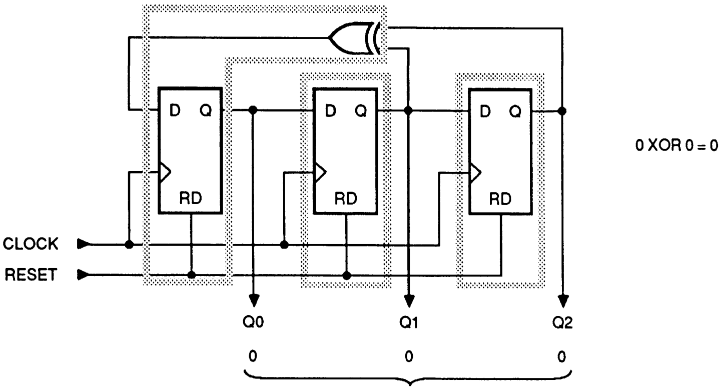


Figure 12. Johnson counters are useful for implementing clock dividers or counter pre-scalars within an LCA. Decoding their states with a two-input AND function provides glitch-free outputs. Johnson counters can operate at nearly the toggle rate of the LCA.



0010022 29

Figure 13. The counting sequence for a three-bit LFSR using XNOR (exclusive-NOR) feedback. All of the possible "skip" paths are indicated. Also shown is the "stuck" state.



0010023 14

Figure 14. A simple XOR LFSR counter which will be "stuck" in state 000 after configuration since all registers are reset.

DESIGN EXAMPLE 8. A modulo five Linear Feedback Shift Register (LFSR) counter.

Figure 15 shows the schematic for a three-bit LFSR counter which implements a modulo-five (divide by five) counter. To generate a modulo five output from a three-bit LFSR counter, two states must be skipped as indicated by Equation 4. The initial state from which the counter can jump two states is 101. By decoding the state (011) just prior the initial skip state (101), the sense of the feedback into the first bit can be inverted when fed into the XNOR feedback path. In operation, the counter will skip from state 011 to state 001, thus implementing a modulo five counter.

The registers within the counter will initialize to state 000 after configuration or after RESET is asserted. The counter sequence never enters the "stuck" state (111) because of the XNOR feedback.

DESIGN EXAMPLE 9. Adding another state to a Linear Feedback Shift Register.

Linear Feedback Shift Registers can normally only count to 2^n–1 possible states. However, with the addition of a small amount of additional logic, an LFSR counter will instead count to the full 2^n states. This gives the LFSR counter the same capability as a full binary counter. To give the LFSR an additional counter state, all but the last bit of the "stuck" state is decoded and included in the feedback path. Again, a three-bit LFSR counter will be used as an example.

The "stuck" state for an XNOR feedback LFSR counter is the all ones state or 111. To make a three-bit LFSR counter count to eight instead of seven, all but the last bit of the "stuck" state must be decoded (ANDed together) and included as part of the XNOR feedback.

$(2^n - 1)$ 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

Table 3. Possible feedback combinations for LFSR counters of three to ten bits in length.

This circuit is shown in Figure 17. The LFSR counter begins operation as it normally would if it did not have the extra logic. The additional state is inserted just after state 110. The extra decoding inverts the sense of the feedback to produce the additional state 111. In the previous example, state 111 was the "stuck" state. However, the additional logic again inverts the normal sense of the feedback to produce state 011. From there, the LFSR counts as it normally would except that it now is a modulo eight counter instead of a modulo seven counter as it would be without the extra logic. Notice the new counting sequence which is also shown in Figure 16.

The macro library included with the Xilinx XACT Development System includes another LFSR counter which is a modulo 256 counter with clock enable and reset-direct. This macro can be found in the macro library under **C256FC-rd**.

UP/DOWN COUNTERS

Another form of digital counter is the UP/DOWN counter. In operation, a counter bit will toggle either if all of the previous counter bits are HIGH and the direction is UP, or if all of the previous counter bits are LOW and the direction is DOWN. Therefore each CLB or counter bit of an UP/DOWN counter must:

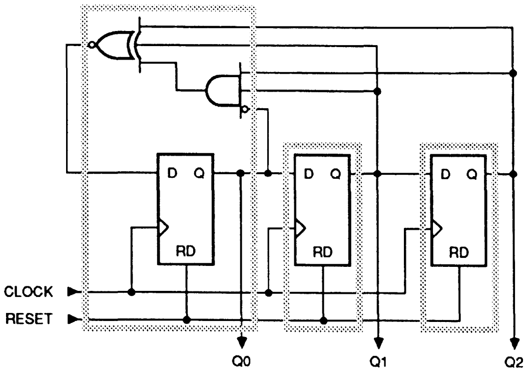
- Toggle Qn if all Q0 to Qn–1 are HIGH and direction is UP, or
- Toggle Qn if all Q0 to Qn–1 are LOW and direction is DOWN

Adding clock enable, reset, parallel enable and various other control inputs can make the algorithm for the counter fairly complex. Since an UP/DOWN counter is complex to begin with, every effort should be made to minimize the number of control signals required for the design. For example, if the counter only need be reset at initialization, use the fact that all registers within the LCA are reset upon configuration. Whenever possible, build ripple-carry UP/DOWN counters, as opposed to lookahead-carry types.

DESIGN EXAMPLE 10. A 13-bit binary UP/DOWN counter with synchronous RESET.

This design example describes two possible implementations of the same counter function. One implementation is performance-driven and therefore requires a lookahead-carry UP/DOWN counter. In the second, the amount of resources required for the counter function are minimized by using a ripple-carry UP/DOWN counter.

SCHEMATIC



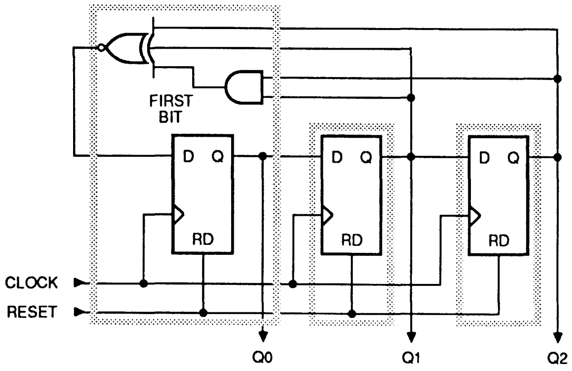
STATE TABLE

Q0	Q1	Q2
0	0	0
1	0	0
1	1	0
0	1	1
0	0	1

0010023 15

Figure 15. A modulo-five Linear Feedback Shift Register (LFSR). The counting sequence for an LFSR counter is a non-binary, pseudo-random pattern.

SCHEMATIC



STATE TABLE

2^N STATES

Q0	Q1	Q2
0	0	0
1	0	0
1	1	0
1	1	1
0	1	1
1	0	1
0	1	0
0	0	1

"STUCK" STATE APPEARS IN AN LFSR COUNTING SEQUENCE IF ALL BUT THE FIRST BIT (Q2) OF THE "STUCK" STATE IS DECODED AND INCLUDED IN THE FEEDBACK PATH

0010023 16

Figure 16. By ANDing all but the last bit of the "stuck" state and using this value in the feedback path, an LFSR can be forced to count to 2^n possible states instead of the 2n–1 states usually associated with an LFSR counter.

For maximum performance with a binary counter, a lookahead-carry implementation should be used. The general algorithm for binary lookahead-carry UP/DOWN counters is described in Equation 5.

Assumptions:

- RESET is active HIGH
- Direction is UP/$\overline{\mathrm{DOWN}}$

Equation:

$$
\begin{aligned}
&Q0 = \overline{\mathrm{RESET}} \bullet \overline{Q0} && [5]\\
&Q1 = \overline{\mathrm{RESET}} \bullet [Q1 \;@\; \{(\mathrm{UP} \bullet Q0) + (\overline{\mathrm{UP}} \bullet \overline{Q0})\}]\\
&Q2 = \overline{\mathrm{RESET}} \bullet [Q2 \;@\; \{(\mathrm{UP} \bullet Q0 \bullet Q1)\\
&\qquad + (\overline{\mathrm{UP}} \bullet \overline{Q0} \bullet \overline{Q1})\}]\\
&\bullet\\
&\bullet\\
&\bullet\\
&Qn = \overline{\mathrm{RESET}} \bullet [Qn \;@\; \{(\mathrm{UP} \bullet Q0 \bullet \ldots \bullet Qn{-}1)\\
&\qquad + (\overline{\mathrm{UP}} \bullet \overline{Q0} \bullet \ldots \bullet \overline{Qn{-}1})\}]
\end{aligned}
$$

As shown by the general equation, an UP/DOWN counter quickly becomes more complex with each additional bit. This example does not include the other possible control signals such as clock enable or parallel enable. If a 13-bit binary UP/DOWN counter with lookahead carry and RESET were built in either the XC2064 or XC2018, it would require an estimated 42 Configurable Logic Blocks! However, using the –70 (70 MHz) speed grade device, the 13-bit UP/DOWN counter can operate at 15 MHz.

If minimizing LCA resources is the goal, a ripple-carry implementation should be used. In a ripple-carry UP/DOWN counter, all of the required logic can fit into a single CLB. The general equation for a single-bit ripple-carry UP/DOWN counter is indicated in Equation 6.

$$
\begin{aligned}
&Qn = \overline{\mathrm{RESET}} \bullet [Qn \;@\; \{(\mathrm{UP} \bullet Qn{-}1) && [6].\\
&\qquad + (\overline{\mathrm{UP}} \bullet \overline{Qn{-}1})\}]
\end{aligned}
$$

COUNTER A

COUNTER B	MODULO	2	3	4	5	6	7	8	9
	3	6		12	15		21	24	
	4		12		20	12	28		36
	5	10	15	20		30	35	40	45
	6			12	30		42	24	18
	7	14	21	28	35	42		56	63

Table 4. Heterodyne Counter can be built from two counters (counter A and counter B) with different modulos

By cascading multiple single-bit counter segments, the 13-bit UP/DOWN counter can be built with only 13 CLBs. However, it will have lower performance than the lookahead-carry version.

Generally it is best to limit lookahead-carry UP/DOWN counters to a reasonable number of counter bits. With more than ten counter bits, a lookahead-carry UP/DOWN counter starts to consume a great deal of resources. If possible, UP/DOWN counters should be built with cascaded counter segments. The counter segments can be larger than the single-bit implementation shown in Design Example 11 in order to increase performance.

HETERODYNE COUNTERS

Another class of counter is the heterodyne counter. In this category, the terminal counts of two or more counters with different modulos combine to produce a counter of yet another modulo. The AND decoded terminal counts from the first set of counters combine in such a way as to produce an additional counter with a modulo equal to the least common multiple of the counters as described in Equation 7.

$$
\begin{aligned}
&\text{MODULO X} = \text{L.C.M. (MODULO Y,} && [7]\\
&\qquad\qquad \text{MODULO Z, \ldots\ MODULO n)}
\end{aligned}
$$

For example, the terminal counts from a modulo-three counter and a modulo-four counter can be combined to form a modulo-twelve counter (the least common multiple of three and four). This kind of counter can be useful in designs where two other counter modulos already exist or in designs where a counter of one modulo exists and a second counter can be built with few additional resources. To make use of an existing counter, the added counter must have the same clock input as the existing counter.

None of the first set of counters should have a modulo which is a multiple of the other. For example, a modulo-two counter combined with a modulo-four counter only produces a modulo-four counter instead of a modulo-eight as might be thought. This occurs because four is a multiple of two and the least common multiple of both numbers is still four.

The counters used to implement a heterodyne counter may be of different types. For example, a binary lookahead-carry counter can be combined with a Johnson counter or a Johnson counter with a LFSR counter.

Table 4 indicates just a few of the various modulos available by building heterodyne counters using two counters with the indicated modulos. The blank areas indicate counter combinations which are not applicable such as cases in which one modulo is a multiple of the other.

DESIGN EXAMPLE 11. A modulo-15 heterodyne counter.

Assume a design requires a modulo-15 counter. Also assume that a modulo-5 LFSR counter is already required for another portion of logic. Luckily the modulo-5 counter uses the same clock input that the modulo-15 counter requires. Instead of building an entire modulo-15 counter which will require at least four CLBs just for registers, a modulo-15 heterodyne counter can be built with only two additional CLBs.

From Table 4, a modulo-15 counter can be built with a modulo-5 counter and a modulo-3 counter. In the case of the hypothetical design, the modulo-5 counter already exists. Only a modulo-3 counter need be added.

The best method to implement a modulo three counter is with a Johnson counter. The placement and routing is simple, and it can operate at high frequencies. Figure 17a shows the schematic for the modulo-15 heterodyne counter built from one existing counter and one new counter. Figure 17b shows the timing diagram generated by this combination of counters. Notice that both counters coincidently generate a terminal count after fifteen clock cycles and do so only every fifteen clock cycles.

SUMMARY

The flexible array architecture of a Xilinx Logic Cell Array (LCA) allows various implementations of digital counters. This flexibility frees the designer from the limitations imposed by both 74-series devices and the sum-of-products architectures found in Programmable Logic Arrays (PLAs).

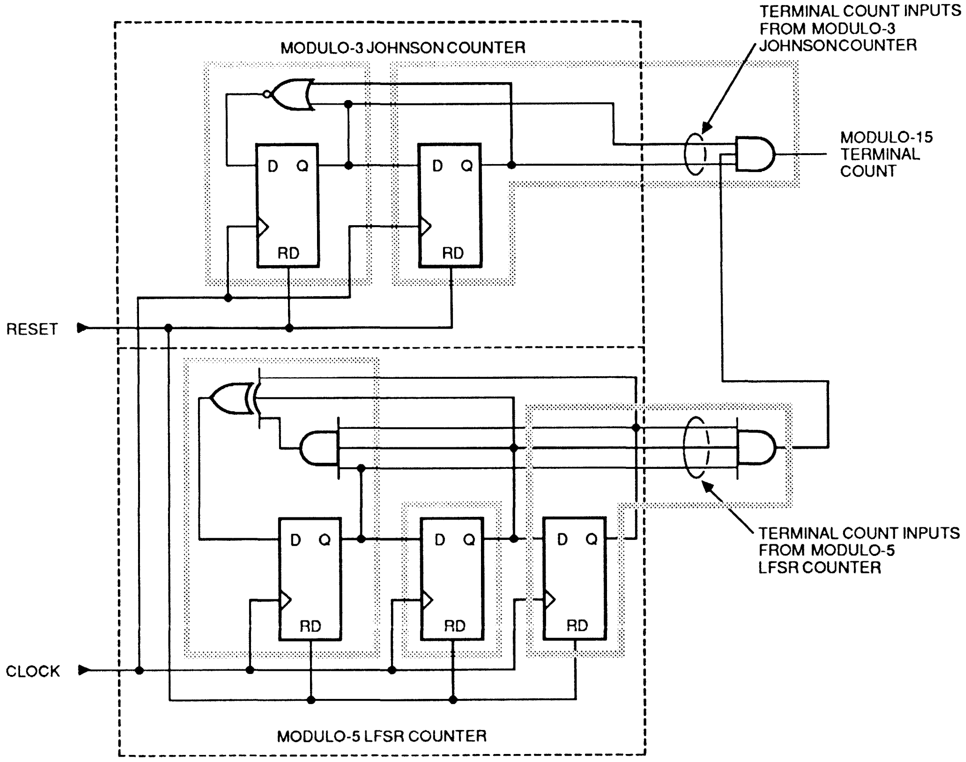
This application note described several counters applicable to Logic Cell Array designs. These counter types included common binary, Johnson, Linear Feedback Shift Register, and heterodyne counters.

By choosing the appropriate counter for a given application, a designer can optimize both resource efficiency (the routing and logic required for a function) and overall performance. Resource efficiency is increased when implementing only the required function and not substituting a one-for-one logic replacement for another logic technology.

Table 5 lists the various counter types and associated comments. It should prove helpful when determining which counter to use for an application. In some applications, overall performance may be the critical need. With others, absolute resource efficiency may by required. With more than one possible implementation available, the designer can tailor the logic to custom-fit his application instead of wasting resources on circuitry that is not used for his application.

TECHNICAL SOURCES:

1. Messina, A. *Considerations for Non-Binary Counter Applications.* Computer Design, November 1972. pp. 99–101.


MODULO-3 JOHNSON COUNTER
TERMINAL COUNT INPUTS FROM MODULO-3 JOHNSONCOUNTER
MODULO-15 TERMINAL COUNT
D
Q
RD
RESET
TERMINAL COUNT INPUTS FROM MODULO-5 LFSR COUNTER
CLOCK
MODULO-5 LFSR COUNTER

0010023 17a

Figure 17a. A modulo-15 heterodyne counter built from a modulo-5 LFSR Counter and a modulo-3 Johnson counter

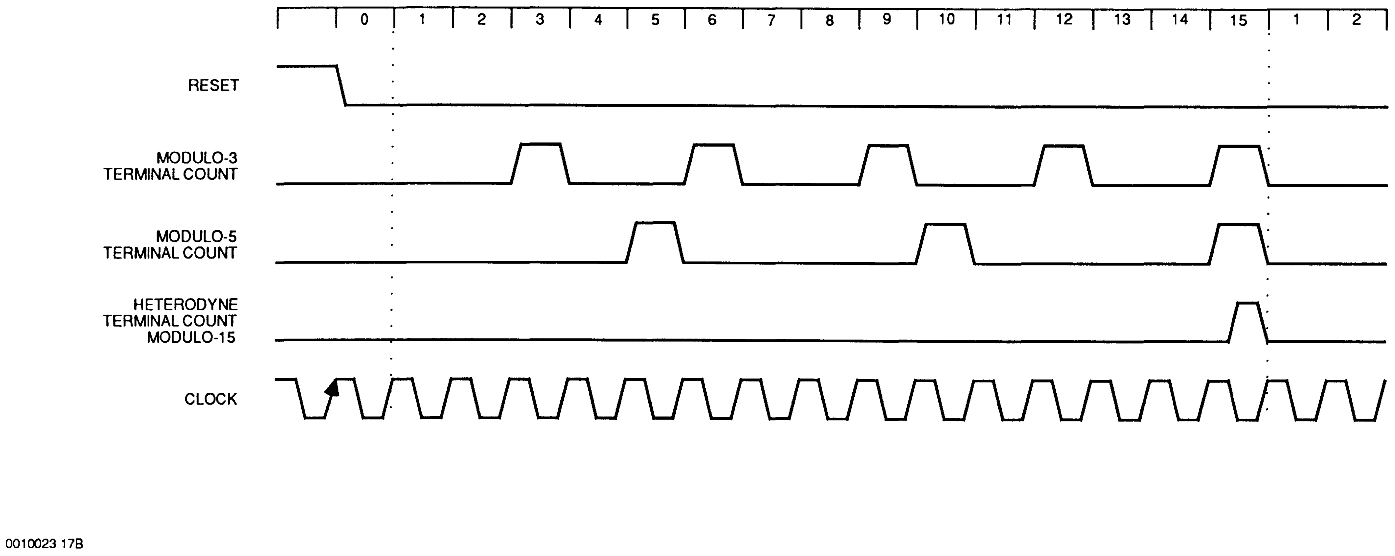


Figure 17b. Timing diagram for the modulo-15 heterodyne counter. Notice that the terminal counts from the first two counters combine to produce a third terminal count every fifteen clock cycles.

COUNTING STYLE	BINARY			NON-BINARY	
Counter Type	**Ripple**	**Ripple-Carry**	**Lookahead-Carry**	**Johnson**	**Linear Feedback Shift Register**
Modulo	2^N	2^N	2^N	2N	2^N-1
CLB Efficiency	Excellent	Good	Good-Fair Decreases slowly with increasing modulo	Good below modulo 6 Poor above modulo 12	Good
Routing Efficiency	Excellent	Good	Good-Fair Decreases with increasing modulo	Excellent	Good
Performance	Very poor 1 CLB delay per counter bit	Fair Decreases with increasing modulo	Good Gradually decreases with increasing modulo	Excellent One CLB delay	Good Gradually decreasing with increasing modulo
Best-Use	Not suggested Use only where silicon efficiency demands	General binary Counters with fair to good performance	Higher-performance binary counters	Very high-performance counters with low modulo	Higher-performance counters with large modulos
Advantage(s)	Low resource requirements Easy routing	Easy to use with macro library Good, all-around counters	Highest performing binary counter	Very high-performance Easy routing Glitch-free decoding	Good performance at high modulos
Disadvantage(s)	Asynchronous Slow	Decreasing performance with increasing modulo	Requires additional logic and routing resources	Low register efficiency above modulo 12	Non-binary, pseudo-random counting sequence

Table 4. A summary of the various counter types and their applications in LCA designs.

Metastability Analysis Of Logic Cell™ Array Flip-Flops

INTRODUCTION

Metastability is a condition of unstable flip-flop output caused by changes in the input data at or near the critical clock edge. A metastable condition may be induced during attempts to clock flip-flops with data which is not synchronized with the clocking signal. Flip-flop outputs may oscillate or exhibit an intermediate output level during a period of metastability. The probability of a metastable condition existing and its duration depend on many factors, only some of which the user can control. In implementing systems with any technology, an understanding of metastability is necessary to insure proper operation in clocking asynchronous signals.

One critical element in examining metastability is the loop delay of the flip-flop in question. Loop delay is the time required for a signal at any point in the flip-flop to propagate through the flip-flop circuit and cause a reinforcement of the signal at its starting point. Figure 1 shows one type of flip-flop with the loop delay path indicated. A change in the state of a node in the flip-flop, the input for example, requires one loop delay to be held by the flip-flop. In a metastable condition, an internal node, typically in the input stage, attains an intermediate level as a result of the data signal changing while the clock is changing. The intermediate level, neither "1" nor "0", is propagated around the loop, forcing the output into a metastable state. The flip-flop will only achieve a "1" or "0" output when a node with the intermediate level becomes "1" or "0" and the new value is propagated through the loop forcing the output out of the metastable condition. Movement of internal nodes away from the intermediate level is a random activity and therefore cannot be guaranteed.

Another method to illustrate metastability is to plot the worst-case-clock to flip-flop-output delay versus the delay from stable data to the clock edge. Figure 2 shows this type of plot for a typical flip-flop. 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 output delay at the critical-data-to-clock relationship may be "infinitely" long.

ANALYSIS OF A LOGIC CELL ARRAY CIRCUIT

The 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 Logic Cell Arrays, some additional control is possible. The probability of a flip-flop passing through a metastable region can be calculated as[1]

$$\text{Probability} = 1 - e^{(-\text{settling time / loop delay})}$$

The probability of a flip-flop remaining in the metastable region is then:

$$\text{Probability of Error} = e^{(-\text{settling time / loop delay})}$$

For the circuit shown in Figure 3, settling time is the difference between the worst case delay from a clock edge clocking the flip-flop in Configurable Logic Block (CLB) 1, output propagating to the flip-flop in CLB 2 with setup time and the delay from one clock edge to

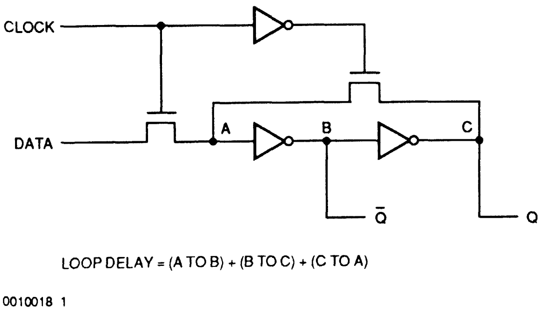


Figure 1. Flip-Flop Implementation

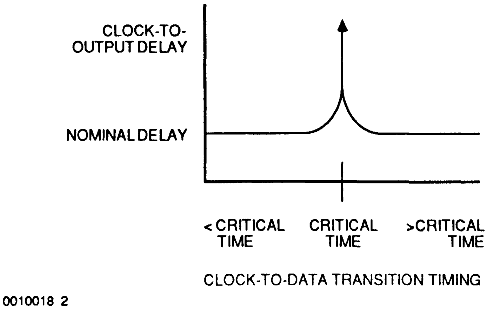


Figure 2. Flip-Flop Output Critical Timing

the next. The maximum delay path must be considered because it would produce the lowest settling time with a correspondingly higher probability of error.

Examining a specific case for which the XC2064 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:

Flip-flop clock to output delay	20	ns
Interconnect delay	15	
Flip-flop setup time	12	
Flip-flop loop delay	2	
Clock Period (10 MHz.)	100	

For this example, a value of error probability can be calculated.

$$\begin{aligned} P\,[\text{Error}] &= e^{(-[100-(20+15+12)]/2)} \\ &= e^{(-53/2)} \\ &= 3.1 \times 10^{-12} \end{aligned}$$

This 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 would be representative of a repetitive clock sampling an asynchronous signal, the time between failures can be calculated by the following relationship:

$$\text{Failures Per Time Period} = \text{Probability per Event X Events per Time}$$

The time between failures is then the inverse of these failures per time-period value. For our example,

$$\begin{aligned} \text{Failures per Time Period} &= (3.1 \times 10^{-12}) \times (1 \times 10^{7}) \\ &= 3.1 \times 10^{-5} \end{aligned}$$

$$\begin{aligned} \text{Time BetweenFailures} &= 1 / (3.1 \times 10^{-5}) \text{ sec} \\ &= 3.3 \times 10^{4} \text{ sec} \\ &= 8.96 \text{ hrs} \end{aligned}$$

This number would indicate a very short time between errors for sampling an event at 10 MHz. It must be realized that 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 would not be the case. Actual asynchronous events occur only a small percent of the time of operation, not at each clock edge as this calculation assumes.

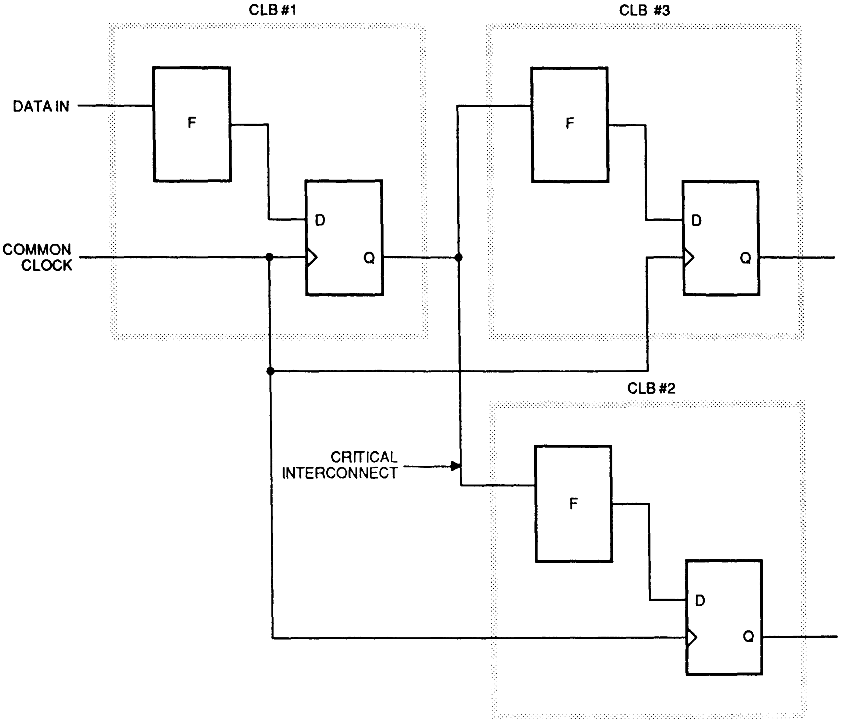


Figure 3. LCA Implementation

Based on this type of calculation, several methods to improve this error performance are possible:

1. Reduce interconnect delay
2. Use higher speed device
3. Decrease the clock rate

Reduce Interconnect Delay

If direct connect can be utilized in this critical path area, the 15 nanosecond delay for interconnect could be reduced to 0. The effect on the error probability can be easily seen.

$$\begin{aligned} P\,[\text{Error}] &= e^{(-[100-(20+12)]/2)} \\ &= e^{(-68/2)} \\ &= 1.71 \text{ X } 10^{-15} \end{aligned}$$

For the same 10 MHz clock, this results in a failure period;

$$\begin{aligned} \text{Time Between Failures} &= 1/(1.71 \text{ X } 10^{-15}) \text{ X } (1 \text{ X } 10^{7}) \text{ sec} \\ &= 5.83 \text{ X } 10^{7} \text{ sec} \\ &= 675 \text{ days} \end{aligned}$$

Use a Faster Device

Use of a faster device will improve all of the device-related performance parameters. Moving to the next higher speed grade would result in the following critical parameters:

Flip-flop clock to output delay	15.0	ns
Interconnect delay	7.0	
Flip-flop setup time	8.0	
Flip-flop loop delay	1.5	

The clock period remains the same at 100 ns.

The new error probability becomes:

$$\begin{aligned} P\,[\text{Error}] &= e^{(-[100-(15+7+8)]/1.5)} \\ &= e^{(-46.667)} \\ &= 5.4 \text{ X } 10^{-21} \end{aligned}$$

For the 10 Mhz clock rate, this results in a new failure period:

$$\begin{aligned} \text{Failure Period} &= 1/(5.4 \text{ X } 10^{-21})\text{X}(1 \text{ X } 10^{7}) \text{ sec} \\ &= 1.85 \text{ X } 10^{13} \text{ sec} \\ &= 2.14 \text{ X } 10^{8} \text{ days} \\ &= (\text{approx. } 563 \text{ years}) \end{aligned}$$

Change Clock Rate

If the clock rate which is used to perform the sampling can be reduced, a dramatic reduction of the failure rate results. If the clock rate were reduced from 10 MHz to 5 MHz, error performance can be examined with the following parameters:

Clock to flip-flop output	20	ns
Interconnect delay to second block	15	
Setup time for second flip-flop	12	
Flip-flop loop delay	2	
Clock period	200	(5 MHz)

$$\begin{aligned} P\,[\text{Error}] &= e^{(-[200-(20+15+12)]/2)} \\ &= e^{(-153/2)} \\ &= 5.98 \text{ X } 10^{-34} \end{aligned}$$

For 5 MHz clocking, the failure period is calculated as:

$$\begin{aligned} \text{Failure Period} &= 1/(5.98 \text{ X } 10^{-34})(5 \text{ X } 10^{6}) \text{ sec} \\ &= 3.35 \text{ X } 10^{26} \text{ sec} \\ &= 3.87 \text{ X } 10^{21} \text{ days} \\ &= (\text{approx. } 1.1 \text{ X } 10^{19} \text{ years!}) \end{aligned}$$

OTHER CONSIDERATIONS

Flip-flops in the Logic Cell Array family have been specifically designed to reduce the loop delay to a practical minimum to reduce 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 output of the flip-flop may be loaded differently depending on how the user has connected the device. This difference in loading can significantly complicate the analysis of the flip-flop's metastability behavior. In the Logic Cell Array, all of the flip-flops are immediately followed by a buffer, prior to any user-programmable connections. This buffer serves to isolate the flip-flop from any variations in loading that could adversely affect its metastability behavior. For the user, this significantly simplifies analysis of metastability effects in the system.

CONCLUSION

As shown by these examples, the probability of a failure based on a metastable condition and the subsequent system failure period, can vary widely and are dependent on several factors. When using the Logic Cell Array to implement system level functions, the user has significant control over some of the critical parameters necessary to provide sufficient immunity to metastable conditions.

[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.

Reading Back Logic Cell™ Array Configuration Programs

Users of Logic Cell Arrays (LCAs) who wish to verify the data loading and storage of the configuration program for the Logic Cell Array can perform a data readback on the device(s) after programming. Performing this programming data readback verification requires a knowlege of the format and manipulation of the configuration program or bitstream. The purpose of this application brief is to provide the information needed to understand the bitstream format for loading and readback verification. Information regarding the specific contents of the bitstream is beyond the scope of this application brief.

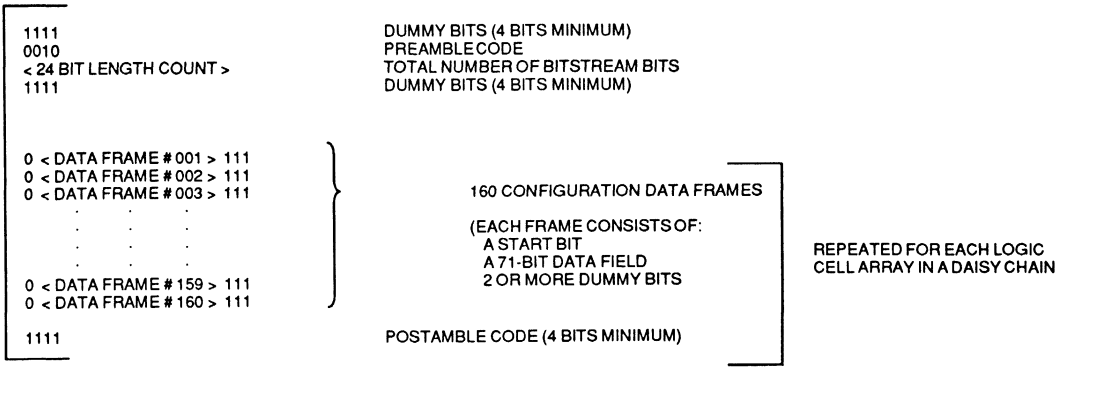
PROGRAM BITSTREAM

The first step in understanding the readback verification process is to examine the composition of the bitstream that is to be loaded into the device. In this discussion, we are assuming that the device is being configured from a processor, either in peripheral or slave mode. The connections to the device and the timing needed to perform the configuration are discussed in the Applications Note *Methods of Configuring the LCA*. Regardless of the configuration method, the bitstream data is the same; our bitstream is assumed to be in a PROM file created with the MAKEPROM command and formatted for Intel MCS86 compatibility.

Figure 1 shows the data format of the PROM file. The information preceeding the first data field is required to initialize the configuration logic on the LCA for the proper bitstream length. Each subsequent data field provides configuration information for a portion of the device. The beginning of a PROM file for an XC2064 is shown in Figure 2. Note that the first byte of the data field (underlined in line 2) is hex 4F. The required leading 1s are in the low nibble, with the preamble in the high nibble. The bits are arranged in this fashion to simplify the connections from an external PROM or ROM to the LCA in master mode.

Although the LCA's internal memory is always loaded serially, master mode reads the configuration program in parallel directly from an external memory device, such as a PROM, and serializes the data internally for loading into the memory cells. The PROM connections provide the least significant bit of the byte (D0) as the bit which is serialized first. The consequence of this is that the data bits in the individual bytes in the PROM file are reversed from the order in which they will be interpreted by the device. During loading of the configuration data with a processor, the PROM file data is read one byte at a time and is supplied to the LCA one bit at a time, beginning with the least significant bit (D0).

Consider the next three bytes of the PROM file, Figure 2; the hexadecimal 00F460 represents the 24-bit binary length count of 0000000000101111100000110 (12038 decimal), i.e., the total number clock cycles required to load this bitstream. Three aditional clocks are required to complete configuration and activate the device. The



1111 — DUMMY BITS (4 BITS MINIMUM)
0010 — PREAMBLE CODE
< 24 BIT LENGTH COUNT > — TOTAL NUMBER OF BITSTREAM BITS
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 START BIT
A 71-BIT DATA FIELD
2 OR MORE DUMMY BITS

1111 — POSTAMBLE CODE (4 BITS MINIMUM)

REPEATED FOR EACH LOGIC CELL ARRAY IN A DAISY CHAIN

Figure 1. Configuration Data Arrangement for the XC 2064

0010003 13

fifth byte (hexadecimal EF) contains the four pad 1s, the start bit, and the first 3 bitsof the 71-bit data field. All of the data bits in the configuration data fields are the complement of the bits needed to control the elements associated with the memory cell. The LCA performs an inversion of the incoming data, so actual data bits stored in the memory cells will be the complement of the input data.

Internal Data Storage

Data that is supplied to the Logic Cell Array during configuration is shifted into a 71-bit shift register. When the shift register is filled, it is written into the internal memory cells as a single 71-bit word. In the XC2064 there are 160 words of 71 bits each, comprising a total of 11,360 bits of programming data. For the XC2018, there are 196 words of 87 bits; a total of 17,052 bits.

Readback

Readback allows the user to extract the configuration program from the Logic Cell Array, even while the device is operating. This data may be used to verify that the contents of the memory cells have not been changed since the last programming cycle. In addition, the readback data contains the state of all of the storage elements in the logic blocks, as well as the state of the input connection point on each I/O block.

The readback process is accomplished without using any of the user I/O pins. CCLK, M0 and M1 are used to read the data in a serial fashion. The readback 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. Figure 3 shows this data reading process.

Individual frames of data are read back in the same sequence that they are supplied to the device. In the readback serial data stream, the individual bits are the true sense of the internally stored data bit; the bits in the programming stream are inverted from those stored internally. The initial data frame is preceeded by a dummy clock cycle and two dummy bits whose state is unknown. After the first data frame, there will be a stop bit, 0, and a single start bit, 1, prior to the next frame. After the last frame, there is a single 0 stop bit. Even if additional CCLK cycles are applied after the last data frame is read, the M1/RD output will be three-stated; the pin is not driven after the final stop bit.

Readback Data Contents

After the configuration program has been read back, it may be compared to the input data stream to determine if the device is correctly configured. Input data dummy bits and start bits and readback data start and stop bits need to be removed, either as part of the programming/readback process or after the readback is complete.

In the programming and readback bitstreams, some of

```
:020000020000FC
:100000004F00F460EFFAF3F3FFF7C5FFFF7FD39CD7
:10001000A5EBBB5975F7FFFB3F7FFEFEFEB5FCFF6E
:10002000DFBD59AFBDBE4FFFFBFEFEDFBFBFAF5F01
:100030007F7FF7F7FF7FFFFDFDFFFBFFBFFFFFFFA8
:10004000CBD7FFB7FFFFFDFFFFBF3EFFFF7FFFEFF7
:10005000FFFFBDFFFFFFB7FF7BFFFFFFFFFFDFEFEE
:10006000FFFDFBFFFFFFDFFFFFFFFF9D7F7FFFFE29
:10007000FCFAFD7DEFEEFFFFFFFFFFFFFFFF7FF45
:10008000FFFFFFFFFFFF7F3FFFFFFFFFFFFFFFFFC0
:10009000FEF9EFEFDFFFD7BFBFFFCFDFFFFFFFBFEF
:1000A000FFFFFF77EFEFDFDFDFBFBFBFFFF3FF7FB4
:1000B000FFCFBFDFFFFF9DFBFAB7F7F7FFFF7FEF33
:1000C000FAFEFEFDFD7D7B7BFBF7FFFFFFFFFFFBE5
:1000D000FBFB3FF3F7EFE9EFFFFFFFFDBB9BD5D3B54
:1000E0005BFB77F7F7CE7777EFEFAED5D7D77B9F70
:1000F0009F3F3F3E373E3EDE7B67FFFEFEFEEEFF4C
.     .     .     .     .     .     .     .

.     .     .     .     .     .     .     .

.     .     .     .     .     .     .     .
```

Figure 2. Beginning of Typical Hex PROM File

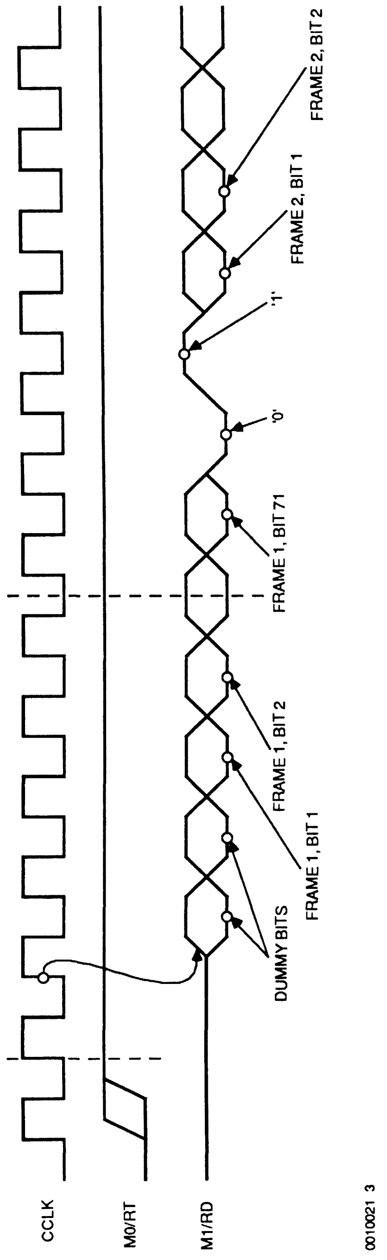


Figure 3. Readback Control Timing

the memory locations do not correspond to actual memory cells in the device. These locations may be unused during both programming and readback. They contain the storage element and input block values during readback. The storage element and input block values are extracted and displayed by the in-system emulator during debugging. For readback to verify configuration, the user must ignore these bit locations since their contents may not be the same as the corresponding positions in the programming bitstream.

Bit positions that are to be ignored in the readback data stream can be generated in a standard XACT bit file with the MAKEMASK command. The .BIT file generated with MAKEMASK can be converted into a PROM file with the MAKEPROM command. The format for the final mask PROM is exactly the same as for the configuration program PROM. Each data bit in the readback data stream that is to be ignored is represented by a "0" in the mask PROM file. Figure 4 shows the beginning of a mask bitstream file for an XC2064 that has been converted into a PROM. Note that this file has the same preamble, length count and pad bits as the regular programmimg data file.

To utilize the mask information, users should strip off the preamble and length count information and extract the appropriate data bits for each data field. Since the PROM format has the data bits arranged with the least signifigant bit in the D0 position, the data bits must also be reordered to put them in the correct sequence (See Figure 5). A simple program could be written to create the mask bit fields for each data field. Note that the end of one data field and the beginning of the next field can be verified by detecting the dummy bits and the start bit between each field.

```
:020000020000FC
:100000004F00F4608FEDEDDD8BDBBBB78770ECEF5D
:10001000DF1FDCBFBF3F9463FFFFFFE9FFFFFFB3BC
:100020001DFFFFFBD7FFF7FFAFED587FDFFEFEBEE2
:10003000FDFD2DC7FAFFFEFFFFFDFF7FB9E6FFF7CD
:10004000BFFFEFFF7FDB35FFBFFFF77FFFFFDFEE77
:10005000F1F3E7A7E7CFCFCF7EBFBC7C7979F9F288
:10006000F2FAFBF575EBEBEBD6D7D7DFA7A74B4F33
:100070004F979E9EFE3E3D5D7A7ABAF4F4F4D7ED3A
:10008000EDDADBDBB5B7B7BD7F7FF7FEFEEEFDFD3A
:10009000FDFDFFFFFFFFFFFFFFFEFEFFBFBF5F7F7B6
:1000A000EBEF6F7FDEDFAFBFBF5F7F7FBB7B78F89B
:1000B000F0F0F0E1E1C1DEEFEFDBDFDFB7BFBFFE65
:1000C0007E7FFFFEFEFEFDFDF5D7F9F9F2F3F3E5C5
:1000D000E7E7BDCECF979F9F2F3F3FEF757FBFFED6
:1000E000FE7EFDFD7DEFFBFBF5F7F7EBEFEF7FFF0E
:1000F000FFEFFFFFDFFFFFFFFBFFFFFFFFFFFFFF44
```

Figure 4. Mask PROM File

	1ST BYTE								2ND BYTE								
PROM FORMAT	D7	D6	D5	D4	D3	D2	D1	D0	D'7	D'6	D'5	D'4	D'3	D'2	D'1	D'0	. . .
READBACK FORMAT	D0	D1	D2	D3	D4	D5	D6	D7	D'0	D'1	D'2	D'3	D'4	D'5	D'6	D'7	. . .

0010021 5

Figure 5. Data Bit Sequence

XILINX

The Programmable Gate Array Company

Table of Contents

Programmable Gate Array Design Examples

A UART Design Example

OVERVIEW

This application note shows the design engineer how to implement a sample complex digital logic function using the Xilinx XC2064 Logic Cell™ Array (LCA). The design implemented is a Universal Asynchronous Receiver Transmitter (UART). The UART design adds a number of complexities in order to illustrate various features and flexibility of the XC2064 LCA including:

- The amount of logic that can be implemented in an LCA.
- The versatility of the LCA's Configurable Logic Blocks (CLBs) and Input/Output Blocks (IOBs) in implementing any logic function.
- The extent of user control of the design process.

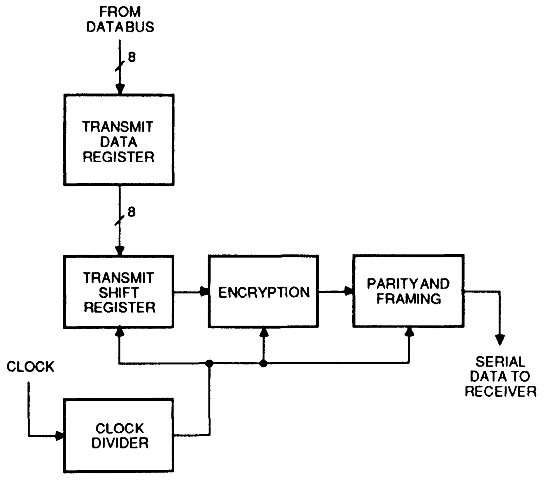
It is important to remember that any logic function having up to four variables can be implemented in any one of the 64 CLBs in an LCA. Therefore the implementation process can be focused on function rather than being constrained to an implementation based on a fixed set of available logic functions.

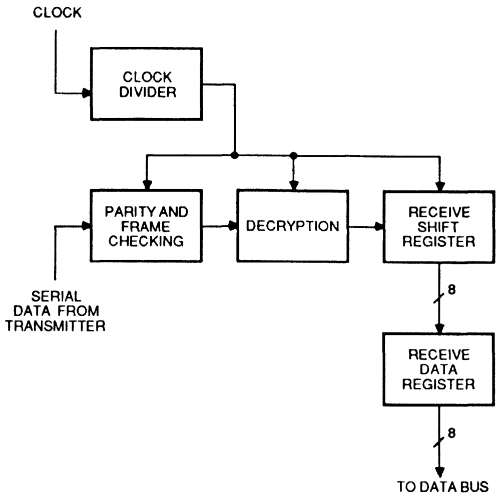
This application note provides:

- A description of the design and operation of the UART
- A suggested methodology for user implementation of logic functions in LCAs
- Examples of how various portions of the UART are implemented using the suggested methodology

The UART in this application has a fixed format and uses eight-bit data words with even parity. It is designed to operate as a typical peripheral device on a microprocessor data bus. Its operation is similar to that of an 8251 configured for asynchronous operation, except that this application incorporates cipher feedback encryption and decryption in the serial data path.

The Transmitter portion of the UART prepares data for





0010006 1 & 2

Figure 1. Block Diagram of UART Transmitter

Figure 2. Block Diagram of UART Reciever

transmission by converting each parallel byte it receives from the microprocessor data bus into a serial data stream. The encryption circuitry encodes the serial data for security and additional logic provides the start bit, parity bit and stop bits (framing). As shown in the block diagram in Figure 1, the transmitter consists of five logic sections:

- Transmit data register
- Transmit shift register
- Encryption
- Parity and framing
- Clock divider

The receiver portion of the UART receives the transmitted frame, decrypts the data, and converts it to a parallel data byte. Additional logic detects parity, framing and overrun errors. Figure 2 shows the five sections of the receiver:

- Parity and framing check
- Decryption
- Receive shift register
- Receive data register
- Clock divider

TRANSMITTER DESIGN

Figure 3 shows the schematic of the transmitter portion of the UART. When TXRDY is high, it indicates that the transmit data register shown at the top of the diagram is ready to be loaded with data. Data are loaded from the processor when signals chip select (CS) and write enable (WE) are activated. Subsequently TXRDY goes low. The internal TLOAD signal then controls the parallel transfer of data from the transmit data register to the transmit shift register below it and resets TXRDY. The ninth-bit at the left end of the transmit shift register is a tag bit. Instead of using a bit counter, the position of the tag bit (a logic zero) followed by logic ones, is used internally by the parity and framing logic to determine when to insert parity and stop bits into the serial data stream.

Note that all of the flip-flops in the transmit data register and transmit shift register operate synchronously with the exception of the asynchronous reset of the data register status flip-flop as indicated by RD on that flip-flop. Also note that control functions have been implemented within the flip-flop data logic. The control function parallel enable is indicated on the logic diagram by PE on the flip-flop symbol; clock enable, by CE; synchronous reset, by R; synchronous set, by S; and the Data input, by D. When power is applied to the UART, an external active-low CLEAR drives a flip-flop which provides the synchronized signal, RESET, to initialize the device.

Clock Divider

As an illustration of the flexibility of the design approach, a cascaded pair of modulo-4 Johnson counters, TCC0/TCC1 and TCC2/TCC3, shown at the bottom of the diagram, is used to divide the clock input by 16 to produce the transmit clock. This was chosen as an alternative to the more common binary weighted counter. When the transmit shift register is empty, the IDLE signal maintains the counters in a (0000) state. Thus, when the transmitter is idle, the clock operates at full rate. When the UART is loaded and begins actively transmitting serial data, the transmit clock operates at one-sixteenth the input clock rate. This fine timing resolution minimizes the time it takes to recognize and load data when the transmitter is idle.

Encryption

Figure 4 shows the schematic for the encryption and decryption registers used in the UART design. To synchronize the encryption registers with the decryption registers of the receiver, the clocks are enabled only when data are being shifted. The encryption circuit exclusive-ORs (XORs) the data with a pseudorandom bit sequence. It is implemented in two stages using an 8-bit Linear-Feedback Shift Register (LFSR) cascaded with a 9-bit LFSR. The output of the Transmit Shift Register is XORed with the feedback bits (1, 3, 5, 8) of encryption register R1. The result is XORed with the feedback bits (1, 5, 8, 9) of encryption register R2.

Parity and Framing Generation

Located in the lower right of the diagram are TSC1 and TSC0, a two-bit state controller that cycles through the following sequence of states:

State		Condition
TSC1	TSC0	
0	0	Data has been loaded into the transmit shift register, signals TDATA DONE and TPARITY DONE are not active.
1	0	TPARITY DONE goes active when the tag bit is detected in the proper position.
1	1	Follows the previous state (10). If TXRDY indicates data is ready to be loaded, TLOAD goes active.
0	1	Follows the previous state (11). Maintains the clock at full rate until TLOAD goes active.

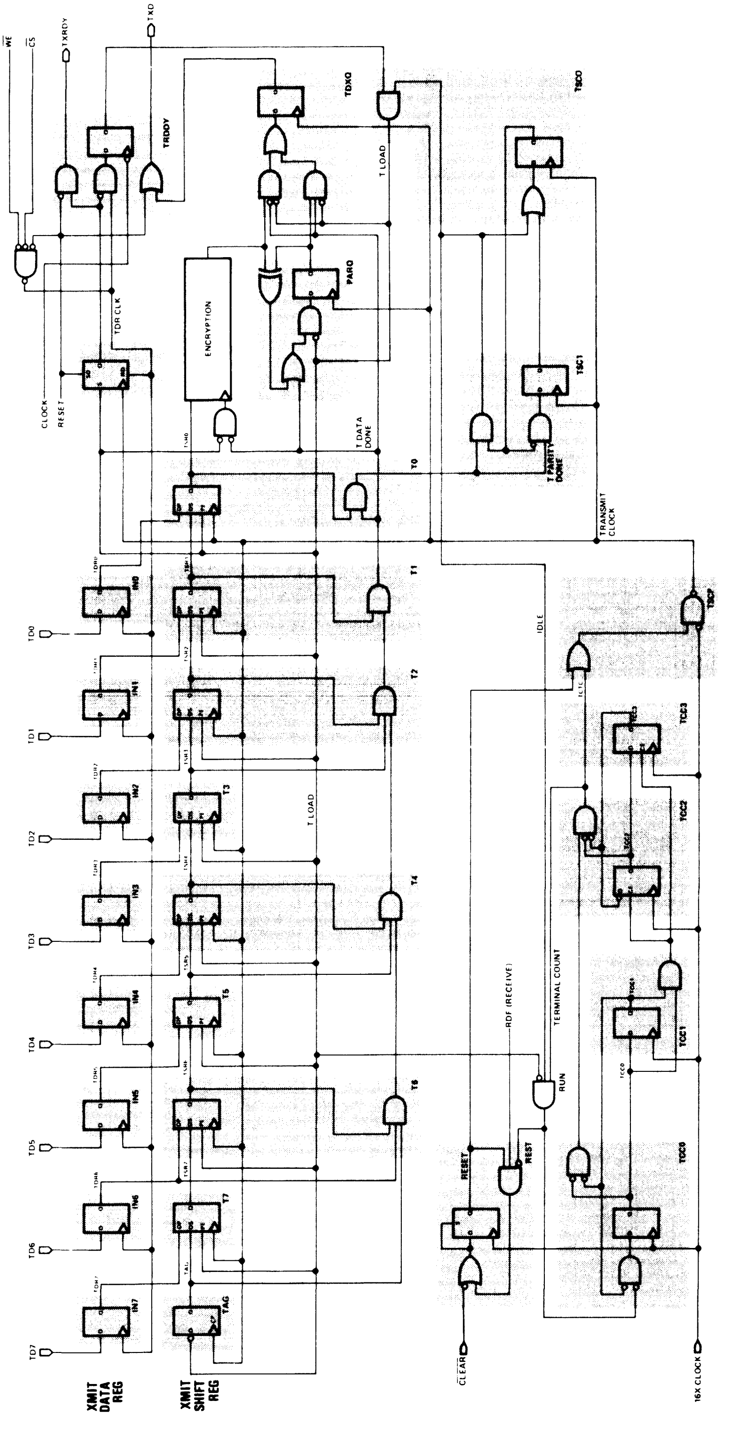


Figure 3. UART Transmitter Schematic

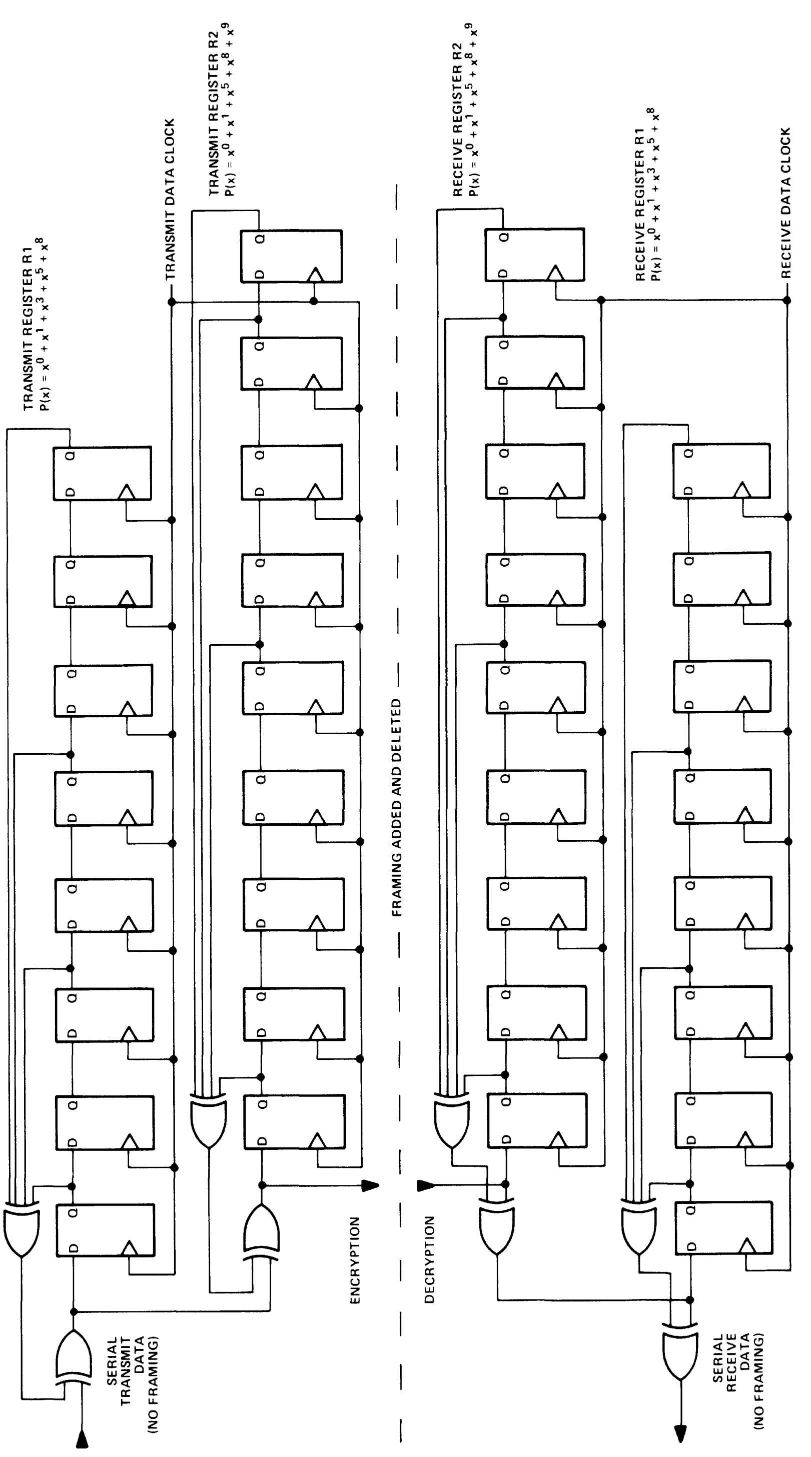

TRANSMIT REGISTER R1
P(x) = x^0 + x^1 + x^3 + x^5 + x^8
TRANSMIT DATA CLOCK
TRANSMIT REGISTER R2
P(x) = x^0 + x^1 + x^5 + x^8 + x^9
RECEIVE REGISTER R2
P(x) = x^0 + x^1 + x^5 + x^8 + x^9
RECEIVE REGISTER R1
P(x) = x^0 + x^1 + x^3 + x^5 + x^8
RECEIVE DATA CLOCK
D
Q
FRAMING ADDED AND DELETED
ENCRYPTION
DECRYPTION
SERIAL TRANSMIT DATA (NO FRAMING)
SERIAL RECEIVE DATA (NO FRAMING)

Figure 4. Encryption/Decryption Schematic

The output of flip-flop PARQ, which is shown below the encryption function, generates parity as it toggles once for each data bit which is a 1. The position of the tag bit is detected by the extent to which the register has filled by all ones. After the data has shifted, PARQ then provides the parity bit and produces the stop bits. TLOAD resets PARQ, TAG and the start bit when the next data word is loaded.

RECEIVER DESIGN

Figure 5 shows the schematic of the receiver portion of the UART. The receiver uses the same 16X clock input as the transmitter. As with the transmitter clock, when the receiver is actively receiving a frame of data, a clock divider located at the bottom of the figure generates a receive clock signal that is one-sixteenth the input clock rate. When the receiver is idle, the clock operates at full rate searching for a start bit.

Data Sampling

The data-input portion at the lower left of the receiver schematic includes a synchronous data filter, the receive data filter (RDF). It provides noise filtering on the input data stream at the 16X clock rate. When a start bit is detected at the data filter output, the clock divider begins counting. After eight clock cycles (mid-bit), the clock divider generates a receive data clock to sample the value of the RDF output. The data is sampled every 16 clock cycles thereafter until a complete frame is shifted in.

Error Detection

Each data bit is shifted into the RSRP flip-flop, where parity is accumulated for comparison with the transmitted value. The framing error logic checks for the correct stop bits when the start bit reaches the last bit of the receive shift register. The Overrun, framing and parity error flags can be read by the processor on data lines D3, D4 and D5, respectively, after RLOAD goes active.

Decryption

At the receiver, the encrypted data is input to a shift register that creates the same feedback data by using the same polynomials as the encryption registers. The feedback data is XORed with the encrypted data to regenerate the original data. With this encryption method, a transmission error causes a difference between the encryption and decryption register data. The decrypted data is incorrect for as long as its takes to flush the error the lngth of the decryption registers.

Receive Shift Register and Data Register

During receiver idle time, the receive shift register located below the receive data register is loaded with all ones so that the start bit (a logic zero) of a frame can be detected when it shifts through to the right end of the register. When the receive shift register is loaded with a complete data byte, RLOAD enables the receive data register. On the next receive clock signal, the data are loaded in parallel into the receive data register and RXRDY output becomes active indicating that the data can be read by the processor. At the same time, the clock divider is reset in preparation for the next data byte and RDF begins monitoring the serial input signal for the next start bit.

LOGIC CELL ARRAY IMPLEMENTATION

Orderly and logical assignment of logic functions to the CLBs and their placement in the LCA contributes to simplified routing and performance optimization in the final implementation of a design. A good design methodology will result in a more effective and efficient implementation. The methodology used in this example involves some preparation before using the Xilinx XACT LCA Development System to implement the design.

This methodology includes three basic steps:

Step 1. Group Logic For CLB and IOB Utilization

The first step in implementing the UART design in an LCA is to examine the schematic diagram to identify CLB segments that will be able to share common inputs. Since the UART design is register-intensive, most CLB groupings consist of a flip-flop function and some associated logic. Figure 5 shows the receiver schematic with the logic functions grouped into CLBs. Shaded boxes indicate the logic to be implemented within single CLBs.

When assigning functions to a CLB on the schematic, note that each CLB is composed of a 4-input combinational logic module, a general-purpose storage element, and routing selection logic. The combinatorial function may be split into two functions, each using up to three of the available variables. Combinatorial functions which share common variables with a flip-flop's function are more effective in the same CLB. I/O Blocks (IOBs) are composed of a three-state output buffer, an input buffer and an input flip-flop. An IOB flip-flop can be used internally if its corresponding pad/pin is not being used for external connections. This was the case for major portions of the encryption/decryption registers.

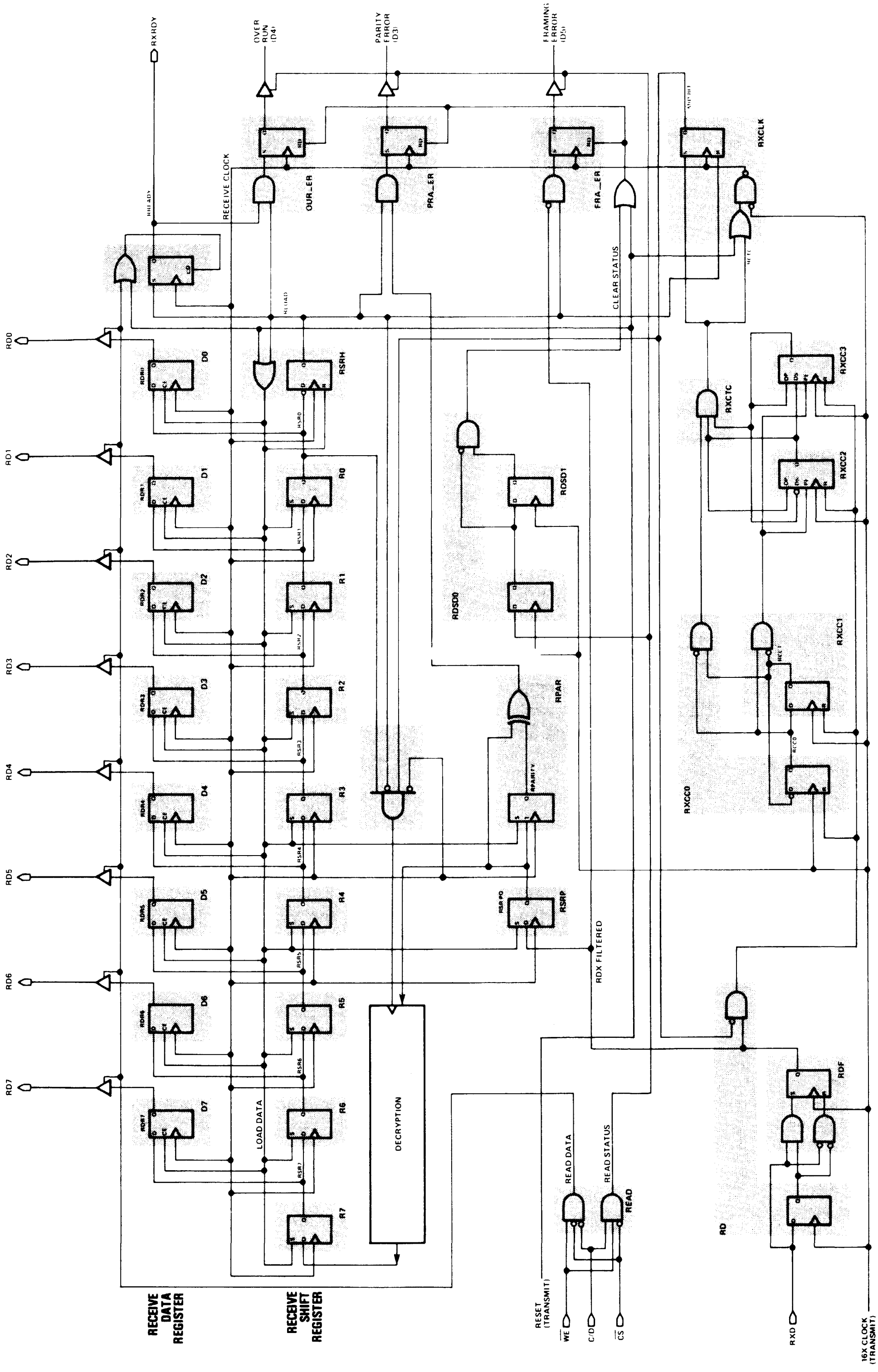


Figure 5. UART Receiver Schematic Showing CLB and One Bit-Slice of Receiver Function

Since the implementation process using XACT will likely reveal alternatives in design, partitioning, or placement for a particular application, it is not necessary that all logic elements be assigned at this point.

Step 2. Assign CLBs and IOBs within the LCA

After grouping logic, a placement plan is prepared. To optimize routing, look for the following when placing CLBs:

- Functions that are repeated (e.g., by data word or bit-slice)
- Good utilization of LCA long lines for common control functions
- Clustering to optimize direct interconnect resources
- Orderly assignment of input and output pins
- Pin requirements for the selected LCA configuration mode

The upper left portion of the receiver diagram is a bit-slice consisting of one bit each of the receive shift register, the receive data register, and the data output buffer. Figure 5 shows this bit-slice function as a outlined box that includes the CLB boxes labeled R7, D7, and the associated I/O buffer. This function is repeated for each of the eight bits and shares common control signals in the design.

Assigning one bit-slice to the CLBs of the top right corner of the LCA, the IOB three-state buffer becomes the receive bit-slice output buffer that connects to the processor data bus. The CLBs in column H and G become the receive data and the receive shift bit-slice, respectively. This horizontal arrangement allows best use of the direct interconnect resources between the CLBs when connections are routed for this function. Repeating this bit-slice function down the right edge of the LCA and extending the function to the full register size allows the use of vertical long lines for the common control signals such as clock, load data and read data signals used by each bit-slice.

Figure 6 shows a completed LCA implementation with CLBs and IOBs placed, configured and routed for the UART application. The receive bit-slice function is indicated by the outlined area at the top right of the LCA. To aid in recognition of the placement of CLBs within the LCA, each CLB is labeled (e.g., r7, d0, fra_er) and corresponds to the CLB groups with the same label in the schematics (e.g., R7, D0, FRA_ER). The UART example allows logical placement of CLBs into transmit and receive halves.

The clock divider is clustered in the center in columns D and E to optimize the direct interconnect resources. It is common for groups of associated blocks to be placed in such clusters.

The transmit data register fits well into the storage elements of the IOBs at the left edge of the LCA. The transmit shift register, together with framing bits and associated logic functions, is placed in a snake-like pattern alternating between columns A and B. This placement allows adjacent CLBs to use direct interconnect resources for the data connections and vertical long lines for common clock and parallel enable signals.

Step 3. Use XACT to Place, Configure and Route CLBs and IOBs

The placement plan prepared in Step 2 facilitates the placement and configuration of CLBs and IOBs with the Xilinx XACT LCA Development System. XACT is an integrated, easy-to-use system that provides complete design automation tools for users to specify and implement designs in an LCA. Placement, configuration and routing of CLBs and IOBs are handled in a graphics-oriented editing environment using the XACT LCA Editor.

Using the LCA Editor for CLB and IOB placement and configuration for an application is an iterative process that employs several facilities in XACT. To efficiently investigate placement alternatives, repetitive functions such as the transmit and receive data and shift registers can be placed, configured, and routed by means of an execution file of edit commands. A file of executable commands can be created and edited using any available text editor. This file contains LCA Editor commands (such as macro calls, add pin and edit block commands) that would normally be used during an editing session.

Standard logic libraries and user-generated macro capabilities can be utilized for faster design entry. User macros can be generated for repetitive functions or groups of CLBs to increase the efficiency of implementation. For example, a macro was generated for the receiver bit-slice function shown in Figure 5. A user macro can be created by choosing the appropriate elements from the Xilinx macro library and/or direct logic definition using XACT facilities and then combining them in a new user macro with the CUTMACRO command.

Once the macro is generated, it is easily used in the LCA implementation. For each occurrence of the function, the macro provides simple and repeatable iteration of the function in CLBs and IOBs on the LCA. Editing of the resulting logic can allow minor variations to individual sections of multiple instances of a macro. For example, in the transmit shift register, signal pin assignments were swapped on some CLBs to optimize routing.

The use of consistent pin sets would have resulted in the use of more routing resources. Figure 6 shows one bit-slice of the transmit register at the left-center edge of the LCA for which a macro was created, repeated and edited to optimize pin assignments.

Attention should be paid to the routing of clock nets. In a fully synchronous design the long line network driven by the global clock buffer (in the upper left corner of the LCA) provides a stable skew-free clock source. Gated clocks and other timing signals may incur routing delays. For example, the most direct routing for the receive clock, generated in CLB RXCLK (row B, column F), would connect the receive clock to a long line in column G for the receive shift register, and to a long line in column F to the error flags, and then from column G to a long line in column H for the receive data register. This route results in a series of small R-C delays due to the impedance of the programmable interconnect points (PIPs) and the high fanout capacitance of the long lines used in the route. The consequence is that the receive clock of column F, and particularly of column H, could incur enough additional delay to cause the next data from the direct routing of the shift register to arrive at the data register before the clock for the current data. This would result in a data hold time violation. The routing can be changed by using the edit net command to connect the receive clock to the long lines of columns F and H and then from H to G. This routing guarantees adequate data hold time for the receive data register inputs.

The method used for routing the transmit clock illustrates a more direct solution to clock routing. The source of transmit clock (tscp in row E, column C) drives a horizontal long line near the bottom of the LCA. The vertical long lines in column A, B and C connect to the horizontal long line to bring transmit clock to the Transmit Registers, the transmit state controller and other logic with equal delays. Figure 6 indicates the routing for the receive and transmit clock by highlighted lines.

The appropriate use of long lines during automatic routing is encouraged, as several techniques are available for placement and configuration. One is to use the edit net command to connect a network source manually to the desired long lines before automatic routing is used to complete the net routing.

DOCUMENTATION

XACT provides complete documentation for the LCA implementation. The documentation is maintained in files that may be printed or viewed on the monitor. These files include the following information:

- LCA File – Contains all configuration, routing and name information
- NET File – List of selected signals, their sources and destinations and propagation delays. Obtained with the Report, QueryNet command.
- BLOCK Files – List of selected CLB and IOB designations, inputs, outputs, clocks, and equations; via Report QueryBlock command.
- LOG File – All Edit commands executed in the last editing session

A .LOG file may be created which is a record of the edit commands of an aborted XACT edit session. The .LOG file can be used to recover an editing session in which a major error was executed. The .LOG file is edited to remove the unwanted commands. Then, with the last saved LCA File, the edited log file can be executed, to restore the last session up to the edited point.

At any point during the design implementation process, XACT can generate the configuration program file that defines the current design. With the XACT development tools, design verification is done through the use of timing analysis and logic simulation. Any required modifications to the design are easily accomplished with the LCA Editor. Also, the design can be physically programmed in an LCA, then analyzed and debugged in the target system using the XACTOR In-Circuit Emulation capabilities.

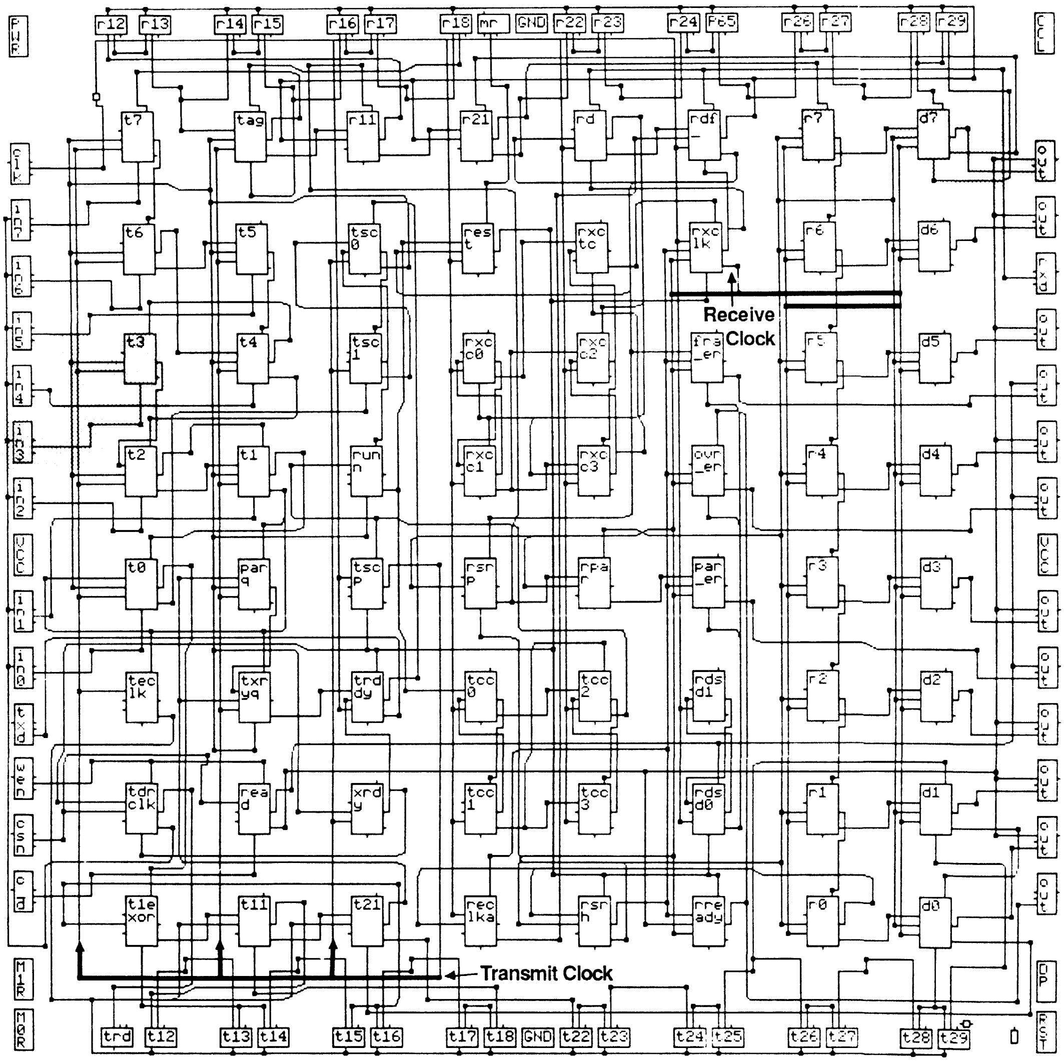


Figure 6. Logic Cell Array Layout with CLB and IOB Placements

NET File Output

```
Querynet: UART1024.LCA (2064PC68-50), XACT 1.22, 13:24:27 SEP 10, 1986

  c_d . . . . . . . .   P24.I (c_d_) . . .    2 GB.D (read)
  clearn. . . . . . .   P2.I (mr). . . . .    2 BD.A (rest)
  clock . . . . . . .   P11.I (clk). . . .    0 CLK.AA.I
  clrstat . . . . . .   GF.Y (rdsd0) . . .   15 DF.D (ovr_er)
                                             16 CF.D (fra_er)
                                             13 EF.D (par_er)
  cout2 . . . . . . .   DA.Y (t2). . . . .    0 DB.B (t1)
  cout4 . . . . . . .   CB.Y (t4). . . . .    9 DA.A (t2)
  cout6 . . . . . . .   BA.X (t6). . . . .    2 CB.B (t4)
  cp. . . . . . . . .   CLK.AA.O . . . . .    0 GF.K (rdsd0)
                                              0 FF.K (rdsd1)
                                              0 DE.K (rxcc3)
                                              0 BF.B (rxclk)
                                              0 HD.B (reclka)
                                              0 EC.B (tscp)
                                              0 FC.K (trddy)
                                              0 GE.K (tcc3)
                                              0 GD.K (tcc1)
                                              0 FE.K (tcc2)
                                              0 FD.K (tcc0)
                                              0 DD.K (rxcc1)
                                              0 BD.K (rest)
                                              0 CD.K (rxcc0)
                                              0 AF.K (rdf_)
                                              0 AE.K (rd)
                                              0 CE.K (rxcc2)
  csn . . . . . . . .   P23.I (csn_) . . .    8 GB.B (read)
                                              2 GA.C (tdrclk)
  encryptionq . . . .   HC.X (t21) . . . .   15 EB.B (parq)
                                             14 FB.C (txryq)
  frame_error . . . .   CF.Y (fra_er). . .    8 P55.O (out_fe)
  midbit. . . . . . .   BF.X (rxclk) . . .    0 AF.D (rdf_)
                                              6 AE.D (rd)
  overrun_error . . .   DF.Y (ovr_er). . .    5 P53.O (out_or)
  parity_error. . . .   EF.Y (par_er). . .    4 P50.O (out_pe)
  pe. . . . . . . . .   EE.Y (rpar). . . .    0 EF.B (par_er)
  rlxor . . . . . . .   AC.Y (r11) . . . .    0 AD.B (r21)
  rcc0. . . . . . . .   CD.X (rxcc0) . . .    0 DD.B (rxcc1)
  rcc1. . . . . . . .   DD.X (rxcc1) . . .    0 CD.C (rxcc0)
  rcc2. . . . . . . .   CE.X (rxcc2) . . .    0 DE.A (rxcc3)
                                              0 BE.C (rxctc)
  rcc3. . . . . . . .   DE.X (rxcc3) . . .    0 CE.C (rxcc2)
                                              7 BE.D (rxctc)
  rcca. . . . . . . .   CD.Y (rxcc0) . . .    2 BE.B (rxctc)
  rccb. . . . . . . .   DD.Y (rxcc1) . . .    6 CE.B (rxcc2)
                                              0 DE.B (rxcc3)
  rcphold . . . . . .   AE.X (rd). . . . .    7 CE.A (rxcc2)
                                             16 DE.C (rxcc3)
                                             16 CD.D (rxcc0)
                                             16 DD.A (rxcc1)
  rctc. . . . . . . .   BE.Y (rxctc) . . .    2 BF.A (rxclk)
  rdf . . . . . . . .   AF.Y (rdf_). . . .   22 ED.A (rsrp)
                                             11 AE.C (rd)
                                             13 CF.B (fra_er)
                                             17 BD.C (rest)
  rdq . . . . . . . .   AE.Y (rd). . . . .    0 AF.B (rdf_)
  rdr0q . . . . . . .   HH.X (d0). . . . .    2 P47.O (out0)
  rdr1q . . . . . . .   GH.X (d1). . . . .    0 P48.O (out1)
  rdr2q . . . . . . .   FH.X (d2). . . . .    0 P49.O (out2)
  rdr3q . . . . . . .   EH.X (d3). . . . .    0 P51.O (out3)
  rdr4q . . . . . . .   DH.X (d4). . . . .    0 P54.O (out4)
  rdr5q . . . . . . .   CH.X (d5). . . . .    0 P56.O (out5)
  rdr6q . . . . . . .   BH.X (d6). . . . .    0 P58.O (out6)
  rdr7q . . . . . . .   AH.X (d7). . . . .    0 P59.O (out7)
```

BLOCK File Output (Short Form)

```
Queryblk: UART1024.LCA (2064PC68-50), XACT 1.22, 13:25:52 SEP 10, 1986

d0        HH  te2x      rsr0qy    xfer      te28      rxcp      rdr0q     te2xor
d1        GH  te25      rsr1qy    xfer      te29      rxcp      rdr1q     te2x
d2        FH  (.a)      rsr2qy    xfer      (.d)      rxcp      rdr2q     (.y)
d3        EH  (.a)      rsr3qy    xfer      (.d)      rxcp      rdr3q     (.y)
d4        DH  (.a)      rsr4qy    xfer      (.d)      rxcp      rdr4q     (.y)
d5        CH  (.a)      rsr5qy    xfer      (.d)      rxcp      rdr5q     (.y)
d6        BH  (.a)      rsr6qy    xfer      (.d)      rxcp      rdr6q     (.y)
d7        AH  re28      rsr7qy    xfer      re29      rxcp      rdr7q     re2xora
fra_er    CF  (.a)      rdf       rload     clrstat   rxcp      (.x)      frame_er
ovr_er    DF  rrdy      (.b)      rload     clrstat   rxcp      (.x)      overrun_
par_er    EF  (.a)      pe        rload     clrstat   rxcp      (.x)      parity_e
parq      EB  tdatadon  encrypti  tload     (.d)      txcp      tparityq  (.y)
r0        HG  rsr1q     (.b)      xfer      (.d)      rxcp      rsr0q     rsr0qy
r1        GG  rsr2q     (.b)      xfer      (.d)      rxcp      rsr1q     rsr1qy
r11       AC  rsrpq     relxora   recp      (.d)      (.k)      rel1      rlxor
r2        FG  rsr3q     (.b)      xfer      (.d)      rxcp      rsr2q     rsr2qy
r21       AD  re2xora   rlxor     recp      (.d)      (.k)      rsriq     re21
r3        EG  rsr4q     (.b)      xfer      (.d)      rxcp      rsr3q     rsr3qy
r4        DG  rsr5q     (.b)      xfer      (.d)      rxcp      rsr4q     rsr4qy
r5        CG  rsr6q     (.b)      xfer      (.d)      rxcp      rsr5q     rsr5qy
r6        BG  rsr7q     (.b)      xfer      (.d)      rxcp      rsr6q     rsr6qy
r7        AG  rsriq     (.b)      xfer      (.d)      rxcp      rsr7q     rsr7qy
rd        AE  rxd       (.b)      rdf       midbit    cp        rcphold   rdq
rdf_      AF  rxd       rdq       recpe     midbit    cp        recp      rdf
rdsd0     GF  read_sta  rdstat    (.c)      reset     cp        rdstat0   clrstat
rdsd1     FF  (.a)      (.b)      rdstat0   (.d)      cp        rdstat    (.y)
read      GB  wen       csn       (.c)      c_d       (.k)      read_sta  read_dat
reclka    HD  rload     cp        rsr0q     (.d)      (.k)      (.x)      recpe
rest      BD  clearn    run       rdf       (.d)      cp        reset     (.y)
rpar      EE  xfer      rsrpq     rxcp      (.d)      (.k)      (.x)      pe
rready    HF  reset     rload     read_dat  (.d)      rxcp      rrdy      (.y)
rsrh      HE  reset     rsr0q     rxcp      (.d)      (.k)      rload     xfer
rsrp      ED  rdf       (.b)      rxcp      xfer      (.k)      (.x)      rsrpq
runn      DC  tsca      tctc      (.c)      tload     (.k)      (.x)      run
rxcc0     CD  (.a)      (.b)      rcc1      rcphold   cp        rcc0      rcca
rxcc1     DD  rcphold   rcc0      (.c)      (.d)      cp        rcc1      rccb
rxcc2     CE  rcphold   rccb      rcc3      (.d)      cp        rcc2      (.y)
rxcc3     DE  rcc2      rccb      rcphold   (.d)      cp        rcc3      (.y)
rxclk     BF  rctc      cp        rload     reset     rxcp      midbit    rxcp
rxctc     BE  (.a)      rcca      rcc2      rcc3      (.k)      (.x)      rctc
t0        EA  tsr1q     tdatadon  tload     tdr0q     txcp      tsr0q     tparityd
t1        DB  tsr2q     cout2     tload     tdr1q     txcp      tsr1q     tdatadon
t11       HB  (.a)      telxor    tecp      te18      (.k)      t11q      tlxor
tlexor    HA  tsr0q     (.b)      te15      te13      (.k)      (.x)      telxor
t2        DA  cout4     tsr3q     tload     tdr2q     txcp      tsr2q     cout2
t21       HC  (.a)      tlxor     tecp      te2xor    (.k)      encrypti  te21
t3        CA  tsr4q     (.b)      tload     tdr3q     txcp      tsr3q     (.y)
t4        CB  tsr5q     cout6     tload     tdr4q     txcp      tsr4q     cout4
t5        BB  (.a)      tsr6q     tload     tdr5q     txcp      tsr5q     (.y)
t6        BA  tsr7q     (.b)      tload     tdr6q     txcp      cout6     tsr6q
t7        AA  tagq      (.b)      tload     tdr7q     txcp      tsr7q     (.y)
tag       AB  re18      re13      tload     re15      txcp      tagq      relxora
tcc0      FD  (.a)      run       tcc1q     (.d)      cp        tcc0q     tc00
tcc1      GD  tcc0q     (.b)      (.c)      (.d)      cp        tcc1q     tc03
tcc2      FE  tc03      tc00      tcc3q     (.d)      cp        tcc2q     tctc
tcc3      GE  tcc2q     tc03      (.c)      (.d)      cp        tcc3q     (.y)

t24       P38 te23      (.t)      tecp      te24
t25       P39 te24      (.t)      tecp      te25
t26       P40 te25      (.t)      tecp      te26
t27       P41 te26      (.t)      tecp      te27
t28       P42 te27      (.t)      tecp      te28
t29       P43 te28      (.t)      tecp      te29
trdy_     P27 trdy      (.t)      (.k)      (.i)
```

BLOCK File Output (Long Form)

```
                  D=
                  K=cp
read      GB  X:G Y:F F:A:B:D G:A:B:D Q: SET: RES: CLK:
              F = A*-B*-D
              G = A*-B*D
                  A=wen                     X=read_status
                  B=csn                     Y=read_data
                  C=
                  D=c_d
                  K=
reclka    HD  X: Y:F F:A:B:C G: Q: SET: RES: CLK:
              F = -A*-B*C
                  A=rload                   X=
                  B=cp                      Y=recpe
                  C=rsr0q
                  D=
                  K=
rest      BD  X:Q Y:Q F:A:C:B:Q Q:FF SET:F RES: CLK:K
              F = -A+-B*C*Q
                  A=clearn                  X=reset
                  B=run                     Y=
                  C=rdf
                  D=
                  K=cp
rpar      EE  X: Y:G F:A:B:Q G:B:Q Q:FF SET: RES: CLK:C
              F = A+(B@Q)
              G = Q@B
                  A=xfer                    X=
                  B=rsrpq                   Y=pe
                  C=rxcp
                  D=
                  K=
rready    HF  X:Q Y: F:B:Q G:C:A Q:FF SET: RES:G CLK:K
              F = B+Q
              G = C+A
                  A=reset                   X=rrdy
                  B=rload                   Y=
                  C=read_data
                  D=
                  K=rxcp
rsrh      HE  X:Q Y:G F:A:B:Q G:A:Q Q:FF SET: RES: CLK:C
              F = -B*-(A+Q)
              G = A+Q
                  A=reset                   X=rload
                  B=rsr0q                   Y=xfer
                  C=rxcp
                  D=
                  K=
rsrp      ED  X: Y:Q F:A:D G: Q:FF SET: RES: CLK:C
              F = A+D
                  A=rdf                     X=
                  B=                        Y=rsrpq
                  C=rxcp
                  D=xfer
                  K=
runn      DC  X: Y:G F: G:A:B:D Q: SET: RES: CLK:
              G = A*B*-D
                  A=tsca                    X=
                  B=tctc                    Y=run
                  C=
                  D=tload
                  K=
rxcc0     CD  X:Q Y:G F:D:C G:C:Q Q:FF SET: RES: CLK:K
              F = -D*-C
              G = C*-Q
                  A=                        X=rcc0
```

File Format Examples

A Printer Buffer Controller

INTRODUCTION

This application note presents a practical application of XILINX's XC2064 Logic Cell™ Array (LCA) in a typical systems application. The objectives are to demonstrate the versatility of the LCA by designing it into a practical and useful application and, in so doing, to present to the reader some useful concepts and techniques for designing with LCAs. The application example presented here is a printer buffer subsystem for driving printers with a parallel, Centronics-type interface, in which the LCA is programmed to act as the printer interface and FIFO (first-in-first-out) buffer controller. As shown in the block diagram in Figure 1, the LCA, together with external byte-wide static RAM, constitutes a complete printer buffer subsystem. This circuit permits a host system's processor to write characters into the FIFO buffer as if it were a high speed printer port. The LCA stores the print data in the external buffer memory until the printer is ready to accept it. Data stored in memory is then read back and delivered to the printer at its maximum rate until the buffer is empty. Once print data is loaded into the buffer, the actual printing operation is independent of and transparent to the processor. The result, from the processor's point of view, is an appreciable speedup of print operations, thereby freeing the processor earlier to perform its next task. For instances in which the entire print file fits into the print buffer, the print operation becomes a separate, parallel task requiring no further attention from the processor.

The print buffer controller can be thought of as being comprised of three major functional areas. They are:

A microprocessor bus interface which includes:

- A data bus (for receiving data and reading status flags)
- A 2-level input FIFO

A parallel printer interface with handshake controls

A circular queue mechanism consisting of:

- Read-address and store-address pointers
- An address comparator for sensing FULL condition
- An address comparator for sensing EMPTY condition
- A static RAM interface
- Associated timing and control logic

IMPLEMENTATION ISSUES

In defining the controller's functions and how these funtions are to be implemented in a LCA, attention must be given to certain physical and electrical limitations which dictate whether the application will fit into a single LCA device shown in Figure 2. The following parameters serve as good indicators in "sizing" a logic circuit:

- The number of input/output pins (IOBs) required
- The number of logic blocks (CLBs) required
- The number of storage elements required (e.g., flip-flops)
- The clock frequency and signal propagation delays

One of the primary criteria in sizing the application is the number of I/O pins required by the circuit; if the number exceeds 58, then an XC2018 or multiple XC2064s may be required. The pin count analysis for the print buffer controller example is outlined in Table 1. Initial interpretation of that summary indicates the number of I/O pins on the XC2064 should be adequate for this application.

Let us examine the circuit requirements for the print buffer controller. Two necessary elements are the host processor interface and the printer interface. The host system's processor should be able to read status information about the buffer's current state (e.g., FULL and BUSY status flags) prior to each write operation in order to prevent buffer overflow and to determine when the print job is complete. The parallel printer interface should be able to recognize and respond to the printer's ACK (Acknowledge) and BUSY control lines with the appropriate handshake.

Another necessary element is the FIFO (first-in first-out) memory. Because the objective was to be able to buffer up to 64K bytes of print data, external RAM devices were used instead of classical FIFO devices because they are more economical and offer higher capacities. The RAM serves as the actual print storage buffer, whereas the LCA provides the control and timing for the RAM as well as controlling the data path. The LCA also controls the addressing of the RAM to make it behave as a FIFO-like circular queue. (For more information on circular queues, see the side box that appears later in this appnote.) Once print data is loaded into the queue, the actual printing operation is independent and transparent to the processor.

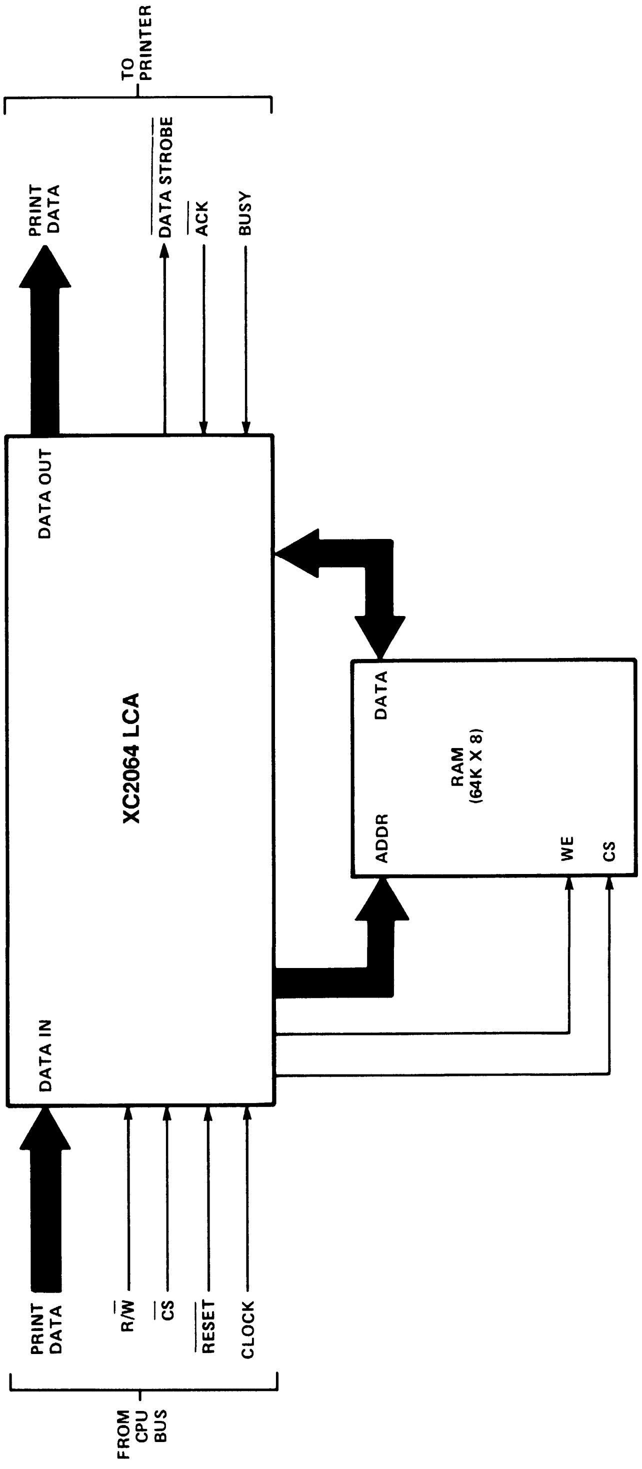


Figure 1. Block Diagram of Print Buffer

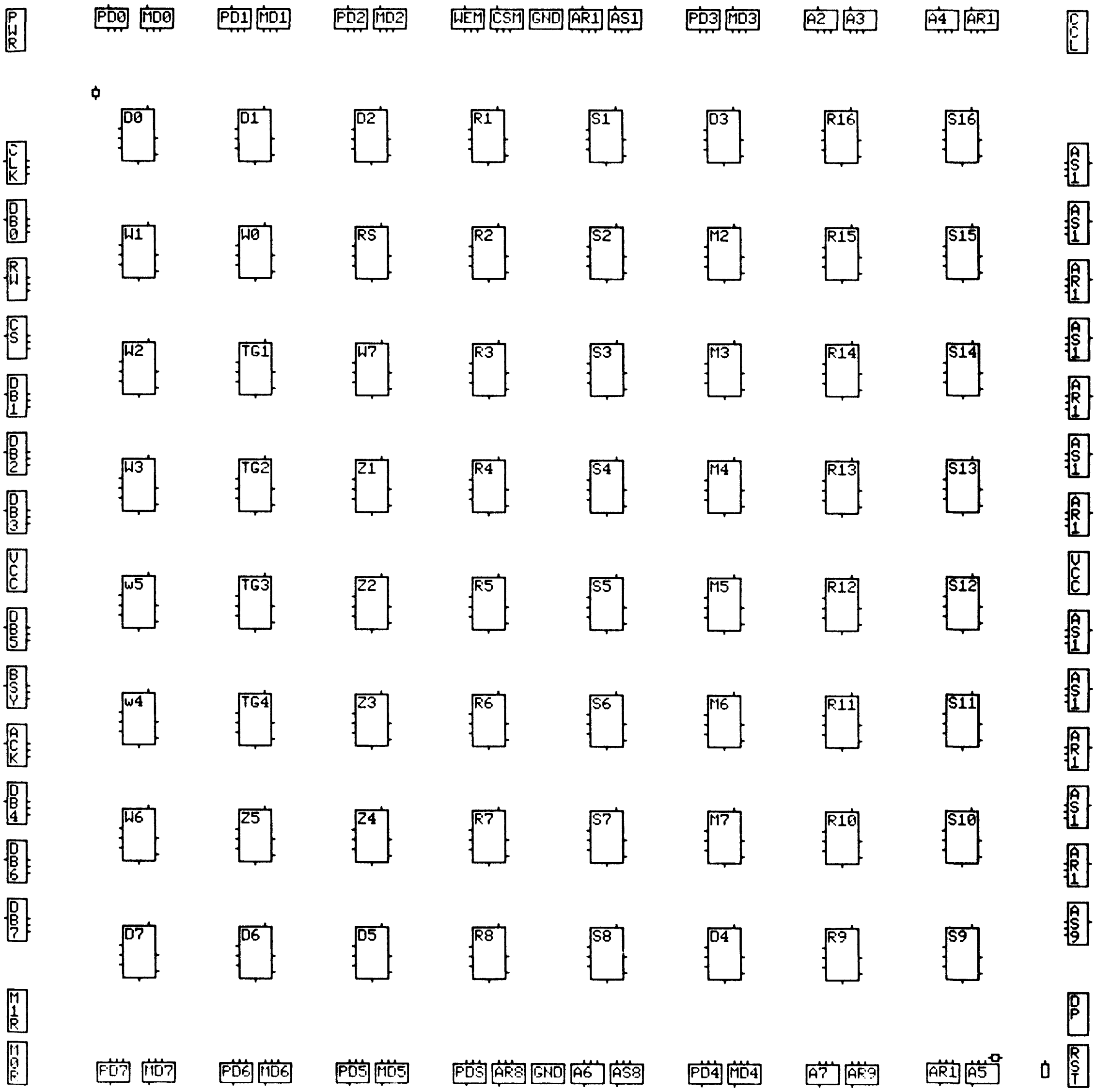


Figure 2. Placement of I/O and Logic Blocks

To better understand what is required by the circular queue's control logic, consider that in order to read and write to the RAM in different locations, two separate 16-bit address counters must be kept: one representing the current **read** pointer, and the second one the current store pointer. The values of these two address counters must be compared in order to determine whether the queue is full or empty. These two addresses are multiplexed to form the 16-bit phys-ical memory address bus. Although this multiplexing is easily accomplished using CLBs (requiring one CLB for each multiplexed address bit), an alternative and more efficient method of multiplexing buses is to use the 3-state output feature of the IOBs. With this approach, the output pins of the two address buses are paired bit-for-bit and tied together externally with only one set of IOBs enabled at any given time. This technique trades off LCA I/O pins for CLBs. For reasons that become apparent later, a combination of these multiplexing techniques was chosen for this design.

Several constraints of the XC2064 LCA must be kept in mind when assigning pinout definitions. One issue concerns assignment of I/O pins and the placement of registered input buses relative to each other. Since all IOB flip-flops on any given "side" of the LCA (i.e., top, bottom, left or right) share a common clock, registered input buses requiring different clock signals must be positioned on different sides of the LCA.

The Data Path

A good starting point in many designs is the consideration of the data path. In this particular application, the XC2064 must be able to accept print data from the processor, store the data into the RAM, retrieve the data from the RAM and then finally deliver that data to the printer interface. Input data from the processor passes through a two-stage FIFO within the controller to speed up the operation and absorb minor data rate variations in storing data into RAM. A one-bit wide "slice" of the data path through the XC2064 is shown in Figure 3 and illustrates how maximum use can be made of the IOB resources. The complete, 8-bit wide data path is shown in Figure 4 along with the logic necessary to control data flow. A write strobe causes data on the processor's data bus to be captured in Register 1, which is comprised of the input flip-flops in the associated IOBs. The data in Register 1 is then synchronously transferred (in FIFO-like fashion) to Register 2 before being presented to the static RAM interface.

Table 1: Pin Requirements For the Print Buffer Controller

Processor bus interface requires:	
a Chip Select pin (CS),	1 pin
a Read/Write pin (R/W), and	1 pin
eight Data bus pins (D0–D7)	8 pins
Parallel (Centronics-type) printer interface requires:	
a Data Strobe output (DS),	1 pin
eight printer Data outputs (PD0–PD7),	8 pins
a printer Busy input (BUSY), and	1 pin
a printer Acknowledge input (ACK)	1 pin
Buffer memory interface requires:	
a memory Chip Select output (CSM),	1 pin
a memory Write Enable output (WEM),	1 pin
a memory Address bus	16 pins
(outputs MA0–MA15),	
a bidirectional memory Data bus (MD0–MD7)	8 pins
Miscellaneous:	
a 10 MHz system clock input (CLOCK)	1 pin
a "master reset" control (RESET)	1 pin

Moving in the other direction, print data read from the RAM are captured in the input (IOB) flip-flops (Reg 3) used for the bidirectional memory data bus pins, MD0–MD7. These data are then presented to the printer interfaces's data lines, PD0–PD7. The bidirectional memory data bus pins are controlled by a timing circuit (see Timing Generation and Figure 5) in such a way that half of the time they are inputs for reading data from RAM, and half of the time they are outputs for writing data to the RAM. The LCA also controls the read/write timing of the RAM by controlling both its CE (Chip Enable) and WE (Write Enable) pins.

Two portions of the data path make use of input flip-flops within the I/O blocks: namely the processor data bus and the RAM data bus. Each set of flip-flops is clocked at different times and under different conditions. Since the XC2064 constrains all IOBs along any one edge of the LCA to share a common clock, these input data bus registers must be grouped together and located on different edges of the LCA. In the final design, the memory data bus pins are split equally between the top and bottom sides of the LCA due to layout considerations. See Figure 11.

TIMING GENERATION

Before discussing details of the data path control logic, it is useful to examine the basic timing employed in that logic. The timing control logic for the data path is shown in Figure 5 along with details of the memory timing. Let us assume we have available a 10 MHz system clock (100 ns clock periods). Since most static RAMs have read and write cycle times of less than 400 ns and a minimum write strobe width of 200 ns or less, these values should accommodate most SRAMs. Slightly faster or slower memory operation is easily attained by adjusting the system clock frequency. A four-stage Johnson counter is used to generate 8 clock states from which the necessary control signals can be decoded in a glitch-free manner. As shown in the timing diagrams, the memory interface continually alternates between 4 states of write cycle followed by 4 states of read cycle. All actual read or write operations with the RAM fit into these time slots. Some static RAMs require nonzero data setup and hold times before and after the write strobe. Consequently, the active-low Write Enable (WE) strobe to the RAM starts one state after the write cycle begins and ends one state before the write cycle ends. Similarly, during read cycles, the data read from the RAM is actually sampled one state prior to the end of the read cycle.

For higher performance systems, this arrangement can be easily modified for tighter timing of read and write cycles by adding more states to the timing sequence and increasing the externally supplied clock frequency.

The CLB partitioning for the timing circuit is shown as shaded areas in Figure 5, with each shaded area representing one CLB. The circuit requires four full CLBs plus a partial CLB which is shared with another design function.

DATA PATH CONTROL LOGIC

Figure 4 shows not only the full data path through the LCA, but also the logic necessary to control the data flow. This logic performs several functions vital to controller operation. They include:

- Coordinating data flow from the host processor to the RAM
- Coordinating data flow from the RAM to the printer interface
- Making status information available at the processor interface
- Sensing and generating the appropriate printer handshake control signals

Note that some of the flip-flops used in Figure 4 are represented as synchronous Set-Reset flip-flops. Although not typically available in other types of logic, these flip-flop types are especially useful in CLB logic since they simplify logic schematics and aid in understanding circuit operation. Their operation is similar to a JKFF, except for the case where both Set and Reset inputs are aserted simultaneously. This potential conflict is resolved by choosing either the Set or Reset to be the dominant condition (indicated on the symbol as a dot after either the S or R label). Figure 6 illustrates how easily these flip-flop types can be implemented using CLBs.

With a clearer understanding of these circuit elements, let us take a closer look at the details of the control logic.

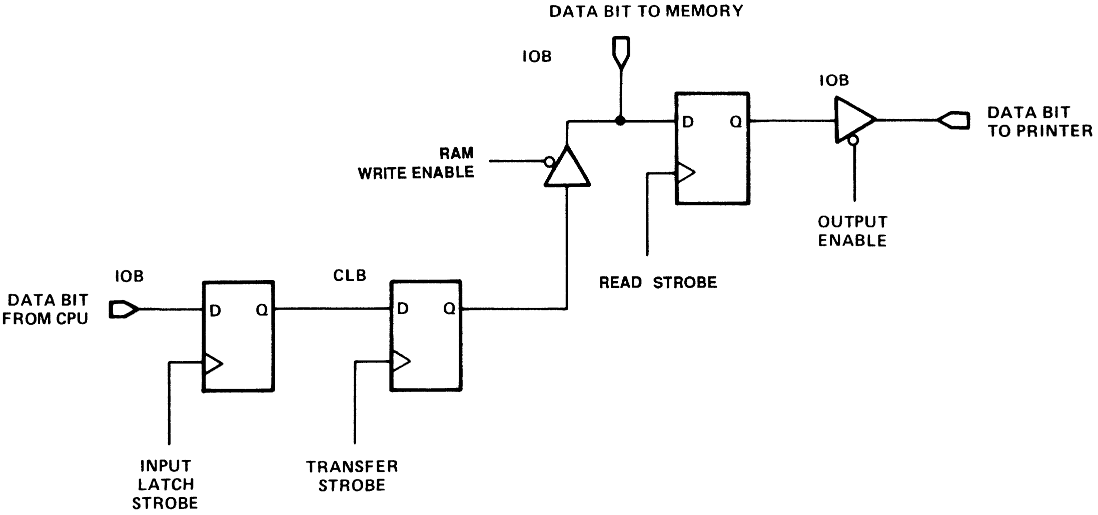


Figure 3. One-Bit Slice of Data Path

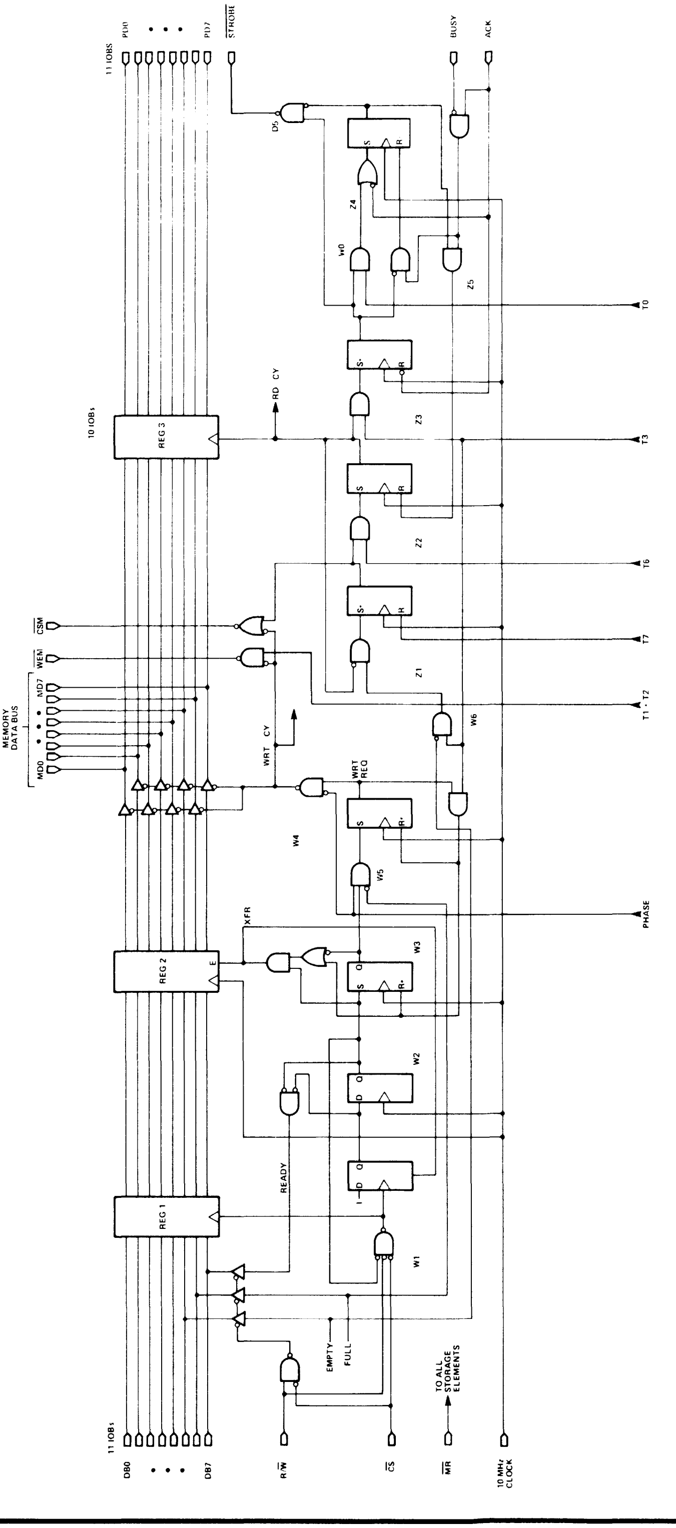
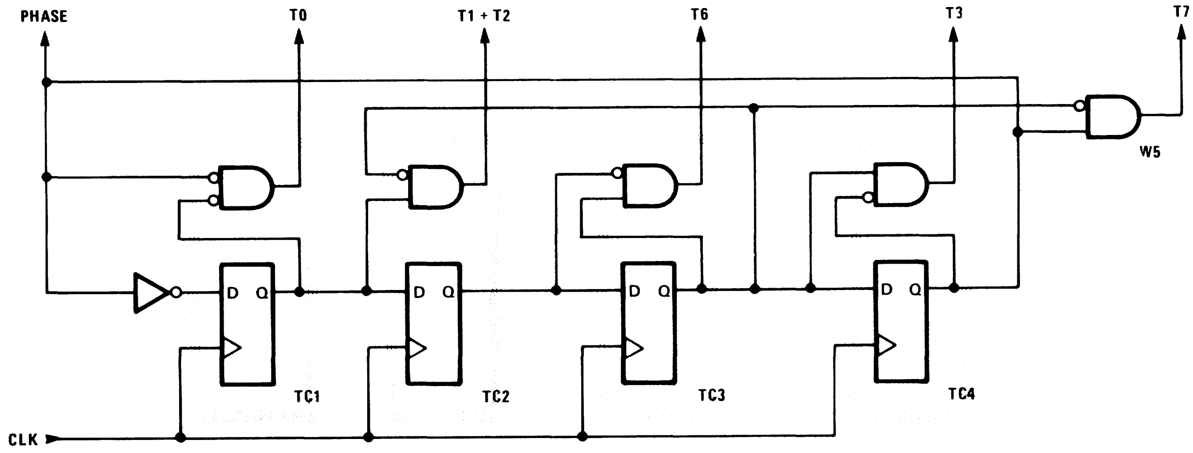


Figure 4. The Data Path and Control Logic



(200 NS) MEMORY WRITE STROBE: 1000 = T1, 1100 = T2

MEMORY WRITE CYCLE (400 NS):
0000 = T0
1000 = T1
1100 = T2
1110 = T3

MEMORY READ CYCLE (400 ns):
1111 = T4
0111 = T5
0011 = T6
0001 = T7

ASSUME 10 MHz CLOCK FREQUENCY

Figure 5. Timing Generation Logic

DETAILED DESCRIPTION OF DATA PATH CONTROL LOGIC

The data presented by the host CPU on the D0–D7 data bus pins is captured by the IOB's input flip-flops (collectively referred to here as Register 1) on the rising (trailing) edge of the active-LOW write pulse as shown in Figure 7. Flip-flop W1 (FF-W1) is set to indicate that new data has been loaded into register Reg 1. This causes the READY status signal to be deasserted, and resynchronizing flip-flop FF-W2 to be set on the following clock edge. The data byte in Reg 1 will be copied into Reg 2 (the FIFO register) on the next clock edge, providing that Reg 2 doesn't currently contain valid data. If Reg 2 does contain valid data, then the new data byte is held in Reg 1 until Reg 2 becomes empty. Once the new data value is loaded into Reg 2, both FF-W1 and then FF-W2 are cleared, causing the READY status line to be asserted again. By sensing READY, the host CPU knows that the next data byte can be written into Reg 1. This two-deep FIFO mechanism permits fast loading of the RAM with only a minimum amount of time during which the interface is not ready.

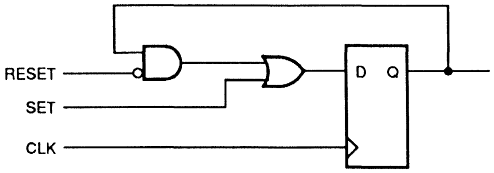
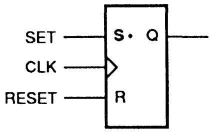
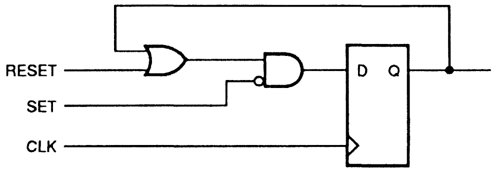
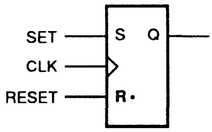
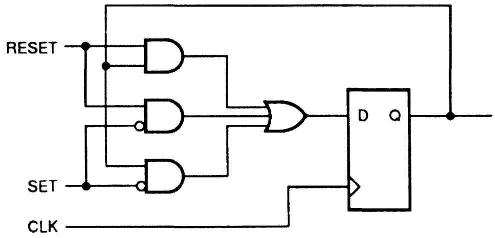
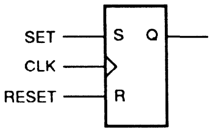
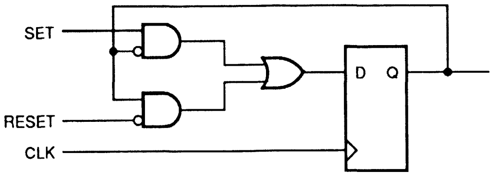
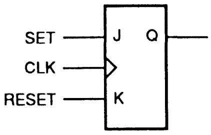
Once new data is present in Reg 2, the next clock edge which occurs prior to the start of the next write PHASE causes the WRT REQuest flag (FF-W4) to be set provided however, that the RAM buffer is not full. This guarantees the integrity of the RAM write operation by insuring that the write cycle is not cut short by starting part way through a write time slot. The next write time slot (defined by the PHASE signal) is then used to present the byte to the memory data bus pins. During this time slot, the Chip Select for the memory (CSM) is asserted LOW, and the Write Enable (WEM) is pulsed LOW during the T1+T2 time states. At the end of the write cycle, the WRT REQ bit (FF-W4) is reset and the Store address pointer incremented.

Reading data from the memory and presenting it to the printer interface is accomplished in a similar fashion, as shown in Figure 8. Whenever the address comparators indicate that the RAM buffer is not EMPTY, the next available read time slot (as again determined by the PHASE signal) is used to read the next data byte from the RAM. In order to insure that an entire read time slot is used and not just a partial one, FF-Z1 is set at the end of T3 and cleared at the end of T7. The FF-Z1 output is used as the read cycle (signal RD CY) component of the RAM's chip select. One time state prior to the end of the read cycle (i.e., at the end of T6), FF-Z2 is set, which causes the data being read from the RAM to be clocked into Reg 3. Once FF-Z2 is set, FF-Z1 will clear on the next clock edge, ending the RAM read cycle and advancing the Read address counter. Flip-flops FF-Z3 and FF-Z4 control the timing and handling of the print data. Since most parallel printers specify a data setup time, a data strobe pulse width and a data hold time of approximately 0.5 µs, these flip-flops are used to generate that particular timing. Assuming a 10 MHz clock is used, FF-Z3 is set at the end of the next T3 state, which is exactly five 100ns periods after data became valid at Reg 3. FF-Z4 is set at T0, exactly five 100ns periods after FF-Z3 went set. The condition in which FF-Z3 is set and FF-Z4 is not yet set is used to generate the data strobe pulse with the proper timing. This meets the data setup time and the data strobe pulse width requirements for most parallel printers. The printer responds some time later with an asynchronous ACK (Acknowledge) handshake signal and possibly a BUSY status signal. When the ACKnowledge signal goes LOW (active), FF-Z3 is cleared. Later, when both ACK and BUSY become inactive, both FF-Z2 and FF-Z4 are also cleared. Until FF-Z2 is cleared, FF-Z1 cannot be set and new data cannot be read from the RAM. Consequently, data overruns cannot occur during the time required by the printer to complete its operation and release its BUSY signal.

Review of Circular Queue Concepts

The familiar first-in-first-out (FIFO) memory is ideally suited to this print buffer application. Unfortunately, dedicated FIFO memory devices are generally expensive and very limited in size. However, an ordinary byte-wide static RAM, together with a moderate amount of control logic, can be made to function like a FIFO. An LCA device can be used effectively to implement this structure, commonly referred to as a circular queue. A circular queue can be thought of as a read-write memory with two address pointers: one pointer referred to as the store pointer, S, and the second one referred to as the read pointer, R. The read pointer indicates the address in the buffer memory where the next character to be read from the queue is located. The store pointer indicates the address where the next character to be placed into the queue will be stored. After a data word is entered or retrieved from the queue, the appropriate address pointer is incremented by one. Whenever one of the pointers reaches the last (highest) address in the read-write memory, it automatically wraps around to point to the first (lowest) memory address. If the memory size is a power of two, then these pointers can be implemented with simple 2^n modulo counters. In order to detect the queue-empty and-queue full conditions and prevent overruns and underruns, some simple rules are followed:

1. Upon initialization, set R = S
2. Whenever R = S, queue is empty; and
   whenever S + 1 = R, queue is full
3. In writing to the queue:
   first verify that S+1 does not equal R; if it does,
   then queue is full, else write the new entry at
   address S and increment S by one.
4. In reading from the queue:
   first verify that R does not equal S; if it does,
   then queue is empty, else read the entry at
   address R and increment R by one.

CIRCUIT DEFINITION	LOGIC SYMBOL	EQUATION AND KARNAUGH MAP
TYPE A: SET-DOMINANT WHEN S=R=1 		NEXT Q = $= SET + \overline{RESET} \cdot Q$ NEXT Q SET 0 1 1 0 1 1 1 0 Q RESET
TYPE B: RESET-DOMINANT WHEN S=R=1 		NEXT Q = $= \overline{RESET} \cdot (SET + Q)$ NEXT Q SET 0 1 0 0 1 1 0 0 Q RESET
TYPE C: HOLD PRESENT STATE WHEN S=R=1 		NEXT Q: $= \overline{RESET} \cdot (SET + Q) + SET \cdot RESET \cdot Q$ $= \overline{RESET} \cdot (SET + Q) + SET \cdot Q$ NEXT Q SET 0 1 0 0 1 1 1 0 Q RESET
TYPE D: CHANGE STATE WHEN S=R=1 (JKFF) 		NEXT Q: $= \overline{RESET} \cdot (SET + Q) + SET \cdot RESET \cdot \overline{Q}$ $= \overline{RESET} \cdot Q + SET \cdot \overline{Q}$ NEXT Q SET 0 1 1 0 1 1 0 0 Q RESET

0010016 1

Figure 6. Flip-flops Used in the Printer Buffer Controller

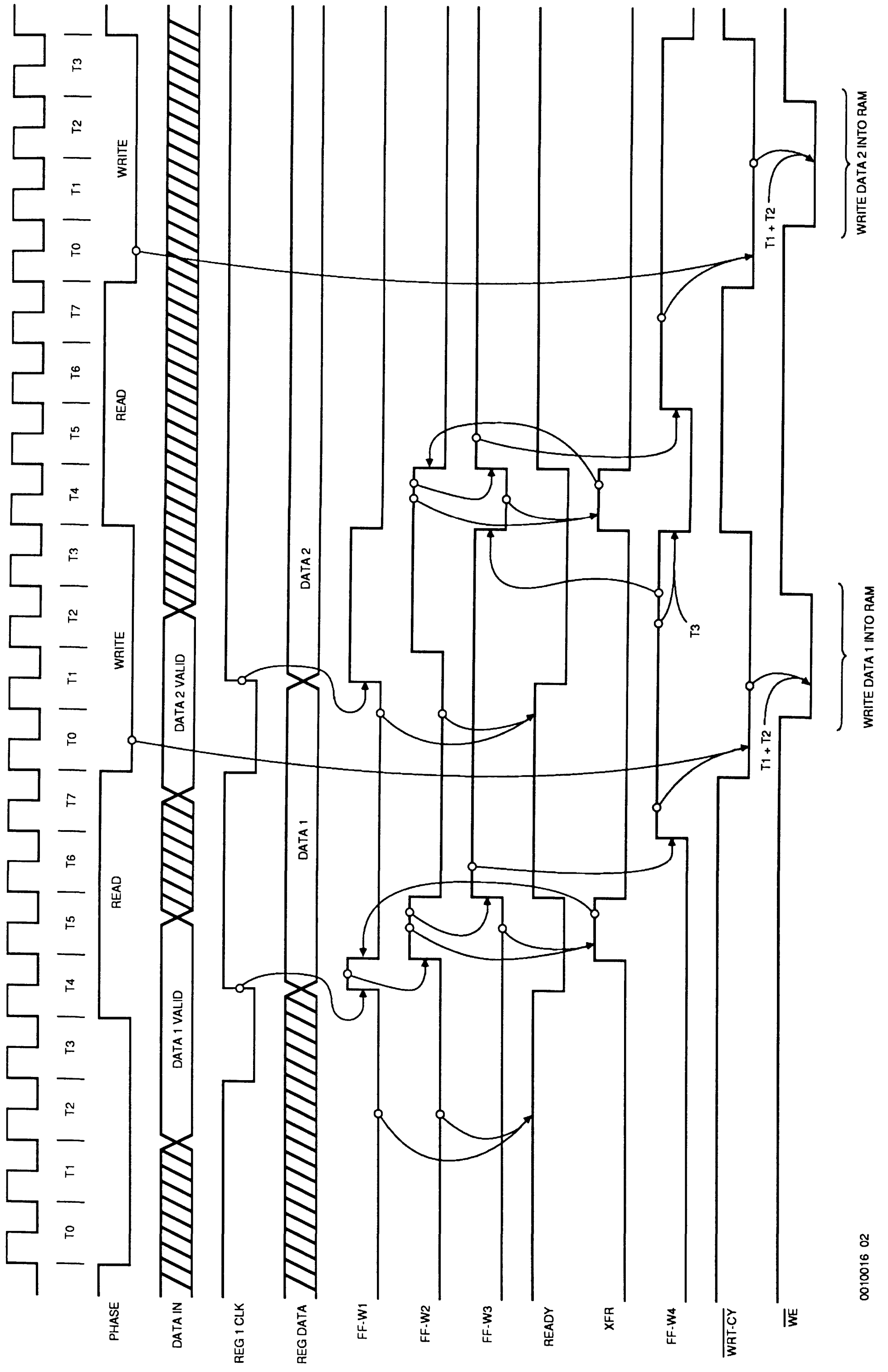


Figure 7. Buffer Write Timing

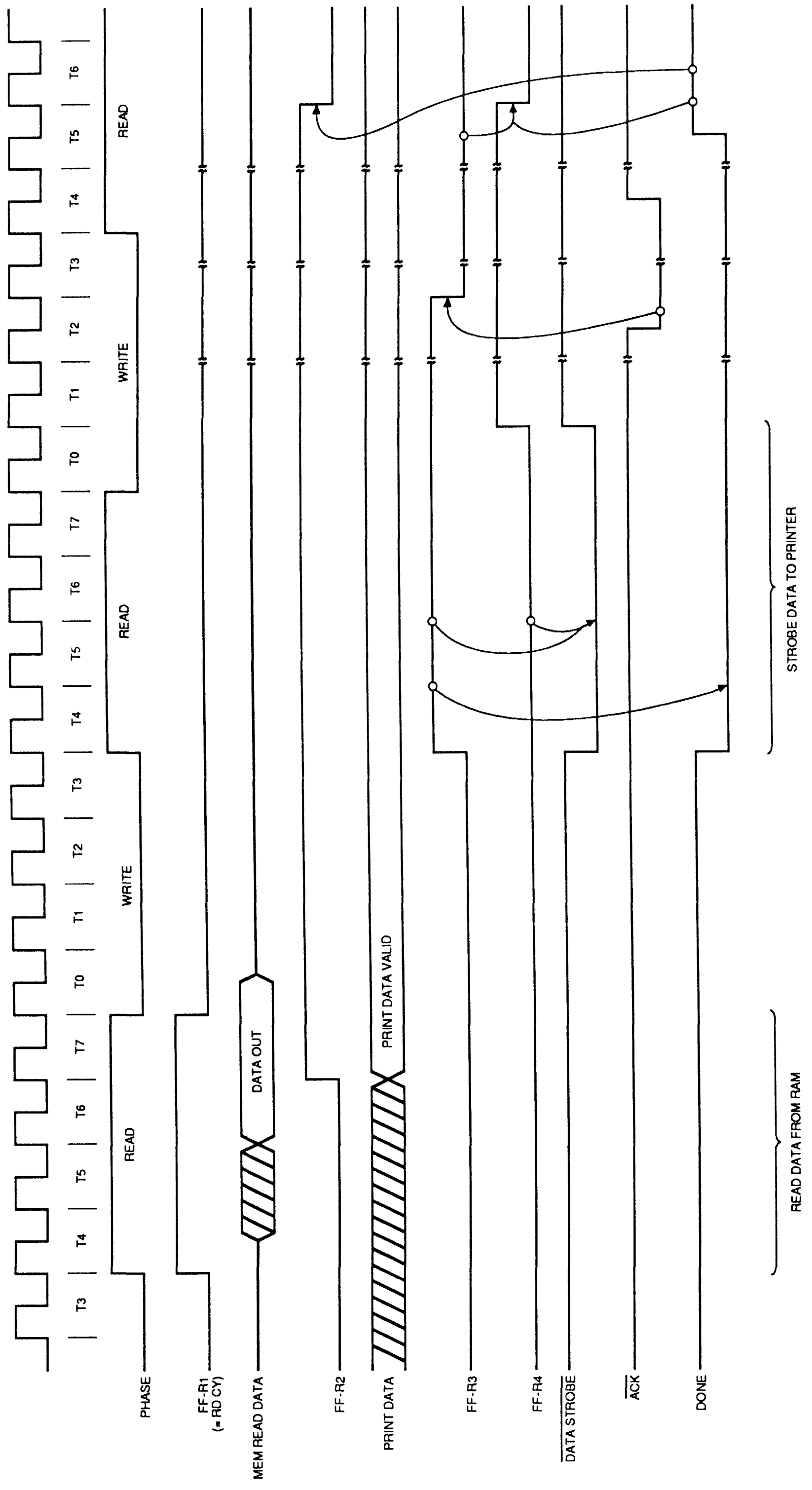


Figure 8. Buffer Read Timing

0010016 03

The RAM's Chip Select (CSM) is generated by ORing, the WRT_CYCLE and READ_CYCLE signals together. In this way, the data bus between the LCA and the RAM is enabled only during memory cycles in which data is being transferred; at all other times the RAM is disabled, resulting in lower power dissipation from the RAM.

ADDRESS COUNTERS

The read and store address pointers required for a circular queue are nothing more than counters. Although almost any type of counter would suffice for this application, careful selection of the right type of counter can lead to some significant savings in terms of CLB usage.

Although binary counters are the most familiar and the easiest to understand, synchronous versions are expensive to implement for larger modulos. Johnson counters, which count through only 2n states rather than 2^n, require an excessive amount of logic. The best counter in this case is a linear feedback shift register (LFSR) counter. The advantages in using LFSR counters in this application outweigh their main disadvantage—a nonconsecutive count sequence. They are very resource-efficient, since they consist merely of an n-bit shift register with bits from certain stages fed back to an exclusive-OR gate at the first stage. With proper selection of the feedback bits, these counters can be made to cycle through $(2^n)-1$ unique count states before repeating.

A potential drawback of the LFSR counter is the possibility of "hanging up" in the excluded count state (usually the all 1s or the all 0s state). Although this state cannot be entered in normal operation, it may possibly occur upon powerup. This situation can be prevented by guaranteeing that this excluded state is not one which occurs upon power-up. Through the use of the LCA's power-on reset which clears all the stages, and by inverting the sense of the feedback bit from the last counter stage, the count sequence commences with the all 0s state and never enters the all 1s state. Since all the LCA's storage elements are cleared during configuration, this condition is assured.

ADDRESS COMPARATORS

Two address comparisons are required to implement a circular queue: one to detect the EMPTY condition (R = S) and one to detect the FULL condition (S+1 = R). (The values of the Read and Store address counters are referred to here as R and S, respectively. See the description of circular queues in the side box.) Whereas the comparison of the two address counter values for equality is straightforward, the comparison of two LFSR counter values for the FULL condition (S+1 = R) appears difficult because of the LFSR's nonconsecutive count sequence. However, it should be clear that the S+1 value of the store pointer is simply the next state value (S prime) of S and is available as the D input of each flip-flop stage of S. This next-state value of each stage is readily obtainable with LFSR counters and makes implementing the address comparators simply a matter of bit-wise exclusive NORing together the appropriate bits of the two address counters.

Although address bus comparators are normally implemented as parallel exclusive-NOR structures, the architecture of the CLB with its dual outputs suggests an alternate, more efficient implementation. A serial implementation in which the result of each one-bit compare was ANDed with the results of the previous stage and passed along to the next stage would also serve the same function. Although this serial implementation is slower then a parallel one (yet still fast enough for our requirements), it has a major advantage over the parallel approach: such a circuit could be created from the same string of CLBs used to implement the two counters. Since each stage of the LFSRs consumes only the CLB's flip-flop and does not make use of the rest of the CLB's available combinational logic resourc-es, the serial address comparators could be implement-ed in those same CLBs. The combined implementation of both 16-bit counters as well as the two 16-bit address comparators is shown in Figure 9.

Note that with the 32 CLBs required to implement the two 16-bit counters, inclusion of the two 16-bit address comparators requires only one additional CLB, for a total of 33 CLBs.

TRADEOFFS IN MULTIPLEXING THE ADDRESS OUTPUTS

The last portion of the print buffer controller to be designed is the address output multiplexing logic. After implementing all the other sections of the entire print buffer controller circuit, only 6 CLBs and 26 IOBs remain to implement the address multiplexing logic. If the address multiplexing were accomplished entirely with 3-state outputs, then 32 IOBs would be required, six more than the 26 available. The six unused CLBs could be used as six one-bit wide multiplexers, but this number is not sufficient either. However, this would reduce the IOB requirement by six, from 32 IOBs to 26, which is precisely the number available. With a combination of the two multiplexing techniques, it is possible to multiplex two 16-bit address buses and access a full 64K byte print buffer memory. Every CLB and IOB is utilized in this application.

The XC2064's RESET pin acts as a master reset control for all flip-flops and latches during the LCA's user-operating mode. Consequently, it can also be used to

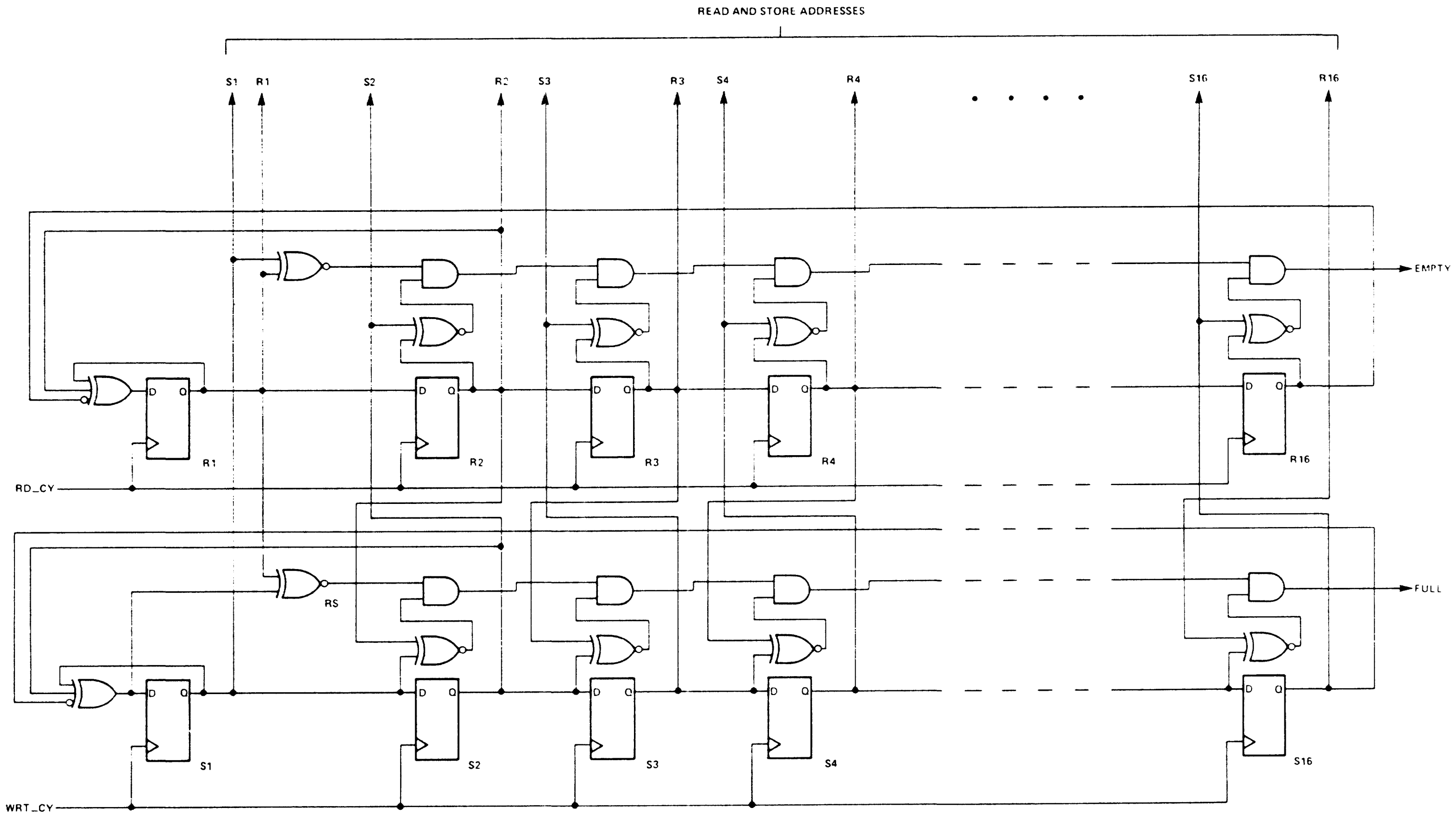


Figure 9. Address Counters and Comparator Logic

reset the overall operation of the controller, and it can be thought of as a 59th I/O pin in this application.

DETERMINING THE CLB AND IOB PLACEMENTS

One of the important aspects of designing with LCAs is the placement of the IOBs and CLBs within the LCA. Defining which I/O pins serve which function is a matter of great interest to the system designer, since it directly affects printed circuit board layout and routing and, possibly, board space. Less obvious but just as important to the designer, however, is the judicious placement of CLBs within the LCA. Proper placement of CLBs is just as important as good logic design, since it directly affects how much logic can be packed into a single LCA device. This, in turn, affects the total number of ICs and ultimately the board size. Several IOB and CLB placements can be analyzed in order to determine the best solution for a particular application. The placement of CLBs and IOBs in this design example required several passes before selection of the final placement presented here. The factors affecting this particular layout are indicative of those affecting many designs. For example:

- Wherever possible, an attempt was made to make maximal use of direct interconnect options. This was done to free general-purpose routing resources for other nets.
- The two counters were positioned in vertical columns to use metal long lines for clock distribution and to minimize clock skew.
- There is a high degree of connectivity between each CLB stage of one address counter and the corresponding CLB stage of the other counter due to the two address comparisons taking place. As a result, the pairs of corresponding CLB stages of the two counters are positioned adjacent to each other to minimize interconnect path lengths. See Figure 5. Since the pattern of interconnect between CLB pairs is repeated many times in the design, this pattern was carefully analyzed and optimized wherever possible.
- The read and store address buses must be multiplexed to create a single physical address bus of Figure 10. Since only six CLBs were available for address multiplexing, only six pairs of address bits could be multiplexed in CLB logic. The remaining address bits are multiplexed at the I/O pins. Consequently, the CLBs comprising the address counters are positioned along the right side of the LCA with outputs on the top, bottom and right sides of the LCA. Bits of the innermost counter stages required longer path lengths to get to IOBs and, therefore, were chosen for logical multiplexing. This allowed routing one output signal to one IOB instead of routing two signals out to two IOBs. This resulted in considerably less routing congestion.
- Since IOBs on each side of the LCA share a common input flip-flop clock, the IOBs (Reg1) for the processor data bus and the IOBs for the RAM data bus (Reg 3) must be located on separate sides of the LCA. The processor data bus IOBs were placed on the left side of the LCA. Because of CLB placement decisions and the large number of IOBs required by the memory address bus, the data bus to the RAM buffer was split with four IOBs on the top and four IOBs on the bottom. The eight CLBs associated with Reg 2 in the data path were likewise split four along the top and four along the bottom. See Figure 5.

CONFIGURING THE LCA

The LCA configuration program is 12,040 bits or 1,505 bytes in length. This configuration program can be loaded either from a host CPU or automatically upon power-up from an external memory (e.g., an EPROM). The specific configuration mode employed by the LCA is determined by the strapping of two pins on the device, M0 and M1, to either Vcc or ground.

In designs in which it is possible to let the host CPU handle the configuring, the LCA can be configured from the system bus as though it were a peripheral device. This is accomplished through use of three Chip Select pins, a serial DIN (Data In) pin, and a WRT (Write Strobe) pin. In this mode, the configuration program is written into the LCA serially, one bit at a time. If the CPU were not available to handle configuration, then the self-programming (Master) mode could have been chosen.

In this application example, let us assume that configuration programming can be conveniently handled by the host CPU. In this case, the peripheral configuration mode is the most appropriate. With this mode, the configuration data can be stored anywhere in the system—possibly sharing space with program code in a “bootup” ROM or EPROM, or stored on a disk. This way, configuring the LCA can be made part of the bootup process. To use the peripheral mode, Mode Select pins M0 and M1 must be tied HIGH and LOW, respectively. The device pins used for configuration then are: DIN, DOUT, CCLK, three Chip Selects and the WRT strobe. Since all of these pins are used as address outputs for the RAM in the final design, activity on these lines during the configuration process will not present a problem. However, since 5 of these 7 pins are inputs which are driven from the CPU's system bus during configuration, they must be isolated from the system bus once configuration is completed in order to prevent signal contentions. This is easily accomplished with a single three-state buffer package.

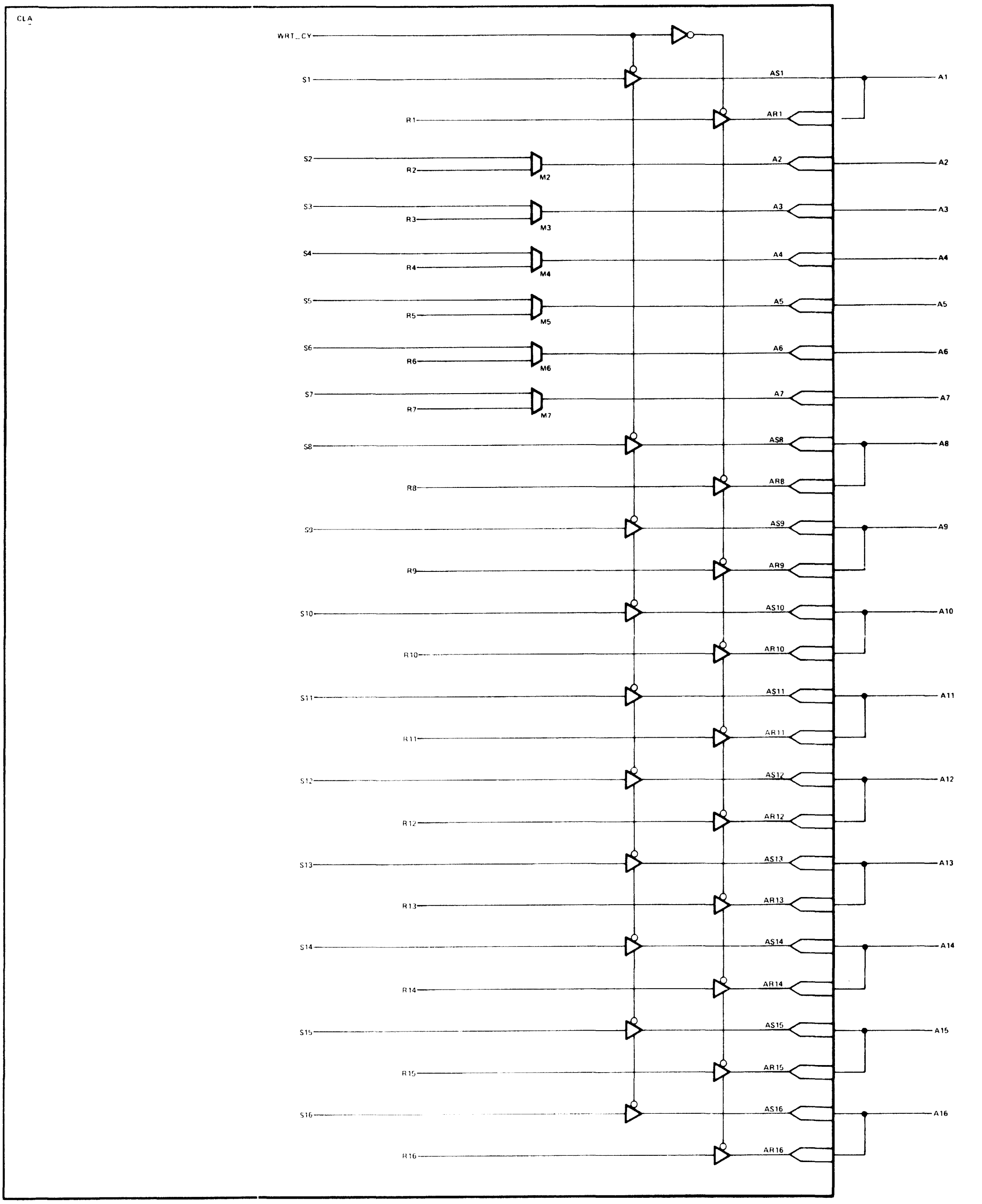


Figure 10. Read Address and Write Address Multiplexing

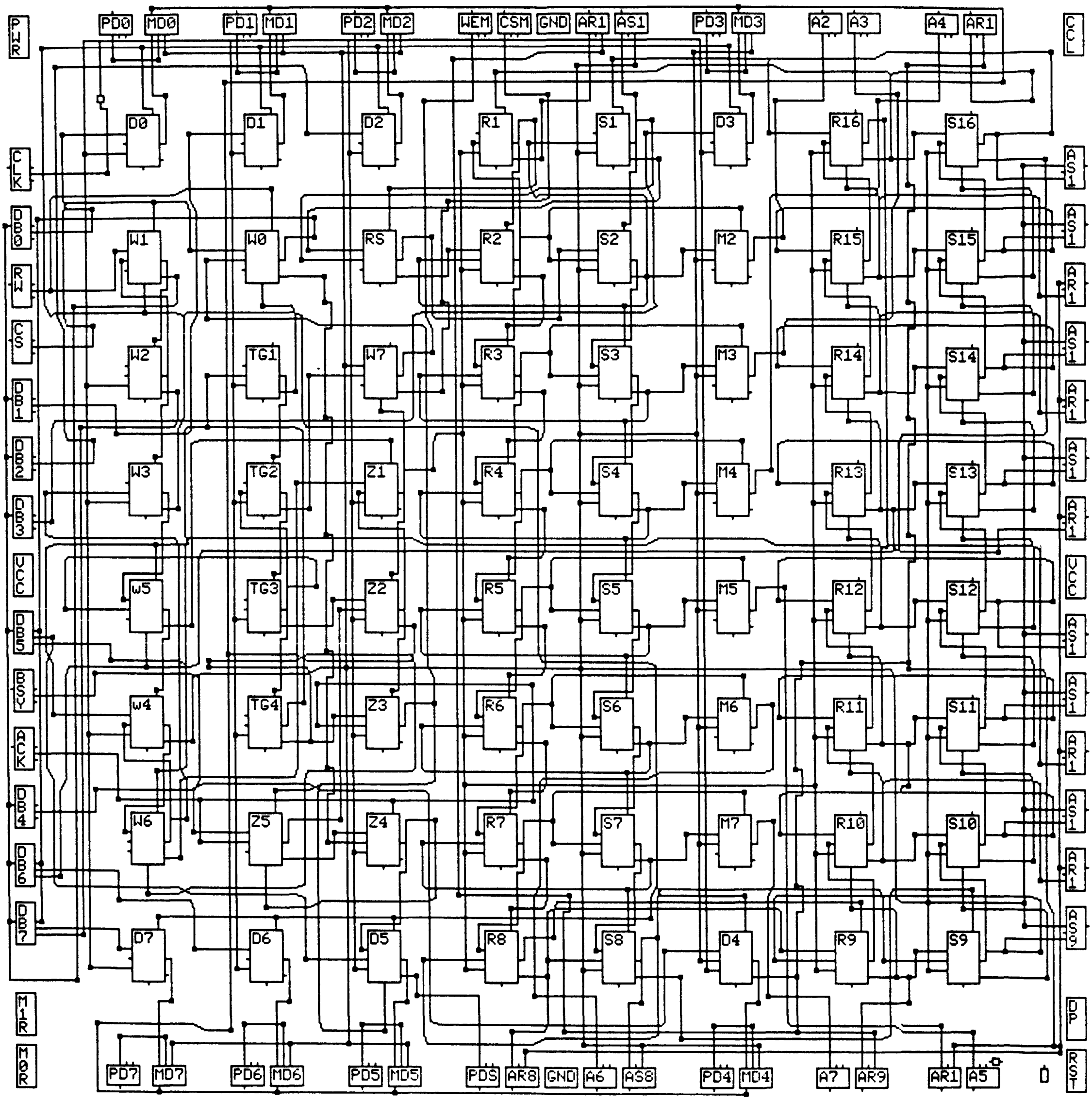


Figure 11. Placed and Routed Printer Buffer Controller LCA Design

DESIGN ALTERNATIVES

A number of design alternatives are possible with the printer buffer controller example we have chosen. Modifications could have been made in a number of different areas, including the following:

- Using a smaller address space (i.e., smaller counters and comparators)
- Eliminating the input FIFO stage (Reg 2)
- Implementing all the address multiplexing with CLBs rather than IOBs
- Using a Johnson counter with more states for finer timing resolution
- Using the on-chip oscillator rather than an external clock
- Using a processor bus interface with separate read and write strobes
- Supporting other printer signals (e.g., Out-of-paper, Off-line, Fault)
- Supporting a different style printer interface altogether
- Supporting an interrupt output to the host when the queue goes empty

COMPARISION WITH ALTERNATIVE TECHNOLOGIES

One method of evaluating the efficiency of the LCA in a given application is to consider the same logic design implemented with standard product SSI/MSI logic devices. The summary of SSI/MSI components in Table 4 represents an equivalent 74LSxx logic family implementation of the print buffer controller design. Implementations in 4000 series CMOS or 74HCxx family CMOS would likely result in even higher package counts, since those logic families lack the breadth of the 74LSxx product family.

Another useful way of gauging the effectiveness of the LCA as a vehicle for implementing logic circuits is to consider the number of equivalent NAND gates required to implement the same design in a gate array. The usual procedure is to compute the number of equivalent 2-input NAND gates required to implement the same logic function. This type of calculation places a numerical measure on the amount of logic required to implement a specific circuit. An analysis of the printer buffer controller design, based on the macrocell library of a major gate array vendor, shows that the amount of logic necessary for a gate array implementation of this same circuit would be approximately 838 equivalent gates.

Typically, gate array vendors reserve a margin of safety to guarantee routability. As a result, only about 80% of the total gates on a device are available for use by customer's logic. This number can vary slightly from design to design and from vendor to vendor, but usually it is near 80%. If we apply this factor to the actual computed gate count of 838, we see that this application design should require:

$$\frac{\text{gate count}}{0.8} = \frac{838}{0.8} = 1047 \text{ gates.}$$

Accordingly, the printer buffer controller would require a gate array of at least 1,000 gates.

The cost advantages of a gate array solution are often outweighed by the high costs and lengthy time delays required to produce a working prototype. The sequence of events which a designer must go through before working silicon can be delivered typically can take months, with the risk of future delays if errors are encountered. For many smaller gate array applications (such as this printer buffer controller), the LCA combines high density with the advantages of user programmability.

Table 4: An Equivalent Implementation Using SSI/MSI (74LSxx)

Component Description	Number of Elements	Device Number	Pkgs Req'd
Counters and comparators section:			
octal shift reg. w/clear and 3-state outputs	-	74LS299	4
quad exclusive NOR (oc)	32	74LS266	8
quad exclusive OR	4	74LS86	1
hex inverters	3	74LS04	1
pull-up resistors	2	resistors	-
Data Path section:			
octal DFF register w/clear	24	74LS273	3
quad 3-state buffers	3	74LS125A	1
octal 3-state buffers	8	74LS244	1
Timing Generation section:			
quad DFF w/clear	4	74LS175	1
quad 2-input ANDs	5	74LS08	1
Control logic section:			
dual DFF w/ clear	8	74LS74	4
quad 2-input ANDs	9	74LS08	3
quad 2-input NANDs	6	74LS00	2
quad 2-input ORs	3	74LS32	1
quad 2-input NORs	12	74LS02	3
triple 3-input NORs	2	74LS27	1
hex inverters	6	74LS04	1
Total Packages Required:			36

SUMMARY

This application note illustrates how an LCA could be designed into a printer buffer controller. Some of the many aspects of designing with LCAs have been presented and discussed. These same issues are applicable to a wide range of LCA applications. Some useful techniques for optimizing the efficiency of the LCA's logic and interconnect resources have also been described.

A Seven-Segment Display Driver

INTRODUCTION

The Xilinx XC2064 and XC2018 Logic Cell™ Arrays (LCA) easily accommodate a wide variety of different logic structures. This application note describes the design and entry of two such structures: A binary-to-seven-segment display driver and a BCD- (binary coded decimal) to-seven-segment display driver. Although a designer would most likely use an off-the-shelf LED display driver, this application note describes the method of entering a display driver design (more as an example than an actual application). With the Xilinx XACT™ Development System, building the second driver design (BCD) is a simple matter of editing the first (binary).

This application note also describes the Karnaugh map logic entry method available in the XACT Development System. For this design, the Karnaugh map entry saves time by simplifying the logic entry. The engineer can enter his logic directly through the Karnaugh map instead of deriving lengthy equations.

Since the XACT Development System supports macros, a designer may build a logic design from higher level functions. Besides the extensive Xilinx macro library, a designer can choose from custom user-created macro functions. This application note demonstrates how to create two LED driver macros to add to the designer's personal macro library. These LED drivers could then be integrated with other logic structures. For example, a designer might decide to include the drivers with a custom front-panel controller or a custom clock-timer display to minimize overall system chip count.

DESCRIPTION OF THE TWO DESIGNS

Binary-to-Seven-Segment Display Driver.

The first design, as graphically described in Figure 1, consists of logic to decode a four-bit binary input with binary values ranging from 0000B to 1111B (0 to 15 decimal). The decoded values drive the seven segments of the LED display to display the corresponding hexadecimal character (0 through F). The "character set" for the binary-to-seven-segment display is shown in Figure 2. Alternate characters representations are acceptable—especially for the "9", "b" and "d" symbols.

The LED segment drivers have registered outputs driven by the Strobe clock input. The logic drives common cathode LED displays. In other words, a logic "1" lights a particular segment. Conversely, in common anode displays, a logic "0" drives a segment.

BCD-to-Seven-Segment Display Driver

The binary-to-seven-segment display driver created in the first design can be easily edited with the XACT Development System to create a BCD-to-seven-segment display driver. The only modification required involves displaying an error character (in this case, an "e") any time the four-bit value exceeds 1001B (9 decimal). The complete "character set" for the BCD display driver is shown in Figure 3.

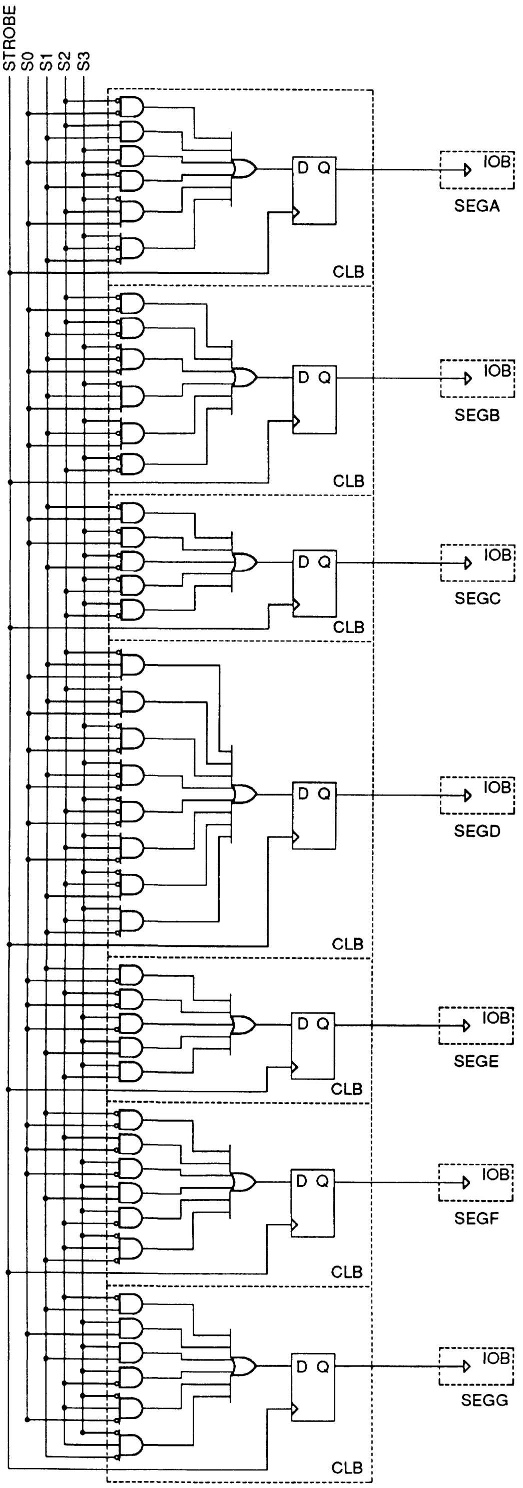
KARNAUGH MAPS WITHIN THE XACT DEVELOPMENT SYSTEM

For ease-of-use, the XACT Development System supports two forms of logic entry while editing a Configurable Logic Block (CLB). The first is standard Boolean equation entry. The second involves editing values within a Karnaugh map. The two entry methods are tightly coupled, and changes entered under one method are reflected in the other. For example an equation entered in standard Boolean form will be processed by the development system so that the equation also appears in its Karnaugh map representation.

The XACT Development System supports Karnaugh maps with one to four input variables. Editing entries within the Karnaugh map involves placing the mouse cursor in the appropriate entry box and then toggling that box making the entry true or false. Both the truth table and the Boolean equation line are updated to reflect any change entered through the Karnaugh map.

Karnaugh Map Background

Karnaugh maps were originally developed by M. Karnaugh in the early 1950s. Karnaugh maps aid in analyzing and minimizing Boolean logic expressions.



However because of the unique logic architecture of a Logic Cell Array, there is no need to minimize logic within a CLB. Logic within an LCA can be directly implemented as a Karnaugh map. Therfore logic minimization using Karnaugh maps will not be discussed here.

For LCA designs, Karnaugh maps provide a quick and concise shorthand notation for defining combinatorial logic functions (although Boolean expressions are possible). This application note describes how to use the Karnaugh entry method to enter the design for the LED driver quickly. Using the Karnaugh map entry method saves a long and tedious step—converting the Karnaugh map into a reduced Boolean expression.

Description of Karnaugh Maps

Within the XACT Development System, Karnaugh maps graphically describe the combinatorial logic inside a CLB. They may have from one to four input variables. The maps consist of a matrix of cells each corresponding to a single minterm of the four possible input variables.

As a simple example, Figure 4 shows a two-input Karnaugh map along with its truth table. The bars on the edges of the Karnaugh map indicate the row or column

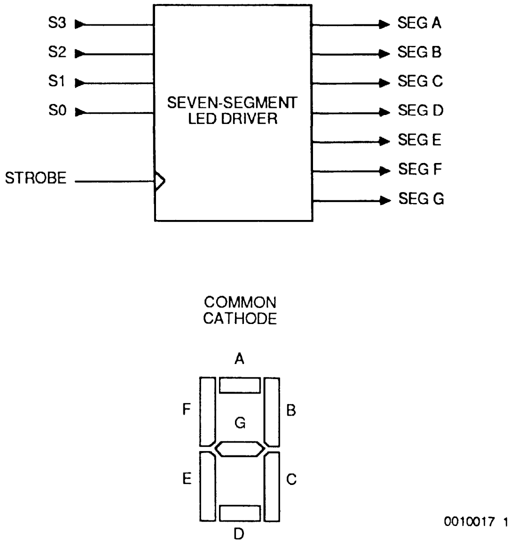


Figure 1. Basic Seven-Segment LED Driver

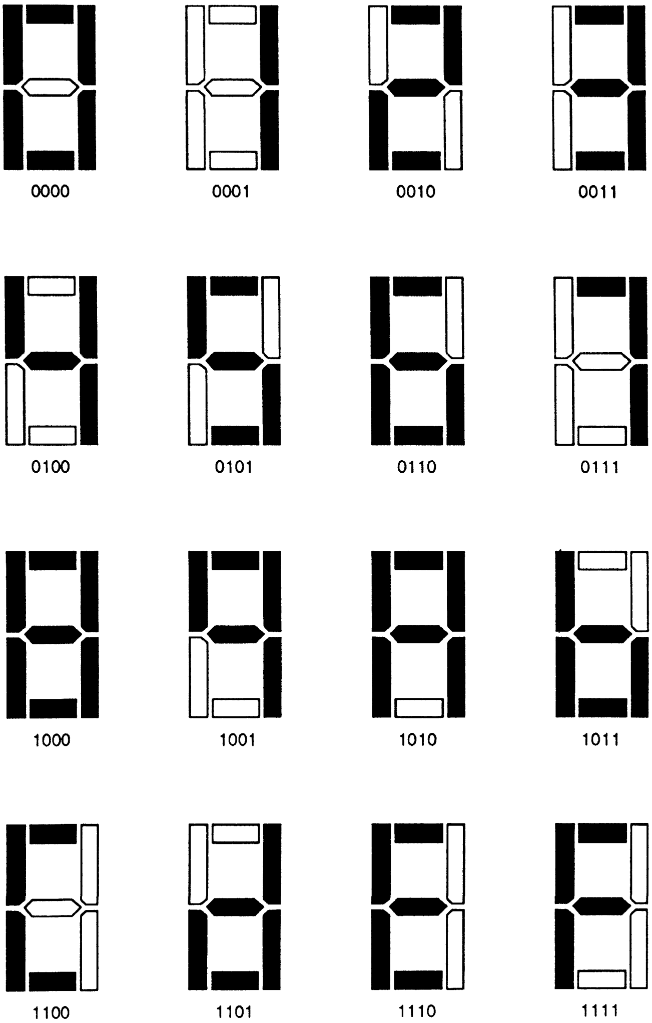
for which the specified input variable is true (logic "1"). Notice how the values from the truth table correspond to entries within the Karnaugh map. The first line of the truth table (A=0 and B=0) maps into the upper left-hand entry of the Karnaugh map, where again A=0 and B=0. The user can "direct map" this value into the Karnaugh map, or he can use the decimal value of the inputs (zero for A=0, B=0) and "position map" the value into position zero of the Karnaugh map.

To enter the simple equation in Figure 4, the user either enters the Boolean equation **"A@B"** or toggles the upper right-hand and lower left-hand entries in the Karnaugh map to the ON state using the mouse.

This technique becomes even more powerful with each additional input variable. For example Figure 5 shows this same technique applied to a four-input Karnaugh map. For some functions, the Karnaugh map represents a shorthand logic notation over standard Boolean equations.

BINARY-TO-SEVEN-SEGMENT DISPLAY DRIVER

Table 1 lists all of the possible inputs for the seven-segment display driver, including which of the seven segments is driven by each stimulus. An asterisk indicates that the segment output is driven for the specified combination of inputs S3 through S0 .



0010017 2

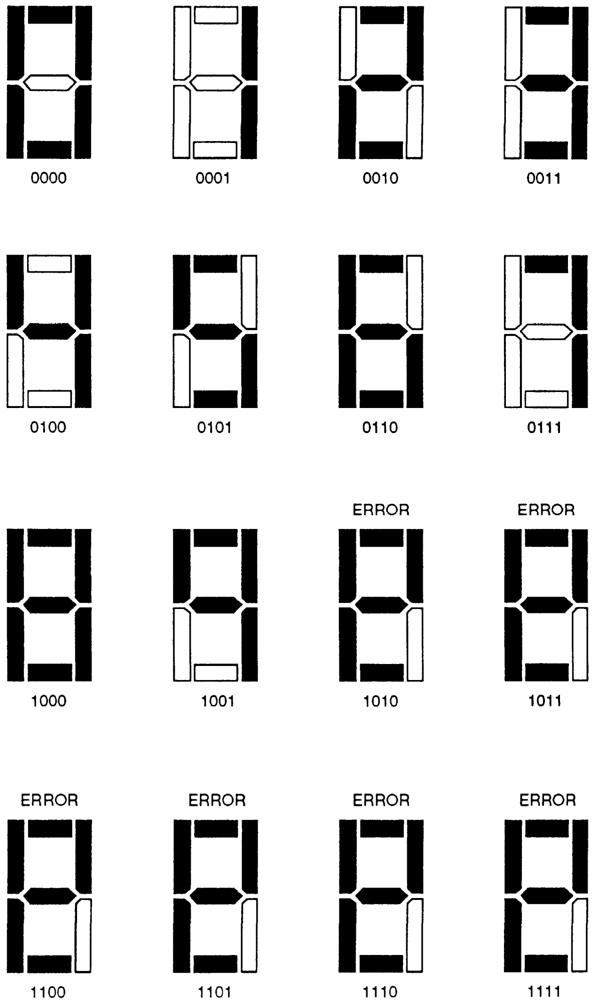
Figure 2. Binary-to-Seven-Segment Display Character Set

Obviously, converting this table into a series of Boolean logic equations would be tedious. A much simpler method involves entering this information directly into the CLBs using the XACT Editor's Karnaugh map entry facility.

Before delving too deeply into the design entry method, some conventions must first be established. For example if the XACT command sequence **Screen(Show(World))** appears, then select the **Screen** command using the mouse, then the **Show** command, and lastly the **World** command. A **Done** command is implied after each sequence unless the development system prompts for more inputs. From the keyboard, the user merely types **Show World** (the first command, **Screen**, does not need to be entered if typing the command from the keyboard) to accomplish the same goal.

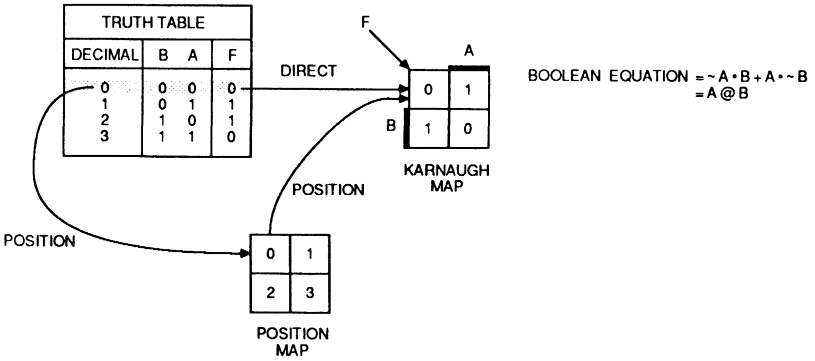
Keyboard command abbreviations are often indicated by uppercase highlighted characters in the menus. Abreviations are documented in the Editor section of the *XACT LCA Development System Manual.*

To generate a blank Karnaugh map within a Configurable Logic Block (CLB), first select **Blk(EditBlk)** with the mouse or simply type **EB** with the keyboard. Then select the CLB in position AH (upper right-hand corner) to work on. This CLB will drive the Segment A (SEGA) signal line of the LED driver.



0010017 2

Figure 3. BCD-to-Seven-Segment Display Character Set



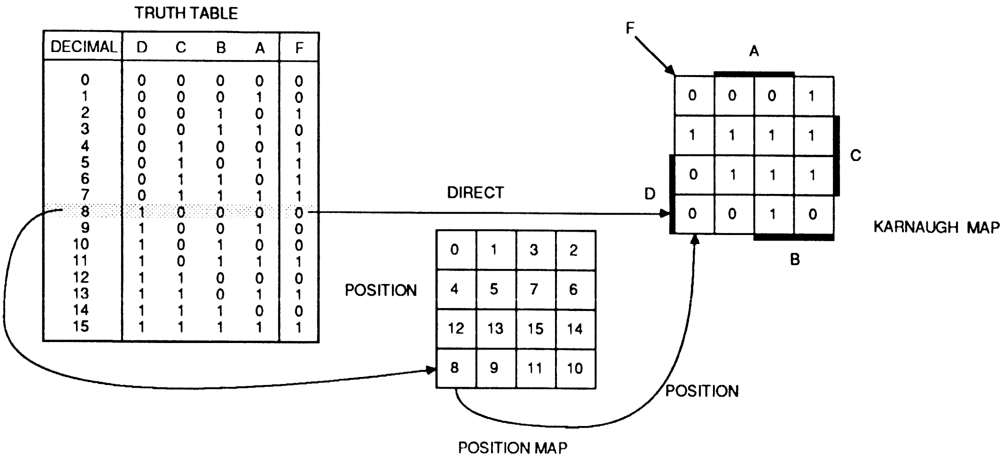
DECIMAL	B	A	F
0	0	0	0
1	0	1	1
2	1	0	1
3	1	1	0

	A
0	1
1	0

0	1
2	3

0010017 4

Figure 4. Two-Variable Karnaugh Map



DECIMAL	D	C	B	A	F
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

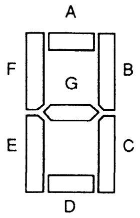
0	0	0	1
1	1	1	1
0	1	1	1
0	0	1	0

0	1	3	2
4	5	7	6
12	13	15	14
8	9	11	10

0010017 5

Figure 5. Four-Variable Karnaugh Map

DECIMAL	S3	S2	S1	S0	A	B	C	D	E	F	G
0	0	0	0	0	*	*	*	*	*	*	
1	0	0	0	1		*	*				
2	0	0	1	0	*	*		*	*		*
3	0	0	1	1	*	*	*	*			*
4	0	1	0	0		*	*			*	*
5	0	1	0	1	*		*	*		*	*
6	0	1	1	0	*		*	*	*	*	*
7	0	1	1	1	*	*	*				
8	1	0	0	0	*	*	*	*	*	*	*
9	1	0	0	1	*	*	*			*	*
10	1	0	1	0	*	*	*		*	*	*
11	1	0	1	1			*	*	*	*	*
12	1	1	0	0	*			*	*	*	
13	1	1	0	1		*	*	*	*		*
14	1	1	1	0	*			*	*	*	*
15	1	1	1	1	*				*	*	*



0010017 13

Table 1. Input Table for Binary-To-Seven-Segment Display

Since each segment of the LED driver requires a function of four input variables, change the base configuration of the CLB from its default configuration (two functions, each using three of the five available input variables). Select **Config(Base(F))** to do this. Now the CLB is configured to be a single function using four out of the five possible variables.

To create a blank Karnaugh map, select **Config(Order(F(A(B(C(D)))))** (remember to add the implied **Done**). The display should appear like Figure 6 with a blank Karnaugh map appearing in the lower left hand corner.

Configure the SEGA CLB by selecting "Q" to drive the X output. In addition, use the "K" clock input to clock the storage element.

To save effort entering this design, copy the configuration in the SEGA CLB to six other CLBs (one each for the remaining six LED segments). First, select **Screen(Switch)** to switch to the physical interconnect editor. Then, select **Blk(Copyblk)** and click on block AH (which is the SEGA block). Copy this block to CLB blocks BH through GH (all along the right-hand edge of the die). Remember to select **Done** when complete. Again select **Screen(Switch)** to continue editing the SEGA CLB.

Using the information in Table 1, the user can enter the design directly into the Karnaugh map. For clarity and an as example, entering the data for Segment A (SEGA) of the seven-segment display is described.

Table 2 contains the binary values of the input variables A, B, C, and D as well as the corresponding logic output F for SEGA. All of the possible values ranging from 0000B to 1111B are numbered with their corresponding decimal values. In addition, Figure 7a shows where each of the binary inputs is located within the Karnaugh map using the decimal values of the 4-bit binary input. Figure 7b details how the logic for SEGA maps into the Karnaugh map.

Using the mouse, place the mouse cursor inside the upper left block of the Karnaugh map and toggle that block on (bright yellow) with the mouse select button. The logic for SEGA (when all of the inputs are logic "0") has just been set. The truth table to the left of the Karnaugh map and the Boolean equation line at the bottom of the screen should both reflect the change made in the Karnaugh map.

Continue toggling on the indicated blocks for the remaining values, just as shown in Figure 7b. This is much simpler than typing the Boolean equation for SEGA:

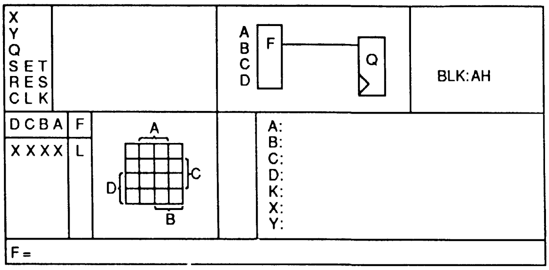
SEGA = ~C*~A+C*B+D*~A+~D*B+~D*C*A*~C*~B

Repeat this same process for the six remaining CLBs. The Karnaugh maps should appear like those shown in Figure 8.

Now that each of the segment drivers has been configured, name each CLB with its corresponding segment identity. For Segment A, type

NAMEB HA SEGA (Enter).

The name SEGA should appear in block AH. To save the designer time and effort, the development system keeps a running stack of the latest command line entries. This way, a user can pull up the last few commands and edit them to enter similar commands. For example to name Segment B, merely press the up cursor button. The line previously entered will appear on the command line. Edit the command line using the



0010017 6

Figure 6. Blank Karnaugh Map in CLB Display

DECIMAL	D	C	B	A	F
0	0	0	0	0	*
1	0	0	0	1	
2	0	0	1	0	*
3	0	0	1	1	*
4	0	1	0	0	
5	0	1	0	1	*
6	0	1	1	0	*
7	0	1	1	1	*
8	1	0	0	0	*
9	1	0	0	1	*
10	1	0	1	0	*
11	1	0	1	1	
12	1	1	0	0	*
13	1	1	0	1	
14	1	1	1	0	*
15	1	1	1	1	*

0010017 14

Table 2. Truth Table Specifically for Segment A

left and right cursor keys and the insert (Ins) and delete (Del) keys to read

NAMEB HB SEGB.

Now press the ENTER key. Continue naming the remaining blocks in the same manner.

Add the required networks using the Addnet command. Connect a network called **S0** to the A-input of each CLB in the design, one called **S1** to the B-input, one called **S2** to the C-input, and one called **S3** to the D-input. Attach all of the K clock lines together on a net called **Strobe**.

For the X-output from each CLB, add an output network using a name specific for each segment output. For example, attach a network called **A_Out** to the X-output from the SEGA CLB.

VALIDATING THE DESIGN

Before a section of logic is saved as a macro function, its function should be thoroughly verified. Without verification, logic bugs could potentially be passed from one design to another (a veritable digital design influenza). Using the macro library, a designer builds his logic in debugged modules.

Modular logic design accomplishes many of the same goals as modular software design. Modularity helps speed development by isolating errors to specific modules, making the system easier to debug. By using validated modules throughout his design, the designer reduces the time and frustration of the hardware debug cycle. Obviously validating macros is important.

The minimum validation that should be applied to a macro function is the Design Rule Checker (DRC). To invoke the DRC for the LED driver design just entered, select **Misc(DRC)** from the LCA Editor. Warnings about missing sources or loads may be ignored if the specified networks are known not to have them. For example, in this design, the DRC indicates that networks S0, S1, S2, S3, and Strobe have no sources while networks A_Out through G_Out have no loads. Warnings about PIPs (programmable interconnect points) may be ignored if some of the networks have not been routed (as is the case with this design example).

A much better way to validate the design involves either the simulator (P-SILOS™), the in-circuit emulator (XACTOR™), or the download cable. To validate the design with the simulator, run the simulation generator program (SimGen in the Executive Program menu). Then, using any standard text editor, edit the data file (<filename>.DAT) to include valid stimuli to test the design. Figure 9 shows the stimuli used to verify the LED driver. Running a simulation based on these stimuli produces the graphical output shown in Figure 10.

Emulation is even simpler. It involves downloading the design into a system and testing it there for functionality.

After verifying the design, enter the LCA Editor by Selecting **Editlca** from the main Executive menu. Now, to save the binary-to-seven-segment display drive in the macro library, select **Misc(Cutmacro)**. When the development system prompts for a file name, enter 7SEG_BLC for a seven-segment display driver (7SEG_) using binary (B) input with latched output (L) to a common cathode (C) display.

To save the desired portion of logic under this macro name, select the seven CLBs used in this design using the mouse. Select blocks AH through GH. Select **Done** when complete. The **Cutmacro** command automatically picks up the routing attached to the specified blocks (as well as any corresponding block and net names) and creates an ASCII text file containing all the macro information. A portion of the ASCII macro file called 7SEG_BLC.MAC appears in Figure 11. The **Cutmacro** command also appends the file extension .MAC to a macro file.

Once in ASCII form, the designer can use any standard text editor to edit the newly created macro file. For example, the designer may decide to change the default network or block names.

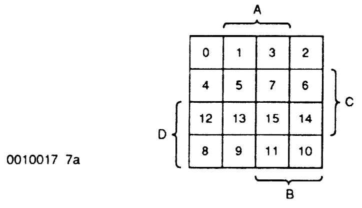


Figure 7a. Position Map for a Four-Variable Karnaugh Map

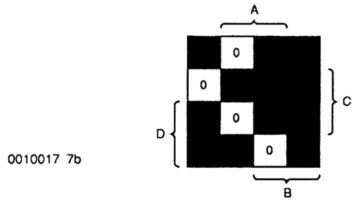


Figure 7b. Segment A Logic Mapped Into Karnaugh Map

The newly created macro file can now be used just like any standard macro library entry merely by selecting **Misc(Macro))** and choosing macro 7SEG_BLC.

BCD-TO-SEVEN-SEGMENT DISPLAY DRIVER

Table 3 indicates which of the seven-segments are lighted by each of the sixteen input combinations. Notice that combinations 10 through 15 are identical (the error symbol "e"). Modifying the binary display driver to make it a BCD display driver is quite simple. Positions 10 through 15 in each Karnaugh map for Segments A through G must be modified. To display the error symbol, all of the LED segments except for Segment C (SEGC) are lighted. The edited Karnaugh maps for the BCD driver appear in Figure 12.

Edit the Karnaugh maps for all of the LED segments (SEGA through SEGG) to reflect the changes and then perform a Design Rule Check to verify the design.

Once verified, save this new design as another macro, this time called **BCD_7LC** for BCD to seven-segment display (BCD_7) with latched outputs (L) for a common cathode display (C).

OTHER POSSIBLE MODIFICATIONS

Common-Anode Displays

The logic for both LED display drivers presented here specifies active-high outputs (positive logic). Positive logic drives common-cathode displays by sending a logic "1" to the appropriate segment. To drive common-anode displays, however, the logic sense must be inverted. In other words, the positive logic equations must be converted to negative logic equations.

Simply inverting every entry within each Karnaugh map for each CLB accomplishes this. However, in this case, editing the Boolean equation is much easier. While editing the correct CLB, select **Config(Editeq(F))** and use the left and right cursor keys and the insert (Ins) and delete (Del) keys to edit the equation. For example, to invert the equation for SEGA, change the equation from

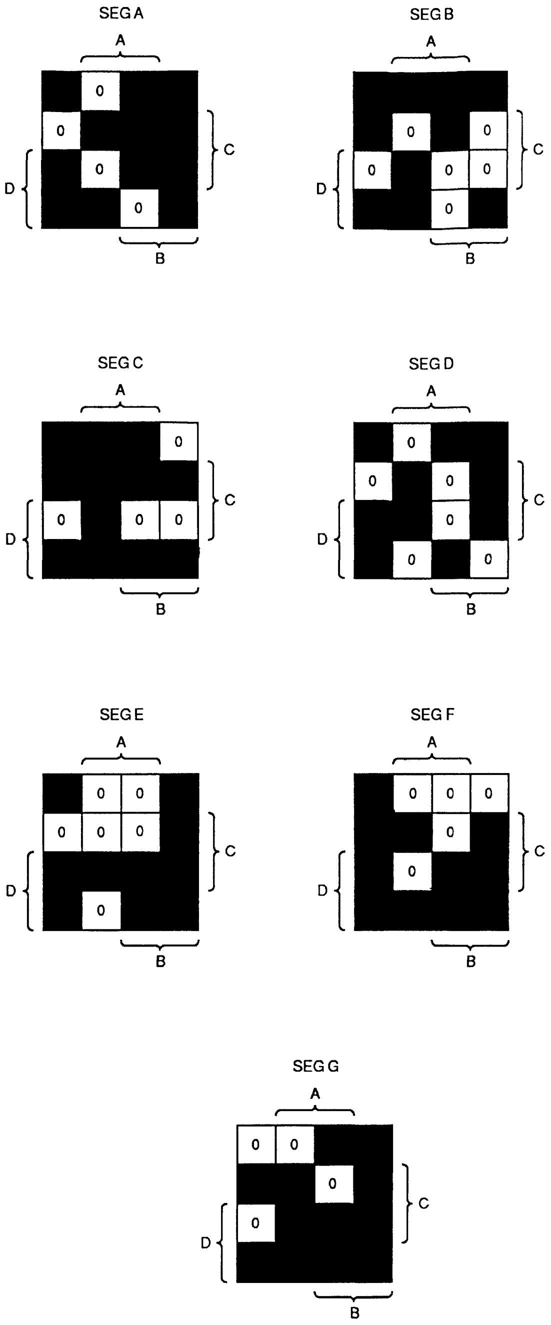
F = ~C*~A+C*B+D*~A+~D*B+~D*C*A+D*~C*~B

to

F = ~(~C*~A+C*B+D*~A+~D*B+~D*C*A+D*~C*~B)

by adding a "~(" to the front of the equation and a ")" to the back of the equation. Do this for each segment.

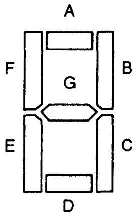
Thus, two new macros can be created—one for a binary-to-seven-segment display driver for common-anode



0010017 8

Figure 8. Karnaugh Map Seven-Segment Display (Binary Inputs)

DECIMAL	S3	S2	S1	S0	A	B	C	D	E	F	G
0	0	0	0	0	*	*	*	*	*	*	
1	0	0	0	1		*	*				
2	0	0	1	0	*	*		*	*		*
3	0	0	1	1	*	*	*	*			*
4	0	1	0	0		*	*			*	*
5	0	1	0	1	*		*	*		*	*
6	0	1	1	0	*		*	*	*	*	*
7	0	1	1	1	*	*	*				
8	1	0	0	0	*	*	*	*	*	*	*
9	1	0	0	1	*	*	*			*	*
10	1	0	1	0	*	*		*	*	*	*
11	1	0	1	1	*	*		*	*	*	*
12	1	1	0	0	*	*		*	*	*	*
13	1	1	0	1	*	*		*	*	*	*
14	1	1	1	0	*	*		*	*	*	*
15	1	1	1	1	*	*		*	*	*	*



0010017 15

Table 3. Input Table for BCD-To-Seven-Segment Display

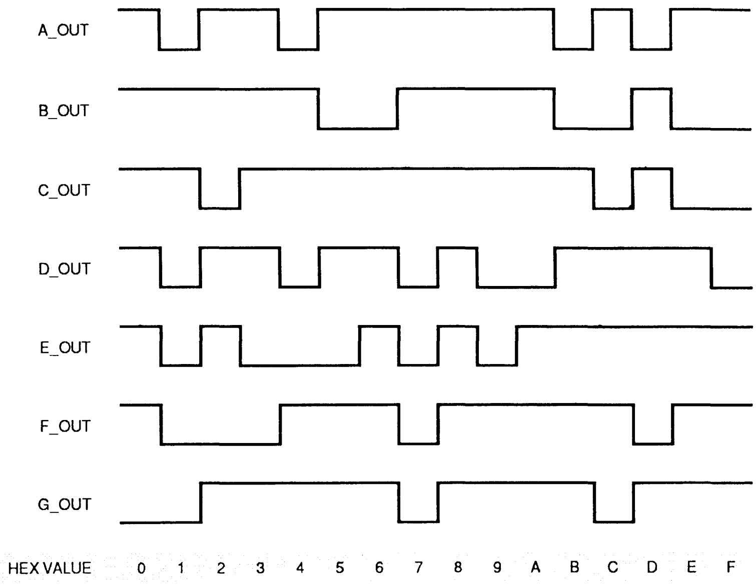
```
$
$ Simulation file for design 'KARANAUGH.LCA' type '2064c68-1'
$ Created by XACT Ver. 1.1 at 15:01:43 MAR 23, 1986
$
!INPUT KARANAUGH.sim

$ INPUTS:
Strobe       .CLK 0 S0 100 S1 200 S0 .REP 0
GLOBAL.RESET .CLK 0 S1 1 S0 $ Initial pulse to reset latches

.PATTERN S3 S2 S1 S0
0         0  0  0  0    $hex 0
500       0  0  0  1    $hex 1
1000      0  0  1  0    $hex 2
1500      0  0  1  1    $hex 3
2000      0  1  0  0    $hex 4
2500      0  1  0  1    $hex 5
3000      0  1  1  0    $hex 6
3500      0  1  1  1    $hex 7
4000      1  0  0  0    $hex 8
4500      1  0  0  1    $hex 9
5000      1  0  1  1    $hex A
5500      1  0  1  1    $hex B
6000      1  1  0  0    $hex C
6500      1  1  0  1    $hex D
7000      1  1  1  0    $hex E
7500      1  1  1  1    $hex F
.EOP

.MONITOR Strobe S0 S1 S2 S3 ; A_Out B_Out C_Out D_Out E_Out F_Out G_Out
.GRAPH Strobe S0 S1 S2 S3 ; A_Out B_Out C_Out D_Out E_Out F_Out G_Out
```

Figure 9. Simulator Stimulus File to Validate Binary-to-Seven-Segment Display Design



0010017 10

		A B C D E F G				A B C D E F G	
		O O O O O O O				O O O O O O O	
	S S S S	U U U U U U U			S S S S	U U U U U U U	
TIME	3 2 1 0	T T T T T T T		TIME	3 2 1 0	T T T T T T T	
0	0 0 0 0	1 1 1 1 1 1 0	0	4035	1 0 0 0	1 1 1 1 1 1 1	8
500	0 0 0 1	1 1 1 1 1 1 0		4500	1 0 0 0	1 1 1 1 1 1 1	
535	0 0 0 1	0 1 1 0 0 0 0	1	4535	1 0 0 1	1 1 1 0 0 1 1	9
1000	0 0 1 0	0 1 1 0 0 0 0		5000	1 0 0 1	1 1 1 0 0 1 1	
1035	0 0 1 0	1 1 0 1 1 0 1	2	5035	1 0 1 0	1 1 1 0 1 1 1	A
1500	0 0 1 1	1 1 0 1 1 0 1		5500	1 0 1 0	1 1 1 0 1 1 1	
1535	0 0 1 1	1 1 1 1 0 0 1	3	5535	1 0 1 1	0 0 1 1 1 1 1	C
2000	0 1 0 0	1 1 1 1 0 0 1		6000	1 0 1 1	0 0 1 1 1 1 1	
2035	0 1 0 0	0 1 1 0 0 1 1	4	6035	1 1 0 0	1 0 0 1 1 1 0	C
2500	0 1 0 1	0 1 1 0 0 1 1		6500	1 1 0 0	1 0 0 1 1 1 0	
2535	0 1 0 1	1 0 1 1 0 1 1	5	6535	1 1 0 1	0 1 1 1 1 0 1	D
3000	0 1 1 0	1 0 1 1 0 1 1		7000	1 1 0 1	0 1 1 1 1 0 1	
3035	0 1 1 0	1 0 1 1 1 1 1	6	7035	1 1 1 0	1 0 0 1 1 1 1	E
3500	0 1 1 1	1 0 1 1 1 1 1		7500	1 1 1 0	1 0 0 1 1 1 1	
3535	0 1 1 1	1 1 1 0 0 0 0	7	7535	1 1 1 1	1 0 0 0 1 1 1	F
4000	1 0 0 0	1 1 1 0 0 0 0					

0010017 10A

Figure 10. Timing Diagram Output from P-SILOS for Binary-to Seven-Segment Display Driver (Diagram is used to validate logic)

```
; Cutmacro: KARANAUGH.LCA, XACT 1.1, 15:47:57 MAR 23, 1986
Parameter NAME ? Enter instance name:
Parameter NET S3 Select S3 net
Parameter NET S2 Select S2 net:
Parameter NET S1 Select S1 net:
Parameter NET S0 Select S0 net:
Parameter NET Strobe Select Strobe block:
Parameter CLB ? Select SEGA block:
Parameter CLB ? Select SEGB block:
Parameter CLB ? Select SEGC block:
Parameter CLB ? Select SEGD block:
Parameter CLB ? Select SEGE block:
Parameter CLB ? Select SEGF block:
Parameter CLB ? Select SEGG block:
Editblk %7
C X:F Y: F:A:B:C:D Q:FF SET: RES: CLK:K
Eq F = ~C*~A+C*B+D*~A+~D*B+~D*C*A+D*~C*~B
Endblk
Editblk %8
•
•
•
Editblk %13
Base F
Config X:F Y: F:A:B:C:D Q:FF SET: RES: CLK:K
Equate F = ~C*B+D*A+D*B+D*~C+~D*C*~A+~D*C*~B
Endblk
Addpin %2 %7.A %8.A %9.A %10.A %11.A %12.A %13.A
Addpin %3 %13.B %12.B %11.B %10.B %9.B %8.B %7.B
Addpin %4 %7.C %8.C %9.C %10.C %11.C %12.C %13.C
Addpin %5 %13.D %12.D %11.D %10.D %9.D %8.D %7.D
Addpin %6 %7.K %8.K %9.K %10.K %11.K %12.K %13.K
```

Parameterized nets, default names and prompt messages

Parameterized blocks and prompt messages

Logic Definition

Parameterized net names and connections

Figure 11. Portion of Newly Created LED Driver Macro (7SEG_BLC)

displays and one to display BCD data on a common-anode display.

Direct Combinatorial Outputs

For both of the designs presented here, latched outputs drive each segment LED. For direct combinatorial outputs (i.e. nonlatched), edit each CLB as follows: For combinatorial output, the X output must originate from Function F and not from the storage element "Q". Select "F" in the select field for the X output instead of "Q".

SUMMARY

Karnaugh map entry saves time over standard Boolean equation entry especially for complex expressions of four variables. Two seven-segment display driver designs demonstrated the Karnaugh map entry technique.

The **Cutmacro** command allows a user to create and save user-defined logic macros. In this application, two seven-segment display drivers created using the XACT Development System were saved as macros.

Macro functions allow a designer to build modular logic designs. Modular design helps speed development by reducing errors. A designer should thoroughly validate a user-defined macro before saving it. Both the simulator (P-SILOS), the download cable, and the in-circuit emulator (XACTOR) are helpful in validating designs.

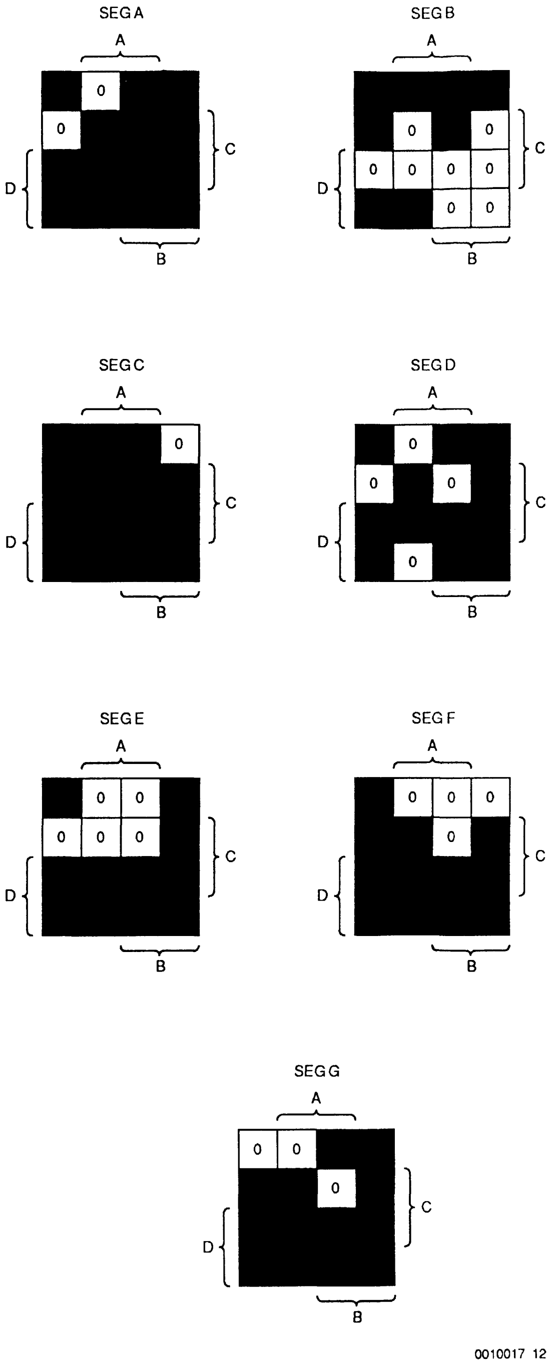


Figure 12. Karnaugh Maps for Seven-Segment Display (BCD Inputs)

Field Hardware/Software Updates Made Cost Effective

INTRODUCTION

With the high cost of travel and labor, many companies are looking for alternatives to the conventional field service person as the method of providing field updates for their products. A solution which is becoming more popular is the use of EEPROM (Electrically Erasable Programmable Read Only Memory) or Battery Backed-up RAM as a method of storing control software. Remote loading of this memory allows changes to be made without the need to physically travel to a customer's site. This solution only addresses the need to provide software updates, but does not address the need for additional hardware to perform the update, or the potential requirement to update the hardware itself. Addition of extra hardware to perform software updating requires circuit board space and other specialized functions which may outweigh the potential benefit from a remote update capability.

Logic Cell™ Arrays (LCAs) allow the hardware designer a degree of flexibility, similar to that of the software designer, for making rapid functional changes. With appropriate configurations, the Logic Cell Array may provide the additional hardware and control functions necessary to perform remote upgrading for both software and hardware. This logic may even be shared with the logic used for normal operation.

DESCRIPTION

Figure 1 shows a block diagram of a design for a remote EEPROM programmer The EEPROM could be loaded with either software modifications or hardware changes. If hardware changes are to be loaded into the EEPROM, a re-load of the Logic Cell Array would be performed after the EEPROM is programmed to insure that the Logic Cell Array has the correct configuration. The location of the address and data pins on the Logic Cell Array (see Figure 2) have been chosen to correspond to pins of the same function during configuration, insuring that proper loading can be performed without the need for any external circuitry.

The incoming serial data stream is converted to eight-bit bytes by a shift register, while a three bit counter tracks the number of bits received. When eight data bits have been received they are loaded into an eight bit data buffer and the address generator is incremented. The address generator, a 16 bit counter, is held in a reset state until the first data byte has been received and loaded into the data buffer. This insures that the first byte of data will be programmed into hexadecimal location 0000 of the EEPROM. One clock cycle after the data is loaded and the address counter incremented, the WE signal is asserted for one clock cycle. This allows an entire byte time for the write process of the EEPROM to be completed. (When writing to EEPROM, care should be taken in selecting a baud rate slow enough to allow ample time for the write cycle, approximately 10 ms. Typically 800 baud or less would guarantee adequate time for the write cycle.)

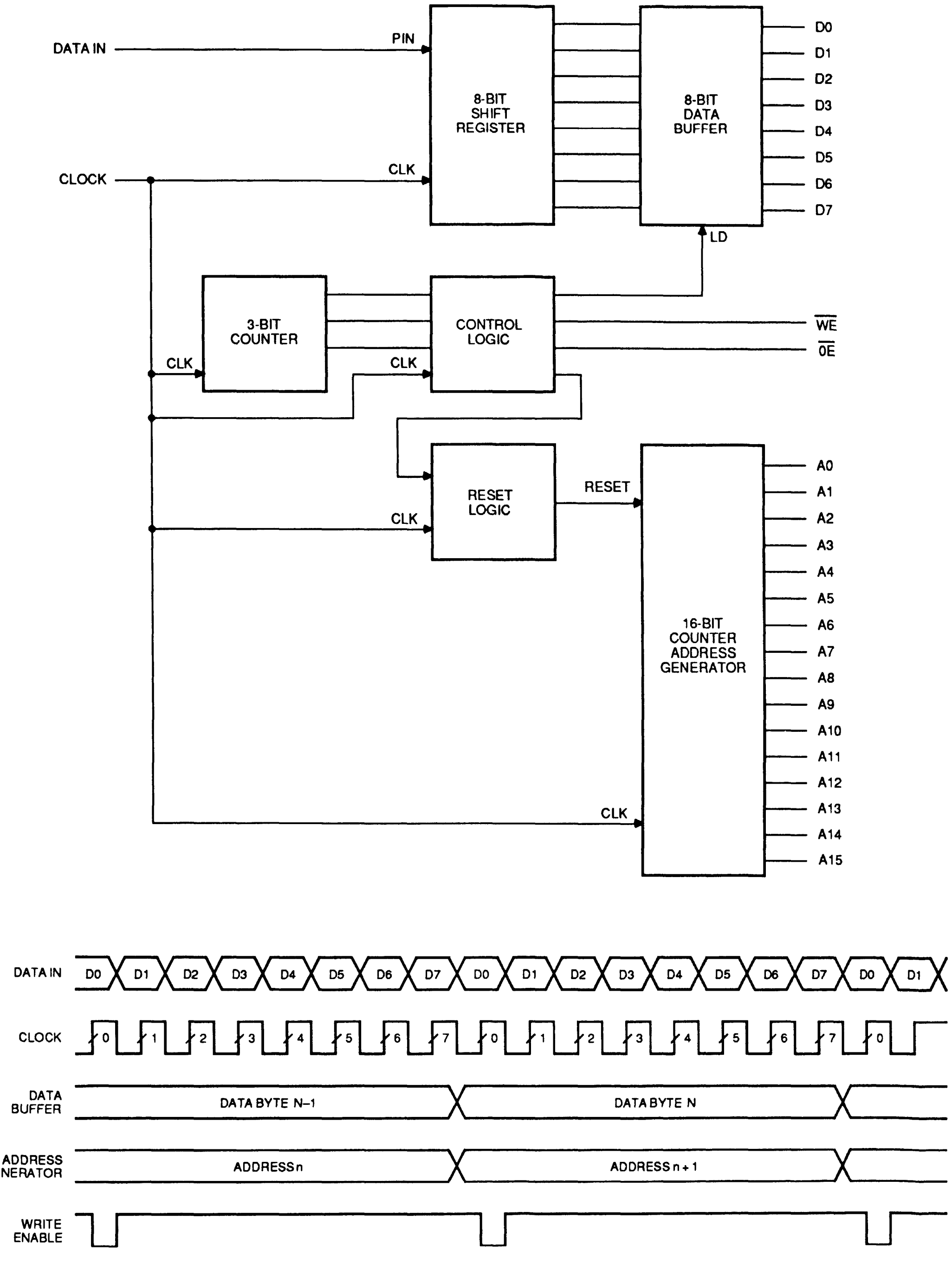
This circuit requires less than two-thirds of an XC2064 for implementation of the EEPROM programming function. The remaining logic could be used for additional features such as:

1. Parity and framing checking
2. Detecting start and stop bits
3. Logic to extract and load a starting address into the address generator
4. Additional address bit generation capability to extend the size of addressable memory beyond 64K bytes.
5. Logic to detect a command to initiate a re-configuration cycle of the Logic Cell Array
6. Logic to perform a readback of the contents of the EEPROM for verification

Additional information on the design of a complete UART device with the Logic Cell Array is available from Xilinx.

CONCLUSION

With the flexibility and capabilities of the Logic Cell Array, systems designers have new alternatives for remote operation, diagnostics, loading and control. The economic benefits of precluding the requirement for field service visits to remote locations can easily justify the incremental design effort required to add the capabilities to the remote system using Logic Cell Arrays.



0010020 1A&B

Figure 1. E^2 PROGRAMMER

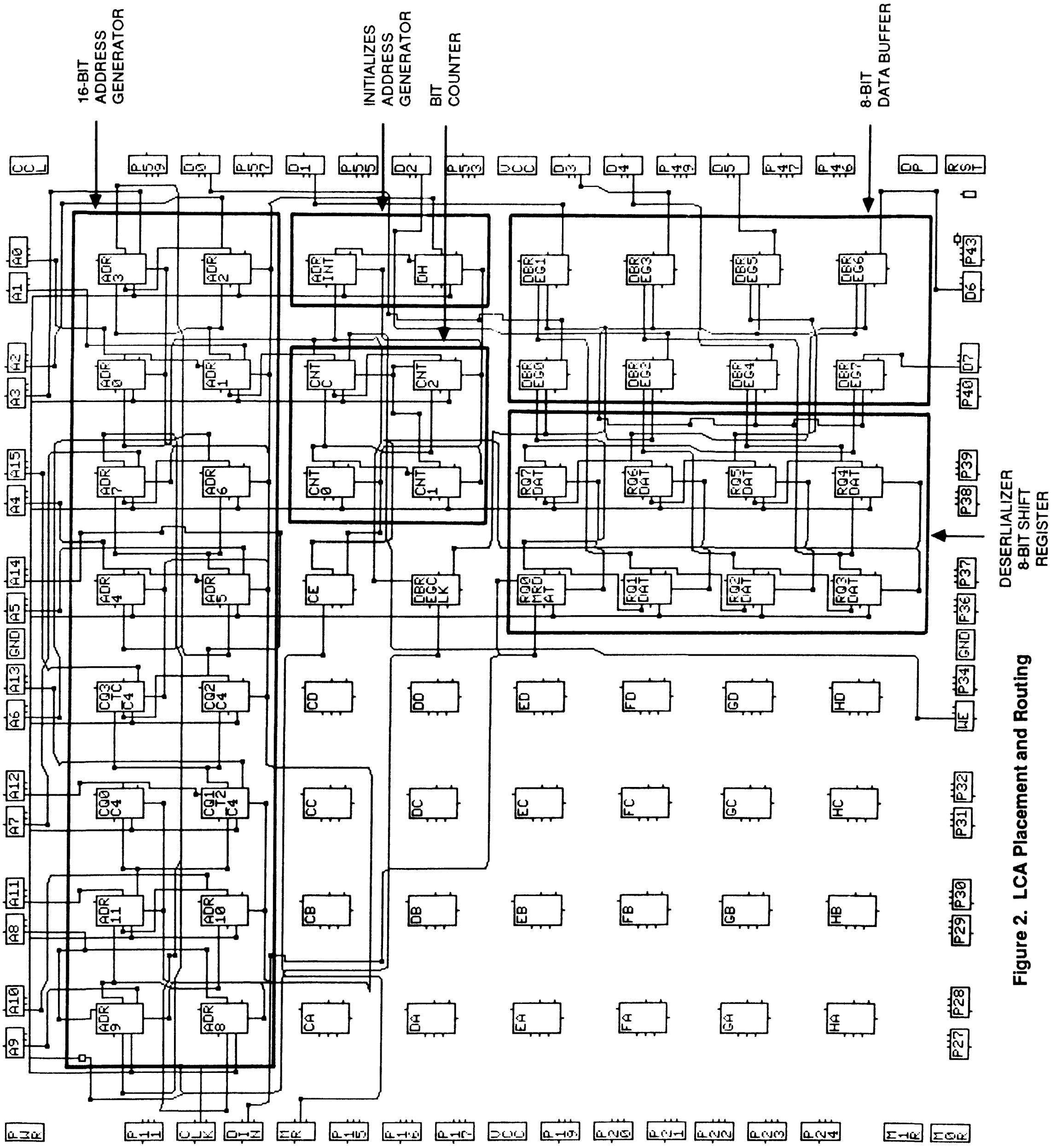


Figure 2. LCA Placement and Routing

A T1 Communicatons Interface

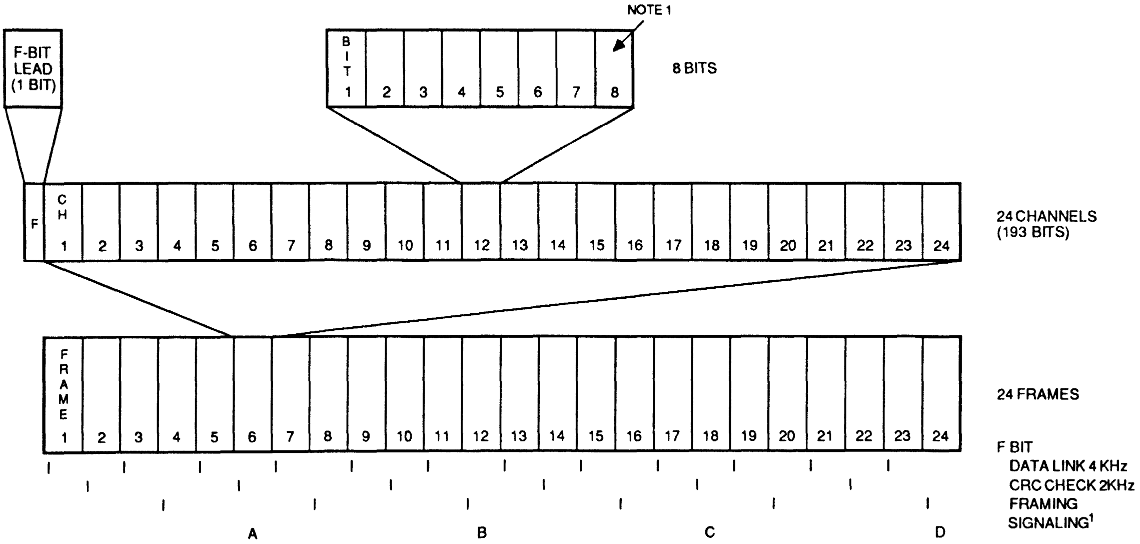
INTRODUCTION

Data communications is increasingly an integral part of successful businesses. Communication standards are being established, modified and reestablished to meet evermore demanding conditions. Among these is a requirement by the business community to curtail operating expenses. A natural way to govern the cost of new capabilities is to use existing facilities and expand their scope. The recent push to add digital data communications to equipment already used for voice transfer is an example of this expansion. The original T1 transmission, repeating and reception standard has been extended by including data transmission interleaved with the previously defined voice transmission. This application note describes the implementaion of T1-compatible logic design in a Xilinx Logic Cell™ Array (LCA).

THE T1 STANDARD

The T1 transmission standard is based on a 1.544-MHz sampling frequency. The data is time-division multiplexed into "multiframes". Each multiframe consists of 12 frames of information. Each frame is divided into 24 channels of 8-bit data bytes. The length of the frame is extended to a 193rd bit by adding a framing bit (F-bit) for synchronization. As discussed later, an extension to this basic standard has been defined and is referred to as the Extended Framing Format. This format consists of 24 frames, each organized as the same 24 8-bit channels (see Figure 1). This set of 24 frames is called a Superframe.

The framing bit consists of two signals, Fs and Ft, transmitted alternately in the first bit position of each frame. Combining Fs and Ft forms an 8-KHz signal used



1. Bit 8 (LSB) of each channel of frames 6, 12, 18 and 24 is replaced by A, B, C and D signaling, respectively.

0010026 1

Figure 1. Extended Framing Format

for "robbed-bit" signaling and mainframe synchronization. The eighth bit (Least Signifigant Bit or LSB) of each channel of frame 6 will be replaced by "A" signaling information and the LSBs of frame 12 will be replaced by "B" signaling information. This information is used to indicate off-hook status, busy, and other telephone related information.

For voice transmission, the robbed-bit signaling does not dramatically effect the receiver's ability to recover the entire voice-quality signal. However in the use of the facilities for data transmission, this bit replacement would be unacceptable. Data transmission precludes the insertion of signaling information into the bit stream.

In 1981, AT&T and other contributing corporations defined an extension of the basic T1 transmission format. In AT&T Technical Advisory 70, the Extended Framing Format was defined as (1) extending the multiframe from 12 to 24 frames, (2) adding C and D signaling information into frames 18 and 24, respectively, and (3) redefining the 8-KHz framing pattern into three subsections, now referred to as the Fe bit.

The Fe bit consists of 2-KHz framing, 2-KHz cyclic redundancy check, and 4-KHz data link patterns. The framing pattern consists of a 001011 pattern transmitted in frames 4, 8, 12, 16, 20 and 24. The CRC-6 signal is transmitted in frames 2, 6, 10, 14, 18 and 22. The data link information is transmitted as the leading bit in alternate frames starting with frame 1. The data link information is transmitted using the X.25 level 2 protocol.

The standard also includes a specific requirement of maintaining a certain "ones density" on the transmission line. The T1 repeaters used on the telephone lines require enough energy (ones data and low-to-high transitions) to regenerate the signal and send the signal on to a subsequent repeater. The T1 specification allows the transmission of all 0 data channels by transmitting a specific bipolar violation signal instead of the standard alternate mark insertion (AMI) signal. Known as the B8ZS signal, this specific bipolar violation pattern can be distinguished by the receiver as an all-zero data channel rather than a transmission line problem.

T1 TRANSMITTER OPERATION

As shown in the block diagram in Figure 2, the T1 Extended Framing Format transmitter consists of six major subsections: the data input registers, the framing bit register, the signaling bit and data shift register, the counter based modulo 193 and 24 timing generators, the bit select multiplexer, and the bipolar generation circuitry. These subsections work together to generate a T1 transmission frame.

T1 TRANSMITTER BLOCK DIAGRAM

At system reset, the timing generator counters (CQ0–CQ7, FR0–FR4) are synchronized and reset. The latched synchronization input, SYNCIN, can be used to hold off operation. The counters are held in the reset state by using the asynchronous reset capability of the Configurable Logic Block. At any time, the SYNCIN can be used to issue this reset request. External circuitry should monitor SYNCOUT for acknowledgment.

The initial end-of-frame (EOF) pulse from the EOF decoder causes the bit select multiplexer to transmit the first framing bit, a data link information signal. This circuitry is also used to load the Fe bit shift register with its framing pattern. A second pulse is then issued to the frame counter to bypass the second unused state.

The bit counter then processes 24 bytes of channel data using the input latch, data request register, shift register, signaling bit registers and the associated circuitry. The bit-select multiplexer monitors the 3 LSBs of the bit counter to issue a request for a new channel byte, to present channel data being sent and to insert signaling information when required.

The shift register decodes the position of a preset flag bit that is shifted through the register. The flag decode logic issues a data request (DREQ) signal and the data request register loads subsequent channel byte information. Whenever the zero detection logic detects an insufficient ones density in the data stream, it signals for an external B8ZS zero substitution in lieu of the bipolar generation circuitry output.

Each time the EOF signal is asserted, it causes the data shift register to pause. This allows the subsequent framing bit to be inserted into the bit stream as defined by the content of the frame decoder. Subsequent frames of channel data are transmitted using the same sequence.

AN LCA BASED T1 TRANSMITTER DESIGN

The Data Input Register

The data input register section includes the 8-bit input register, the 9-bit data shift register (8 data bits plus a flag bit), the zero detect logic section and the data request register. By utilizing eight of the 58 input registers of the XC2064 Logic Cell Array (LCA), the Input Latch was implemented without using any of the 64 Configurable Logic Blocks (CLBs). To minimize the use of LCA resources, the designer should place all data inputs on a common side of the die. This orientation requires only one of the Input/Output Block clocks to latch data. The other three sides are available for the framing and signaling inputs.

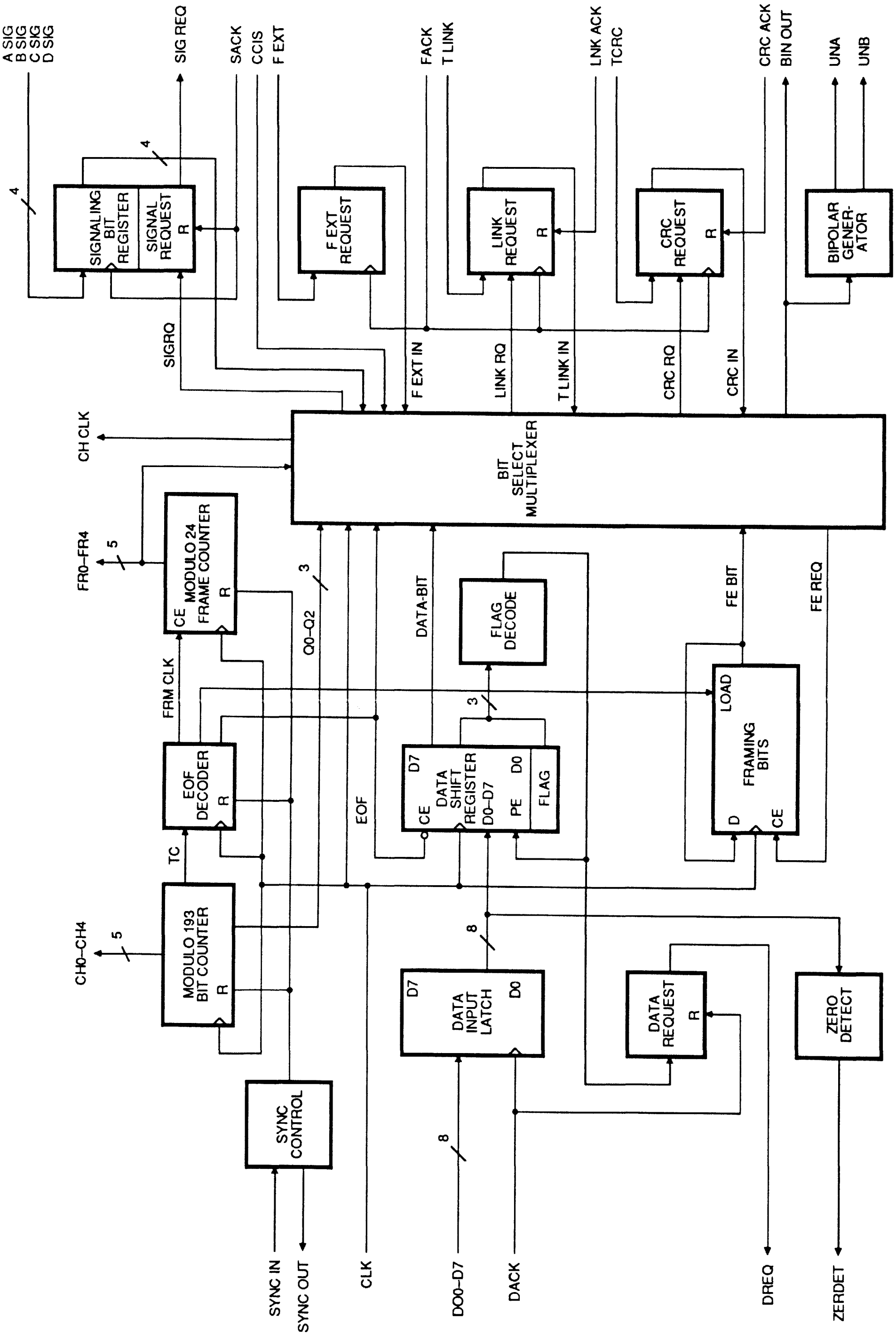


Figure 2. T1 Transmitter Block Diagram

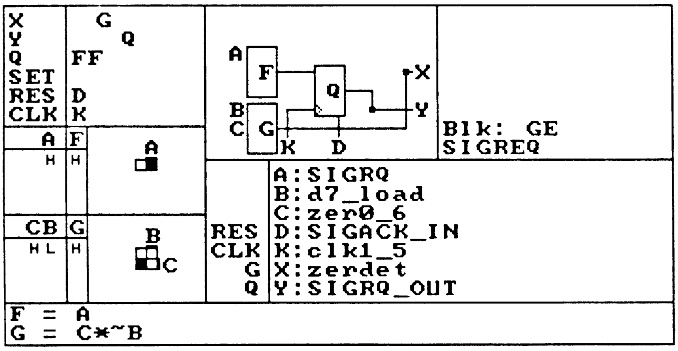
0010026 2

By generating the DREQ output, the data request register requests a byte of channel information. Figure 3 shows an XACT "editblock" screen of the Configurable Logic Block implementing this function. External circuitry should respond by placing the data byte on the D0–D7 inputs and loads it into the input latch by asserting the DACK input. DACK also resets the request register. If the zero detect logic does not indicate an all-zero data byte, the byte is transferred into the shift register after the previous byte has finished. The decode logic in the data request register uses the flag bit to indicate that the previous byte has been sent.

A modified version of the RS8PR macro (eight bit shift register with parallel load and synchronous reset) from the XACT macro library was used to implement the Data shift register of this application. Figure 4 shows the eighth bit implementation where pin B functions as a clock enable, pin C as load enable. The eight data bits are loaded into the data shift register in parallel and then shifted out, one bit at a time onto the bit-select multiplexer. Replacing the synchronous reset with a clock-enable function will allow the end-of-frame (EOF) signal to force a pause in the shift, allowing the framing bit to be inserted into the bit stream.

The Framing Bit Registers

The extended framing Fe bits are placed in a 6-bit shift register shown left of the bit select multiplexer of Figure 2. The framing bit registers for data lnk and CRC information use a similar handshake to the request/acknowledge scheme used in the data input register. The F ext register's framing information is controlled with the common channel interoffice signaling (CCIS) input. All three registers, shown on the right of Figure 2, request new information during the frame prior to the frame in which they are used. External circuitry must respond to these requests within the alotted time for the data link and CRC cases.

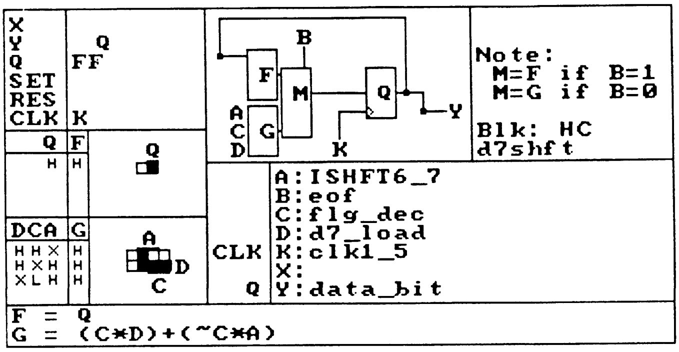


```
Blk: GE
SIGREQ

    A:SIGRQ
    B:d7_load
    C:zer0_6
RES D:SIGACK_IN
CLK K:clk1_5
  G X:zerdet
  Q Y:SIGRQ_OUT

F = A
G = C*~B
```

Figure 3. Input Register Signal Request Block



```
Note:
 M=F if B=1
 M=G if B=0

Blk: HC
d7shft

    A:ISHFT6_7
    B:eof
    C:flg_dec
    D:d7_load
CLK K:clk1_5
    X:
  Q Y:data_bit

F = Q
G = (C*D)+(~C*A)
```

Figure 4. The Eighth Bit of the Input Shift Register

The Signaling Bit Registers

The signaling bit register in the upper right of Figure 2 provides a request line and is loaded with an acknowledge signal. The A, B, C and D signaling information respectively, is substituted for the LSB of every channel byte of the 6th, 12th, 18th and 24th frames of the superframe. Externally connecting the D0 input to the signaling inputs would facilitate the removal of signaling information. Signaling information is used only in voice transmission.

The Timing Generation Section

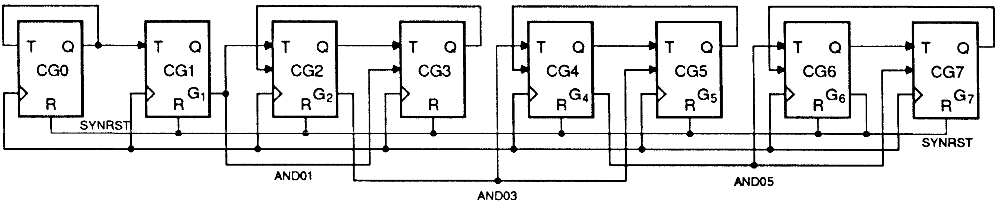
The timing generation section (top of Figure 2) includes the modulo-193 bit counter and the modulo-24 frame counter. The modulo-193 counter is a conventional 8-bit binary counter with a special synchronous reset capability. The modulo-24 counter uses a "hold bit" technique to reduce the 5-bit, 32-state counter to the desired 24 state count. The LCA flexibility allows the user to tailor circuitry to the exact configuration needed.

The Modulo-193 Bit Counter

Figure 5 shows the bit counter. The 8-bit counter is reset synchronously after it cycles through 193 states. The end-of-frame (EOF) decoder indicates that the 193rd state has been reached. The EOF signal is used in many places throughout the design. EOF allows the frame counter to increment by performing a clock-enable function. It is also used to allow the proper framing bit to be inserted by holding off the data shift register for one clock cycle. Figure 6 shows the configuration for one of the CLBs used in this counter.

The Modulo-24 Frame Counter

Figure 7 shows the 5-bit binary counter sequence used for the frame counter. Whenever the three least significant bits (FR0, FR1, FR2) reach all-ones state, the FR1 bit will be held. After every six sequential frame-counter clock cycles, the next two binary states are skipped. In this way, four cycles of six states are used to generate a 24-state counter. Figure 7 also indicates the



$G_1 = Q_0 \bullet Q_1 = \text{AND01}$

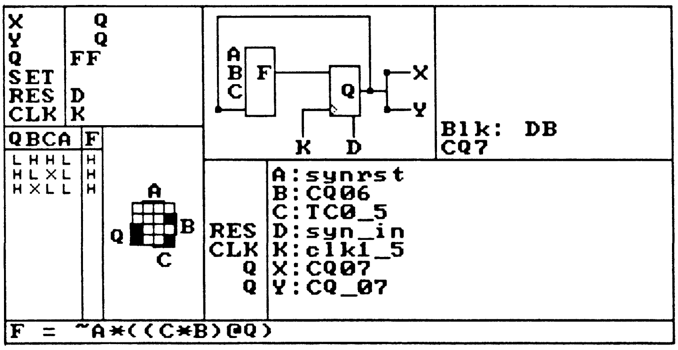
$G_2 = \text{AND01} \bullet Q_2 \bullet Q_3$

$G_4 = \text{AND03} \bullet Q_4 \bullet Q_5 = \text{AND05}$

$G_6 = \overline{\text{AND05}} \bullet Q_6 \bullet Q_7 = \text{SYNRST}$

0010026 5

Figure 5. Modulo 193 Bit Counter



X Q
Y Q
Q FF
SET
RES D
CLK K

QBCA	F
L H H L	H
H L X L	H
H X L L	H

Blk: DB
CQ7

RES A:synrst
CLK B:CQ06
Q C:TC0_5
Q D:syn_in
K:clk1_5
X:CQ07
Y:CQ_07

F = ~A*((C*B)@Q)

Figure 6. Eighth Bit of Modulo 193 Counter

relationship of each encoded value and its corresponding frame value. The CHCLK signal can be used to synchronize external decoder circuitry. Internal decoding could be included in this design, but is left up to the user to implement. Figure 8 shows an example CLB configuration for this counter.

The Bit-Select Multiplexer

The bit-select multiplexer (see Figure 9) uses a frame decoder (see Figure 10) with other logic and registers to determine whether data, signaling or framing information is to be sent. The presence of the EOF signal in the multiplexer determines which source of information should appear at the output signal.

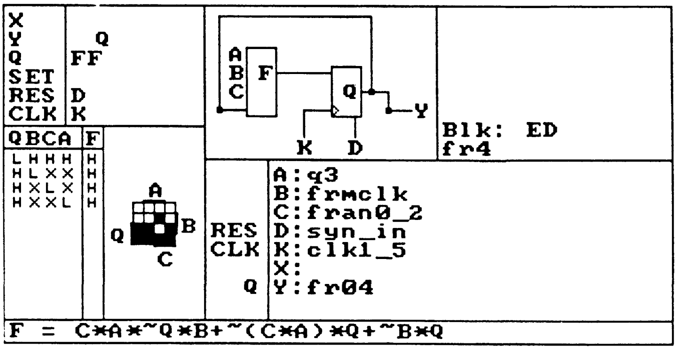
The CCIS signal is used to insert an externally generated framing bit into the bit stream when the EOF signal is active.

	PREVIOUS STATE					NEXT STATE					FRAME
	FR 4	FR 3	FR 2	FR 1	FR 0	FR 4	FR 3	FR 2	FR 1	FR 0	
	0	0	0	1	0	0	0	0	1	1	1
	0	0	0	1	1	0	0	1	0	0	2
	0	0	1	0	0	0	0	1	0	1	3
	0	0	1	0	1	0	0	1	1	0	4
*	0	0	1	1	0	0	0	1	1	1	5
	0	0	1	1	1	0	1	0	1	0	6
	0	1	0	1	0	0	1	0	1	1	7
	0	1	0	1	1	0	1	1	0	0	8
	0	1	1	0	0	0	1	1	0	1	9
	0	1	1	0	1	0	1	1	1	0	10
	0	1	1	1	0	0	1	1	1	1	11
*	0	1	1	1	1	1	0	0	1	0	12
	1	0	0	1	0	1	0	0	1	1	13
	1	0	0	1	1	1	0	1	0	0	14
	1	0	1	0	0	1	0	1	0	1	15
	1	0	1	0	1	1	0	1	1	0	16
	1	0	1	1	0	1	0	1	1	1	17
*	1	0	1	1	1	1	1	0	1	0	18
	1	1	0	1	0	1	1	0	1	1	19
	1	1	0	1	1	1	1	1	0	0	20
	1	1	1	0	0	1	1	1	0	1	21
	1	1	1	0	1	1	1	1	1	0	22
	1	1	1	1	0	1	1	1	1	1	23
*	1	1	1	1	1	0	0	0	1	0	24

* Next state holds FR1 high, therefore skipping two states four times during sequence.

0010026 7

Figure 7. Modulo-24 Frame Counter



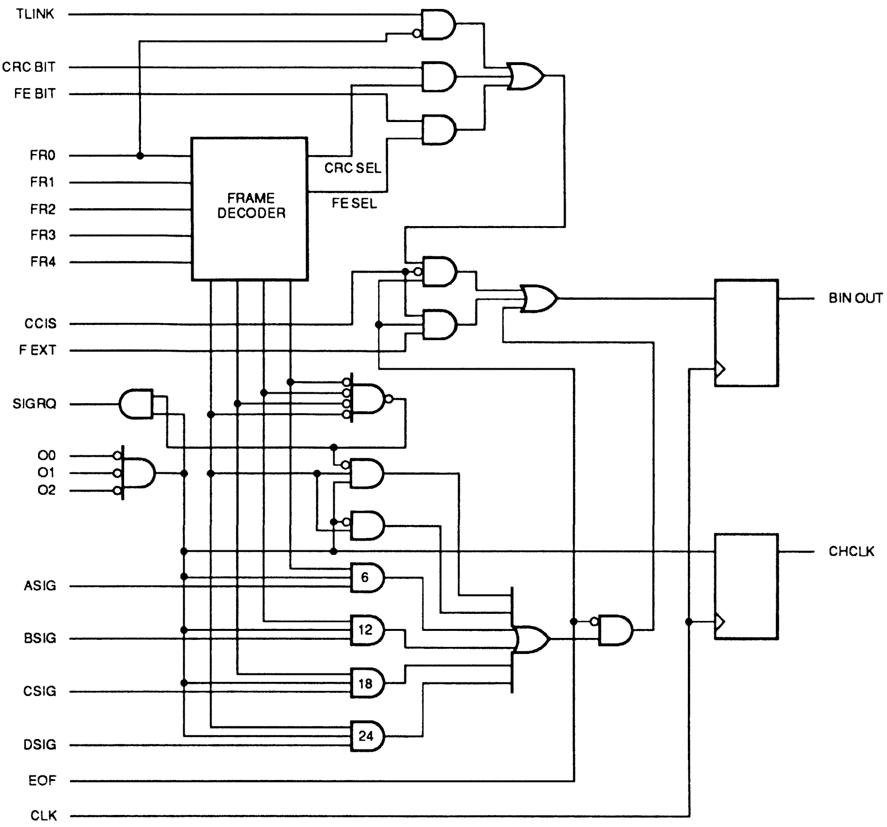
QBCA	F
L H H H	H
H L X X	H
H X L X	H
H X X L	H

Blk: ED
fr4

A:q3
B:frmclk
C:fran0_2
D:syn_in
K:clk1_5
X:
Y:fr04

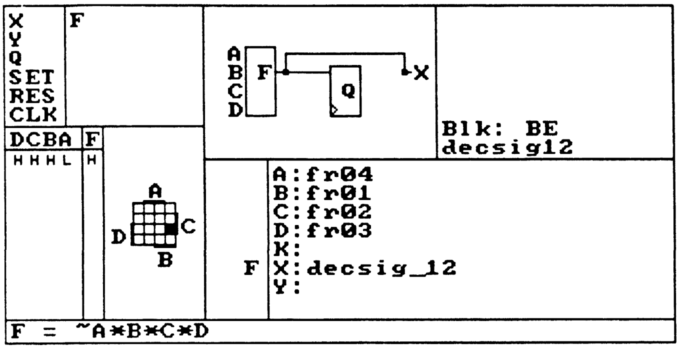
F = C*A*~Q*B+~(C*A)*Q+~B*Q

Figure 8. The Fifth Bit of the Frame Counter



0010026 9

Figure 9. Bit-Select Multiplexer



Blk: BE
decsig12

A:fr04
B:fr01
C:fr02
D:fr03
K:
F X:decsig_12
Y:

F = ~A*B*C*D

Figure 10. Frame-12 Decoder Example

The framing decoder, CCIS, FeEXT, TLINK, TCRC and the associated logic will determine the proper bit to transmit. The frame decoder defines whether the Fe (frames 4, 8, 12, 16, 20 and 24) or CRC bit (frames 2, 6, 10, 14, 18, and 22) is to be inserted into the framing bit position. When EOF is inactive, the least significant bit in the frame counter determines that the data link bit is to be inserted into the bit stream. CCIS and EOF are also used to gate transmissions of the framing bit.

The framing decoder, the ASIG-DSIG, DATA BIT and the three least significant bits of the Modulo-193 counter determine the proper data bit to be transmitted. When the bit counter inputs are all in a zero state, the gating functions allow the previously sampled A, B, C or D signaling data to be inserted into the bit stream during the 6, 12, 18 and 24th frames, respectively. This signaling bit replaces the LSB from the data byte in the transmitted bit stream.

Bipolar Generation

The bipolar generation logic converts the binary output signal from the bit-select multiplexer into a pair of unipolar outputs. The bipolar generation logic utilizes an array of control logic along with a pair of registers to convert NRZ coded data into two unipolar alternate mark insertion (AMI) coded data. A third register indicates which unipolar output last received the mark pulse.

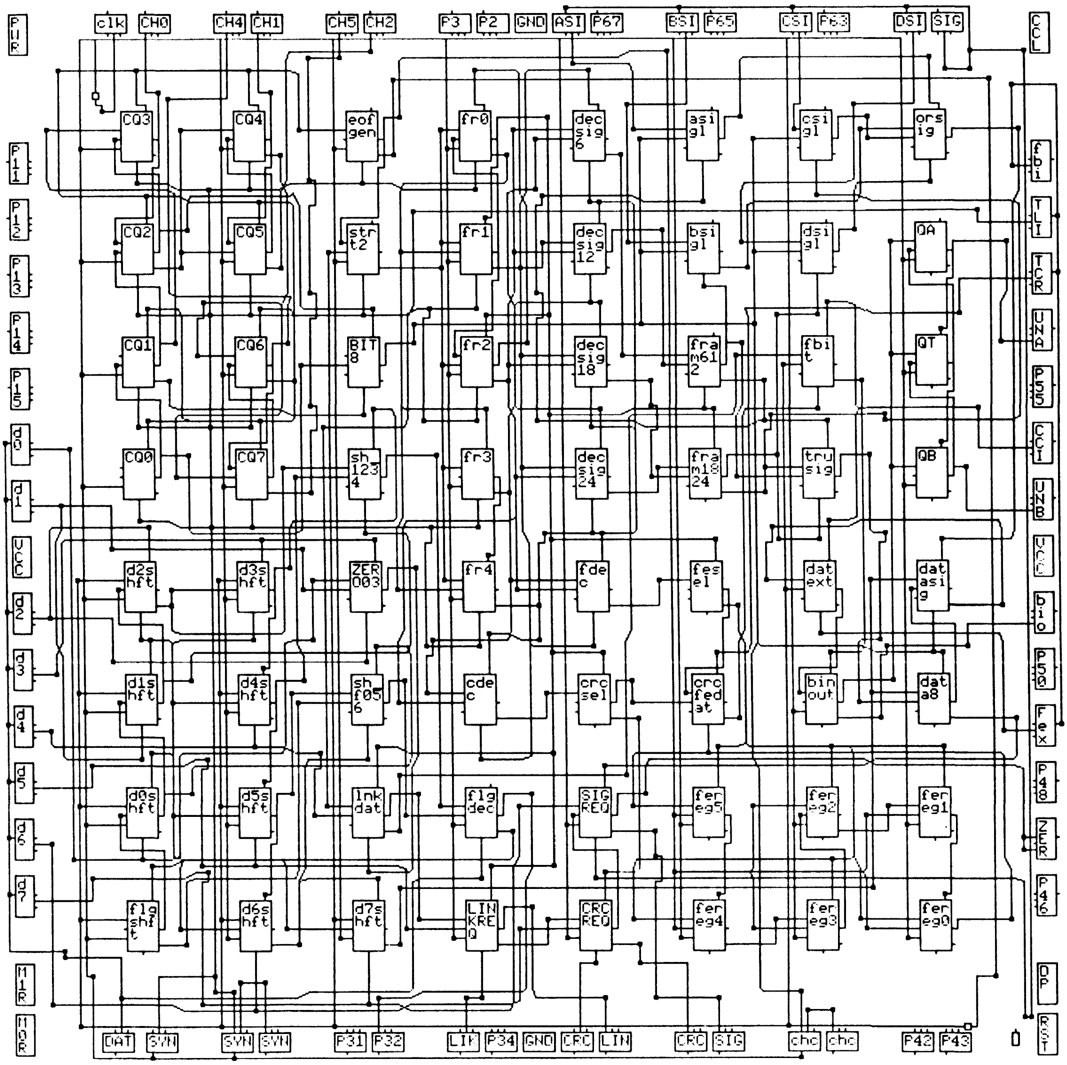
These outputs can be externally tied to a T1 Line Interface Unit (LIU). The LIU will provide the paired unipolar output conversion to a bipolar signal. This LIU will also equalize the T1 transmitter circuitry to the transmission line.

T1 TRANSMITTER DESIGN EXTENSIONS

Many features of the T1 standard could be incorporated into designs similar to the one shown in this application note. One simple extension is the generation of the data format required for the CCITT (European version) version of the T1 standard. This version uses 256 bit frames (32 time slots of 8-bit bytes) for data transmission along with defined specific time slots sending signaling information. For flexibility, both designs could reside in configuration-program form, allowing the system to select which standard to use.

Other possibilities could be insertion of a B8ZS zero-suppression standard control section or alternate forms such as the HDB3 standard used in the CCITT format. Idle code transmission, digital milliwatt (the transmission of a repetitive data sequence in a individual channel), and internal CRC generation are all examples of further design extensions.

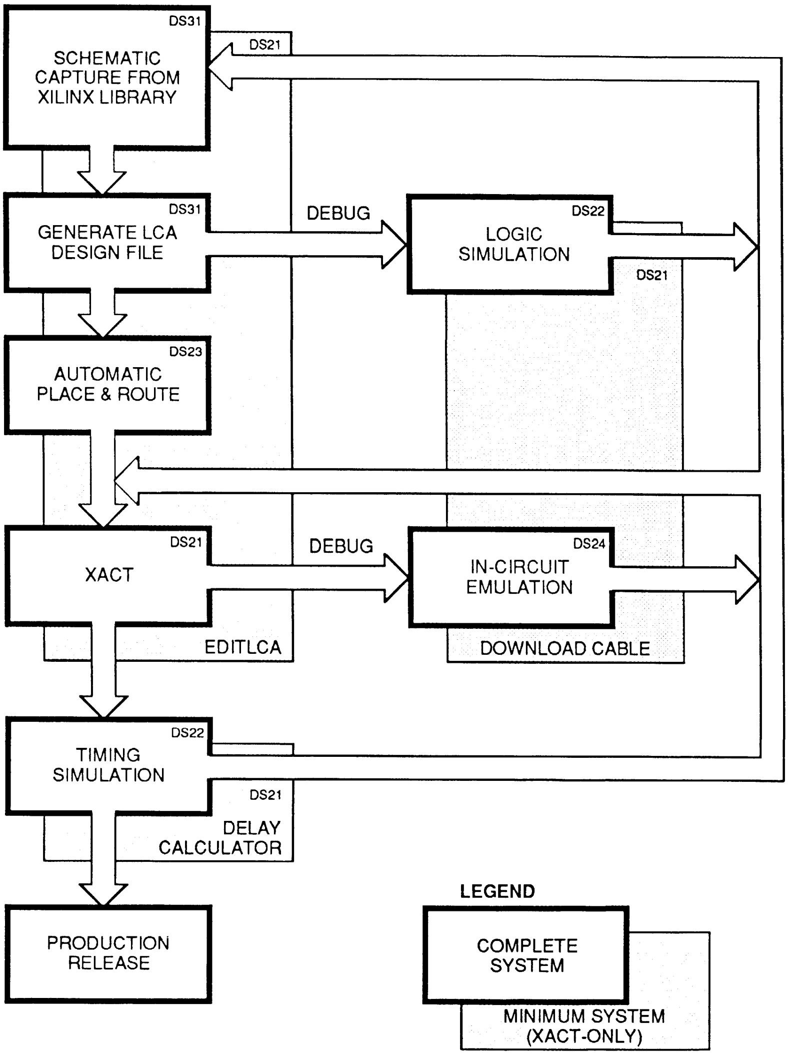
With the advantages of LCA flexibility, a single EPROM could hold the configuration data for many different versions of the design. This would allow support of many transmission variations using the same common transmit hardware.



Placed and Routed T1 LCA Design

XILINX

The Programmable Gate Array Company



Logic Cell Array

Development System Options

The DS21 XACT Design Editor provides all capabilities required for Logic Cell Array design. Additional development system options provide enhanced designer productivity during design entry, placement and routing, and design verification.

XILINX

Table of Contents

The Programmable Gate Array

Using Programmable Gate Arrays

Programmable Gate Array Design Examples

Programmable Gate Array Development Systems

Appendices

PC System Configurations

INTRODUCTION

Xilinx provides a Development System which facilitates the design of systems which incorporate the Xilinx Logic Cell™ Array (LCA). All of its software is designed to run on two widely available, low-cost workstations: the IBM® PC/XT™, and PC/AT™ computers. The purpose of this application note is to advise users of the software how they should configure their PC to make the best use of the software. For the most part, this consists of choosing among the myriad of options which may be used with a standard PC/XT or PC/AT.

First, let's make sure what we mean by a PC/XT or PC/AT. There are many so-called clones of the IBM PCs. The difference in cost between one of these clones and the original can be significant. In addition, some clones offer features and/or performance greater than that of the original. For these reasons, purchase of a clone may be a cost-effective alternative. Every clone tested so far, which claims to be PC compatible, has run the current Xilinx software.

MINIMUM CONFIGURATION

The minimum configuration required to run the Xilinx XACT™ Design Editor for the XC2064 LCA is as follows:

- 1 - PC/XT
- 1 - Hard disk drive
- 1 - Floppy disk drive
- 640K Bytes of memory
- 1 - RS-232C serial port
- 1 - Centronics printer parallel port
- 1 - Color Graphics Adapter (CGA)
- 1 - Color Graphics Display (CGD)
- 1 - Mouse

This is a minimum configuration. For many applications, the performance of the software in this configuration is adequate. Additional equipment may be needed to use some Development Systems options, or to design with larger arrays.

RECOMMENDED CONFIGURATION

A more powerful configuration may result in significant improvements in productivity:

- 1 - PC/AT
- 1 - Hard disk drive
- 1 - Floppy disk drive
- 640K Bytes of memory
- 2 - RS-232C serial ports
- 1 - Centronics printer parallel port
- 1 - EMS memory card with at least 256K
- 1 - Enhanced Graphics Adapter (EGA)
- 1 - Enhanced Graphics Display (EGD)
- 1 - Mouse

The first advantage of this configuration is performance. The performance of the software in this configuration is much improved over the minimum configuration. This is true both because of the improved performance of the 80286 over the 8088 and the AT hard disk over the XT hard disk.

The second advantage of this configuration concerns the display. Xilinx software only requires the CGA and CGD. However, the FutureNet™ schematic capture package, which can be used in conjunction with the Xilinx software requires the EGA and EGD. In addition, future versions of the Xilinx software will take advantage of the higher resolution of the EGA and EGD if available. The second serial I/O port provides a connection for the XACTOR™ In-Circuit Emulator.

The final advantage of this configuration concerns future LCAs. Currently, we have squeezed both the XC2064 family and the XC2018 family of LCAs into the memory available on the PC/XT. Xilinx software to support larger LCAs will require the IBM PC/AT class of machines running a new version of MS-DOS™ which will use the protected mode of the 80286. The recommended configuration will provide support for the larger LCAs now in design.

SPECIFIC HARDWARE RECOMMENDATIONS

Growth in the PC market has resulted in many options for each item in the configuration. The following is a description of the recommendations for each item:

PC/XT

There are many machines available in this category. Prices and quality vary greatly, but as noted previously, every compatible clone tested so far will run the current Xilinx software. The trade-off here is between quality and price. The IBM PC/XT has a proven record of reliability and the IBM keyboard has a better "feel" than the keyboard found on most clones. On the other hand, a clone can be purchased for less than 1/3 the cost of the IBM.

In addition to the system unit and keyboard, the PC/XT should include the following standard equipment:

- 256K Bytes of memory on the motherboard
- 1 - 360KB Floppy disk drive
- 1 - 10MB Hard disk drive
- 1 - Serial/parallel interface card

The standard equipment helps to meet some of the other configuration requirements. If these are not included, they would have to be purchased separately.

PC/AT

There are many machines which fall into this category as well. However, there are a few more things to consider about clones of the PC/AT. Every clone of this type tested so far will run the current Xilinx software under versions of MS-DOS up to and including 3.2. However, since the protected mode version of MS-DOS is not yet available it is possible that a clone would not execute properly in that mode. One should try to get a guarantee from the manufacturer that their clone will work in the protected mode of MS-DOS, when that version arrives. Without such a guarantee, buying that clone will entail some risk—it will probably execute the current Xilinx software, but may not execute future versions of the software which require the protected mode MS-DOS .

One of the most compelling reasons for selecting a PC/AT clone is that many of them are considerably faster than the the original. IBM has 2 PC/ATs: one runs at 6 MHz and the other at 8 MHz. This speed improvement is quite noticeable when executing the Xilinx software. However, there are some potential pitfalls to selecting a faster machine. The biggest problem seems to be that not all add-on cards will behave properly in a faster PC/AT. Specifically, problems have been observed with serial interface cards. The problem has been observed both in cards from IBM and in others.

In addition to the system unit and keyboard, the PC/AT should include the following standard equipment:

- 512K Bytes of memory on the motherboard
- 1 - 1.2MB Floppy disk drive
- 1 - 20MB Hard disk drive
- 1 - Serial/parallel interface card

As with the PC/XT, the standard equipment helps to meet some of the other configuration requirements. If these are not included, they must be purchased separately.

640K of Memory

The memory requirement is met differently depending on whether one is starting with a PC/XT or PC/AT.

On a PC/XT, the 640K memory requirement is usually met with a multifunction card which has 384K of memory and a serial port. The most widely used such card is the AST Six-Pak™. Another way to meet this requirement is with a multifunction card which provides a serial port, 384K of memory and 256K of EMS memory. An example of this kind of multifunction card is the Intel Above™Board/PS.

On a PC/AT, this requirement is usually met with a multifunction card which has 128K of memory and a serial port. The most widely used such card is the AST Advantage™. Another way to meet this requirement is with a multifunction card which provides a serial port, 128K of memory and 256K of EMS memory. An example of this kind of multifunction card is the Intel AboveBoard AT/PS. If you should decide to get separate cards for the additional memory and an extra serial port be careful when selecting serial port cards. Several serial cards, both from IBM and others, do not work in PC/AT's running at more than 6 MHz.

EMS Memory Card

An EMS memory card extends the PC memory beyond 640K bytes. Sometime ago Lotus, Microsoft and Intel proposed a software standard for accessing a 64K page frame with a virtually unlimited number of pages. This standard is the "Expanded Memory Specification" or EMS. Because of the names of the companies involved, it is also sometimes referred to as the "LIM" standard. Hardware manufacturers have produced cards and related software device drivers which meet this standard. Software manufacturers like Xilinx have developed software which uses this standard to access memory in excess of the standard 640K. In the current

Xilinx software, the EMS memory is used to expand the available memory so there is room for the extra data necessary to support larger LCAs, specifically the XC2018 family of LCAs.

There are several different cards which meet the EMS standard:

- Intel AboveBoard
- Intel AboveBoard/PS
- AST RamPage
- Techmar Maestro

This is not a complete list. There are several other cards on the market. Since there is a published standard for these cards, any card which claims to be an "EMS Standard" card should work with the Xilinx software. Usually, each card has 2 types: one type for the PC/XT and another for the PC/AT.

For the PC/AT, some extra features should be considered when purchasing an EMS card. These involve the protected mode MS-DOS which will be available in the future. The EMS card should be reconfigurable to be "*extended* memory" rather than "*expanded* memory" for use with protected mode MS-DOS when it becomes available. The 8088 processor, which is found in the PC/XT (and its cousin the 8086) can directly address only 1MB of memory. The 80286, which is found in the PC/AT, can address up to 16MB of memory when executing in its "protected" mode. IBM coined the term "extended memory" to refer to that memory in a PC/AT which is above the 1MB limit and can only be accessed by the 80286 in protected mode. An EMS card which can be reconfigured as "extended memory" will be useful when the protected mode MS-DOS is introduced. The Intel AboveBoard/AT is one EMS card which offers this feature.

Color Graphics Adapter (CGA) and Color Graphics Display (CGD)

As first defined by IBM, the PC included two different display options. The first was the Monochrome Display and the second was the Color Graphics Display (CGD). The interfacing of these displays to the PC was done with two different cards: the Monochrome Display Adapter (MDA) and the Color Graphics Adapter (CGA). The Monochrome Display and MDA can only be used to display text. The text is of very high quality because the font is 9X14 pixels , i.e. 126 dots (9 times 14) were used to represent each text character. The Monochrome Display and its adapter were not designed to display graphics. The CGD *and* CGA were designed to display text and graphics. The text however is of poorer quality than the Monochrome Display because the font used is 8X8 pixels, i.e. 64 dots (8 times 8) were used to represent each text character. The CGD can display graphics by allowing individual pixels, or dots, in a 320X200 grid to be manipulated. Each dot can be one of four colors. These two displays and adapters became the established standards. Since then, many other displays and adapters have been designed. Most of them are compatible with one of the original displays and offer other additional features. IBM designed a new display called the Enhanced Graphics Display (EGD) and an adaptor called the Enhanced Graphics Adapter (EGA) which emulates the older CGD and CGA. The EGD and EGA display text in a 8X14 pixel font and can display graphics in a native mode by allowing a 640X350 grid of pixels to be manipulated.

Which is the right display for a system to be used for designing with LCAs? The minimum requirement is a display and an adapter that are compatible with the CGD and CGA. However, there are several reasons to choose the EGD and EGA. The first relates to the FutureNet™ schematic capture package. FutureNet requires a display and adapter that are compatible to the EGD and EGA. The EGD and EGA's text display is much better than that of the CGD and CGA. With the EGD and EGA, you can use your machine for other applications which require only a text display. When the EGD and EGA are used by the current Xilinx software, graphics will be displayed in the CGD and CGA compatible mode and text will be displayed in the native EGD and EGA mode. In the future however, Xilinx software which will use the EGD and EGA in the higher resolution native graphics mode will be available.

Mouse

Prior to release 1.3 of the Xilinx software, the choice of a mouse was easy: only the PC Mouse from Mouse Systems was supported. Starting with release 1.3 of the Xilinx software however, several different kinds of mice are supported:

- PC Mouse from Mouse Systems
- MicroSoft serial mouse
- MicroSoft parallel mouse
- LogiTech mouse
- FutureNet mouse

In addition to these mice, the software will also support any mouse which is either compatible to the PC Mouse or has a device driver which makes it look like a MicroSoft Mouse. The choices here are driven by cost and by the requirements of other software used with the Xilinx software. With the FutureNet schematic capture system one should use the FutureNet Mouse which comes with that system.

	XACT	XACTOR	Auto Place and Route (APR)	P-SILOS Simulator	Schematic Entry	
					Futurenet	
System Memory[1,2]	640K	640K	640K	640K	512K	
Floppy Disk	360K	360K	360K	360K	360K	
Hard Disk[3]	10MB	10MB	10MB	10MB	10MB	
Mouse	Mouse Systems, MicroSoft, FutureNet, or none		None	None	FutureNet	
Display and Adapter Card[4]	Standard Color Display and Adapter		Monochrome or Standard Color		Enhanced Color with 256K	
Serial Port	1 (Mouse)	2 (Mouse, XACTOR)	0	0	0	
Parallel Port[5]	1 (Download Cable, Printer, Security Key)	1 (Printer, Security Key)	1 (Printer, Security Key)	1 (Printer, Security Key)	0	
Expanded Memory Card[6]	With 256K for XC2018 Designs	None	With 256K for XC2018 Designs	None	None	

1. The XACT, XACTOR and APR programs will utilize expanded memory, if available.
2. XACT, XACTOR and Auto Place and Route, require 600K Bytes of available system memory to operate on a full design.
3. XACT, XACTOR and P-SILOS combined require 3 M Bytes of Hard disk. Auto Place and Route requires an additional 1 M Byte.
4. The XACT and XACTOR Program will utilize the enhanced color capability, if available.
5. The following printer types are supported: OKI92, IBM Graphics, HP Laser, MX80, MX100, FX80 and FX100.
6. EMS card not required for XC2064 designs.

Table 1. Minimum PC Configurations for IBM PC XT/AT or Compatibles with DOS 2.1 or Later

Fundamentals of In-circuit Emulation and Simulation

This Application Note describes the operation of two design verification tools for designers using Logic Cell™ Arrays: the XACTOR™ In-Circuit Emulator, and the P-SILOS logic simulator.

The Logic Cell™ Array (LCA) combines a high-performance, general-purpose gate array architecture with user programmability. Since the Logic Cell Array is user-programmed, designs can be verified in real-time in a system during development.

Certain innovative provisions of the Logic Cell Array extend this in-circuit verification capability to in-circuit emulation. Emulation permits reading the state of internal logic and I/O latches during operation, as well as temporarily configuring unused I/O pins to monitor internal nodes during real-time operation.

While in-circuit emulation adds a unique and valuable capability to an ASIC designer's repertoire, simulation remains a useful complementary tool for verification of critical paths and worst-case timing analysis. Simulation may also be used to verify logic modules before a prototype system is available for in-circuit emulation. With the Logic Cell Array (LCA), however, ASIC design success is no longer dependent on painstaking and exhaustive simulation.

DESIGN METHODOLOGY

The Logic Cell Array design methodology is based on a natural, iterative design approach in which successive design implementations can be readily verified in-circuit, as described in a design example below.

The implementation of a Logic Cell Array design has three stages: design entry and partitioning, placement and routing, and compilation of the configuration program. First the design is entered, using either a PC-based schematic capture package with a Xilinx library, or with the Xilinx XACT™ LCA Design Editor. Schematic entry permits entering the design with standard logic symbols such as gates and latches. The XACT editor is a graphical environment which permits direct design entry using equations or Karnaugh maps for each LCA Configurable Logic Block.

The design is then partitioned into blocks of logic corresponding to the LCA's Configurable Logic Blocks. For schematic entry users this is done by automatic gate-to-LCA partitioning software provided with the schematic library. For XACT users, the partitioning is performed by the designer during design entry as each logic or I/O block is configured.

As with a conventional gate array, each partitioned logic and I/O block is "placed" by assigning it to a specific physical location within the Logic Cell Array. Then the interconnect networks between blocks are routed. When a design is entered or edited with XACT, placement of each logic and I/O block is done implicitly as the block is configured to have the desired function. Routing software within XACT then automatically allocates programmable routing resources for each net in the design. Use of special interconnect resources such as direct interconnect and long lines for time-critical nets can also be specified by the designer.

As an alternative to interactive placement and routing with XACT, designers may use a separate Xilinx software package on their PC to place and route a design which has been entered and partitioned with either the schematic entry package or XACT. Placement and routing constraints (such as forcing a net onto a long line) may be specified schematically with special attributes, or with a constraints text file.

Once a design has been placed and routed, it is automatically compiled by the XACT system into a configuration program. For quick checks of performance during design entry, the design editor's built-in delay calculator provides a timing analysis using calculations based on actual logic placement and routing.

Once a design has been implemented, it can be verified using the in-circuit emulator. A design is emulated by programming an emulation pod that is plugged into the target system. The heart of the in-circuit emulator is a Logic Cell Array within an emulation pod (Figure 1). The emulation LCA, connected to the target system by a flex cable, then performs the same function as an LCA plugged directly into the target system.

This realtime operation of the LCA in the target system provides the ultimate verification of an LCA design's interaction with the other elements in the system.

Another advantage of emulation is that the LCA is also under control of the designer's PC. The basic functions of the emulation system are to program the emulation pod's LCA with the designer's configuration, to monitor and control the LCA's programming and control signals, and to readback and display the LCA's latch states. The emulation controller can manage up to four emulation pods concurrently.

Due to the speed with which a design can be quickly modified in-circuit using XACTOR, it is frequently useful to implement temporary debugging circuitry in a design during development—such as temporarily connecting unused I/O blocks to internal nodes for viewing with a logic analyzer or an oscilloscope.

Once the design function has been verified in-circuit, the designer may simulate critical paths to ensure correct timing under worst-case conditions. Since the design's functionality has been verified in-circuit at this point, a complete functional simulation is not generally necessary. By eliminating simulation for complete functional design verification, the computer resources required for simulation are significantly reduced. An IBM PC provides adequate performance for timing simulation of critical paths.

Simulation can also be performed on unrouted and partially routed designs. An unrouted net will be simulated with zero interconnect delay, permitting a check of the design's logic. On extremely dense designs this logical check is useful since it permits designers to verify the design's logic before performing final placement and routing. This is especially useful for designs entered with a XILINX-supported schematic capture package which have been partitioned but not placed and routed. An unrouted design may also be created in XACT by turning autoroute off before entering the design.

DESIGN EXAMPLE

The simple dual-speed, variable modulo counter of Figure 2 was designed to illustrate the basic operation of the XACTOR™ in-circuit emulator and the P-SILOS™ simulator. All input and output pins correspond to switches and LEDs on the Xilinx DB01 Demo Board for readers who wish to experiment further with the design.

The circuit counts until its four outputs reach 0000, at which point it begins counting again at the number specified by the four parallel data inputs, D3–D0. The counter's up/down direction is selected by the switch input UP_DN, and its speed is selected by SW_SPEED. SW_SPEED = HIGH will cause the counter to run at the crystal's 1 MHz speed; SW_SPEED = LOW will select a divide-by-eight circuit to run the counter at 125 KHz.

For example, if D3–D0 are set to 0011 (3), UP_DN=LOW (down), and SW_SPEED=HIGH (fast), then the counter will repeat the sequence

3 - 2 - 1 - 0 - 3 - 2 ...

at the crystal's 1 MHz frequency. Each time the count equals zero a terminal count (TC_OUT) goes high for one clock cycle.

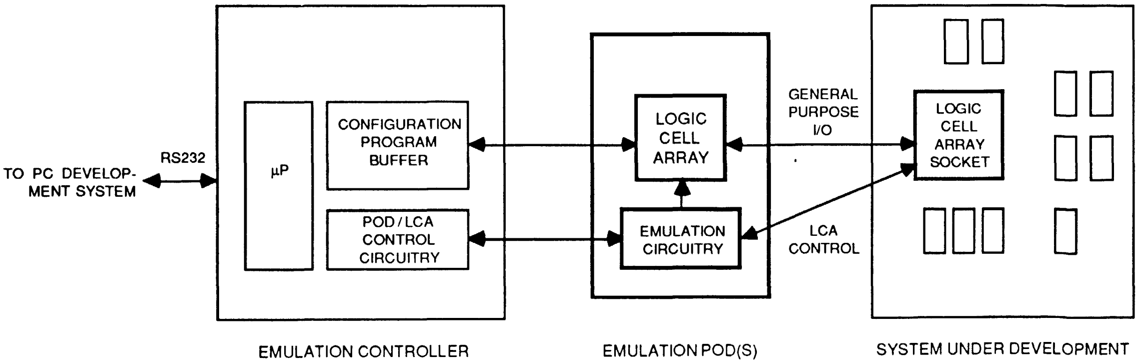


Figure 1. Emulation POD Interface

The up/down counter is constructed from the C16BUD-RD macro supplied with XACT. The HI_LO logic block provides a HIGH signal which constantly enables the counter's clock enable input (CLKENA), and a LOW signal which constantly disables the counter's reset line. In a design where space is critical, a more efficient solution would have been to modify the counter blocks to eliminate the unused clock enable and reset functions.

The four counter outputs (CNT_Q3–CNT_Q0) are routed to the outputs Q3_OUT–Q0_OUT, and to the parallel enable decode block, PE_TC. Each time the counter reaches zero, PE_TC drives the counter's PARENA input (and the TC_OUT output pin) HIGH, loading the counter with the four D3–D0 bits.

The crystal oscillator clock and divide-by-eight prescaler are implemented with the GXTL macro and a 3-stage binary ripple counter. A multiplexing circuit controlled by SW_SPEED selects between the full-speed clock and the output of the prescaler. SW_SPEED is synchronized with the crystal clock at the I/O block to prevent clock glitches during speed changes. While not required for a design running at only 1 MHz, the output of the dual-speed clock is routed to the CLK.AA clock buffer to reduce skew at the counter's clock input.

IN-CIRCUIT EMULATION

Installation and operation of the XACTOR In-Circuit Emulator is detailed in chapter 8 of the XACT LCA Development System user manual. Hardware installation consists of connecting the emulation controller to a serial port of the PC-based XACT development system, and connecting one or more emulation pods between the controller and the target system under development, as shown in Figure 3.

In addition, XACTOR users should ensure that the emulation pod assembly contains an LCA of the same speed grade specified for their design. The emulation Logic Cell Array can be easily removed from its pod assembly and replaced with an LCA of the appropriate speed.

After hardware installation, the XACTOR in-circuit emulator is invoked from the XACT development system Executive by selecting the XACTOR program from the

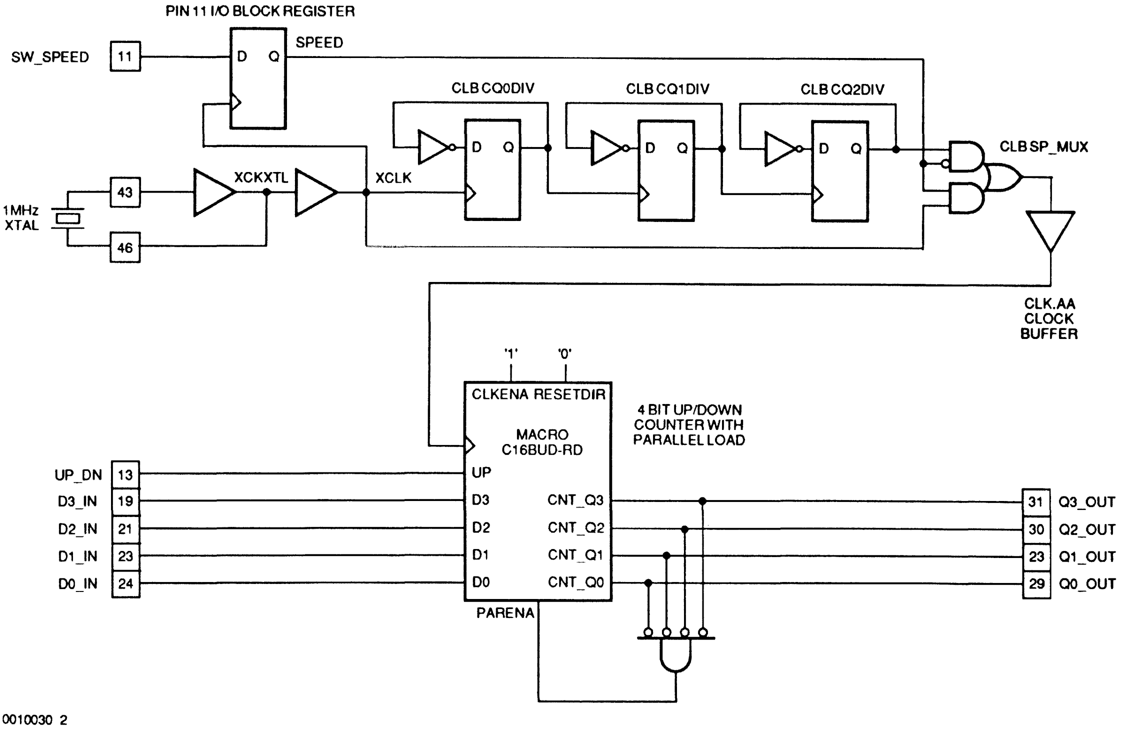


Figure 2. Dual Speed, Variable Modulo Counter

PROGRAM menu. Once the XACTOR software is running (Figure 4), emulating a Logic Cell Array design typically consists of:

1. Turning ON the system under development,
2. Loading LCA configuration program(s) into the emulation controller, and
3. Programming the appropriate emulation pod(s) with their configuration program(s).

Once programmed, a pod provides the same function as a Logic Cell Array plugged into the target system.

Loading the configuration program(s) into the XACTOR emulation controller is initiated with the LOADBITS command in the SETUP menu. A menu of all configuration bitstream files on the PC will be displayed. As each file is selected it is loaded into the emulation controller. The DONE command is used to indicate that all required bitstreams are loaded into the controller.

Any of the loaded bitstreams can then be used to program any pod connected to the controller. First the POD command in the XACTOR menu is used to specify a pod to program. Then the XACTOR menu's PROGRAM command is selected. A menu of the bitstreams in the controller is displayed, and the user is asked for both a primary and alternate bitstream. The same bitstream should be selected for a pod's primary and alternate bitstream (unless XACTOR's dual bitstream capability is being utilized as described in the XACTOR documentation).

If the AUTOLOAD setting is OFF, the pod will be programmed as soon as the PROGRAM command is completed. The emulation pod's Logic Cell Array will then operate as configured in the designer's circuit, in real-time.

If programming is to be initiated by the target system, using the configuration program stored in the emulation controller, the AUTOLOAD setting must be ON. When the target system's Done/Program line is pulsed LOW, the emulation pod will be programmed with the specified configuration program.

READBACK

The Logic Cell Array has a built-in readback feature which permits reading the configuration and logic latch state of a configured LCA during operation. Since readback permits viewing the state of each latch in a design, this feature is especially useful when debugging counters and state machines.

For example, in the dual-speed counter it is possible to read and display the state of each of the counter latches. Although the readback process is performed transparently while the Logic Cell Array is operating, it is typically necessary to temporarily inhibit further state changes during the readback period (typically 12 milliseconds or less). This is because readback is sequentially performed on each column of LCA elements.

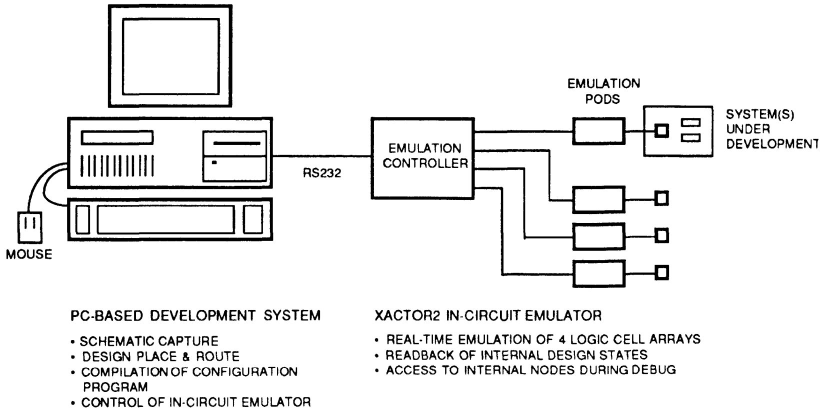

EMULATION PODS
SYSTEM(S) UNDER DEVELOPMENT
EMULATION CONTROLLER
RS232
MOUSE
PC-BASED DEVELOPMENT SYSTEM
• SCHEMATIC CAPTURE
• DESIGN PLACE & ROUTE
• COMPILATION OF CONFIGURATION PROGRAM
• CONTROL OF IN-CIRCUIT EMULATOR
XACTOR2 IN-CIRCUIT EMULATOR
• REAL-TIME EMULATION OF 4 LOGIC CELL ARRAYS
• READBACK OF INTERNAL DESIGN STATES
• ACCESS TO INTERNAL NODES DURING DEBUG
0010030 3

Figure 3. In-circuit Emulation Development System

During design debugging, therefore, it is useful to add circuitry which permits single-stepping the design clock(s). When a clock signal is generated externally, single-step circuitry may be added either within the LCA design, or externally. When a design uses the LCA's built-in crystal oscillator, as in the dual-speed counter example, the circuitry can be added internally, or the crystal can be replaced with an external clock source which can be single-stepped. In the DCOUNTER design, the crystal oscillator was temporarily replaced with a debounced switch to single step through several count sequences.

Figures 5 and 6 show the result of performing a readback of the design example between two consecutive clock pulses. The latches corresponding to each design element can be identified by comparing the readback figures with the "world view" of the LCA design in Figure 7. Note that in both cases the cursor has been placed over the CQ0CNT block, so the status line (third line from the bottom of the screen) shows the block's name, position, and current logic state.

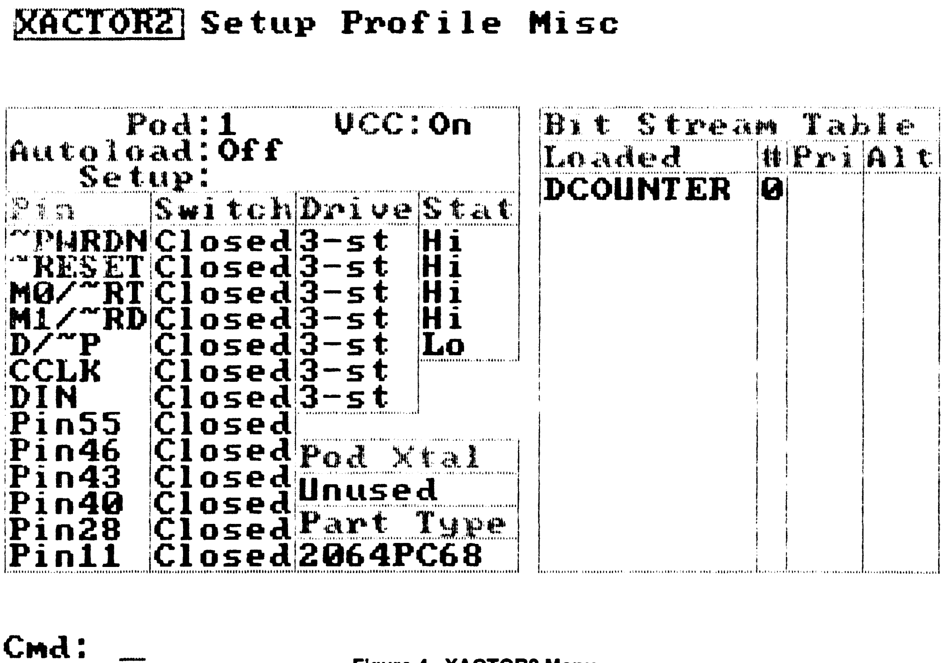
Although not specifically used in this design example, XACTOR has provisions for isolating and monitoring several I/O and control lines. The operation of these features is described in the XACTOR documentation.

SIMULATION

Once the function of a design has been verified in-circuit, designers can simulate critical paths to ensure that the design will function under worst-case process, voltage, and temperature conditions. Simulation is also a useful tool for determining the Logic Cell Array speed grade required for a particular application. The following discussion, while based upon a specific design example, is intended to illustrate the general mechanics of using P-SILOS.

Before simulating a design, the designer must create a simulation network description of the design, and a simulation setup file that defines the design's inputs and specifies which logic nodes to monitor and graph (Figure 8).

The SIMGEN program in the XACT Executive automatically creates a simulation network description of the design that is currently loaded into XACT. The network includes delay parameters corresponding to routing and logic delays, setup times, and hold times based upon the selected speed grade operating under worst-case temperature, voltage, and processing conditions. Unrouted LCA nets will be simulated with zero interconnect delay, permitting logic verification of unrouted logic paths.

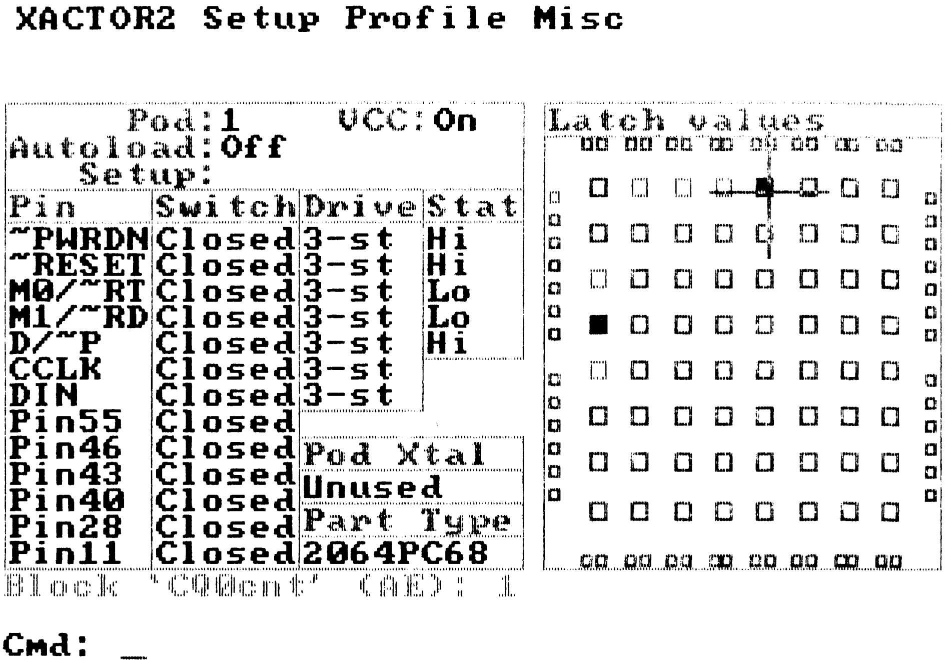


```
XACTOR2 Setup Profile Misc

     Pod:1     VCC:On          Bit Stream Table
Autoload:Off                   Loaded    |# |Pri |Alt
  Setup:                       DCOUNTER  |0 |    |
Pin    |Switch|Drive|Stat
~PWRDN |Closed|3-st |Hi
~RESET |Closed|3-st |Hi
M0/~RT |Closed|3-st |Hi
M1/~RD |Closed|3-st |Hi
D/~P   |Closed|3-st |Lo
CCLK   |Closed|3-st
DIN    |Closed|3-st
Pin55  |Closed
Pin46  |Closed|Pod Xtal
Pin43  |Closed|Unused
Pin40  |Closed|
Pin28  |Closed|Part Type
Pin11  |Closed|2064PC68

Cmd: _
```

Figure 4. XACTOR2 Menu

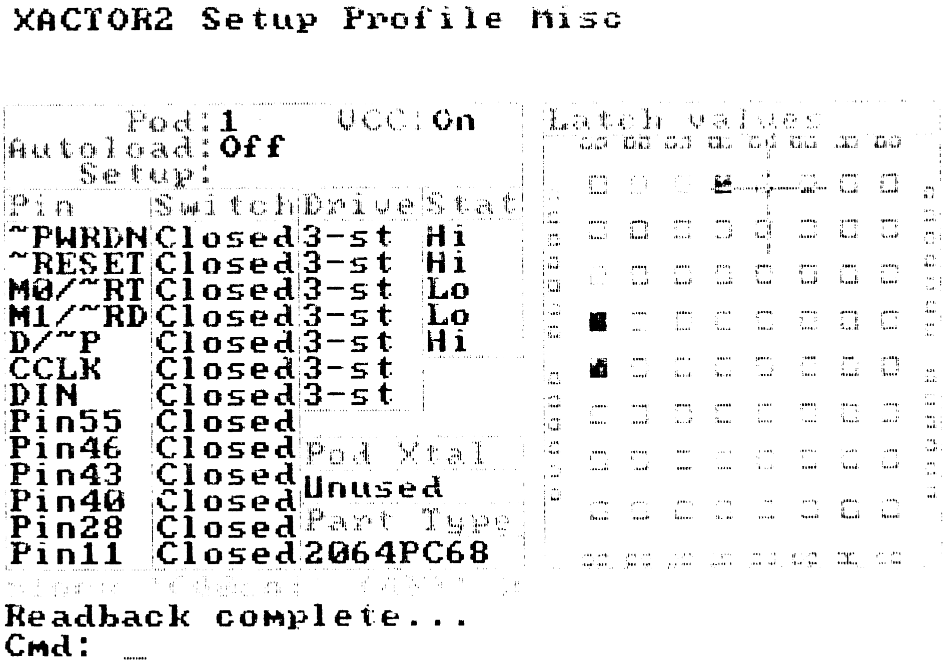


```
XACTOR2 Setup Profile Misc

     Pod:1      VCC:On
Autoload:Off
   Setup:
Pin      Switch Drive Stat
~PWRDN  Closed 3-st  Hi
~RESET  Closed 3-st  Hi
M0/~RT  Closed 3-st  Lo
M1/~RD  Closed 3-st  Lo
D/~P    Closed 3-st  Hi
CCLK    Closed 3-st
DIN     Closed 3-st
Pin55   Closed
Pin46   Closed Pod Xtal
Pin43   Closed Unused
Pin40   Closed Part Type
Pin28   Closed 2064PC68
Pin11   Closed
Block 'C00ent' (AE): 1

Cmd: _
```

Figure 5. LCA READBACK, Counter = 0001 Prescale Counter = 010



```
XACTOR2 Setup Profile Misc

     Pod:1      VCC:On
Autoload:Off
   Setup:
Pin      Switch Drive Stat
~PWRDN  Closed 3-st  Hi
~RESET  Closed 3-st  Hi
M0/~RT  Closed 3-st  Lo
M1/~RD  Closed 3-st  Lo
D/~P    Closed 3-st  Hi
CCLK    Closed 3-st
DIN     Closed 3-st
Pin55   Closed
Pin46   Closed Pod Xtal
Pin43   Closed Unused
Pin40   Closed Part Type
Pin28   Closed 2064PC68
Pin11   Closed
[illegible]
Readback complete...
Cmd: _
```

Figure 6. LCA READBACK, Counter = 0010 Prescale Counter = 011

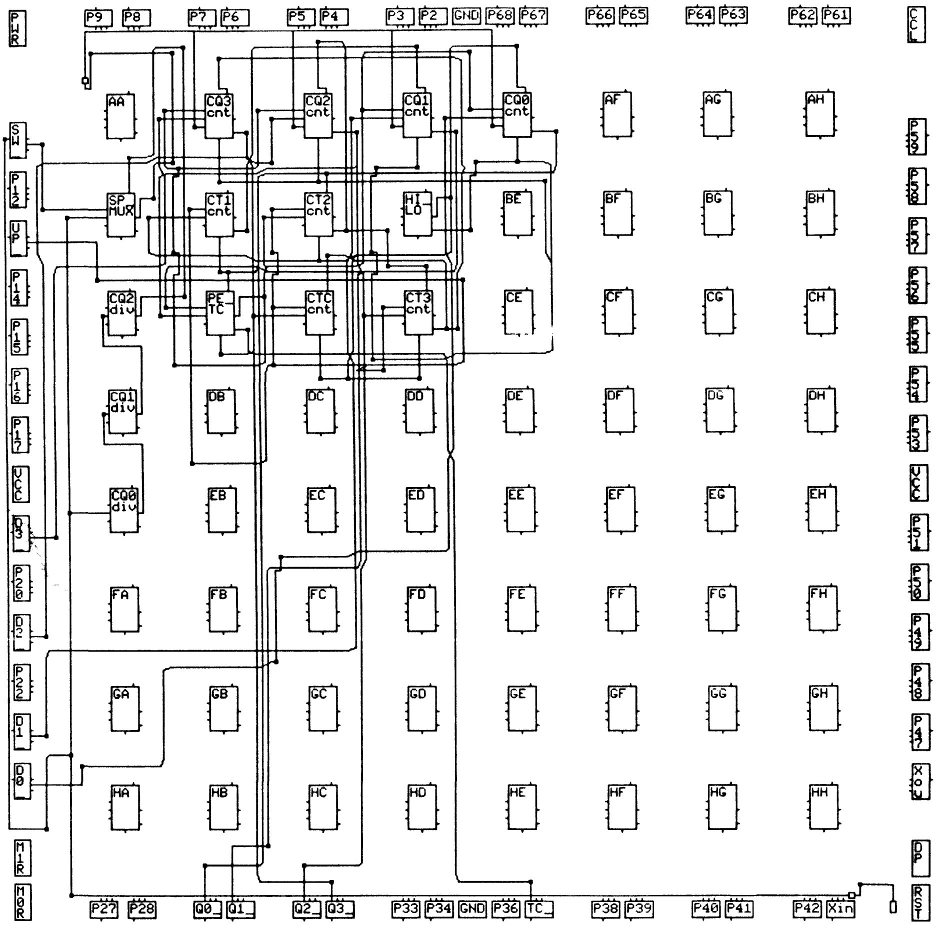


Figure 7. DCOUNTER.LCA Design Example

When executed while the LCA design DCOUNTER.LCA is loaded, SIMGEN writes a simulation network description to DCOUNTER.SIM. The first time the network file is created for a design, SIMGEN also creates a default simulation setup file (DCOUNTER.DAT). Since the setup file is typically customized with a text editor by the designer, SIMGEN will not overwrite the file on successive design iterations. It is therefore necessary for the designer to ensure that the .DAT file is kept up to date if subsequent design changes add or remove logic nodes. This is especially important to remember when using a hierarchical schematic capture package since the hierarchy names may change when new levels are added.

The default setup file created by SIMGEN for the DCOUNTER design example is shown in Figure 9. Lines must be less than 80 characters, and any text after a comment character ($) is ignored (such as the four line header which records the device type used to implement the design). The first command line is a command to P-SILOS to read the simulation network file DCOUNTER.SIM.

The following eight lines define each of the design's I/O block inputs as clocks with an initially unknown state. If a crystal oscillator is used, the crystal oscillator clock line (XCLKXTL in design example) is set to a default frequency of 500 KHz. A global reset input, GLOBAL RESET-, is always created and included which is defined to pulse LOW for 1 ns, corresponding to the Logic Cell Array's initial logic reset after configuration. The signal corresponds to the LCA's $\overline{\text{RESET}}$ pin.

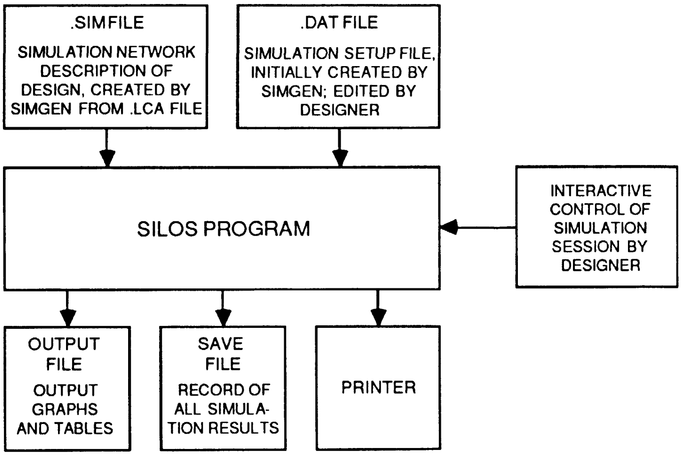
The .MONITOR instruction, which lists each I/O block input and output, initially assumes P-SILOS should monitor and display each input and output node during a simulation run. As described below, additional (internal) nodes can be added to the .MONITOR instruction so designers can monitor the state of any desired nodes as simulation progresses.

SIMULATION SETUP

Before beginning simulation, designers can use a text editor to modify the default setup file to specify initial input states.

In practice, the designer will need to assign to each input an initial logic state. A node's logic state is defined by its level and its strength, of which 12 combinations are possible. A node can be driven to a LOW level (0), a HIGH level (1), or can be unknown (*). A node's strength indicates its effective resistance, or how easily charge can be added to or removed from a node. Allowable logic strengths are supply level (S), driving level (D), resistive (R), or high-impedance (Z).

An I/O block configured as an input is typically specified by the P-SILOS user to be driven by a supply strength (S), since it is assumed that the output driving the LCA's


.SIMFILE
SIMULATION NETWORK DESCRIPTION OF DESIGN, CREATED BY SIMGEN FROM .LCA FILE
.DAT FILE
SIMULATION SETUP FILE, INITIALLY CREATED BY SIMGEN; EDITED BY DESIGNER
SILOS PROGRAM
INTERACTIVE CONTROL OF SIMULATION SESSION BY DESIGNER
OUTPUT FILE
OUTPUT GRAPHS AND TABLES
SAVE FILE
RECORD OF ALL SIMULATION RESULTS
PRINTER

0010030 8

Figure 8. Silos Inputs and Outputs

inputs can source or sink infinite charge. Internal logic signals and LCA outputs default to driving strengths (D).

An external output driving a bidirectional I/O block may be specified as resistive (R) strength so the LCA output, when enabled, will override the external source. If simulating output contention is desired, the external output can be alternately modeled as a driving (D) strength when active, and high impedance (Z) when inactive. If contention occurs between the two driving outputs, the output state will appear as D* during simulation. This simulation technique places the burden of establishing the strength of the external output upon the P-SILOS user, but may result in a more accurate simulation of the bidirectional signals.

The logic states P-SILOS can use are:

	Supply	Driving	Resistive	High-Z
High	S1	D1	R1	Z1
Unknown	S*	D*	R*	Z*
Low	S0	D0	R0	Z0

The example of Figure 10, which is an edited version of the default setup file, illustrates several useful P-SILOS instructions.

CLOCKS AND PATTERNS

For simulation purposes, a Logic Cell Array crystal oscillator is not modeled. Instead the designer directly defines the state of the crystal buffer's output (XCLKXTL in the design example). The command

```
XCLKXTL .CLK 0 S0 500 S1 1000 S0  .REP 0
```

is used to define the crystal oscillator output as a supply-strength 1 MHz clock which is low for 500 nanoseconds, high for 500 nanoseconds, and then repeats the pattern indefinitely from time 0. For inputs which are not regular, any number of transitions at arbitrary intervals may be specified with .CLK.

The .PATTERN instruction permits specifying input logic patterns as a group. This instruction is especially useful in conjunction with the .HEX or .OCT instructions which group nodes into a logic bus. For example the .HEX instruction in DCOUNTER.DAT specifies that the four parallel load inputs can be grouped together into a group called DBUS, whose levels can be specified with a hex digit. Setting DBUS to 5 within a .PATTERN instruction is equivalent is setting to D3=0, D2=1, D1=0, and D0=1. The .PATTERN instruction in DCOUNTER.DAT (Figure 10) defines a series of input transitions on DBUS, UP_DN (the count direction control), and SW_SPEED (the count speed select) which test the basic operation of the design.

MONITOR, TABLE, GRAPH

The setup file should generally include a .MONITOR instruction so that key node transitions are displayed during simulation runs.

While calculating simulation results, P-SILOS will display the changes on the nodes specified in the .MONITOR instruction. This is useful to monitor the results of simulation as they occur. For example, if a lengthy simulation sequence is being performed it is helpful to monitor the key outputs to ensure that the simulation is progressing as intended.

Each node in the table is displayed in one column. Semicolons can be inserted to create blank columns that can make the output display more readable. When the list of nodes is longer than 79 characters, additional lines can be used as long as the first non-blank character is a continuation character (+).

The .TABLE instruction defines the nodes that are recorded for later display with OUTPUT instructions. The syntax and display format for the .TABLE instruction is the same as for .MONITOR. Typically designers include all nodes of potential interest in the .TABLE instruction since these transitions are generally the ones scrutinized after simulation to verify design performance.

The nodes which P-SILOS will display in its graphical format are specified in the setup file's .GRAPH instruction. The syntax is the same as for .TABLE, and the node list is frequently the same, since one typically wants to view the same nodes in both tabular and graphic format. Several .GRAPH instructions may be used to produce separate graphs, although it is generally simpler to maintain one longer graph when graphs are to be printed and stored to disk.

STARTING SIMULATION

Once the setup file (.DAT) has been created, P-SILOS can be invoked from within XACT using the PROGRAM menu's SILOS command, or from DOS by typing SILOS. The simulation setup file DCOUNTER.DAT is loaded using the command INPUT DCOUNTER; the setup file then loads the simulation network DCOUNTER.SIM as specified in the file's INPUT command.

```
$
$ Simulation file for design 'DCOUNTER.LCA' type '2064PC68-50'
$ Created by XACT Ver. 1.23 at 20:52:26 SEP 1, 1986
$
!INPUT DCOUNTER.sim

$ INPUTS:
D0_IN         .CLK 0 S*
D1_IN         .CLK 0 S*
D2_IN         .CLK 0 S*
D3_IN         .CLK 0 S*
GLOBALRESET-  .CLK 0 S0 1 S1 $ Initial pulse to reset latches
SW_SPEED      .CLK 0 S*
UP_DN         .CLK 0 S*
XCLKXTL       .CLK 0 S0  1000 S1  2000 S0 .REP 0 $ Osc output pin

.MONITOR D0_IN D1_IN D2_IN D3_IN SW_SPEED UP_DN XCLKXTL ; Q0_OUT TC_OUT cnt_TC
+ Q1_OUT Q2_OUT Q3_OUT
```

Figure 9. Default DCOUNTER.DAT File Created by Simgen

```
$
$ Simulation file for design 'DCOUNTER.LCA' type '2064c68-50'
$ Created by XACT Ver. 1.23 at 22:01:31 SEP 1, 1986
$
!INPUT DCOUNTER.sim              $ READ SIMULATION NETWORK FILE

.HEX DBUS=D3_IN,D2_IN,D1_IN,D0_IN    $ DEFINE PARALLEL LOAD DATA 'BUS'


$ INPUTS:

.PATTERN      DBUS  UP_DN    SW_SPEED
0             5     1        1         $ LOAD-COUNT=5, UP, FAST
15200         5     1        0         $ SLOW DOWN
20200         5     0        0         $ REVERSE DIRECTION
35200         8     1        1         $ LOAD-COUNT=8, UP, FAST
.EOP

GLOBALRESET- .CLK 0 S0 1 S1 $ Initial pulse to reset latches

XCLKXTL      .CLK 0 S0 500 S1 1000 S0 .REP 0   $ 1 MHZ CLOCK


$  OUTPUT NODE DEFINITIONS TO WATCH WHILE SIMULATION RUNNING
.MONITOR UP_DN SPEED XCLKXTL XCLK CNTCLK ; DIV_Q2 DIV_Q1 DIV_Q0 ;
+ TC_OUT D3 D2 D1 D0 ;
+ Q3_OUT Q2_OUT Q1_OUT Q0_OUT cnt_TC

$  OUTPUT NODES TO SHOW WITH 'TYPE OUTPUT'
.TABLE   UP_DN SPEED XCLKXTL XCLK CNTCLK ; DIV_Q2 DIV_Q1 DIV_Q0 ;
+ TC_OUT D3 D2 D1 D0 ;
+ Q3_OUT Q2_OUT Q1_OUT Q0_OUT cnt_TC

$  OUTPUT NODES TO GRAPH WITH 'TYPE GRAPH'
.GRAPH   CNTCLK ; UP_DN ; SPEED ; TC_OUT ; D3 ; D2 ; D1 ; D0 ;
+  TC_OUT; Q3_OUT ; Q2_OUT ; Q1_OUT ; Q0_OUT ; CNTCLK

$  ADDITIONAL SILOS OPTIONS
.FORMAT .GFORMAT=1       $ ENABLE PRINTING GRAPHICS ON EPSON PRINTER
.FILE .SAVE=DSAVE        $ 'SAVE' COMMAND WILL SAVE SIMULATION IN DSAVE.SIM
.FILE .STORE=DCOUNTER.OUT    $  'STORE' WILL PUT OUTPUT IN DCOUNTER.OUT
```

Figure 10. DCOUNTER.DAT Edited by Designers to Initialize Simulation

If any errors or warnings are encountered while loading the .DAT and .SIM files, the command TYPE ERROR or TYPE WARNING will display a description of the problems P-SILOS encountered. If needed, the P-SILOS help facility is activated by typing HELP for a list of help topics, or HELP <keyword> for help on a specific topic.

A simulation session generally consists of running a simulation for some period of time based upon the inputs specified in the .DAT setup file, then proceeding interactively by changing the state of one or more inputs and continuing simulation.

The SIM command specifies how many nanoseconds of operation should be simulated without interruption, starting from the current logic state. In the DCOUNTER design, the command

```
SIM 40000
```

simulates for 40000 nanoseconds (or 40 clock cycles for a 1 MHz clock). P-SILOS also recognizes K (x1000) and M (x1000000) suffixes, so

```
SIM 40K
```

is equivalent. All nodes specified in the setup file's .MONITOR instruction will be output to the screen during simulation each time one of the nodes changes state (Figure 11).

If it is necessary to abort the simulation process, the standard DOS Control-Break keyboard input will stop the simulation and return to the P-SILOS prompt. Control-C will terminate P-SILOS completely, returning the user to DOS. Control-Break is generally the preferred method of aborting a simulation.

Input nodes can also be manually set to a state with the SET command. For example, to resume the above counter simulation, but counting in the opposite direction, the commands

```
SET UP_DN=0
SIM 5K
```

will set the up/down control line to 0 (DOWN) and resume simulation for another 5 microseconds.

TABLE OUTPUTS

The TYPE command is used to direct previously calculated simulation results to the PC screen, such as tables of simulation results specified by a .TABLE command, or graphs specified by a .GRAPH instruction.

After a simulation run, the command TYPE OUTPUT can be used to display the state sequences of the nodes defined in the .TABLE instruction. Users can select the time interval to display, and whether to display the nodes at fixed regular intervals or only when one of the nodes changes state.

```
        USXXC DDD TDDDD QQQQC
        PPCCN III C3210 3210N
        _ELLT VVV _     ____T
        DEKKC ___ O     OOOO_
        NDX L QQQ U     UUUUT
          T K 210 T     TTTTC
          L
TIME
   0    10000 000 10101 00000
 500    10100 000 10101 00000
 504    10110 000 10101 00000
 522    11110 000 10101 00000
 555    11111 000 10101 00000
 608    11111 000 10101 01000
 614    11111 000 10101 01010
 638    11111 000 00101 01010
1000    11011 000 00101 01010
1004    11001 000 00101 01010
1025    11001 001 00101 01010
1037    11000 001 00101 01010
1500    11100 001 00101 01010
1504    11110 001 00101 01010
1537    11111 001 00101 01010
1592    11111 001 00101 01110
1596    11111 001 00101 01100
2000    11011 001 00101 01100
2004    11001 001 00101 01100
2025    11001 000 00101 01100
2037    11000 000 00101 01100
2044    11000 010 00101 01100
2500    11100 010 00101 01100
2504    11110 010 00101 01100
2537    11111 010 00101 01100
2596    11111 010 00101 01110
3000    11011 010 00101 01110
3004    11001 010 00101 01110
3025    11001 011 00101 01110
3037    11000 011 00101 01110
3500    11100 011 00101 01110
3504    11110 011 00101 01110
3537    11111 011 00101 01110
3571    11111 011 00101 01111
3587    11111 011 00101 11111
3590    11111 011 00101 10111
3592    11111 011 00101 10011
3594    11111 011 00101 10010
3596    11111 011 00101 10000
4000    11011 011 00101 10000
4004    11001 011 00101 10000
4025    11001 010 00101 10000
4037    11000 010 00101 10000
4044    11000 000 00101 10000
4063    11000 100 00101 10000
4500    11100 100 00101 10000
4504    11110 100 00101 10000
4537    11111 100 00101 10000
4596    11111 100 00101 10010
5000    11011 100 00101 10010
```

Figure 11. .MONITOR Outputs During Simulation

A review of the state changes from time 20K to 25K in 250 ns intervals is displayed with the command

```
TYPE OUTPUT 20K TO 25K STEP 250
```

while the command

```
TYPE OUTPUT 20K TO 30K ON CHANGE
```

will display a table showing each node transition between time 20K and 30K.

Tabular results can also be printed or saved on disk as 132 column ASCII text with the PRINT OUTPUT or STORE OUTPUT command. If only a narrow (80 column) printer is available, NPRINT and NSTORE are used instead.

The STORE commands will append the output in the file STORE.OUT unless the .FILE .STORE instruction is used to specify a different disk file. For example, if the instruction

```
.FILE .STORE=DCOUNTER.OUT
```

has been executed (perhaps in the setup file), then the command

```
NSTORE OUTPUT 0 TO 6.5K ON CHANGE
```

will append to the file DCOUNTER.OUT a table listing each node change between time 0 and 6.5 microseconds for the nodes listed in the .TABLE instruction.

GRAPHICAL OUTPUTS

The TYPE GRAPH command will display the nodes (specified with the .GRAPH instruction) for any prior time interval. P-SILOS displays the graph using text characters so graphing is possible with either a monochrome or color graphics adapter card. The starting and ending times to graph are specified as in the command

```
TYPE GRAPH 0 TO 40K
```

which produces the graph of Figure 12. Once displayed the graph can be scrolled with the graph mode's UP, DOWN, RIGHT, and LEFT commands. There are zoom commands (IN and OUT) which permit displaying more detail or wider time frames as well. The TIME command is a command to specify a specific starting and ending time to be displayed on the graph. For example, once in the graph mode, typing

```
TIME 2750 4750
```

at the graph prompt will display the indicated 2 microsecond simulation period on the display. This command is especially useful for examining small delays while checking critical paths.

P-SILOS will also PRINT, NPRINT, STORE and NSTORE graphs, provided that printing graphs has been enabled with the instruction

```
.FORMAT .GFORMAT=1
```

which causes subsequent graphical output for each node to be displayed (or printed, or stored) on a single text line. This command is not required if an IBM compatible printer is used to print screen dumps using the PC's PRTSC (print screen) command.

As before, the STORE commands will append the graphs to the current output file.

BREAKPOINTS

Simulation breakpoints allow users to automatically stop a simulation run if specified conditions are met. Simulation normally continues until the last specified SIM time point (or until Control-Break is pressed). Users can also select node conditions that will terminate a simulation. For instance, the DCOUNTER design simulation can be halted in the event that the parallel load data is set to 0 and PARENA is enabled since this state will loop indefinitely. The instruction to set this breakpoint condition would be

```
.BREAK D3_IN=0 D2_IN=0 D1_IN=0 D0_IN=0 PARENA=1
```

SAVE/EXIT

The EXIT command will not automatically save the current simulation records before exiting to DOS. If it is necessary to later return to P-SILOS to continue from the current simulation point, the SAVE command can be used to save all previous simulation results before EXITing. When P-SILOS is executed again, the GET command will restore all simulation results from disk, and the user can continue the simulation session as well as review previous simulation results.

By default, SAVE and GET use files with various extensions appended to the default name SAVE (such as SAVE.SIM and SAVE.ERR). However, the instruction

```
.FILE .SAVE=CNTSAVE
```

would set the default SAVE filename to CNTSAVE (or any other specified filename). The next SAVE command will save all simulation results up to the current state in files named CNTSAVE. When P-SILOS is invoked later, the same instruction should be entered to set the default filename again before using the GET command; otherwise GET will attempt to read the default files (SAVE.SIM).

SUMMARY

A useful methodology for Logic Cell Array users is the combination of in-circuit emulation for real-time functional design verification, and simulation for critical path verification. The combination of these two verification tools can significantly improve design productivity.

The XACTOR in-circuit emulator simplifies initial development work by permitting direct control of up to four LCAs connected to a system under development. The readback facility is especially suitable to debugging complex state sequences, as it gives a snapshot of the internal operation of the Logic Cell Array during operation.

Simulation, while no longer required for exhaustive function verification, is recommended for examining critical path performance under worst-case conditions. P-SILOS simulates performance based upon actual delays derived from the placement and routing of designs; it permits interactive or batch control of input patterns and the simulation session.

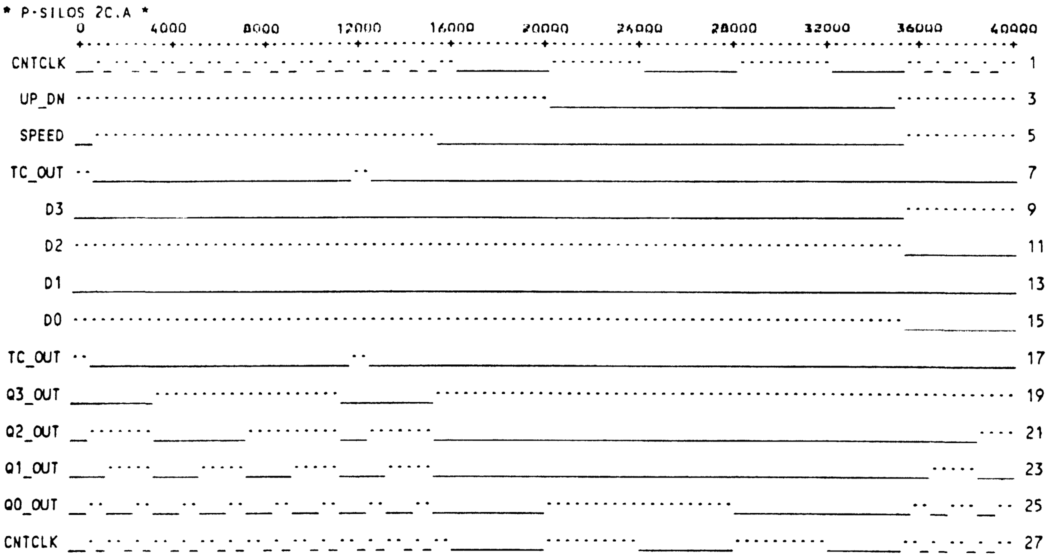


Figure 12. Simulation Graph of DCOUNTER Design Example

XC-DS21
DESIGN EDITOR

Product Brief

FEATURES

- Runs on an IBM® PC/XT™, PC/AT™ or compatible computer
- Complete basic system for design using 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 logical and physical configuration information
- Download cable to transfer configuration programs from PC to LCA in target system
- Compatible hardware and software options to enhance design productivity

GENERAL

The XACT™ Design Editor provides users with a complete design and development system for specification and implementation of designs using Xilinx 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.

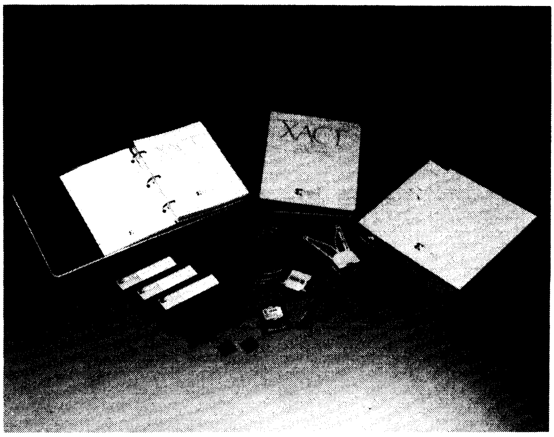
Interactive point-to-point timing delay calculation is provided for timing analysis and critical path determination. This ability enables the user to quickly identify and correct timing problems while the design is in progress.

Automatic generation of simulator 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.

Xilinx provides ongoing support for XACT users. For the first year, software updates are included. After that, the user may purchase the XC-SC21 Annual Support Agreement to continue to receive the latest software releases. XACT users also receive a technical bulletin which includes information about Logic Cell Arrays, software updates and hints for designers. In addition, Xilinx operates an electronic bulletin board to provide software enhancements and interactive factory support.



XACT MACRO LIBRARY

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 Oscillator	(1 + 2IOB)

PADS		IOBs
PIN	Input Pad	(1)
PINQ	Input Pad w/Storage	(1)
PIO	Input/Output Pad	(1)
PIOQ	Input/Output Pad w/Input Storage	(1)
PIOC	Input/Output Pad w/'Open Collector'	(1)
PIOQC	Input/Output Pad w/Input Storage, 'Open Collector'	(1)
POUT	Output Pad	(1)
POUTC	Output Pad w/'Open Collector'	(1)
POUTZ	Output Pad w/3-State Control	(1)
PREG	Output Pad w/Input Storage	(1)

LATCHES		CLBs
LD	Data Latch	(1)
LD-rd	Data Latch w/ResetDir	(1)
LD-sd	Data Latch w/SetDir	(1)
LD-srd	Data Latch w/SetDir, ResetDir	(1)
LDM	Data Latch w/2-Input Data Mux	(1)
LDM-rd	Data Latch w/2-Input Data Mux, ResetDir	(1)
LDM-sd	Data Latch w/2-Input Data Mux, SetDir	(1)

FLIP-FLOPS		CLBs
FD	D Flip-Flop	(1)
FDR	D Flip-Flop w/Reset	(1)
FDS	D Flip-Flop w/Set	(1)
FD-rd	D Flip-Flop w/ResetDir	(1)
FD-sd	D Flip-Flop w/SetDir	(1)
FD-srd	D Flip-Flop w/SetDir, ResetDir	(1)
FDC	D Flip-Flop w/ClkEna	(1)
FDCR	D Flip-Flop w/ClkEna, Reset	(1)
FDCS	D Flip-Flop w/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 w/Set Dominate	(1)
FRS	Set-Reset Flip-Flop w/Reset Dominate	(1)
FJK	J-K Flip-Flop	(1)
FJKS	J-K Flip-Flop w/Synchronous Set	(1)
FJK-rd	J-K (Set-Reset) Flip/Flop w/ResetDir	(1)
FJK-sd	J-K (Set-Reset) Flip/Flop w/SetDir	(1)
FJK-srd	J-K (Set-Reset) Flip/Flop w/SetDir, ResetDir	(1)
FT0	Self Toggle Flip-Flop	(1)
FT0R	Self Toggle Flip-Flop w/Reset	(1)
FT	Toggle Flip-Flop	(1)
FTP	Toggle Flip-Flop w/ParEna	(1)
FTP-rd	Toggle Flip-Flop w/ParEna, ResetDir	(1)
FTR	Toggle Flip-Flop w/Reset	(1)
FTS	Toggle Flip-Flop w/Set	(1)
FT2	2-Input Toggle Flip-Flop	(1)
FT2R	2-Input Toggle Flip-Flop w/Reset	(1)

DECODERS		CLBs
D2-4	1-of-4 Decoder	(2)
D2-4E	1-of-4 Decoder, w/Ena	(2)
74-139	1-of-4 Single Decoder w/Low Output, Ena	(4)
D3-8	1-of-8 Decoder	(5)
D3-8E	1-of-8 Decoder w/Ena	(6)
74-138	1-of-8 Decoder w/Enables, Low Output	(7)
74-42	1-of-10 Decoder w/Low Output	(8)

MULTIPLEXERS		CLBs
M3-1	3-to-1 Mux	(2)
M3-1E	3-to-1 Mux w/Ena	(2)
M4-1	4-to-1 Mux	(3)
M4-1E	4-to-1 Mux w/Ena	(3)
74-352	4-to-1 Mux w/Low Output, Ena	(3)
M8-1	8-to-1 Mux	(7)
M8-1E	8-to-1 Mux w/Ena	(7)
74-151	8-to-1 Mux w/Ena, Complementary Outputs	(7)
74-152	8-to-1 Mux w/Low Output	(7)

REGISTERS

Data Registers

		CLBs
RD4	4-Bit Data Register	(4)
RD8	8-Bit Data Register	(8)
RD8CR	8-Bit Data Register w/ClkEna, Reset	(8)

Serial to Parallel

		CLBs
RS4	4-Bit Shift Register	(4)
74-195	4-Bit Serial to Parallel Shift Register w/ParEna, Reset	(5)
74-194	4-Bit Bi-Directional Shift Register w/ClkEna,ParEna, ResetDir	(12)
RS8	8-Bit Shift Register	(8)
RS8CR	8-Bit Shift Register w/ClkEna, Reset	(8)
RS8PR	8-Bit Shift Register w/ParEna, Reset	(8)
RS8R	8-Bit Shift Register w/Reset	(8)
74-164	8-Bit Serial to Parallel Shift Register w/ResetDir	(8)

COUNTERS

Modulo 2

		CLBs
C2BCR	1-Bit Binary Counters w/ClkEna, Reset	(1)
C2BC-rd	1-Bit Binary Counters w/ClkEna, ResetDir	(1)
C2BP	1-Bit Binary Counters w/ParEna	(1)
C2BR	1-Bit Binary Counters w/Reset	(1)
C2B-rd	1-Bit Binary Counters w/ResetDir	(1)

Modulo 4

C4BCP	2-Bit Binary Counters w/ClkEna, ParEna	(3)
C4BCR	2-Bit Binary Counters w/ClkEna, Reset	(2)
C4BC-rd	2-Bit Binary Counters w/ClkEna, ResetDir	(2)
C4JCR	2-Bit Johnson Counters w/ClkEna, Reset	(2)

Modulo 6

C6JCR	3-Bit Johnson Counter w/ClkEna, Reset	(3)

Modulo 8

C8BCP	3-Bit Binary Counters w/ClkEna, ParEna	(5)
C8BCR	3-Bit Binary Counters w/ClkEna, Reset	(4)
C8BC-rd	3-Bit Binary Counters w/ClkEna, ResetDir	(4)
C8JCR	3-Bit Johnson Counters w/ClkEna, Reset	(4)

Modulo 10

C10BC-rd	4-Bit BCD Counter w/ClkEna, ResetDir	(4)
C10BCP-rd	4-Bit BCD Counter w/ClkEna, ParEna, ResetDir	(7)
74-160	4-Bit BCD Counter w/ClkEna, ParEna, ResetDir	(8)
C10BP-rd	4-Bit BCD Counter w/ParEna, ResetDir	(6)
C10JCR	5-Bit Johnson Counter w/ClkEna, Reset	(5)

Modulo 12

C12JCR	6-Bit Johnson Counter w/ClkEna, Reset	(6)

Modulo 16

C16BA-rd	4-Bit Binary Ripple Counter w/ResetDir	(4)
C16BC-rd	4-Bit Binary Counter w/ClkEna, ResetDir	(4)
C16BCPR	4-Bit Binary Counter w/ClkEna, ParEna, Reset	(10)
C16BCP-rd	4-Bit Binary Counter w/ClkEna, ParEna, ResetDir	(6)
74-161	4-Bit Binary Counter w/ResetDir	(8)
C16BP-rd	4-Bit Binary Counter w/ParEna, ResetDir	(5)
C16BUD-rd	4-Bit Binary Up-Down Counter w/ParEna, ResetDir	(8)
C16JCR	8-Bit Johnson Counter w/ClkEna, Rese	(8)

Modulo 256

C256FC-rd	8-Bit Modulo 256 Feedback Shift Register\ w/ClkEna, ResetDir	(9)

XACT EXECUTIVE COMMAND SUMMARY

PROGRAM MENU

Quit
DRC
MakeBits
Simgen
Xprint
EditLCA
CONVERT
MakeProm
Macgen
Dos

DESIGNS MENU

Directory
Part
Read
Save
Design
Speed
File

PROFILE MENU

Settings
ReadProfile
Mouse
Printer
SaveProfile
Cursor
Keydef
Execute

XACT DESIGN EDITOR COMMAND SUMMARY

NET MENU

QueryNet
NameNet
Route
DelNet
Hilight
FlagNet
AddNet
EditNet
UnRoute
JoinNet
UnHilight

PIN MENU

AddPin
ClearPin
MovePin
SwapSig
RoutePin
SwapPin
DelPin

BLOCK MENU

QueryBlk
MoveBlk
SwapBlk
EndBlk
ClearBlk
NameBlk
CopyBlk
EditBlk
DelBlk

CONFIG MENU

Base
Equate
EditEq
Cdata
Config
Clear
Order

SCREEN MENU

Show
Print
Find
Swltch
Cursor
Redraw
QueryGrid

MISC MENU

Exit
File
Report
Execute
Paste
Macro
Delay
Speed
Save
Dos
Cut
CutMacro
DRC

PROFILE MENU

Settings
ReadProfile
Cursor
Keydef
Printer
SaveProfile
Show
AutoTime
Mouse
AutoRoute

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
- 10MB Hard Disk
- IBM or compatible Color Graphics Adapter and Display
- 1 Serial Interface Port
- 1 Parallel Interface Port
- Mouse Systems™, Microsoft® or compatible mouse

ORDERING INFORMATION

Further information is available from your local Hamilton/Avnet or other Xilinx distributor sales office or by contacting the nearest Xilinx sales representative.

Part Number	Description
XC-DS21	XACT Design Editor System
XC-SC21	XACT Annual Support Agreement

XC-DS22
P-SILOS™ SIMULATOR

Product Brief

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 designs or design 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 grade 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 the debugging easy for both the circuit function and timing.

SYSTEM REQUIRMENTS

Minimum System Configuration

IBM PC/XT, PC/AT or compatible computer with:

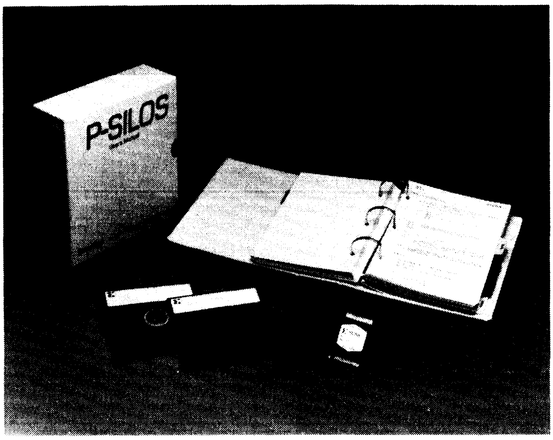
- MS-DOS™ 2.1 or higher
- 640K Bytes RAM
- 1 Diskette Drive
- 10MB Hard Disk
- 1 Parallel Interface Port

Refer to the DS21 Design Editor Product Brief for additional equipment required for systems which will also run the XACT Design Editor

ORDERING INFORMATION

Further information is available from your local Hamilton/Avnet or other Xilinx distributor sales office, or by contacting the nearest Xilinx sales representative.

Part Number	Description
XC-DS22	P-SILOS Simulator



XC-DS23 AUTOMATIC PLACEMENT AND ROUTING PROGRAM

Product Brief

FEATURES

- Automatic placement and routing of logic to minimize design cycle time
- User control over placement of logic blocks
- User specification of critical paths
- Netlist inputs from either schematic capture or XACT™
- May be used in conjunction with schematic capture or with the XACT Design Editor
- Runs on IBM® PC/XT™, PC/AT™ or compatible personal computer

GENERAL

The Automatic Placement and Routing program enhances the productivity of designers using Logic Cell 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 developed incrementally can also take advantage of Automatic Placement and Routing. Partial Logic Cell Array 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.

Xilinx provides ongoing support for users of the Automatic Placement and Routing program. For the first year, software updates are included. After that, the user may purchase the XC-SC23 Annual Support Agreement to continue to receive the latest software releases.

SYSTEM REQUIRMENTS

Minimum System Configuration

IBM PC/XT, PC/AT or compatible computer with:

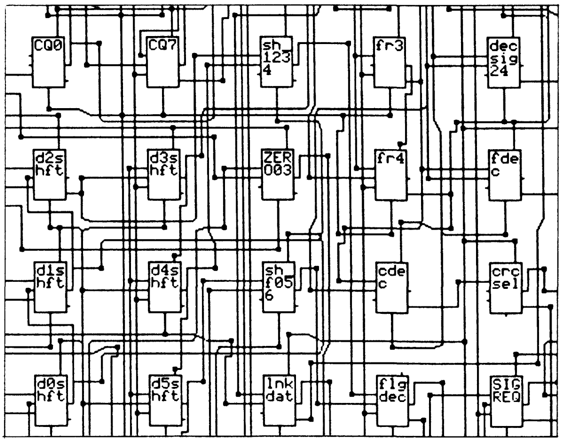
- MS-DOS™ 2.1 or higher
- 640K Bytes RAM
- 1 Diskette Drive
- 10MB Hard Disk
- 1 Parallel Interface Port

Refer to the DS21 Design Editor Product Brief for additional equipment required for systems which will also run the XACT Design Editor

ORDERING INFORMATION

Further information is available from your local Hamilton/ Avnet or other Xilinx distributor sales office, or by contacting the nearest Xilinx sales representative.

Part Number	Description
XC-DS23	Xilinx Automatic Placement and Routing Program
XC-SC23	Xilinx Automatic Placement and Routing Program Annual Support Agreement



XC-DS24, XC-DS26, XC-DS27 XACTOR™ IN-CIRCUIT EMULATOR

Product Brief

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.

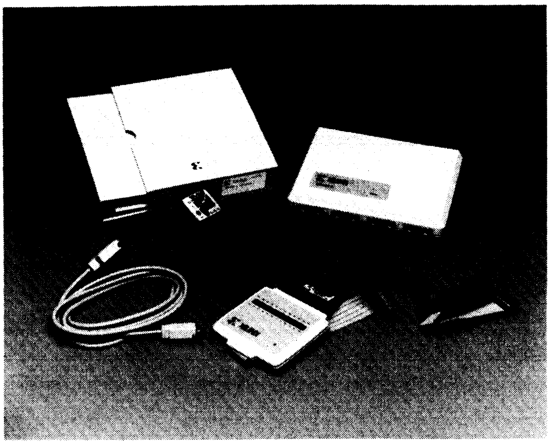
The emulation system is composed of a microcomputer-based controller, and from one to four universal emulation pods each with an emulation header. One pod and header is included with the system. The controller is connected to the host PC through a serial port and provides local storage of configuration 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 PODS

Additional pods may be connected to the XACTOR in-circuit emulator controller, up to a maximum of four pods per controller. Pod headers 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 with:

- MS-DOS™ 2.1 or higher
- 640K Bytes RAM
- 1 Diskette Drive
- 10MB Hard Disk
- IBM Color Graphics Adapter and Display
- 2 Serial Interface Ports
- 1 Parallel Interface Port
- Mouse Systems™, Microsoft or compatible mouse

ORDERING INFORMATION

Further information is available from your local Hamilton/Avnet or other Xilinx distributor sales office, or by contacting the nearest Xilinx sales representative.

Part Number	Description
XC-DS24	XACTOR In-Circuit Emulator with 1 pod and PC68 header
XC-DS26	Universal Emulation Pod
XC-DS27-PD48	Emulation Header for 48 pin DIP
XC-DS27-PC68	Emulation Header for 68 pin PLCC
XC-DS27-PG68	Emulation Header for 68 pin PGA
XC-DS27-PC84	Emulation Header for 84 pin PLCC

XC-DS31 FUTURENET DASH™ SCHEMATIC DESIGNER LIBRARY

Product Brief

FEATURES

- Design entry via the FutureNet DASH™ Schematic Designer
- Macro library of over 100 standard logic family equivalents derived from the XACT™ Macro Library
- Library of logic symbols including all two-input, three-input and four-input AND, OR and XOR gates plus storage, input/output and clock elements
- User control for flagging critical paths for the XC-DS23 Automatic Placement and Routing Program
- Automatic partitioning and conversion of schematic drawings to a Xilinx Logic Cell™ Array design file
- Output compatibility with XACT Design Editor and the Automatic Placement and Routing Program
- Runs on an IBM® PC/XT™, PC/AT™ or compatible personal computer

GENERAL

Schematic entry and automatic partitioning of Logic Cell Array designs shortens product development times. Complex designs can be specified schematically and quickly implemented for in-circuit design verification.

Xilinx's FutureNet DASH Schematic Designer Library provides the symbol library and conversion utility to permit designers to enter Logic Cell Array designs with the FutureNet DASH Schematic Designer. The Xilinx library provides the logic, I/O, and macro symbols to be used in the schematic. A Xilinx conversion utility automatically partitions the schematic into a Logic Cell Array design.

Once partitioned, the design may be placed and routed with the XC-DS23 Automatic Placement and Routing Program or with XACT. The Xilinx symbol library includes symbols to flag critical data and clock signals which the Automatic Placement and Routing Program uses to prioritize those signals for minimum delay.

Xilinx provides ongoing support for users of the FutureNet DASH Schematic Designer Library. For the first year, software updates are included. After that, the user may purchase the XC-SC31 Annual Support Agreement to continue to receive the latest software releases.

SYSTEM REQUIREMENTS

Minimum System Configuration
IBM PC/AT, PC/XT or compatible computer with:

- FutureNet DASH-2 or later and associated hardware including mouse, Enhanced Graphics Adapter and Display
- MS-DOS™ 2.1 or higher
- 640K Bytes RAM
- 1 Diskette drive
- 10MB hard disk

Refer to the DS21 Design Editor Product Brief for additional equipment required for systems which will also run the XACT Design Editor.

ORDERING INFORMATION

Further information is available from your local Hamilton/Avnet or other Xilinx distributor sales office, or by contacting the nearest Xilinx sales representative.

Part Number	Description
XC-DS31	FutureNet DASH Schematic Design Library
XC-SC31	FutureNet DASH Schematic Design Library Annual Support Agreement

EK01 Logic Cell™ Array Evaluation Kit

Product Brief

The Xilinx Logic Cell Arrray is a high-performance CMOS user-programmable gate array. The Xilinx 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 Xilinx 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.

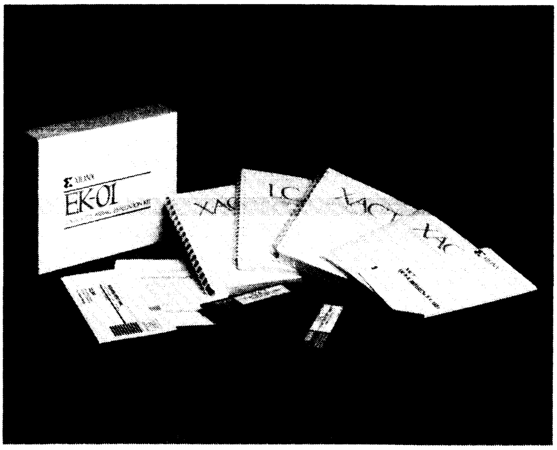
Functional defintion of Configurable Logic Blocks (CLBs), and their internal routing, I/O Block (IOB) definitions, and interconnection are all done within an integrated graphics oriented system. Interactive placement and automatic routing of logic and I/O elements are accomplished quickly and easily via an easy-to-learn user interface.

Designs are captured with a graphics-oriented design editor, using either a mouse or keyboard entry or driven from command files. User functions are specified in terms of CLB definitions and interconnections. Standard logic functlions 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 REQUIRMENTS

Minimum System Configuration
IBM PC/XT, PC/AT or compatible

- MS-DOS™ 2.1 or higher
- 640K Bytes of RAM
- 1 Diskette drive
- 10MB Hard disk
- IBM or compatible Color Graphics Adapter and Display
- 1 Serial interface port
- Mouse Systems™, Microsoft® or compatible mouse

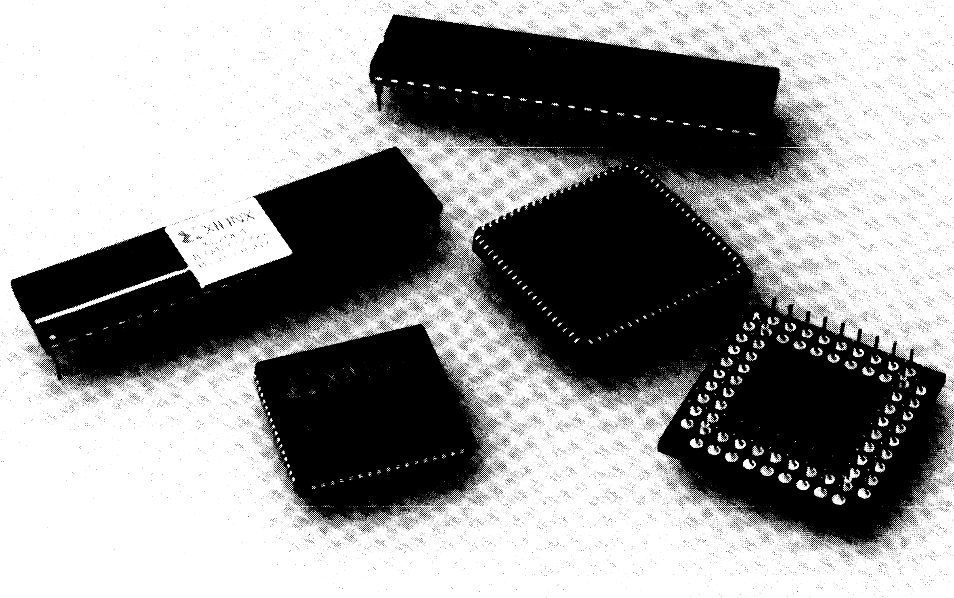
ORDERING INFORMATION

Further information is available from your local Hamilton/ Avnet or other Xilinx distributor sales office, or by contacting the nearest Xilinx sales representative.

Part Number	Description
XC-EK01	Logic Cell Array Evaluation Kit

XILINX

The Programmable Gate Array Company



XILINX

Table of Contents

The Programmable Gate Array

Using Programmable Gate Arrays

Programmable Gate Array Design Examples

Programmable Gate Array Development Systems

Appendices

Technical References

Article Reprints

Application-Specific ICs, Relying on RAM, Implement Almost Any Logic Function, Electronic Design, Oct. 31, 1985.

Printer Buffer Proves RAM-based Logic's Strength and Versatility, Electronic Design, Nov. 14, 1985.

Articles to Reference

Programmable Logic Devices—1986 Technology Forcast, Electronic Design, Jan. 9, 1986.

Dynamically Configurable LCAs Become a Reality, Electronics Weekly (England), Jan. 15, 1986.

Field-Programmable Logic: A New Market Force, Electronics, Jan. 22, 1986.

PLDs Slow Advance of Gate Arrays in Low-End Designs, Computer Design, Feb. 1, 1986.

Logic Cell Arrays: High Density, User-Programmable ASICs, Electronic Component News, March 1986.

Application-Specific ICs, Electronic Component News, April 1986.

Electro/86 ASIC Issue, Electronic Design, May 1, 1986.

Logic Cell™ Arrays; Ram-Programmable Logic Devices, Electronic Engineering Times, May 12, 1986.

Semicustom IC Offers New Possibilities for Software Protection, EDN, June 12, 1986.

ASICs: Take your Pick, Digital Design, June 1986.

Programmable Logic Declares War on Gate Arrays, Digital Design, July 1986.

The Logic Cell™ Array: Birth of a new Technology, Nikkei Electronics, Sep. 1986.

New Architectures Mean New Applications for PLDs, Electronic Engineering Times, Sep. 29, 1986.

In-Circuit Emulation for ASICs, VLSI Systems Design, Oct. 1986.

Conference Papers

Architecture Enhancements with Logic Cell™ Arrays, Southcon '86.

Using Dynamic Reconfigurable Logic in the XC2064, Electro '86.

A User Programmable Reconfigurable Logic Array, Custom Integrated Circuits Conference

Logic Cell Arrays—A Better ASIC Approach, Midcon '86.

Complete ASIC: The Logic Cell™ Array, Wescon '86.

Glossary

Active High – A high-voltage active sense.

Active Low – A low-voltage active sense. Signals which are active low have their names preceded by a tilde (~) or an overbar.

Active Sense – The voltage level (high or low) associated with the Active State (logical "1").

Active Edge – A signal transition upon which actions are dependent; a low-to-high or high-to-low transition used to initiate an action.

ASIC – Application Specific Integrated Circuit. An integrated circuit tailored for a specific use by a single IC customer.

Assert – To cause a signal to change from its inactive to its active state.

Asynchronous – Not synchronized to a clock signal. An asynchronous input affects its circuit output directly.

Bidis – A set of bidirectional buffers located in the LCA **general** interconnect and programmed automatically by the development system to provide signal buffering.

Bitstream – The object form of an LCA configuration program, organized serially and including length count and other control information.

Buffer – (1) A structure for intermediate data storage. (2) A device for isolating a signal in a circuit.

CLB – See Configurable Logic Block.

Clear – To force to a logical "0". See also Reset.

Combinatorial – A logic operation who's output is a direct function of a set of input variables, i.e., not dependent on a timing signal.

Configurable Logic Block (CLB) – A subunit of an LCA that contains configurable combinational-logic and data-storage circuitry.

Configuration Program – The data required by an LCA to determine the user-specified functions of the CLBs, the IOBs and the interconnection networks.

Configuration Logic – The circuitry of the LCA that automatically recognizes, receives, and stores the configuration program and signals the completion of the configuration process.

Configuration Mode – The mode used to load the configuration program into an LCA. The configuration mode is determined by the states of inputs M0, M1 and M2 at the conclusion of initialization. See also Peripheral Mode, Master Mode and Slave Mode.

Daisy Chain – Several devices connected in such a way that configuration program data move serially from one device to the next.

DASH™ – A FutureNet® schematic software program.

De-assert – To cause a signal to change from its active to its inactive state.

Direct Interconnect – Dedicated interconnect that can directly connect adjacent CLBs, IOBs and outputs. This type of interconnect provides the shortest propagation delay between such points and uses minimal interconnection networks.

Enable – To allow a circuit to respond to an input. For example, a clock enable signal allows a circuit to respond to its clock input.

Flip-Flop – A storage device whose output assumes a high or low state according to the states of its inputs and is synchronized to a clock transition.

FutureNet™ – A schematic capture program which may be used for schematic definition of a logic design when used with the Xilinx Auto Place and Route Program.

Gate Array – An integrated circuit which uses factory programmed metal interconnections to define the logic function.

General Interconnect – Horizontal and vertical metal segments joining LCA switching matrices.

Input/Output Block (IOB) – A subunit of an LCA that can be configured to connect the internal circuitry to an external package pin. It contains elements for input-data capture and for three-state output.

IOB – See Input/Output Block

Johnson Counter – A synchronous counter implemented as a shift register whose input is the inverse of the shifted-out end bit.

Latch – A logic device which transfers the data of its input to its output when load enable is active, and retains its value when load enable is inactive.

LCA – See Logic Cell Array

Length Count – Part of the configuration program 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.

Linear Feedback Shift Register (LFSR) – A synchronous counter that is a shift register whose input bit is computed by XORing several bits of the current state.

Logic Cell™ Array – A user-programmable gate array which can be configured (programmed) to perform a range of logic functions.

Long Lines – Interconnect that routes a signal to several destinations in an LCA. This type of interconnect runs the full length or width of an LCA and is often used for clock signals.

Macro – A file containing XACT™ editor commands which may be executed by name and given a set of parameters for netnames, locations etc. in order to generate a logic element from a library.

Master Mode – A configuration mode in which the LCA receive parallel (byte-wide) configuration data from an external memory. The LCA generates read-addresses and automatically serializes (internally) the data for internal storage.

PAL® – A Monolithic Memories fusible link integrated circuit which implements logic using sums of programmable product terms.

Peripheral Mode – A configuration mode in which the LCA acts as a peripheral device. An external device, such as a processor, loads the configuration data bits serially into the LCA.

PIP – Programmable interconnect point, a configuration memory-controlled pass transistor used to control programmable interconnections in a Logic Cell Array.

PLCC – Plastic Leaded Chip Carrier. An integrated circuit package with J-bend leads suitable for socket or surface mounting.

Power-Down State – An idle current condition of the LCA in which its 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 (0010) that signals the start of the configuration program for LCAs.

Product – The result of a logical AND of two or more variables, i.e., logic inputs.

Readback – To cause the LCA to output its configuration information.

Re-configure – To program an LCA that already contains a configuration program with a new configuration program while power is continually supplied.

Register – A group of related latches or flip-flops used to store and sometimes to manipulate data.

Reset – (1) To initialize all storage elements of a device to a starting condition. (2) To force one or more storage elements to logical "0". This may be a synchronous (clocked) or an asynchronous (direct) operation.

Set – To force one or more storage elements to logical "1". This may be a synchronous (clocked) or an asynchronous (direct) operation.

Silos® – An optional logic and timing simulator developed by Simucad Corp. and supported by XACT for use in design verification of Logic Cell Array designs.

Slave Mode – A configuration mode in which the LCA receives a serial configuration program and all control signals from another device, frequently another LCA.

State – The condition of one or a set of flip-flops.

State Machine – A set of flip-flops whose next state and next outputs are functions of its current state and a set of inputs.

Storage Element – An IOB or CLB portion that can store data. The storage element of a CLB can be programmed to act as a latch or a flip-flop.

Sum of products – The result of a logical OR of two or more logical AND operations.

Synchronous – Restricting output changes to those initiated by a transition of a timing signal.

Three-State – (1) Able to function as an output, bidirectional connection or no connection (high impedance). (2) The high-impedance state.

Toggle – To change to the opposite state (e.g. active to inactive).

Toggle rate – The maximum clock frequency at which a flip-flop storage element will toggle properly.

XACT™ – Xilinx Advanced CAD Technology. A set of computer programs that lets a designer specify, develop, and debug the configuration of an LCA using interactive computer graphics.

XACTOR™ – An optional in-circuit emulator used for real-time functional verification of LCA designs.

XILINX

Sales Offices

SALES OFFICES

XILINX, INC.
2069 Hamilton Avenue
San Jose, CA 95125
(408) 559-7778
TWX: 510-600-8750
FAX: 408-559-7114

XILINX, INC.
20 Mall Road, Suite 469
Burlington, MA 01803
(617) 229-7799
TWX: 510-601-2067
FAX: 617-273-0228

XILINX, INC.
7 Great Valley Parkway
Suite 202
Malvern, PA 19335
(215) 251-6863

XILINX, INC.
300 N. Martingale Road
Suite 500
Schaumburg, IL 60173
(312) 490-1972
TWX: 910-997-0078
FAX: 312-490-1985

DOMESTIC SALES

ALABAMA

Technology Marketing Associates, Inc.
3315 So. Memorial Pkwy.
Bldg. 100
P.O. Box 4112
Huntsville, AL 35801
(205) 883-7893
TWX: 510-101-1668
FAX: 205-882-6162

ARIZONA

Quatra Associates
4645 S. Lakeshore Dr.,
Suite 1
Tempe, AZ 85282
(602) 820-7050
TWX: 910-950-1153
FAX: 602-820-7054

ARKANSAS

Bonser-Philhower Sales
4614 S. Knoxville Avenue
Tulsa, OK 74135
(918) 744-9964
TWX: 510-600-5274

CALIFORNIA

Quad Rep Central
24007 Venture Blvd.,
Suite 134
Calabassas, CA 91302
(818) 887-3711
TLX: 858201
FAX: 818-887-4219

Quad Rep South
18004 Skypark Circle,
Suite 200
Irvine, CA 92714
(714) 261-8141
TWX: 910-997-3655
FAX: 714-261-6706

SR Electronics
7585 Ronson Rd., Suite 100
San Diego, CA 92111
(619) 560-8330
FAX: 619-560-9156

Norcomp
3350 Scott Blvd., Suite 24
Santa Clara, CA 95054
(408) 727-7707
TWX: 510-600-1477
FAX: 408-986-1947

COLORADO

Front Range Marketing
3100 Arapahoe Rd.,
Suite 404
Boulder, CO 80303
(303) 443-4780
TWX: 910-940-3442
FAX: 303-447-0371

CONNECTICUT

Lindco Associates, Inc.
10 Main St. South, Suite 20
Southbury, CT 06488
(203) 264-7200
TWX: 4991204

DELAWARE

Micro Comp, Inc.
1421 S. Caton Avenue
Baltimore, MD 21227
(301) 644-5700
TWX: 510-600-9460
FAX: 301-644-5707

FLORIDA

Technology Marketing Associates, Inc.
8000 Orange Ave.,
Suite 100
Orlando, FL 32809
(305) 857-3760
TWX: 510-600-4721
FAX: 305-857-6412

Technology Marketing Associates, Inc.
1280 S.W. 36th Ave.,
Suite 201
Pompano Beach, FL 33069
(305) 977-9006
TWX: 510-601-0120
FAX: 305-977-9044

Technology Marketing Associates, Inc.
1300 S. Harbor City Blvd.
Suite 8
Melbourne, FL 32901
(305) 676-3776
TWX: 510 600-3742
FAX: 305-676-4231

Technology Marketing Associates, Inc.
12360 66th St. No., Suite M
Largo, FL 33543
(813) 536-3796
TWX: 510-600-4463
FAX: 813-539-7082

GEORGIA

Technology Marketing Associates, Inc.
175 W. Wieuca Road N.E.
Suite 209
Atlanta, GA 30342
(404) 257-0374
TWX: 510-600-2444
FAX: 404-843-8705

DAHO (Southwest)

Thorson Company Northwest
12301 N.E. 10th Place
Bellevue, WA 98005
(206) 455-9180

ILLINOIS

Beta Technology Sales, Inc.
501 Mitchell Road
Glendale Heights, IL 60139
(312) 790-9868
TWX: 62885853

IOWA

Advanced Technical Sales
375 Collins Road N.E.
Cedar Rapids, IA 52402
(319) 365-3150
FAX: 319-393-7258

KANSAS

Advanced Technical Sales
9550 E. Lincoln #609
Wichita, KS 67207
(316) 682-2769
FAX: 316-689-8971

Advanced Technical Sales
601 N. Mur-Len, Suite 8
Olathe, KS 66062
(913) 782-8702
FAX: 913-782-8641
TWX: 9103506002

KENTUCKY

Bear Marketing, Inc.
1563 East Dorothy Lane
Suite 104
Kettering, OH 45429
(513) 299-5877

LOUISIANA (Northern)

Bonser-Philhower Sales
689 W. Renner Rd., Suite C
Richardson, TX 75080
(214) 234-8483
TWX: 910-867-4752
FAX: 214-437-0897

LOUISIANA (Southern)

Bonser-Philhower Sales
11321 Richmond, Suite 100A
Houston, TX 77082
(713) 531-4144
TWX: 910-350-3451

MAINE

Mill-Bern Associates, Inc.
120 Cambridge St, Suite 8
Burlington, MA 01803
(617) 273-1313
TWX: 710-332-0077
FAX: 617-229-7797

MARYLAND

Micro Comp, Inc.
1421 S. Caton Avenue
Baltimore, MD 21227
(301) 644-5700
TWX: 301-644- 570719
FAX: 301-644-5707

MASSACHUSETTS

Mill-Bern Associates, Inc.
120 Cambridge St, Suite 8
Burlington, MA 01803
(617) 273-1313
TWX: 710-332-0077
FAX: 617-229-7797

MICHIGAN

A.P. Associates
9903 Webber
P.O. Box 777
Brighton, MI 48116
(313) 229-6550
TWX: 816-287-310

MINNESOTA

Com-Tek
6525 City West Parkway
Eden Prairie, MN 55344
(612) 941-7181
TWX: 310-431-0122
FAX: 612-941-4322

MISSOURI

Advanced Technical Sales
1810 Craig Road, Suite 125
St. Louis, MO 36146
(314) 878-2921
FAX: 314-878-1994

NEVADA

Norcomp
(Excluding Clark County)
3350 Scott Blvd., Suite 24
Santa Clara, CA 95054
(408) 727-7707
TWX: 510-600-1477

Quatra Associates
(Clark County)
4645 S. Lakeshore Dr.,
Suite 1
Tempe, AZ 85282
(918) 820-7050

NEW HAMPSHIRE

Mill-Bern Associates, Inc.
120 Cambridge St., Suite 8
Burlington, MA 01803
(617) 273-1313
TWX: 710-332-0077
FAX: 617-229-7797

NEW JERSEY (Northern)

Parallax
734 Walt Whitman Road
Mellville, NY 11747
(516) 351-1000

NEW JERSEY (Southern)

Delta Technical Sales, Inc.
3901 Commerce Avenue
Suite 180
Willow Grove, PA 19090
(215) 657-7250
TWX: 510-601-1856

NEW MEXICO

Quatra Associates
9704 Admiral Dewey N.E.
Albuquerque, NM 87111
(505) 821-1455

NEW YORK (Metro)

Parallax
734 Walt Whitman Road
Mellville, NY 11747
(516) 351-1000

NEW YORK

Gen-Tech Electronics
4855 Executive Drive
Liverpool, NY 13088
(315) 451-3480
TWX: 710-545-0250
FAX: 315-451-9088

Gen-Tech Electronics
41 Burning Tree Lane
Penfield, NY 14526
(716) 381-5159

Gen-Tech Electronics
70 Sandoris Circle
Rochester, NY 14622
(716) 467-5016

Gen-Tech Electronics
Drake Road
Pleasant Valley, NY 12569
(914) 635-3233

Gen-Tech Electronics
5 Arbutus Lane
Binghampton, NY 13901
(607) 648-3686

NORTH CAROLINA

The Novus Group, Inc.
5337 Trestlewood Lane
Raleigh, NC 27610
(919) 833-7771
TWX: 510-600-0558

NORTH DAKOTA

Com-Tek
6525 City West Parkway
Eden Prairie, MN 55344
(612) 941-7181
TWX: 310-431-0122
FAX: 612-941-4322

OHIO

Bear Marketing, Inc.
P.O. Box 427
3623 Brecksville Road
Richfield, OH 44286
(216) 659-3131

Bear Marketing, Inc.
1563 East Dorothy Lane
Suite 104
Kettering, OH 45429
(513) 299-5877

OKLAHOMA

Bonser-Philhower Sales
4614 S. Knoxville Avenue
Tulsa, OK 74153
(918) 744-9964
TWX: 510-600-5274

OREGON

Thorson Company Northwest
6700 S.W. 105th Ave.,
Suite 104
Beaverton, OR 97005
(503) 644-5900

PENNSYLVANIA (Eastern)

Delta Technical Sales, Inc.
3901 Commerce Avenue
Suite 180
Willow Grove, PA 19090
(215) 657-7250
TWX: 510-601-1856

PENNSYLVANIA (Western)

Bear Marketing, Inc
829 Greenfield Avenue
Pittsburg, PA 15217
(412) 521-4469

PUERTO RICO

Mill-Bern Associates, Inc.
120 Cambridge St, Suite 8
Burlington, MA 01803
(617) 273-1313
TWX: 710-332-0077
FAX: 617-229-7797

Technology Marketing
Associates, Inc.
1280 S.W. 36th Ave.,
Suite 201
Pompano Beach, FL 33069
(305) 977-9006
TWX: 510-601-0120
FAX: 305-977-9044

RHODE ISLAND

Mill-Bern Associates, Inc.
120 Cambridge St, Suite 8
Burlington, MA 01803
(617) 273-1313
TWX: 710-332-0077
FAX: 617-229-7797

SOUTH CAROLINA

The Novus Group, Inc.
5337 Trestlewood Lane
Raleigh, NC 27610
(919) 833-7771
TWX: 510-600-0558

SOUTH DAKOTA

Com-Tek
6525 City West Parkway
Eden Prairie, MN 55344
(612) 941-7181
TWX: 310-431-0122
FAX: 612-941-4322

TEXAS

Bonser-Philhower Sales
8200 MoPac Expwy.,
Suite 120
Austin, TX 78759
(512) 346-9186
TWX: 910-997-8141

Bonser-Philhower Sales
11321 Richmond, Suite 100A
Houston, TX 77082
(713) 531-4144
TWX: 910-350-3451

Bonser-Philhower Sales
689 W. Renner Rd., Suite C
Richardson, TX 75080
(214) 234-8483
TWX: 910-867-4752
FAX: 214-437-0879

TEXAS (El Paso County)

Quatra Associates
9704 Admiral Dewey N.E.
Albuquerque, NM 87111
(505) 821-1455

UTAH

Front Range Marketing
2520 South State St.,
Suite 117
Salt Lake City, UT 84115
(801) 364-6481

VERMONT

Mill-Bern Associates, Inc.
120 Cambridge St, Suite 8
Burlington, MA 01803
(617) 273-1313
TWX: 710-332-0077
FAX: 617-229-7797

VIRGINA

Micro Comp, Inc.
1421 S. Caton Avenue
Baltimore, MD 21227
(301) 644-5700
TWX: 510-600-9460
FAX: 301-644-5707

Micro Comp, Inc.
Rt. 2, Box 390
Huddleston, VA 24104
(703) 297-6295

WASHINGTON

Thorson Company Northwest
12301 N.E. 10th Place
Bellevue, WA 98005
(206) 455-9180
FAX: 206-455-9185

WASHINGTON (Vancouver)

Thorson Company Northwest
6700 S.W. 105th Ave.,
Suite 104
Beaverton, OR 97005
(503) 644-5900

WASHINGTON D.C.

Micro Comp, Inc.
1421 S. Caton Avenue
Baltimore, MD 21227
(301) 644-5700
TWX: 510-600-9460
FAX: 301-644-5707

WEST VIRGINA

Bear Marketing, Inc.
1563 East Dorothy Lane
Suite 104
Kettering, OH 45429
(513) 299-5877

WISCONSIN (Western)

Com-Tek
6525 City West Parkway
Eden Prairie, MN 55344
(612) 941-7181
TWX: 310-431-0122
FAX: 612-941-4322

WISCONSIN (Eastern)

Beta Technology Sales, Inc.
9401 Beloit, Suite 304C
Mllwaukee, WI 53227
(414) 543-6609

CANADA

BRITISH COLUMBIA

Thorson Company Northwest
12301 N.E. 10th Place
Bellevue, WA 98005
(206) 455-9180

ONTARIO

Electro Source, Inc.
The Bell Mews, Suite 233
39 Robertson Road
Nepean, Ontario K2H 8R2
(613) 726-1452
FAX: 613-726-8834

Electro Source, Inc.
215 Carlingview Dr.,
Suite 303
Rexdale, Ontario M9W 5XB
(416) 675-4490
TWX: 06-989271
FAX: 416-675-6871

QUEBEC

Electro Source
4045 Boul Ste. Gean, Suite
315
Dollard, des Ormeaux
Quebec, HG9 1X4

INTERNATIONAL SALES

SOUTHEAST ASIA

Excel Associates, Ltd.
1502 Austin Tower
22-26A Austin Avenue
Tsimshatsui, Kowloon
Hong Kong
Tel: 852-3-7210900
FAX: 852-3-696826
TLX:30841

UK

Ambar Cascom, Ltd.
Rabans Close,
Aylesbury, Bucks HP193RS
England
Tel: 029634141
TLX: 837427
FAX: (02) 9629670

FRANCE

Reptronic
11, Escalier des Ulis
91400 Orsay, France
Tel: 16-9288700
TLX: 610969F

R.T.F. (Radio Television
Francaise S.A.)
13, rue Lhote
33000 Bordeaux, France
Tel: 16-56-52-99-59
TLX: 560627
FAX: 16-56-48-17-83

R.T.F. Quest
3, rue de Paris
35510 Cesson Sevigne,
France
Tel: 16-99-83-84-85
TLX: 741127
FAX: 16-99-83-80-83

R.T.F. Sud Quest
Avenue de la Mairie
31320 Escalquens, France
Tel: 16-61-81-51-57
TLX: 520927F
FAX: 16-61-81-22-36

R.T.F. Gentilly
9, rue d'Arcueil
94253 Gentilly Cedex,
France
Tel: 46-64-11-01
TLX: 2010169F
FAX: 46-64-41-99

R.T.F. Rhone Alpes
St. Mury, Le Vaucanson
38240 Maylan, France
Tel: 16-76-90-11-88
TLX: 980796
FAX: 16-76-41-04-09

GERMANY

Metronik
Semerteichstrasse 92
4600 Dortmund 30
Dortmund, Germany
Tel: (0231) 423037/38
TLX: 8227082

Metronik
Siemensstrasse 4-6
6805 Heddesheim
Mannheim, Germany
Tel: (06203) 4701-03
TLX: 465053

Metronik
Leonhardsweg 2
Postfach 13 28
8025 Unterhaching
Munich, Germany
Tel: (089) 611080
TLX: 897434
FAX: 89-611 6468

Metronik
Laufamholzstr. 118
8500 Nurnberg 30
Nurnberg, Germany
Tel: (Ø911) 590061/62
TLX: 626205

Metronik
Lowenstr. 37
7000 Stuttgart 70
Stuttgart, Germany
Tel: (0711) 764033/35
TLX: 7255228

ISRAEL

Hitek, Ltd.
19, Keren Hayesod St.
POB 563
Herzlia B, 46105 Israel
Tel: 972-53-72538
FAX: 972-3-236926
TLX: 361360

ITALY

ACSIS S.R.L.
Via Alberto Mario. 26
20149 Milano, Italy
Tel: (02) 4390832
TLX: 326566
FAX: (02) 4697607

Celdis Italiana S.P.A.
Via F.ill Gracchi 36
20092 Cinisello Balsamo
Milano, Italy
Tel: (02) 61-839-1
TLX: 334887
FAX: (02) 61-735-13

Celdis Italiana S.P.A.
Via Massarenti 219/4
40138 Bologna, Italy
Tel: (051) 53-333-6

Celdis Italiana S.P.A.
Via Savelli 15
35100 Padova, Italy
Tel: (049) 77-209-9

Celdis Italiana S.P.A.
Via G. Pitre´ 11
00162 Roma, Italy
Tel: (06) 42-897-1

Celdis Italiana S.P.A.
Via Mombarcaro 96
10136 Torino, Italy
Tel: (011) 32-993-88

JAPAN

Okura & Co., Ltd.
6-12, Ginza Nichome
Chuo-Ku
Tokyo, 104 Japan
Tel: 03-566-6361
TWX: J22306
FAX: 03-563-5447

NETHERLANDS

Rodelco Electronics
Takkebijsters 2
P.O. Box 6824
4802 HV Breda
Tel: 76-784911
TLX: 54195
FAX: 76-710029

SWEDEN

Sattco AB
Dalvagen 10
S-171 36 Solno
Stockholm Sweden
Tel: 46 87 340040
TLX: 11588
FAX: 46-8-7349155

SWITZERLAND

Data Comp AG
Silbernstrasse 10
CH-8953 Dietikon
Tel: 01-7405140
Telex: 827750
FAX: 01-7413423

XILINX

2069 Hamilton Avenue
San Jose, CA 95125
(408) 559-7778
TWX: 510-600-8750
FAX: 408-559-7114

Distributed By

Hamilton/Avnet
Locations throughout the
U.S. and Canada.

Additional Sales Offices
Opening Soon

Notes

Notes