DataBook

Altera Corporation
2610 Orchard Parkway
San Jose, CA 95134-2020
(408) 984-2800
Applications Hotline:
1 (800) 800-EPLD
Marketing Hotline:
1 (800) SOS-EPLD

About this Data Book

October 1990

This ***Data Book*** contains complete information about Altera products:

Section 1 provides an introduction to Erasable Programmable Logic Devices (EPLDs). This section describes the principles underlying EPLD architecture, summarizes EPLD families and software tools offered by Altera, and explains the advantages of CMOS EPROM technology. The section also contains a Product Selection Guide for a quick overview of Altera products.

Section 2 describes the EP-series "classic" EPLDs, including A+PLUS software support.

Section 3 describes the EPM5000-series MAX EPLDs, including MAX+PLUS software support.

Section 4 provides preliminary information about the EPM7000-series MAX EPLDs.

Section 5 describes the EPS-series EPLDs, including SAM+PLUS software support. This series includes the Stand-Alone Microsequencer (SAM) and Synchronous Timing Generator (STG) EPLDs.

Section 6 gives an overview of the EPB-series EPLDs, Altera's user-configurable adapter interface chips for the IBM PS/2 Micro Channel. The EPB2001 and EPB2002A EPLDs are described in detail in the ***Micro Channel Adapter Handbook*** (April 1990).

Section 7 describes operating requirements for all Altera EPLDs.

Section 8 describes Altera's development products, including development systems, software utility programs, EDIF netlist interface, software support for Apollo computers, device programmer, adapters, and software warranty. This section also includes a description of third-party support.

Section 9 describes the military products offered by Altera, including the Source Control Drawings (SCDs) for military-qualified EPLDs. The section also contains an application brief about total-dose radiation hardness of Altera EPLDs.

Section 10 provides application notes and briefs for engineers and engineering managers who seek practical ways to reduce design costs, improve design quality, and shorten design cycles.

Section 11 gives information about how to use Altera's Electronic Bulletin Board Service and order Altera products. This section also shows all EPLD package outlines, describes thermal characteristics of EPLDs, explains how to select sockets for J-lead packages, and lists Altera sales offices, representatives, and distributors.

For immediate assistance on technical questions, please call Altera's Applications Hotline at:

1 (800) 800-EPLD

For information on product availability, pricing, and order status, please contact your Altera Representative or Distributor. Phone numbers and addresses of Altera Sales Offices, Representatives, and Distributors are listed at the end of this data book.

Should you have questions that cannot be answered by your Sales Representative or Distributor, please call Altera's Marketing Hotline at:

1 (800) SOS-EPLD

or contact Altera by FAX at:

408-248-6924

Contents

October 1990

Product Index

October 1990

EPLDs

EPLDs (continued)

Development Tools

Development Tools (continued)


ADERA

Contents

October 1990

Section 1 Introduction to Altera EPLDs

1

Programmable Logic Overview

October 1990, ver. 1

Data Sheet

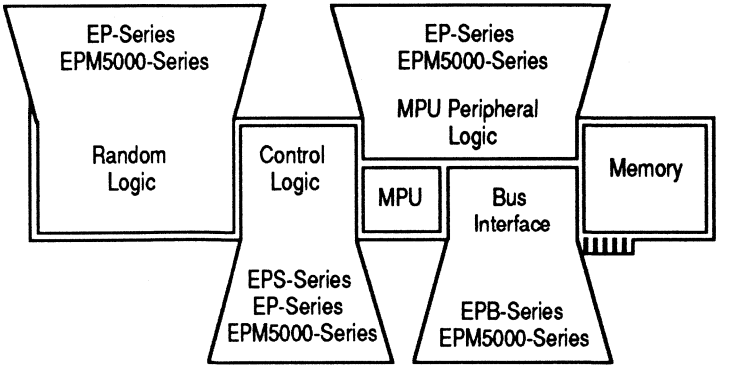
Introduction

Programmable Logic Devices (also described as PLD, PAL, PLA, FPLA, EPLD, EEPLD, LCA, and FPGA devices), combine the logistical advantages of standard, fixed integrated circuits with the architectural flexibility of custom devices. These devices allow engineers to electrically program standard, off-the-shelf logic elements to meet the specific needs of their applications. Proprietary logic functions can be designed and fabricated in-house, eliminating the long engineering lead times, high tooling costs, complex procurement logistics, and dedicated inventory problems associated with custom Application-Specific Integrated Circuit (ASIC) devices, such as gate arrays and standard cells.

The key to this "off-the-shelf ASIC" capability is CMOS EPROM technology, which is used to create Erasable Programmable Logic Devices (EPLDs). Altera has taken advantage of speed and density advances in CMOS EPROM memory products to create sophisticated EPLDs that solve many logic design problems.

Altera provides the broadest line of CMOS EPLDs in the industry, with products ranging in density from hundreds to thousands of gates, offered in a variety of packages with 20 to 100 pins. Larger EPLDs, with up to 20,000 gates and over 200 pins (the EPM7000 series), are currently under development. These EPLDs, together with Altera development software, enable system manufacturers to create custom logic functions for a wide variety of applications. See Figure 1.

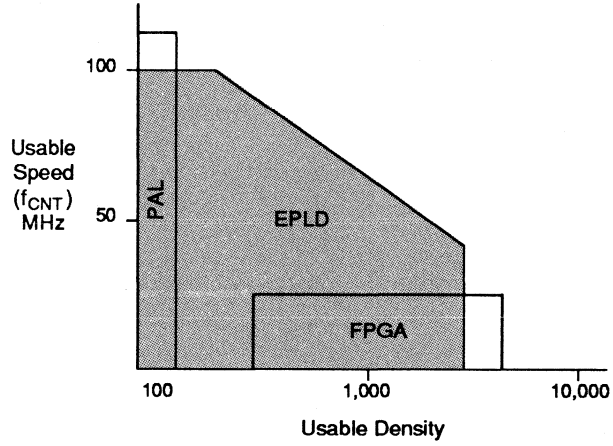
Figure 1. Altera User-Programmable Logic Families



EPLDs can be used to integrate complete printed circuit boards of TTL, PAL, and FPGA devices into a single package. EPLDs are also valuable for prototyping high-density custom devices, which enables designers to test markets and evaluate systems before committing to expensive engineering development cycles and tooling charges. For most of today's applications, EPLDs not only ensure faster time-to-market, but also provide a lower total cost than custom ASIC solutions.

Altera concentrates on creating high-performance device architectures and easy-to-use, highly productive CAE software. Altera products meet the demands of designers who require complete solutions to logic integration that include both PAL speed and FPGA density. See Figure 2.

Figure 2. PLD Speed vs. Density



EPLD Families

Altera offers several families of EPLDs that satisfy many common board- and system-integration needs. EPLD families are divided into two architectural categories: the first provides maximum flexibility for general-purpose logic replacement; the second is specialized for performing specific system design tasks.

- General-purpose EPLDs are available in a variety of integration densities, ranging from PAL replacements to high-density devices that integrate thousands of TTL and random logic gates. These EPLDs are designated with the EP- and EPM- prefixes.
  - The EP-series architecture includes 20- to 68-pin EPLDs that feature zero-standby power, propagation delays (t_{PD}) of 12 ns, and counter frequencies of up to 100 MHz.

- The EPM5000-series Multiple Array MatriX (MAX) architecture includes 20- to 100-pin EPLDs that combine the high speed and ease-of-use of PAL devices with the density of FPGA devices. High-density MAX EPLDs can consolidate 20 to 25 PAL packages and over 100 TTL functions, while offering system clock rates of 50 MHz.

Altera is also developing the next generation of EPLDs—the EPM7000 series—that will provide integration densities of 1,500 to 20,000 gates, with additional increases in system speed.

❒ Function-specific EPLDs provide optimized integration for specific system design tasks. They are classified on the basis of their system design focus and are designated with the EPB- and EPS- prefixes.

- EPB-series EPLDs are bus-oriented devices designed to integrate all the required add-on card logic for a Micro Channel bus interface.

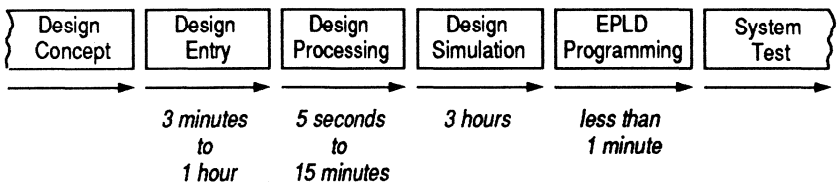
- EPS-series EPLDs offer the logic and speed required for complex control logic, state machines, and imaging and display applications. EPS-series EPLDs include the Stand-Alone Microsequencer (SAM) and Synchronous Timing Generator (STG) EPLDs.

EPLDs are offered in a variety of packages, including the dual in-line package (DIP), J-lead chip carrier (JLCC), small-outline integrated circuit (SOIC), quad flat pack (QFP), and pin-grid array (PGA). EPLDs are available in windowed (erasable) ceramic packages for development, or one-time-programmable plastic versions for high-volume production requirements.

Software Tools

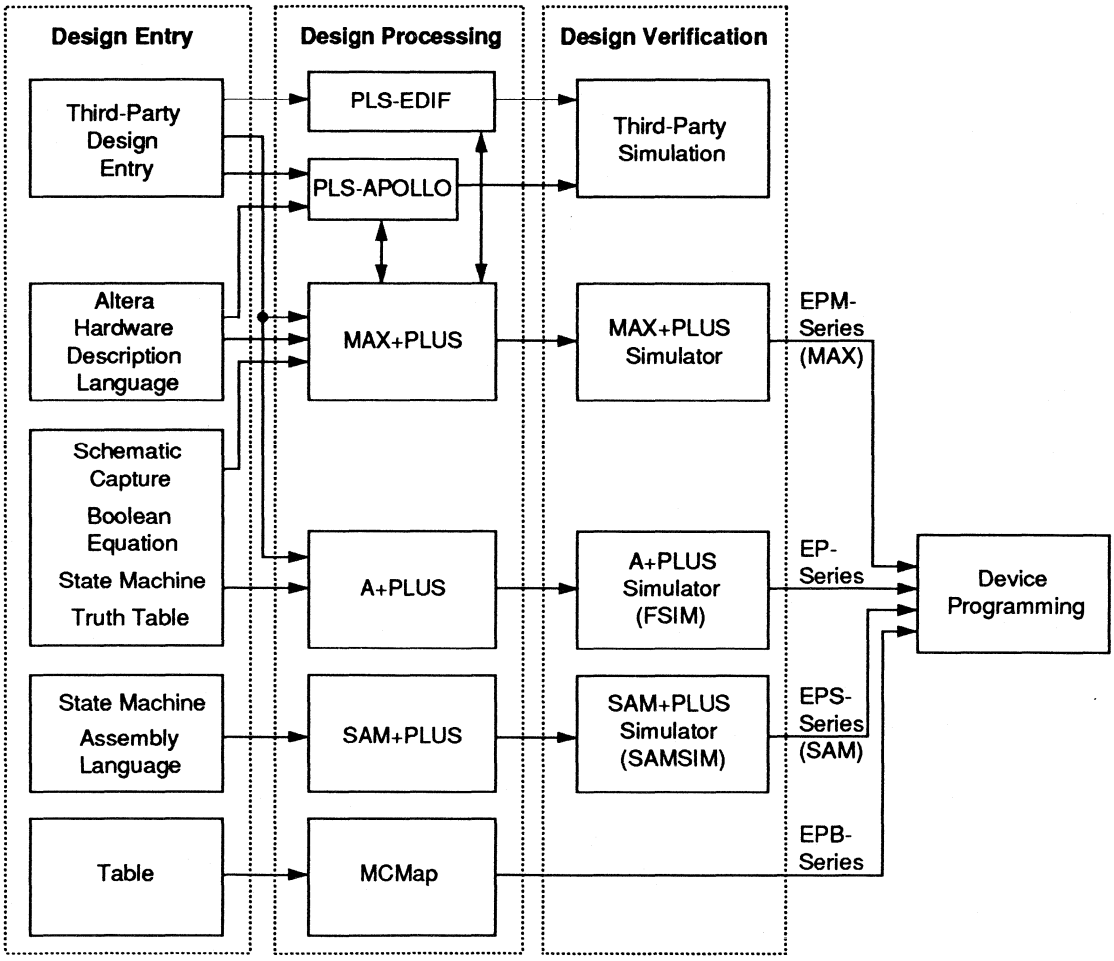
Altera software products are developed together with the EPLD architectures, so features are placed where they are most appropriate—in either software or hardware. The result is efficient software tools that offer familiar design entry methods and rapid design completion. (See Figure 3.) With Altera's CAE development tools, users can take a logic circuit from design entry to device programming in a matter of hours. Design processing is typically completed in minutes, allowing several design iterations to be completed in a single day.

Figure 3. EPLD Design Methodology: From Concept to Silicon in Hours



Altera software is available for PC-AT (or compatible), PS/2, and workstation computers (Apollo, Sun, and IBM). Several design entry options are available: hierarchical schematic capture (with basic gate and complete TTL libraries), the Altera Hardware Description Language (AHDL), Boolean equation, state machine, truth table, netlist, and microcoded assembly language. (See Figure 4.) Design entry methods may be freely combined to create a single EPLD design. Design processors perform minimization and logic synthesis, design fitting (analogous to automatic place-and-route), and generate programming data. Design verification via functional simulation, timing simulation, and delay prediction for speed-critical paths is also available. Hardware for programming EPLDs is offered by Altera and a variety of third-party vendors.

Figure 4. Altera Design Environment



Software interfaces to other design tools are provided by Altera and third-party translators, and via industry-standard EDIF netlists. Many third-party compilers also support Altera EPLDs directly.

EPLD Architecture

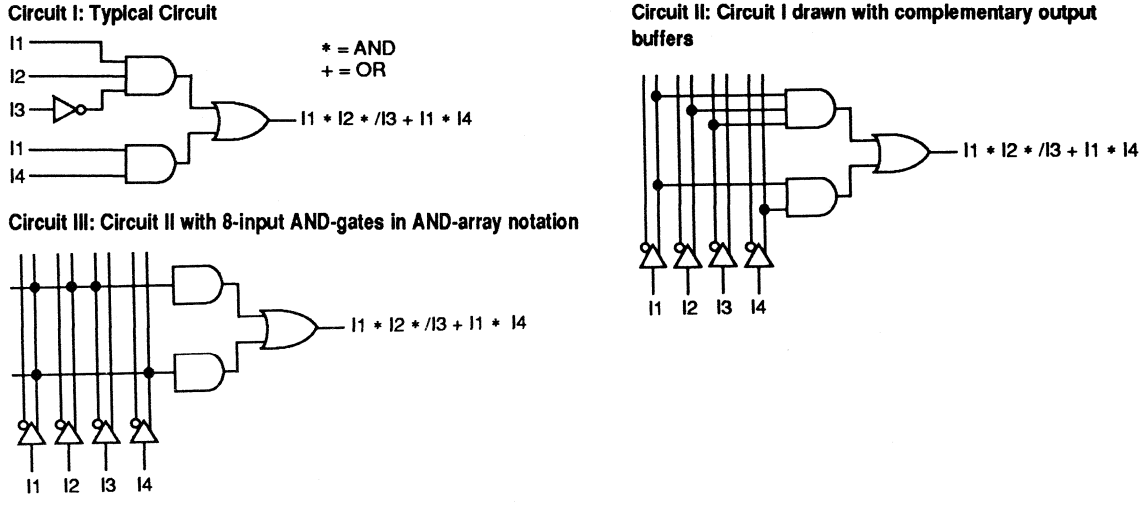
The following discussion of EPLD architecture is provided for interested readers. Note, however, that the Altera approach to logic design eliminates the necessity of mastering the inner complexities of EPLD architectures. The user may work with familiar design entry tools (e.g., TTL functions or a high-level state machine language), and the Altera software automatically translates the design into the format required to fit the EPLD architecture. For detailed architecture and pin-out descriptions for each device, refer to individual EPLD data sheets in this data book.

Basic Concepts

Altera general-purpose EPLDs provide dedicated input pins, user-configurable I/O pins, and programmable flip-flop and clock options that ensure maximum flexibility for integrating random logic functions.

Each EPLD also contains an AND array that provides product terms. A product term is simply an *n*-input AND gate, where *n* is the number of connections. EPLD schematics use a shorthand AND-array notation to represent several large AND gates with common inputs. Figure 5 shows three different representations of the same logic function. Circuit I is presented in classic logic notation; circuit II has been modified to a sum-of-products notation; and circuit III is written in AND-array notation. A dot represents a connection between an input (vertical wire) and one of the 8-input AND gates. No dot implies no connection: the AND gate input is unused and floats to a logic 1.

Figure 5. AND-Array Notation



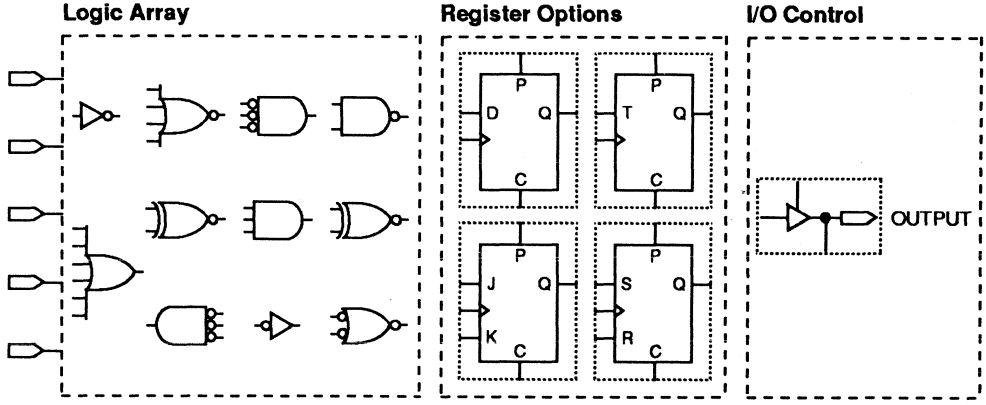
The 2×8 AND-array of circuit III can produce any Boolean function of four variables (provided only two product terms are required) when expressed in sum-of-products form. Any Boolean expression—no matter how complex—can be written in sum-of-products form. Outputs of the two AND gates in Figure 5 are called product terms (or p-terms).

Macrocell Architecture

The fundamental building block of an Altera EPLD is the macrocell. Each macrocell consists of three parts (see Figure 6):

- The logic array implements all combinatorial logic functions.
- The programmable register provides D, T, JK, or SR options (the register can also be bypassed).
- Programmable I/O allows each I/O pin to be configured for dedicated input, output, or bidirectional operation.

Figure 6. The Macrocell

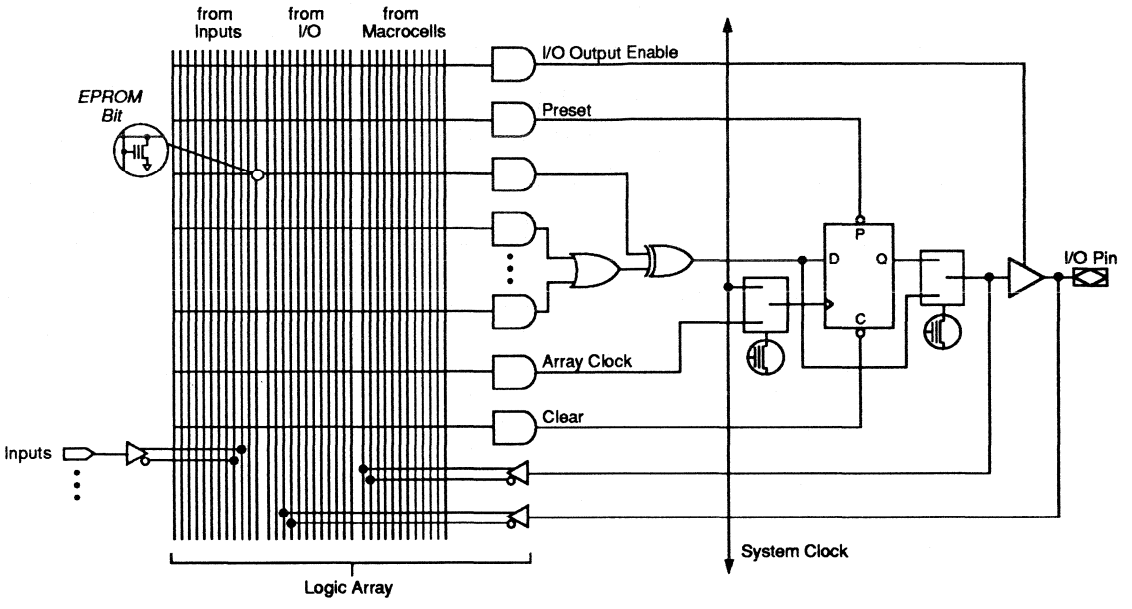


Logic Array

The logic array consists of such a programmable-AND/fixed-OR PLA array. Inputs to the AND array come from the true and complement of the dedicated input and clock pins, and from the macrocell and I/O feedback paths.

For each macrocell, the logic array typically contains 10 product terms that are distributed among the combinatorial and sequential resources. (See Figure 7.) Connections are opened during the programming process. Therefore, any product term may be connected to the true and complement of any array input signal. When both the true and complement of any signal are left intact, a logic low (0) results on the output of the product term. If both the true and complement connection are open, a logical "don't care" results for that input. If all inputs for the product term are programmed opened, a logic high (1) results on the output of the product term.

Figure 7. Detailed EPLD Macrocell Architecture



Several product terms feed a fixed OR whose output connects to an exclusive-OR (XOR) gate. The second input to the XOR function is controlled by a programmable resource (usually a product term) that allows the logic array output to be inverted. Altera software uses this gate to implement active-high or active-low logic, complex mutually exclusive and arithmetic functions, or to reduce the number of product terms to implement a function (by applying De Morgan's inversion). Figure 8 shows an OR function that, in its current form, requires six product terms. By using the "programmable" XOR gate and De Morgan's inversion, the OR function can be transformed into a NAND function:

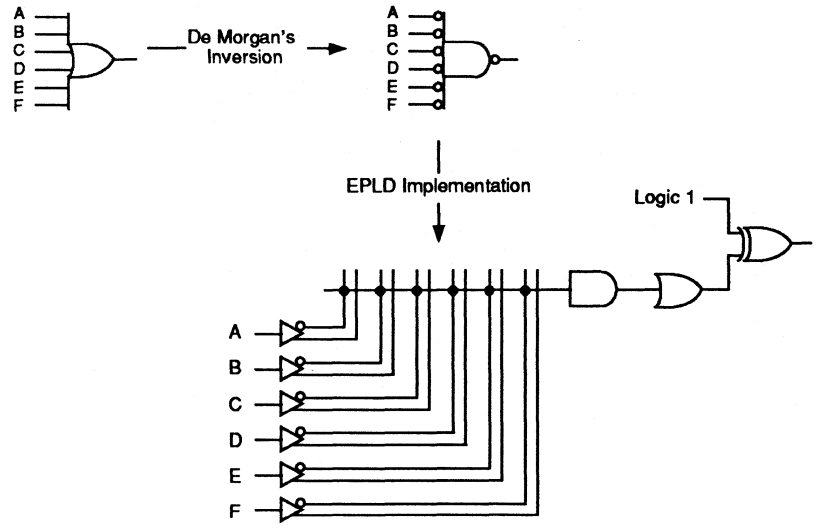
```
A+B+C+D+E+F = /(/A*/B*/C*/D*/E*/F)
```

This inversion from OR to AND translates the equation and reduces the number of fixed-OR terms required in the logic array. Altera software automatically applies De Morgan's inversion and other logic synthesis techniques to optimize the use of the logic array.

Programmable Flip-Flops

Programmable flip-flops are used to create a variety of logic functions that use a minimum of EPLD resources. Each flip-flop can be programmed to provide a conventional D-, JK-, T-, or SR-type function. MAX EPLD flip-flops can also be configured as flow-through latches. If the flip-flop is not required for macrocell logic, it may be simply bypassed. Macrocell flip-

Figure 8. Logic Minimization with De Morgan's Inversion



flops also have an asynchronous Clear and Preset capability that allows complete emulation of any TTL function.

Programmable Clock

In general-purpose EPLDs (except the EP330 and EP320), each internal flip-flop may be clocked from a dedicated system clock (also known as a synchronous clock), any input or I/O pin, or any internal logic function. For each flip-flop, a multiplexer selects either a pin or product-term source for the clock, so that flip-flops can be clocked independently or in user-defined groups. EPLD registers are positive-edge-triggered with data transitions that occur on the rising edge of the dedicated system clock.

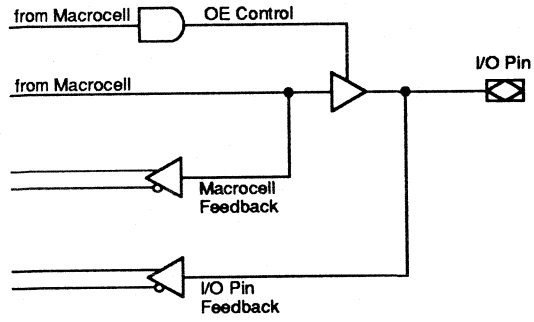
When the clock is driven by a product term, flip-flops can be configured for either positive- or negative-edge-triggered operation. In addition, product-term clocks allow gated-clock and clock-enable logic to be implemented. However, system clock signals have faster clock-to-output delay times than internally generated product-term clock signals.

I/O Control Block

The EPLD I/O control block contains a tri-state buffer controlled by a macrocell product term, and drives the I/O pin (see Figure 9). I/O pins may be configured as dedicated outputs, bidirectional outputs, or as additional dedicated inputs. Most EPLDs have "dual feedback," whereby the macrocell feedback is decoupled from the I/O pin feedback. Dual feedback makes it possible to implement a buried function in the macrocell while the I/O pin is used simultaneously as a dedicated input. Applications that require many buried flip-flops (such as counters, shift registers, and

Figure 9. I/O Control Block

The decoupled I/O control block features dual feedback to maximize use of the EPLD pins.



state machines or bus-oriented functions) are easily accommodated by this programmable I/O control block.

Zero-Power/ Turbo Operation

CMOS technology generally implies lower power dissipation than older bipolar technology. In fact, Altera pioneered true "zero-standby" power operation. By using a unique input-transition detection scheme, most EP-series EPLDs use only microamps during quiescent periods. This feature saves power in applications clocked at low to medium frequencies (< 10 MHz). Each input is connected to a transition-detection circuit consisting of an XOR gate, a delay element, and an OR gate. The trigger output of the OR gate activates logic array power-up on any transition, allowing new input conditions to propagate to EPLD outputs. The logic array is then automatically powered down to await the next transition. The transition-detection circuitry adds an additional 30 to 40% delay to the EPLD input/output path. Consequently, a programmable "Turbo Bit" is provided to disable the input transition detection circuitry and permanently enable the logic array, giving the user a choice of either extra speed or lower power consumption. The EPLD also exhibits better system noise rejection characteristics in the turbo mode, which should be used where noisy environments are a problem. The Turbo Bit is included in the EPLD programming file and is programmed in the same way as any other EPROM bit.

Altera CMOS EPROM Technology

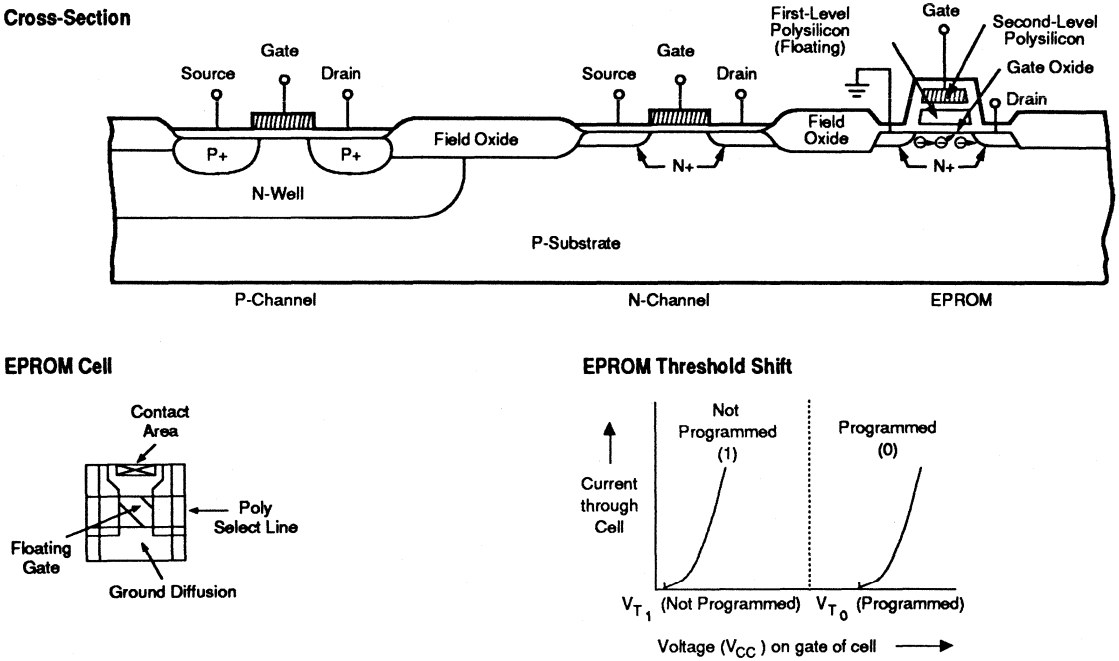
Until Altera invented the first EPLD in 1984 (the EP300), the only technology used for Programmable Logic Devices (PLDs) was bipolar and fuse-based. The active elements on these devices were constructed from traditional bipolar transistors (i.e., TTL), with arrays of fuses providing programmable interconnect structures. These fuse elements consisted of a variety of exotic metal alloys and/or polysilicon structures. However, all relied on opening connections by passing large currents through their small geometries, thereby physically destroying the fuses.

The melting process in bipolar PLD fuses is difficult to control and often results in poor and unpredictable programming yields. Since the process is

irreversible, guaranteed results are impossible. The power-hungry bipolar technology also severely limits integration levels. Altera's pioneering efforts have replaced bipolar technology with CMOS, and fuses with reprogrammable EPROM bits. These bits are much smaller than fuses, electrically programmable, and UV-erasable. EPLDs are fully factory-tested, guaranteeing 100% programming yield at the customer site. CMOS technology also provides low-power operation that allows higher integration levels.

The EPROM cell operates via floating-gate charge injection. The programming process consists of placing sufficient voltage (typically >12 V) on the drain of the transistor to create a strong electric field and energize electrons to jump from the drain region to the floating gate. Electrons are attracted to the floating gate and become trapped when the voltage is removed. If the gate remains at a low voltage during programming, electrons are not attracted and the floating gate remains uncharged. Trapped charge changes the threshold of the EPROM cell from a relatively low value with no charge present ("erased") to a higher value when programmed. Figure 10 shows a basic cross-section of the cell technology.

Figure 10. CMOS EPLD Technology



Within the EPLD's programmable array, a sense amplifier/comparator circuit is placed at the end of each product-term line; by setting a reference voltage into the circuit—halfway between the programmed and unprogrammed levels—the state of the EPROM cells along the product term is sensed and used to select the desired logic function. Low-threshold cells with a logic "1" placed on their select gates (associated input) tend to pull the product-term line down and cause the logic term to go to a "0." Transistors with high thresholds do not conduct even when their gates are at a logic 1, and effectively represent a no-connect. This technology—pioneered with EPROM memory in the early 1970's—made it possible to build Altera EPLDs that can be tested, programmed, and operated reliably. Altera devices currently use state-of-the-art 0.8-micron, CMOS EPROM technology; work is underway to move to even smaller geometries. Because the basic logic array consists of N-channel EPROM transistors, EPLD characteristics are optimized to maximize performance of the N-channel device. This approach minimizes overall input-to-output delays on the chip.

EPROM Cell Margin

To ensure reliable operation in user systems, all EPLDs undergo substantial factory testing prior to shipment. Foremost among these tests are cell-margin tests, which guarantee the in-service retention of EPROM bit programming. Cell-margin testing determines the amount of charge trapped on the floating gate structure.

Charge loss occurs when electrons leak from the floating gate structure over time, and results in a net reduction in programmed cell threshold. Charge gain results from an accumulation of charge on the floating gate, usually caused by electric fields produced by operating the EPLD. Since charge loss and charge gain mechanisms can affect program retention, Altera reliability evaluation includes EPLD burn-in at temperatures of up to 250° C for periods of a week or more. This burn-in period corresponds to >100,000 years of operation at 70° C.

Figure 11. EPROM Cell Margin

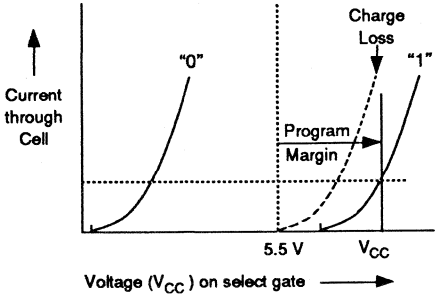


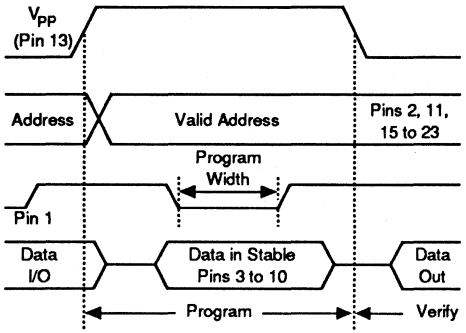
Figure 11 illustrates the concept of cell margin. As mentioned earlier, EPROM arrays depend on cell threshold shifts for correct operation. Zero and One I-V characteristics for the EPROM cell are shown. Program margin is a measure of the spread between the actual device threshold and the minimum required device threshold for correct operation.

To calibrate cell margins, Altera EPLDs are subjected to special test modes that allow EPROM-bit gate voltages to be controlled externally. Cell margins are measured by varying this voltage, a method that accurately monitors cell charge and retention.

Programming

Figure 12 shows a typical programming cycle for Altera EPLDs. The normal programming procedure consists of the following steps:

Figure 12. Programming Waveforms



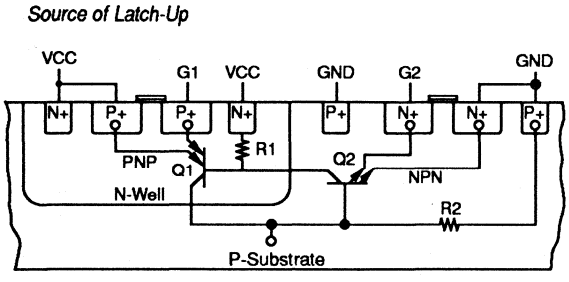
1. The programming pin (V_{PP}) is raised to the super-high-input level (nominally 12.5 V).
2. Row and column addresses are placed on the designated pins.
3. Programming data is placed on the designated pins.
4. The programming algorithm is executed with a sequence of 100-µs programming pulses separated by program verify cycles.
5. Overprogram or margin pulses may be applied to doubly ensure EPLD programming.

The programming operation is typically performed eight bits at a time on either Altera-supplied or other approved programming hardware. Altera EPLDs also feature a Security Bit (i.e., verify-protect bit) that can be programmed to prevent any interrogation of the device's contents. This bit can be set during the programming process to ensure EPLD design security.

Latch-Up

Parasitic bipolar transistors are present in the fundamental structure of CMOS devices. Typically, the base-emitter and base-collector junctions of these transistors are not forward-biased, so the transistors are not turned on. Figure 13 shows a cross-section of a CMOS wafer and primary parasitic transistors. By connecting the P-type substrate to the most negative voltage available on-chip (V_{SS}) and the N-type well structure to the most positive voltage on-chip (V_{CC}), all junctions should, in theory, remain reverse-biased. However, two factors can alter this ideal state.

Figure 13. Parasitic Bipolar Transistors in CMOS



As shown in Figure 13, parasitic resistors also occur in the CMOS structure. These resistors are of no concern as long as currents do not flow through the structure laterally. But if any of the associated diodes turn on for any reason, I-R drops may occur in the structure. The initial turn-on of these diodes usually is the result of power-supply or I/O-pin transients that exceed the limits of V_{SS} and V_{CC}. These transients may be induced by signal ringing and other inductive effects in the system.

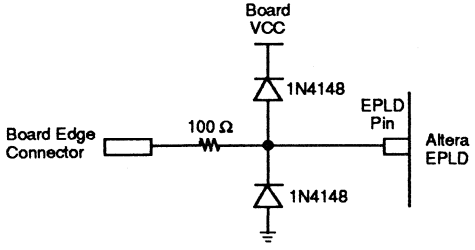
A problem may exist if parasitic structures begin to conduct, since the effect is regenerative and reinforces itself until potentially destructive currents flow. This is the silicon-controlled rectifier (SCR) effect called "latch-up." As current flows through the parasitic transistor, the I-R drop through the resistor increases, further forward-biasing the base-emitter junction, as shown in Figure 13. The cycle continues until the current is limited by drops in the primary current path. However, this current might reach a level that permanently damages internal circuitry.

Altera components have been designed to minimize the effects of latch-up, including power-supply and I/O-pin transients. Under reasonable system operating conditions, all EPLDs are guaranteed to withstand input voltage extremes of between $V_{SS} - 1$ V and $V_{CC} + 1$ V, as well as input currents of 100 mA or less that are forced through the device pins. To minimize the possibility of inducing latch-up, Altera recommends a few general system design guidelines for power and input sequencing to the EPLD. For example, voltages and logic inputs should be applied in the following order:

1. V_{SS} or GND
2. V_{CC} (+5 V)
3. Inputs

When removing power from the EPLD, the order should be reversed: first, inputs are removed or taken low, then V_{CC} is removed or lowered. Simultaneous application of inputs and V_{CC} to the device, which might occur as a power supply ramps during power-up, should be safe. Care should be taken to ensure that inputs cannot rise faster than supply under extreme conditions.

Figure 14. Hot Socket Protection

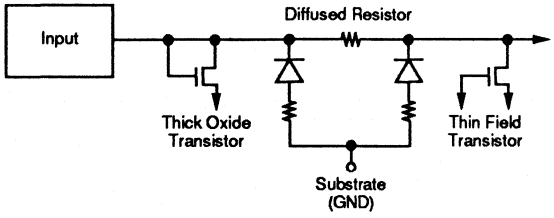


In some applications, boards are "hot-socketed" in the field. The circuitry shown in Figure 14 is recommended to ensure that latch-up-inducing levels are not applied to the EPLD under these conditions. Normally, this circuitry is required only if the EPLD has inputs tied directly to the edge connector. The diodes clamp the inputs at acceptable levels and the series resistor further limits the injection of current into the EPLD input and clamp diodes. This interface provides maximum protection.

Electrostatic Discharge

Electrostatic discharge (ESD) can cause device failure when improper handling occurs. EPLD handling during the programming cycle increases exposure to potential static-induced failure. Voltages into the tens of kilovolts can be generated by the human body during normal activity. Wearing ground straps during device handling and grounding all surfaces that come in contact with components reduce the likelihood of damage.

Figure 15. EPLD Input Protection Structure



Altera components include special structures that reduce the effects of ESD at the pins. Figure 15 shows a typical input structure. Diode structures as well as specialized field-effect transistors shunt harmful voltages to ground before destructive currents will flow. Altera EPLDs typically withstand ESD voltages > 2 kV, and are thus safe under normal handling conditions.

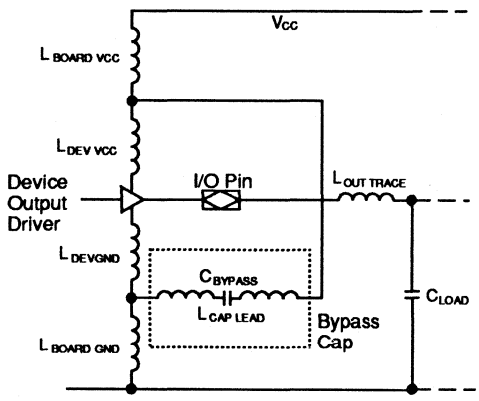
Output Drive Characteristics

The CMOS push-pull output stages used on Altera EPLDs provide good AC and DC load-driving capability in a system environment. I_{OL} and I_{OH} specifications for general-purpose EPLDs are guaranteed at 4 mA to 24 mA, depending on the device. AC output characteristics are typically specified with 35 pF output loads. Additional output capacitive loading affects the device output delay. The timing parameter used is t_{PD} (input-output combinatorial delay). The incremental delay per picofarad of capacitance is typically ≤ 0.1 ns at room temperature.

System Noise

Large switching currents can flow through power supply and output pins during high-performance operation. If a 50-pF capacitor is charged from 0 to 5 V in 10 ns, a dynamic current of 24 mA will flow. If 24 outputs on an EPLD switch simultaneously (for example, in an EP910), the total transient current can exceed 600 mA! This current can severely degrade V_{CC} supply voltage due to the inductive properties of the device and system environment. Figure 16 shows the distribution of typical inductances that can contribute to the problem.

Figure 16. Board-Level Noise Problem



The key to controlling these inductive effects is to adequately decouple the V_{CC} supply to ground at each EPLD with a suitable capacitor or combination of capacitors. This capacitor can then act as a reservoir of charge to supply the transient switching needs of the device. It is recommended that a 0.2 µF capacitor be connected from each V_{CC} pin to ground at the device. High-quality capacitors with low internal and lead inductance (monolithic ceramic or tantalum) should be used, and leads must be kept short to limit series inductance that degrades capacitor effectiveness. Careful decoupling of the power supply is good design practice.

Product Selection Guide

October 1990, ver. 1

Introduction

This Product Selection Guide summarizes the range of available products from Altera:

- ❒ General-purpose EPLDs
- ❒ Function-specific EPLDs
- ❒ Military-qualified EPLDs
- ❒ Programmable logic development systems
- ❒ Programmable logic software
- ❒ Software warranty
- ❒ Programming hardware
- ❒ Programming adapters

For detailed descriptions of the Altera products listed here, refer to the individual data sheets in this data book and to the ***Micro Channel Adapter Handbook***.

EPM-Series EPLDs

Table 1 provides information on the MAX (Multiple Array MatriX) general-purpose family of EPLDs. MAX (or EPM-series) EPLDs are suitable for register-intensive random logic, TTL, and PAL integration.

Table 1. General-Purpose EPM-Series MAX EPLDs

EPLD (1)	Package (2)	Temp. (3)	Speed Option	t_{PD1} (ns)	f_{MAX} (MHz)	I_{CC3} (mA) Active	I_{CC1} (mA) Standby	Macrocells (Registers)	Dedicated Inputs	I/O	Number of Pins
EPM5192	J,L,G,Q,W	C	–1	25	62.5	380	360	192	8	64	84; 100
EPM5192	J,L,G,Q,W	C	–2	30	50.0	380	360	192	8	64	84; 100
EPM5192	J,L,G,Q,W	C,I,M		35	40.0	380	360	192	8	64	84; 100
EPM5130	G,Q,W	C	–1	25	62.5	275	250	128	20	64	100
EPM5130	G,Q,W	C	–2	30	50.0	275	250	128	20	64	100
EPM5130	G,Q,W	C,I,M		35	40.0	275	250	128	20	64	100
EPM5128	J,L,G,Q,W	C	–1	25	62.5	250	225	128	8	52	68
EPM5128	J,L,G,Q,W	C	–2	30	50.0	250	225	128	8	52	68
EPM5128	J,L,G,Q,W	C,I,M		35	40.0	250	225	128	8	52	68
EPM5064	J,L	C	–1	25	62.5	135	125	64	8	28	44
EPM5064	J,L	C	–2	30	50.0	135	125	64	8	28	44
EPM5064	J,L	C,I,M		35	40.0	135	125	64	8	28	44
EPM5032	D,P,J,L,S	C	–1	15	83.3	155	150	32	8	16	28
EPM5032	D,P,J,L,S	C	–2	20	71.4	155	150	32	8	16	28
EPM5032	D,P,J,L,S	C,I,M		25	62.5	155	150	32	8	16	28
EPM5016	D,P,J,L,S	C	–1	15	100.0	115	110	16	8	8	20
EPM5016	D,P,J,L,S	C	–2	17	83.3	115	110	16	8	8	20
EPM5016	D,P,J,L,S	C,I,M		20	62.5	115	110	16	8	8	20

Notes to Table 1:

(1) Preliminary data is shown for some parameters. Consult individual device data sheets for complete information.

(2) Package configurations:
- D: Windowed ceramic dual in-line (CerDIP)
- P: One-time-programmable plastic dual in-line (PDIP)
- J: Windowed ceramic J-lead chip carrier (JLCC)
- L: One-time-programmable plastic J-lead chip carrier (PLCC)
- G: Windowed ceramic pin-grid array (PGA)
- S: One-time-programmable plastic small-outline integrated circuit (SOIC)
- Q: One-time-programmable plastic quad flat pack (PQFP)
- W: Windowed ceramic quad flat pack (WQFP)

(3) C = Commercial (0° C to +70° C); I = Industrial/Automotive (–40° C to +85° C); M = Military (–55° C to +125° C).

EP-Series EPLDs

Table 2 gives information on the "classic" family of general-purpose, zero-standby-power EPLDs. Classic (or EP-series) EPLDs are suitable for random logic, TTL, and PAL integration.

Table 2. General-Purpose EP-Series Classic EPLDs

EPLD (1)	Package (2)	Temp. (3)	Speed Option	t_{PD1} (ns)	f_{MAX} (MHz)	I_{CC3} (mA) Active	I_{CC1} (mA) Standby	Macrocells (Registers)	Dedicated Inputs	I/O	Number of Pins
EP1830	J,L,G	C	–20	20	62.5	0.15	200	48	16	48	68
EP1830	J,L,G	C	–25	25	50.0	0.15	200	48	16	48	68
EP1830	J,L,G	C,I,M	–30	30	41.7	0.15	200	48	16	48	68
EP1810	J,L,G	C	–35	35	40.0	0.15	180	48	16	48	68
EP1810	J,L,G	C,I	–40	40	35.7	0.15	180	48	16	48	68
EP1810	J,L,G	C,I,M	–45	45	33.3	0.15	180	48	16	48	68
EP910	D,P,J,L	C	–30	30	41.7	0.10	80	24	12	24	40; 44
EP910	D,P,J,L	C,I	–35	35	37.0	0.10	80	24	12	24	40; 44
EP910	D,P,J,L	C,I,M	–40	40	32.3	0.10	80	24	12	24	40; 44
EP610A	D,P,J,L,S	C	–12	12	83.3	130	130	16	4	16	24; 28
EP610A	D,P,J,L,S	C	–15	15	83.3	130	130	16	4	16	24; 28
EP630	D,P,J,L,S	C	–15	15	83.0	0.15	90	16	4	16	24; 28
EP630	D,P,J,L,S	C,I,M	–20	20	62.5	0.15	90	16	4	16	24; 28
EP610	D,P,J,L,S	C	–25	25	47.6	0.10	60	16	4	16	24; 28
EP610	D,P,J,L,S	C,I	–30	30	41.7	0.10	60	16	4	16	24; 28
EP610	D,P,J,L,S	C,I,M	–35	35	37.0	0.10	60	16	4	16	24; 28
EP330	D,P,L,S	C	–12	12	125	75	75	8	10	8	20
EP330	D,P,L,S	C,I,M	–15	15	100	75	75	8	10	8	20
EP320	D,P	C	–1	29	45.5	0.15	30	8	10	8	20
EP320	D,P	C	–2	34	40.5	0.15	30	8	10	8	20
EP320	D,P	C,I,M		44	30.3	0.15	30	8	10	8	20

Notes to Table 2:

(1) Preliminary data is shown for some parameters. Consult individual device data sheets for complete information.

(2) Package configurations:
- D: Windowed ceramic dual in-line (CerDIP)
- P: One-time-programmable plastic dual in-line (plastic DIP)
- J: Windowed ceramic J-lead chip carrier (JLCC)
- L: One-time-programmable plastic J-lead chip carrier (PLCC)
- G: Windowed ceramic pin-grid array (PGA)
- S: One-time-programmable plastic small-outline integrated circuit (SOIC)

(3) C = Commercial (0° C to +70° C); I = Industrial/Automotive (–40° C to +85° C); M = Military (–55° C to +125° C).

Function-Specific EPLDs

Table 3 provides information on the function-specific Stand-Alone Microsequencer (SAM) and Synchronous Timing Generator (STG) EPLDs. These EPS-series EPLDs are suitable for implementing high-performance state machines, waveform generators, and control logic.

Table 4 gives information on the Micro Channel bus interface (EPB-series) EPLDs, which provide all the essential functions to interface a PS/2 add-on card with the Micro Channel bus. (Refer to the ***Micro Channel Adapter Handbook*** for detailed information on EPB-series EPLDs.)

Table 3. EPS-Series SAM and STG EPLDs

EPLD (1)	Pkg. (2)	Temp. (3)	Speed Option	f_{MAX} (MHz)	I_{CC2} (mA) Active	I_{CC1} (mA) Standby	Microcode EPROM	Branch EPLD	Stack	Dedicated Inputs	I/O	No. of Pins
EPS448	D,P,J,L	C	–30	30	140	95	448 × 36	768 p-term	15 × 8	8	16	28
EPS448	D,P,J,L	C	–25	25	140	95	448 × 36	768 p-term	15 × 8	8	16	28
EPS448	D,P,J,L	C,I,M	–20	20	140	95	448 × 36	768 p-term	15 × 8	8	16	28
EPS464	J,L,Q,W	C	Preliminary Information—consult factory							4	32	44

Table 4. EPB-Series Micro Channel Bus Interface EPLDs

EPLD (1)	Package (2)	Temperature (3)	Description	Number of Pins
EPB2001	J,L	C	Single-chip interface adapter for PS/2 Micro Channel	84
EPB2002A	L,P	C	DMA arbitration support chip for PS/2 Micro Channel	28

Notes to Tables 3 & 4:

(1) Preliminary data is shown for some parameters. Consult individual device data sheets for complete information.

(2) Package configurations:
- D: Windowed ceramic dual in-line (CerDIP)
- P: One-time-programmable plastic dual in-line (PDIP)
- J: Windowed ceramic J-lead chip carrier (JLCC)
- L: One-time-programmable plastic J-lead chip carrier (PLCC)
- Q: One-time-programmable plastic quad flat pack (PQFP)
- W: Windowed ceramic quad flat pack (WQFP)

(3) C = Commercial (0° C to +70° C); I = Industrial/Automotive (–40° C to +85° C); M = Military (–55° C to +125° C).

Military-Qualified EPLDs

Table 5 provides information on Altera's military-qualified MAX (EPM5000-series) EPLDs; Table 6 gives information on military-qualified classic (EP-series) EPLDs.

Table 5. Military-Qualified EPM-Series MAX EPLDs

EPLD *(1)*	Pkg. *(2)*	Assurance Level *(3)*	t_{PD1} (ns)	f_{MAX} (MHz)	I_{CC3} (mA) Active	I_{CC1} (mA) Standby	Macrocells (Registers)	Dedicated Inputs	I/O	Number of Pins	Altera Mil. Drawing *(4)*
EPM5128	J	B	35	40.0	350	300	128	8	52	68	02D-00827
EPM5128	G	883B	35	40.0	350	300	128	8	52	68	02D-00827
EPM5064	J	883B	35	40.0	225	200	64	8	28	44	02D-00968
EPM5032	D,J	883B	25	62.5	225	200	32	8	16	28	02D-00828
EPM5016	D	883B	20	62.5	175	150	16	8	8	20	02D-00967

Notes to Table 5:

(1) All military-qualified EPLDs are rated to military temperatures (–55° C to +125° C). Preliminary data is shown for some other parameters. Consult individual device data sheets for complete information.

(2) Package configurations:
- D: Windowed ceramic dual in-line (CerDIP)
- J: Windowed ceramic J-lead chip carrier (JLCC)
- G: Windowed ceramic pin-grid array (PGA)

(3) Product Assurance Levels:
- 883B: Processed to MIL-STD-883, current revision.
- B: Fully compliant with deviation to MIL-STD-883, current revision. (Consult Altera for information on specific deviations.)
- DESC: DESC Standard Military Drawing (SMD). Consult Altera or DESC for availability.

(4) A Military Product Drawing (MPD) is prepared in accordance with the appropriate military specification format. When a Source Control Drawing (SCD) is necessary, the appropriate MPD is required for proper SCD preparation.

Table 6. Military-Qualified EP-Series Classic EPLDs

EPLD (1)	Pkg. (2)	Assurance Level (3)	t_{PD1} (ns)	f_{MAX} (MHz)	I_{CC3} (mA) Active	I_{CC1} (mA) Standby	Macrocells (Registers)	Dedicated Inputs	I/O	Number of Pins	Altera Mil. Drawing (4)
EP1810	J	B	45	33.3	240	0.9	48	16	48	68	02D-00782
EP1810	G	883B	45	33.3	240	0.9	48	16	48	68	02D-00782
8946901XX	J	DESC	45	33.3	240	0.9	48	16	48	68	
8946901YC	G	DESC	45	33.3	240	0.9	48	16	48	68	
EP1800	J	B	75	18.2	180		48	16	48	68	02D-00509
EP1800	G	883B	75	18.2	180		48	16	48	68	02D-00205
8854901YC	G	DESC	90	16.1	150		48	16	48	68	
8854902YC	G	DESC	75	18.2	180		48	16	48	68	
EP910	D	883B	40	32.3	150	0.9	24	12	24	40	02D-00935
EP910	J	B	40	32.3	150	0.9	24	12	24	44	02D-00935
EP900	D	883B	60	20.0	100		24	12	24	40	02D-00210
EP900	J	B	60	20.0	100		24	12	24	44	02D-00521
8854801QA	D	DESC	60	20.0	100		24	12	24	40	
8854801XX	J	DESC	60	20.0	100		24	12	24	40	
EP610	D	883B	35	37.0	100	0.9	16	4	16	24	02D-00522
EP610	J	883BX	35	37.0	100	0.9	16	4	16	28	02D-00522
8947601LX	D	DESC	35	37.0	100	0.9	16	4	16	24	
8947601XX	J	DESC	35	37.0	100	0.9	16	4	16	28	
EP600	D	883B	55	22.2	60		16	4	16	24	02D-00194
EP600	J	883BX	55	22.2	60		16	4	16	28	02D-00194
8686401LA	D	DESC	55	22.2	60		16	4	16	24	
8686401XX	J	DESC	55	22.2	60		16	4	16	28	
EP320	D	883B	45	30.3	40		8	10	8	20	02D-00209
EP310	D	883B	50	31.3			8	10	8	20	02D-00179
8863501RA	D	DESC	50	31.3			8	10	8	20	

Notes to Table 6:

(1) All military-qualified EPLDs are rated to military temperatures (-55° C to +125° C). Preliminary data is shown for some other parameters. Consult individual device data sheets for complete information.

(2) Package configurations:
- D: Windowed ceramic dual in-line (CerDIP)
- J: Windowed ceramic J-lead chip carrier (JLCC)
- G: Windowed ceramic pin-grid array (PGA)

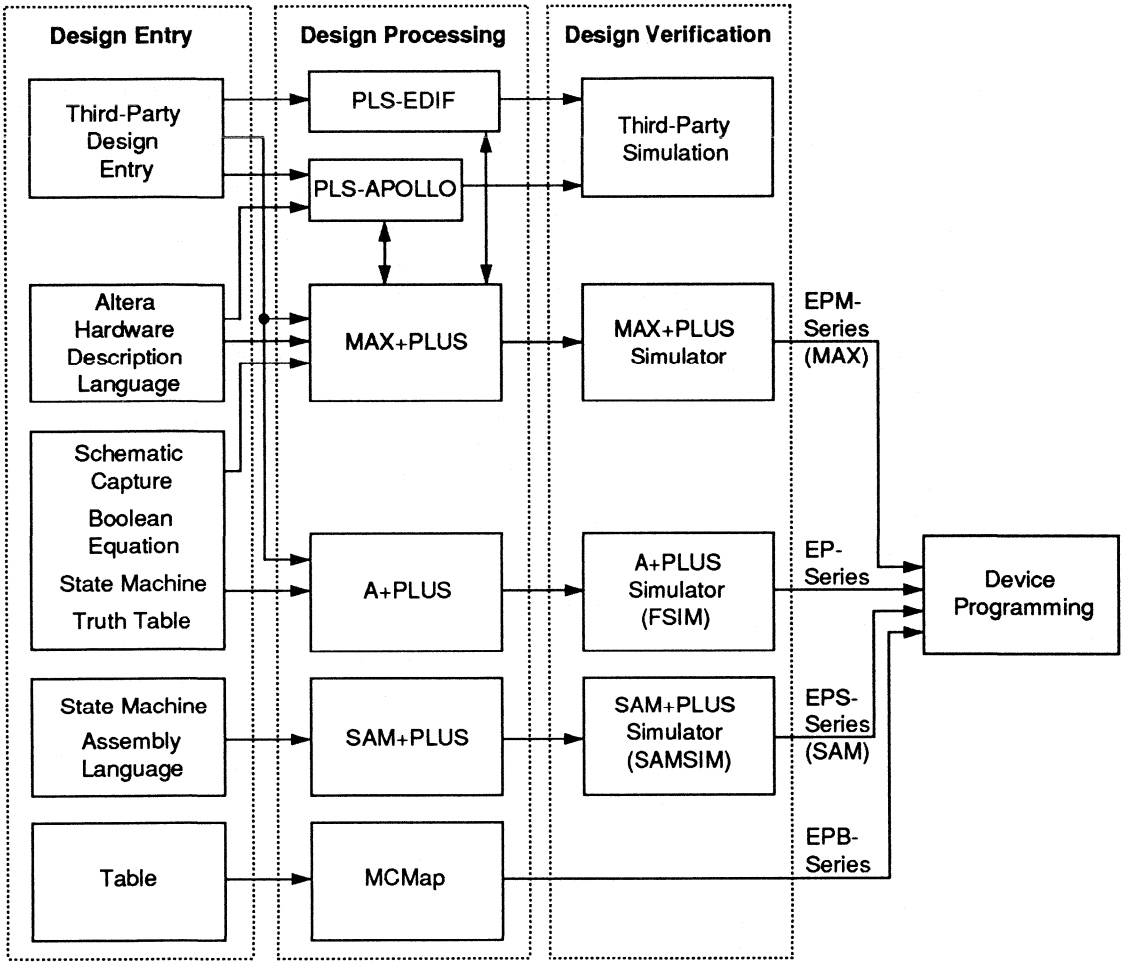
(3) Product Assurance Levels:
- 883B: Processed to MIL-STD-883, current revision.
- 883BX: Processed to MIL-STD-883, current revision with modified J-lead package dimension.
- B: Fully compliant with deviation to MIL-STD-883, current revision. (Consult Altera for information on specific deviations.)
- DESC: DESC Standard Military Drawing (SMD). Consult Altera or DESC for availability.

(4) A Military Product Drawing (MPD) is prepared in accordance with the appropriate military specification format. When a Source Control Drawing (SCD) is necessary, the appropriate MPD is required for proper SCD preparation.

Design Environment

Figure 1 shows the overall design environment provided by Altera development systems, software, hardware, and EPLDs.

Figure 1. Altera Design Environment



1

Development Systems, Software & Hardware

Table 7 shows the software and hardware products available from Altera. Programmable Logic Development Systems (with the PLDS- or PLCAD- prefix) are stand-alone combinations of hardware and software. Programmable Logic Software packages (with the PLS- prefix) are software-only products that may be used together with programming hardware from Altera (e.g., PL-ASAP) or third-party manufacturers.

Table 7. Altera Development Products

Part Number (1)	Software								Hardware	
	MAX+PLUS	A+PLUS	SAM+PLUS	MCMap	LogicMap II	PLAESW-PC	EDIF Interfaces	PLE3-12A	Logic Programmer Card (1)	Adapters (2)
PLDS-ENCORE	✔	✔	✔		✔	✔		✔	✔	✔
PLDS-MAX	✔					✔		✔	✔	✔
PLCAD-SUPREME		✔			✔	✔		✔	✔	✔
PLDS2		✔ (3)			✔	✔		✔	✔	✔
PLDS-SAM			✔		✔	✔		✔	✔	✔
PLDS-MCMAP				✔	✔			✔	✔	✔
PL-ASAP					✔			✔	✔	
PLS-MAX	✔									
PLS-SUPREME		✔			✔					
PLS-SAM			✔		✔					
PLS-MCKIT				✔	✔					✔
PLS-EDIF							✔			
PLS-APOLLO	✔ (4)						✔ (5)			

Notes to Table 7:

(1) (This note applies only to products that include Logic Programmer cards.) The part numbers shown in the left column of this table should be used to order the LP6 Logic Programmer card for IBM PC-AT and compatible computers. To order the LP5 card for IBM PS/2 Model 50, 60, 70, 80, and compatible computers, append "/PS" to the part number shown. The part numbers for individual (stand-alone) Logic Programmer cards are PLP5 and PLP6.

(2) See individual data sheets for details on adapters.

(3) Includes A+PLUS Design Processor only.

(4) Includes MAX+PLUS Compiler and MAX+PLUS TTL MacroFunction Library only.

(5) Includes EDIF netlist reader and writer customized for use with Mentor Graphics CAE tools.

PLDS-ENCORE

PLDS-ENCORE is Altera's most comprehensive EPLD development package. It contains MAX+PLUS, A+PLUS, SAM+PLUS, and LogicMap II software; standard Altera programming hardware (a Logic Programmer card and the PLE3-12A Master Programming Unit); a variety of programming adapters; and several sample EPLDs. PLDS-ENCORE provides all the tools needed to work with Altera EPM- (MAX), EP- (classic), and EPS-series (SAM) EPLDs.

PLDS-MAX & PLS-MAX

PLDS-MAX is a fully integrated programmable logic development system for working with MAX (Multiple Array MatriX) EPLDs. It includes MAX+PLUS design entry, processing, verification, and programming software; standard Altera programming hardware; an assortment of programming adapters; and several sample EPLDs. PLS-MAX is a software-only version of PLDS-MAX.

1

PLCAD-SUPREME & PLS-SUPREME

PLCAD-SUPREME is a full-featured development system for working with EP-series (classic) EPLDs. It includes basic A+PLUS software, A+PLUS software enhancements (LogiCaps schematic capture, State Machine entry, TTL MacroFunctions, Altera Design Librarian [ADLIB], Functional Simulator), LogicMap II software, standard Altera programming hardware, several programming adapters, and selected sample EPLDs. PLS-SUPREME is a software-only version of PLCAD-SUPREME.

PLDS2

PLDS2 is a basic programmable logic development system for working with EP-series EPLDs. It includes basic A+PLUS software (the Altera Design Processor only), LogicMap II software, standard Altera programming hardware, one programming adapter, and sample EPLDs.

PLDS-SAM & PLS-SAM

PLDS-SAM is a programmable logic development system for working with Stand-Alone Microsequencer (SAM) EPLDs. It includes SAM+PLUS and LogicMap II software, standard Altera programming hardware, one programming adapter, and a sample EPLD. PLS-SAM is a software-only version of PLDS-SAM.

PLDS-MCMAP & PLS-MCKIT

PLDS-MCMAP is a programmable logic development system for working with Micro Channel bus interface (EPB-series) EPLDs. It includes MCMap and LogicMap II software, standard Altera programming hardware (a Logic Programmer card and the PLE3-12A Master Programming Unit), one programming adapter, and sample EPLDs. PLS-MCKIT contains all PLDS-MCMAP components except the standard programming hardware.

MAX+PLUS

MAX+PLUS software is a fully integrated system for entering, compiling, simulating, and programming MAX EPLD designs. It features hierarchical graphical and textual design entry, with schematic capture (over 300 basic gate and TTL macrofunctions) and the Altera Hardware Description Language (AHDL) that supports state machine, Boolean equation, and truth table entry methods. The MAX+PLUS design compiler provides

logic minimization, automatic EPLD part selection, architecture optimization, and design fitting (analogous to automatic place-and-route for ASICs). MAX+PLUS software also supports automatic error location, user-defined macros, full timing simulation, delay prediction for speed-critical paths, advanced timing analysis, and a design archiver.

A+PLUS

A+PLUS software transforms input design files into standard JEDEC files for programming EP-series (classic) EPLDs. The Altera Design Processor (ADP) provides logic minimization, automatic EPLD part selection, architecture optimization, and design fitting.

Basic A+PLUS software (available in PLDS2) supports netlist and Boolean equation design entry. A+PLUS software enhancements (provided with PLDS-ENCORE, PLCAD-SUPREME, and PLS-SUPREME) include the LogiCaps schematic capture and state machine design entry methods, over 100 TTL macrofunctions, the Altera Design Librarian (ADLIB) for creating user-defined macrofunctions, and the Functional Simulator (FSIM).

SAM+PLUS

SAM+PLUS software translates state machine or microcode assembly language input design files into standard JEDEC files for programming Stand-Alone Microsequencer (SAM) EPLDs. The SAM Design Processor (SDP) provides logic minimization, architecture optimization, and design fitting. SAM+PLUS also includes SAMSIM, an interactive functional simulator created specifically for verifying SAM designs.

MCMap

MCMap software implements designs for EPB-series EPLDs, which perform all interface functions required between a PS/2 add-on card and the IBM PS/2 Micro Channel bus. It provides interactive, table-driven design entry; performs real-time error checking; and generates a JEDEC file for device programming.

LogicMap II

LogicMap II software uses standard Altera hardware and the JEDEC programming file created by A+PLUS, SAM+PLUS, and MCMap design processing to program Altera EP-series, SAM, and EPB-series EPLDs.

PLS-EDIF

PLS-EDIF is a bidirectional EDIF 2 0 0 netlist interface for transferring designs between PC-based MAX+PLUS software and PC- or workstation-based third-party design entry and logic verification tools. PLS-EDIF supports Dazix, Mentor Graphics, Valid Logic, and Viewlogic CAE tools.

PLS-APOLLO

PLS-APOLLO provides an Apollo workstation-compatible version of the popular MAX+PLUS design compiler, and a bidirectional EDIF 2 0 0 netlist interface for transferring designs between this compiler and Mentor Graphics design entry and logic verification tools. PLS-APOLLO also includes the MAX+PLUS TTL MacroFunction Library (with over 300 basic gate and TTL macrofunctions), which can be used with the Altera Hardware Description Language (AHDL) for state machine, Boolean equation, and truth table design entry.

PLAESW-PC

PLAESW-PC is a renewable, one-year software warranty agreement for PC-based development products. It provides software and documentation updates for all registered owners of Altera's PC-based development systems.

PL-ASAP

PL-ASAP (the Altera Stand-Alone Programmer) contains the hardware and software needed to set up an independent programming station capable of programming all Altera EPLDs. It includes an LP5 or LP6 Logic Programmer card, the PLE3-12A Master Programming Unit, and LogicMap II device programming software. (No design entry, processing, or simulation tools are included.)

LP5 & LP6

The LP5 and LP6 are software-controlled Logic Programmer cards for use with all Altera programmable logic development systems. The LP5 card interfaces with IBM PS/2 Model 50, 60, 70, and 80 (or compatible) personal computers; the LP6 card interfaces with IBM PC-AT (or compatible) personal computers.

PLE3-12A

The PLE3-12A Master Programming Unit serves as the base unit for programming all Altera EPLDs. It can directly program EP310, EP320, and EP330 DIP EPLDs; adapters are required to program all other EPLDs.

Adapters

Table 8 shows the adapters available for Altera EPLDs. (Adapter names consist of the four-letter prefix shown on the left, plus the corresponding number on the right.) Individual adapters can be ordered separately.

Table 8. Programming Adapters

	5192	5130	5128	5064	5032	5024	5016	1830 (1)	1810 (1)	910 (2)	630 (3)	610 (3)	330	448	2001
PLED (dual in-line package)					✔	✔	✔			✔	✔	✔	✔	✔	
PLEJ (J-lead chip carrier)	✔		✔	✔	✔		✔	✔	✔	✔	✔	✔	✔	✔	✔
PLEG (pin-grid array)	✔	✔	✔					✔	✔						
PLEQ (quad flat pack)	✔	✔													
PLES (small-outline IC)					✔	✔	✔				✔		✔		

Notes to Table 8:

(1) Programs EP1800-series EPLDs.
(2) Programs EP900-series EPLDs.
(3) Programs EP600-series EPLDs.

Notes:

Contents

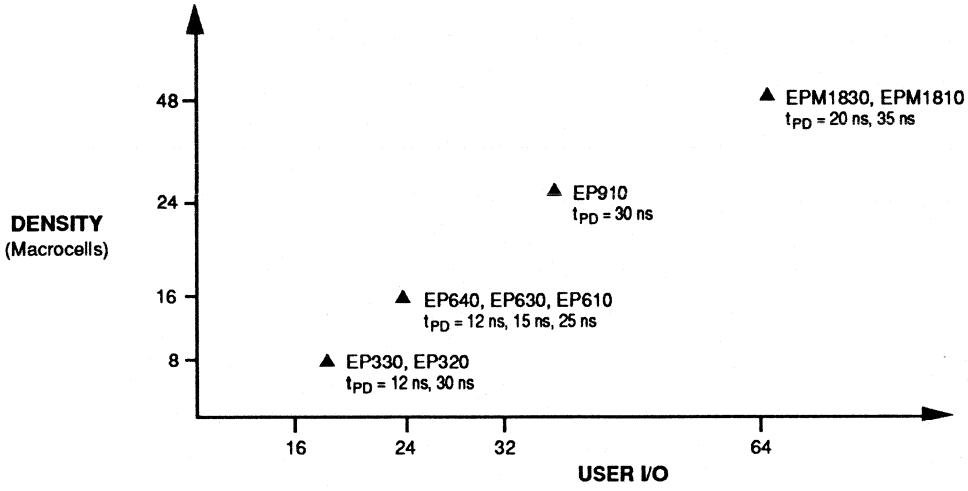
October 1990

Section 2 EP-Series Classic EPLDs

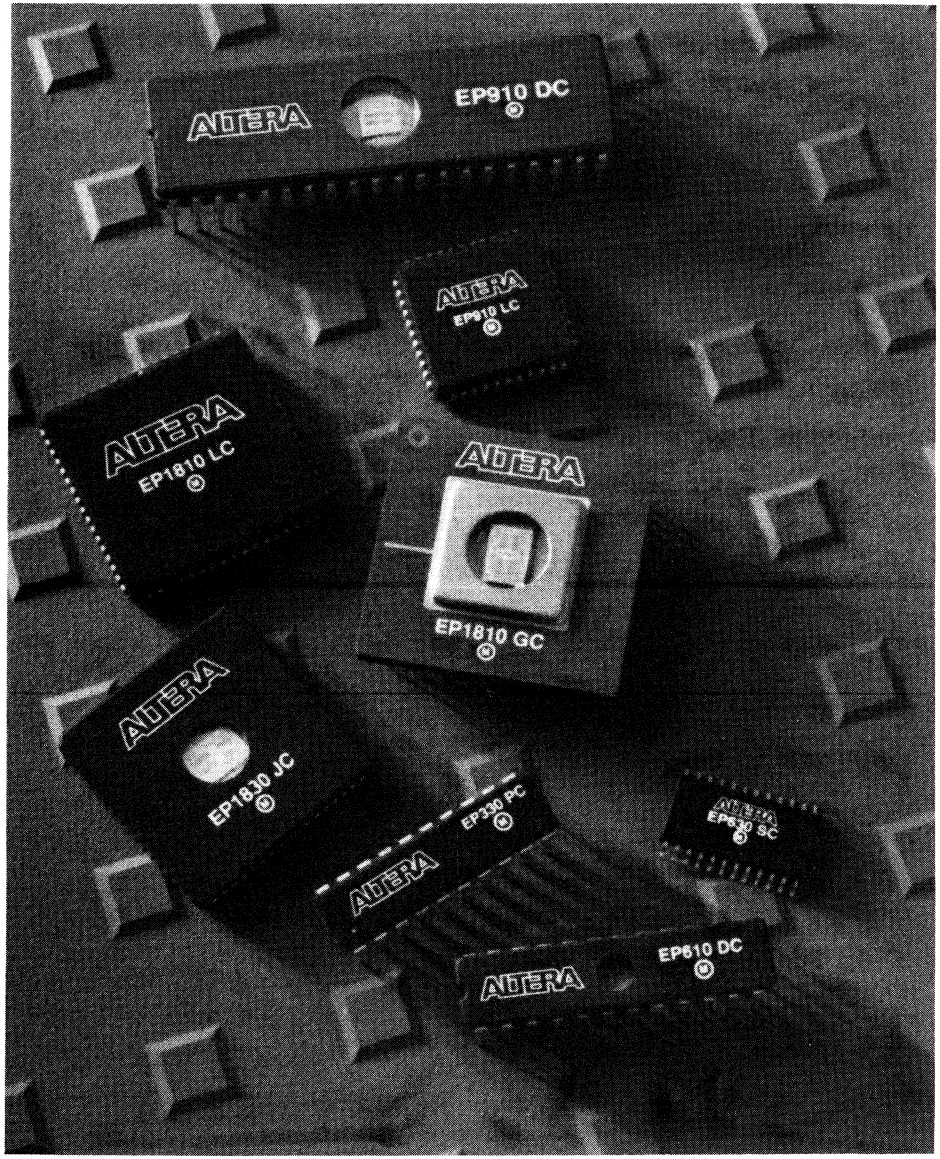
2

EP-Series EPLDs

A High-Speed, Low-Power Integration Solution



- The Altera EP-series classic EPLDs offer the industry's most comprehensive solution to high-speed, low-power logic integration.
- EP-series architecture combines the familiarity of PALs with superior macrocell and I/O flexibility.
- The EP-series structure allows designers to use the same architecture to solve a broad range of integration problems.
- In non-turbo (or standby) mode, these EPLDs consume very low power.
- Non-volatile EPROM technology aids prototype development.
- These EPLDs easily integrate multiple standard 20-pin PAL and GAL devices.
- High pin-to-macrocell ratio is ideal for pin-intensive designs.
- The series provides t_{PD} as low as 12 ns and internal counter rates as high as 100 MHz.
- A full selection of packages is available, including DIP, J-lead, PGA, and SOIC footprints in windowed ceramic and plastic one-time-programmable (OTP) chip-carrier packages.
- EP1800-series designs can be easily converted to custom masked silicon for very-high-volume production.
- EP-series EPLDs are supported with A+PLUS design tools that allow design entry, compilation, simulation, and programming on an IBM PC-AT or PS/2 (and compatible) computer.
- Multiple design entry options are available, including schematic capture, truth table, state machine, Boolean equation, and netlist.
- Logic compilation and fitting is performed in minutes.
- Extensive third-party support is available for design entry, compilation, and programming.


ALTERA
EP910 DC
ALTERA
EP910 LC
ALTERA
EP1810 LC
ALTERA
EP1810 GC
ALTERA
EP1830 JC
ALTERA
EP330 PC
ALTERA
EP630 SC
ALTERA
EP610 DC

EP300-Series EPLDs

High-Performance 8-Macrocell Devices

October 1990, ver. 1 **Data Sheet**

Features

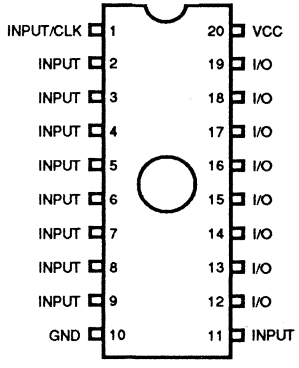
- Low-power, direct replacement for GAL 16V8 and most 20-pin PAL devices
- 8 macrocells with configurable I/O architecture, allowing up to 18 inputs and 8 outputs
- High speed (EP330 t_{PD} = 12 ns)
- EP320 offers "zero power"(typically 10 µA standby)
- 100% generically testable to provide 100% programming yield
- Available in 20-pin windowed ceramic DIP, and plastic DIP, J-lead, and SOIC packages
- A+PLUS software support featuring schematic capture, Boolean equation, state machine, truth table, and netlist design entry methods
- Extensive third-party software and programming support

2

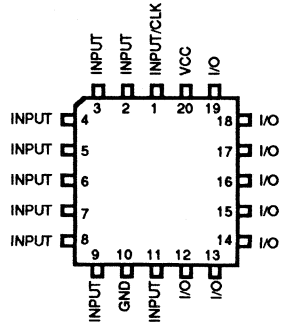
General Description

Altera's EP300-series Erasable Programmable Logic Devices (EPLDs) provides a high-speed, low-power pin-compatible replacement for 20-pin programmable logic devices such as PALs and GALs. EP300-series EPLDs are available in 20-pin windowed ceramic DIP, and plastic one-time-programmable (OTP) DIP, J-lead (PLCC), and SOIC packages. See Figure 1.

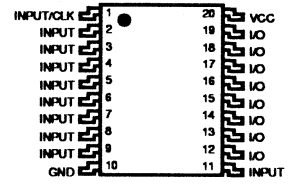
Figure 1. Package Pin-Out Diagrams *Package outlines not drawn to scale.*



DIP



J-Lead



SOIC

DIP	J-Lead	SOIC
Ceramic/Plastic	Plastic	Plastic
EP330 EP320	EP330	EP330

EP300-series EPLDs can accommodate up to 18 inputs and 8 outputs. Each of the 8 macrocells contains a programmable-AND/fixed-OR structure that implements logic with up to 8 product terms. An additional product term in each macrocell controls Output Enable.

Altera's proprietary programmable I/O architecture allows the designer to program output and feedback paths for combinatorial or registered operation in active-high and active-low modes. Thus, EP300-series devices may be configured as drop-in replacements for PAL and GAL devices such as the 16R8 and 16V8. See *Application Note 2 (Replacing 20-Pin PAL and GAL Devices with EP300-Series EPLDs)* in this data book for more information.

The EP300-series CMOS EPROM technology reduces active power consumption to less than 50% of the power required by equivalent bipolar devices, without sacrificing speed. This reduced power consumption makes these EPLDs highly desirable for a wide range of applications. EP300-series EPLDs are 100% generically testable and can be erased with UV light. Designs and design modifications can be implemented quickly, eliminating the need for post-programming testing.

EP300-series EPLDs are programmed with Altera's A+PLUS Development System, which supports schematic capture, Boolean equation, state machine, and netlist design entry. After the design is entered, A+PLUS automatically translates it into logic equations, performs Boolean minimization, and fits it into the EPLD. The device is then programmed in seconds at the designer's desktop to create customized working silicon. In addition, extensive third-party support exists for design entry, design processing, and device programming.

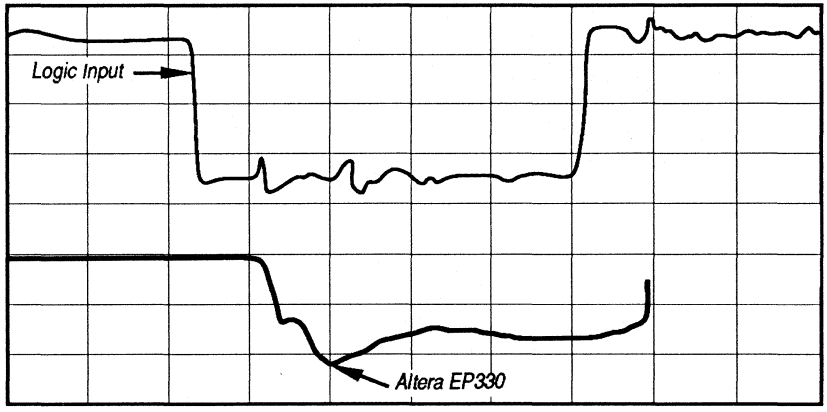
EP300-Series EPLDs

The EP300 series includes the EP330 and EP320 EPLDs. The EP330 and EP320 are JEDEC-file-compatible, allowing a single JEDEC file to be used for programming either device.

EP330

The EP330 combines high performance with low noise. Figure 2 shows the switching performance of the EP330-12. The EPLD's "quiet" outputs allow designs to run fast with high system reliability. In addition, enhanced output current capability (I_{OL} = 24 mA) allows the EP330 to directly integrate designs requiring high-current drive, such as bus interfaces. The EP330 is available with t_{PD} values of 12 ns and 15 ns.

Figure 2. EP330 Output Switching Performance



Timebase = 10.0 ns/division

Channel 1 = 1.000 V/division
Channel 2 = 2.000 V/division

EP320

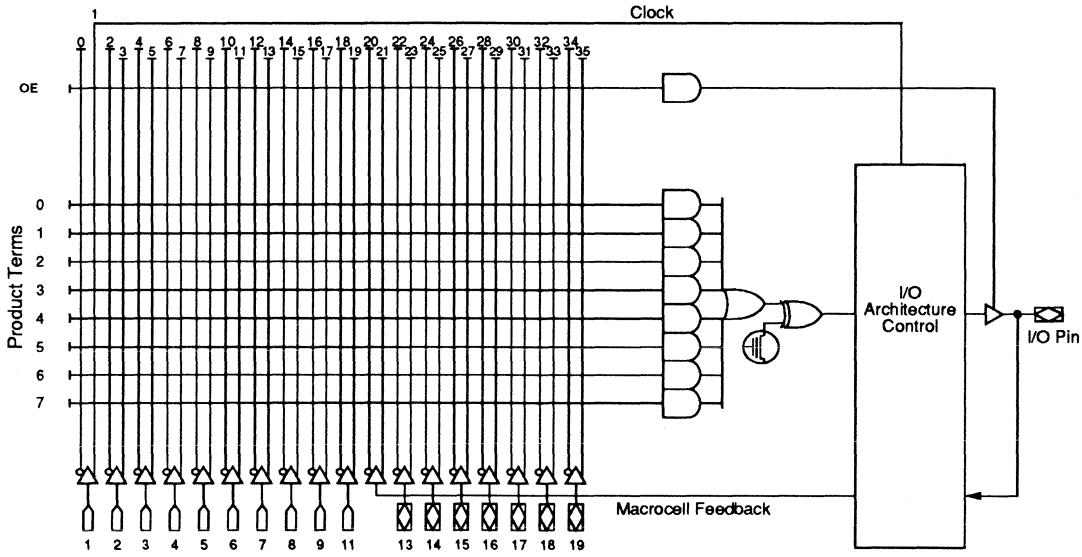
The EP320 is a zero-power EPLD. It typically draws 10 µA when operating in standby mode and 3 mA when operating at 1 MHz. MIL-STD-883B-compliant parts are available. The EP320 is available with t_{PD} values of 30 ns, 35 ns, and 45 ns.

Functional Description

EP300-series EPLDs use CMOS EPROM technology to configure connections in a programmable-AND logic array. EPROM connections are also used to control the output/feedback options, such as registered or combinatorial feedback, in active-high or active-low modes.

Devices in the EP300 series have 10 dedicated data inputs and 8 I/O pins that can be configured for input, output, or bidirectional operation. Figure 3 shows the EP300-series macrocell.

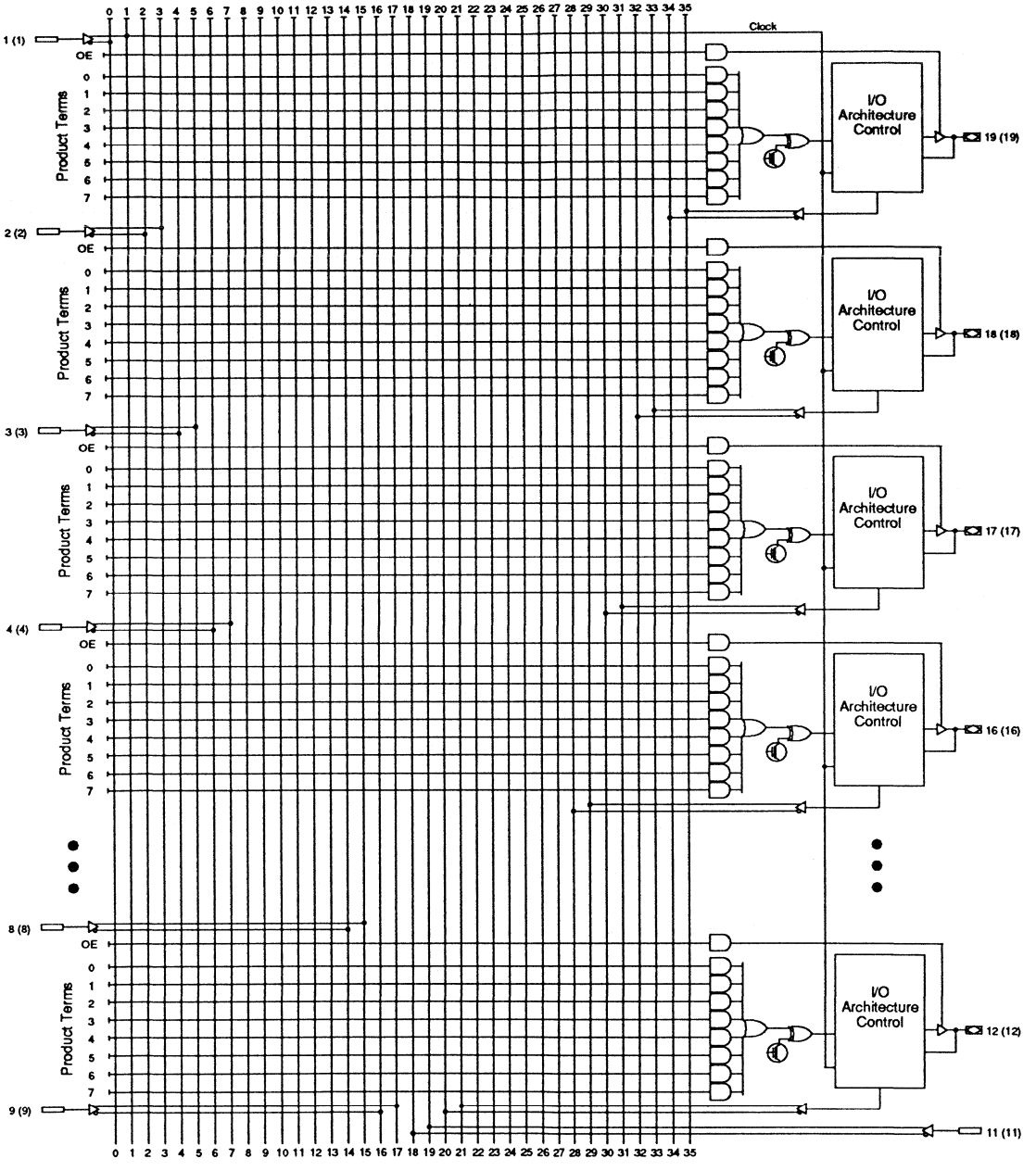
Figure 3. Logic Array Macrocell



The EP330 block diagram is shown in Figure 4. The internal architecture of this device has a sum-of-products (AND/OR) structure. Inputs to the programmable-AND array come from the true and complement signals of the dedicated input pins, and the true and complement forms of the 8 feedback signals that come from the I/O architecture control blocks. The 36-input AND array has 72 product terms distributed equally among the 8 macrocells. Each product term represents a 36-input AND gate.

The outputs of eight product terms are ORed together; then the output of the OR gate is fed as an input to an XOR gate. The XOR function allows the designer to use the invert-select EPROM cell to specify the polarity of the output signal. If the EPROM cell is programmed, the true form of the signal (active high) is passed; if not, the complement of the signal (active low) is passed. The XOR output then feeds the I/O architecture control block, in which the output is configured for registered or combinatorial operation. In registered mode, the output is registered via a positive-edge-

Figure 4. EP300-Series Block Diagram *Numbers in parentheses are for J-lead packages.*



triggered D-type flip-flop. The feedback signal from the output of the flip-flop is also registered. In combinatorial mode, the output is not registered, and the feedback signal comes directly from the I/O pin.

Output Enable Product Term

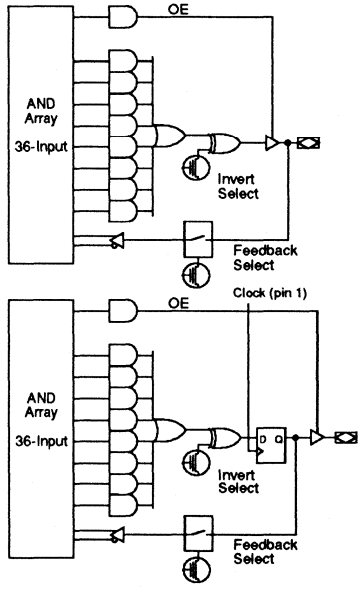
The Output Enable (OE) product term determines whether an output signal is allowed to propagate to the output pin. If the output of the OE product term is high, output to the pin is enabled. If the output is low, the output buffer becomes a high-impedance node and does not allow the output signal to reach the output pin. The I/O pin can then be used as a dedicated input. This OE product term allows true bidirectional operation in combinatorial mode.

EP300-series devices contain eight OE product terms, one for each I/O pin. All outputs can be enabled or disabled simultaneously by using an identically programmed product term at each of the outputs. Outputs can be enabled under other conditions by defining a different OE product term for each output.

I/O Architecture

Figure 5 shows the output configurations available for the eight I/O pins. Both registered and combinatorial outputs may be individually specified for each macrocell. Any I/O pin can be configured as a dedicated input by choosing no output and pin feedback.

Figure 5. I/O Configurations



Combinatorial

I/O Selection

Output/Polarity	Feedback
Combinatorial/High	Pin, None
Combinatorial/Low	Pin, None
None	Pin

Registered

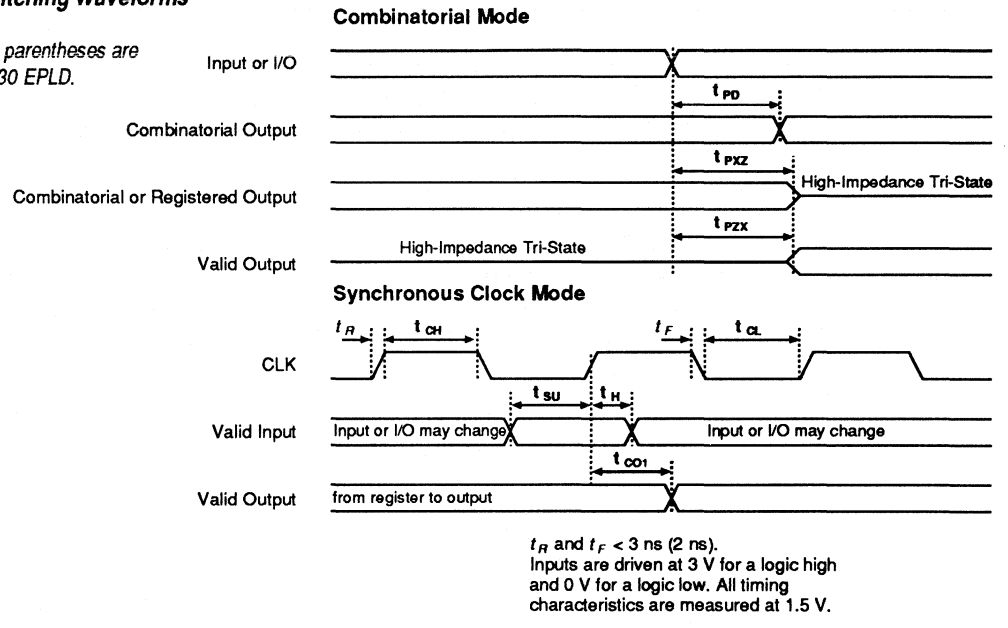
I/O Selection

Output/Polarity	Feedback
D Register/High	D Register, None
D Register/Low	D Register, None
None	D Register

In combinatorial mode, active-high or active-low output polarity with pin feedback or no feedback can be chosen. In registered mode, active-high or active-low output polarity with the internal registered feedback or no feedback are available. In the erased state, I/O architecture is configured for combinatorial active-low output with pin feedback.

The switching waveforms for EP300-series EPLDs are shown in Figure 6.

Figure 6. Switching Waveforms



t_R and t_F < 3 ns (2 ns).
Inputs are driven at 3 V for a logic high and 0 V for a logic low. All timing characteristics are measured at 1.5 V.

Functional Testing

EP300-series EPLDs are fully functionally tested and guaranteed through complete testing of each EPROM bit and all internal logic elements. This testing ensures a 100% programming yield.

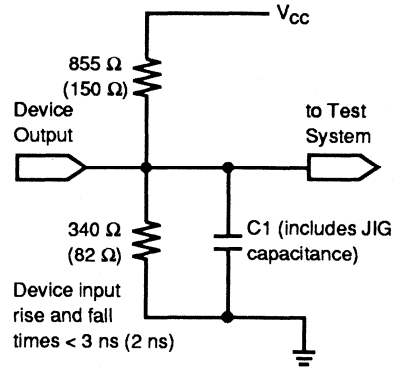
This testing process eliminates traditional problems associated with fuse-programmed circuits. EP300-series EPLDs allow test programming patterns to be used and then erased. The ability to use application-independent, general-purpose tests is called generic testing and is unique to EPLDs.

AC test measurements are performed at the conditions shown in Figure 7.

Figure 7. AC Test Conditions

Power supply transients can affect AC measurements. Simultaneous transitions of multiple outputs should be avoided for accurate measurement. Threshold tests must not be performed under AC conditions. Large-amplitude, fast-ground current transients normally occur as the device outputs discharge the load capacitances. When these transients flow through the parasitic inductance between the device ground pin and the test system ground, it can create significant reductions in observable input noise immunity.

Note: Numbers in parentheses are for the EP330 EPLD.



Design Security

EP300-series EPLDs contain a programmable Security Bit that controls access to the data programmed into the device. If this feature is used, a proprietary design implemented in the device cannot be copied or retrieved. This feature provides a high level of design security by making programmed data within EPROM cells invisible. The Security Bit, along with all other program data, is reset by erasing the device.

Turbo Bit

The EP320 contains a programmable Turbo Bit, set with the A+PLUS software, to control the automatic power-down feature that enables the device's low standby-power mode. When the Turbo Bit is programmed (Turbo = On), the low standby-power mode (I_{CC1}) is disabled, making the circuit less sensitive to V_{CC} noise transients created by the low-power mode power-up/power-down cycle. Typical I_{CC} versus frequency data for both turbo and non-turbo (low power) mode is shown in each EPLD data sheet. All AC values are tested with the Turbo Bit programmed.

If the design requires low-power operation, the Turbo Bit should be disabled (Turbo = Off). When the device is operating in this mode, some AC parameters may increase. To determine worst-case timing, values given in the AC Non-Turbo Adder specifications must be added to the corresponding AC parameter.

Features

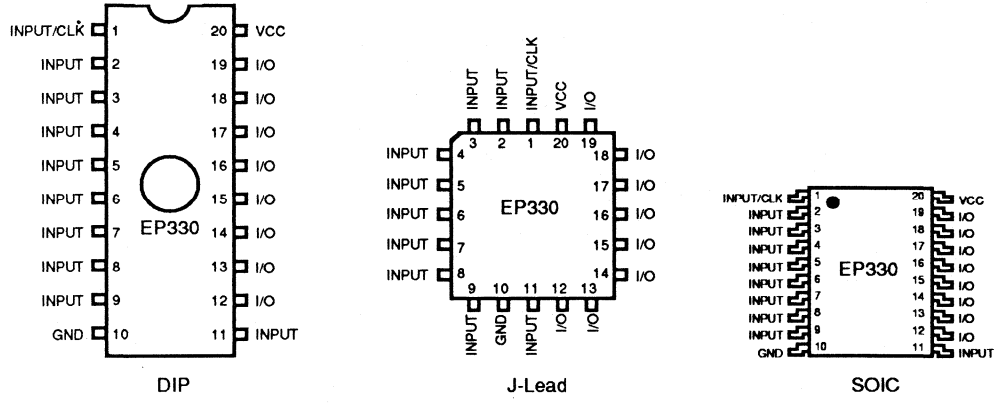
- High-performance 8-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 12 ns
  - Counter frequencies up to 100 MHz
  - Pipelined data rates up to 125 MHz
- Low power; I_{CC} = 45 mA (typical) for an 8-bit counter at 1 MHz
- Available in windowed ceramic and plastic one-time-programmable chip carrier packages
  - 20-pin DIP (ceramic and plastic)
  - 20-pin J-lead (plastic)
  - 20-pin, 300-mil SOIC (plastic)
- Macrocell flip-flops can be individually programmed for registered or combinatorial operation
- "Quiet" outputs minimize output switching noise found in other high-speed CMOS devices

2

Figure 8 shows the pin-outs for the EP330 EPLD.

Figure 8. EP330 Pin-Out Diagrams

Package outlines not drawn to scale.



Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	14.0	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–160	+160	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			800	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75	5.25	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time	*See Note (2)*		20	ns
t_F	Input fall time	*See Note (2)*		20	ns

DC Operating Conditions *See Note (3)*

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –12 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –12 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 24 mA DC			0.5	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND, No load		40	75	mA
I_{CC3}	V_{CC} supply current (active)	V_I = V_{CC} or GND, No load, f = 1.0 MHz, *See Note (4)*		45	75	mA

Capacitance *See Note (5)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		15	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

			EP330-12		EP330-15		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Unit
t_{PD1}	Input to non-registered output	C_1 = 35 pF		12		15	ns
t_{PD2}	I/O input to non-registered output			13		16	ns
t_{PZX}	Input to output enable			12		15	ns
t_{PXZ}	Input to output disable	C_1 = 5 pF, *See Note (6)*		12		15	ns
t_{IO}	I/O input pad and buffer delay			1		1	ns

Synchronous Clock Mode

			EP330-12		EP330-15		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Unit
f_{MAX}	Maximum clock frequency	*See Note (7)*	125		100		MHz
t_{SU}	Input setup time		6		8		ns
t_H	Input hold time		0		0		ns
t_{CH}	Clock high time		4		5		ns
t_{CL}	Clock low time		4		5		ns
t_{CO1}	Clock to output delay			8		10	ns
t_{CNT}	Minimum clock period			10		12	ns
f_{CNT}	Internal maximum frequency	*See Note (4)*	100		83.3		MHz

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) For all clocks: t_R and t_F = 20 ns.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) Measured with a device programmed as an 8-bit counter.
(5) Capacitance measured at 25° C. Sample-tested only. Pin 11 (high-voltage pin during programming) has maximum capacitance of 20 pF.
(6) Sample-tested only for an output change of 500 mV.
(7) The f_{MAX} values represent the highest frequency for pipelined data.

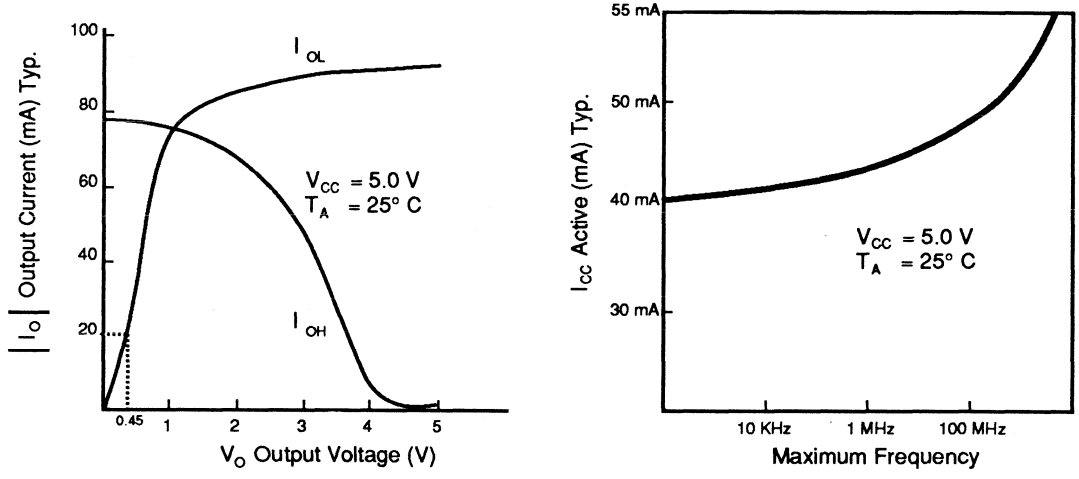
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP330-12, EP330-15
Industrial	(–40° C to 85° C)	Consult factory
Military	(–55° C to 125° C)	Consult factory

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera's Marketing Department by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 9 shows output drive characteristics for EP330 I/O pins and typical supply current versus frequency for the EP330.

Figure 9. EP330 Output Drive Characteristics and I_{CC} vs. Frequency



Features

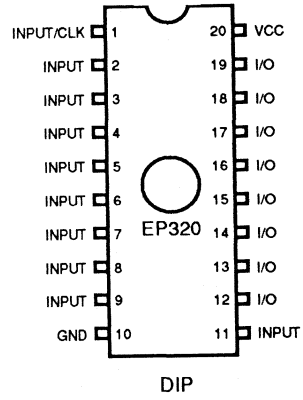
- High-performance 8-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 30 ns
  - Counter frequencies up to 28.6 MHz
  - Pipelined data rates up to 45.5 MHz
- Very low power
  - I_{CC} = 3 mA (typical) for an 8-bit counter at 1 MHz
  - I_{CC} = 10 µA (typical) in standby mode
- Available in 20-pin windowed ceramic and plastic, one-time-programmable dual in-line packages (DIPs)
- Macrocell flip-flops can be individually programmed for registered or combinatorial operation

Figure 10 shows pin-outs for the EP320 EPLD.

2

Figure 10. EP320 Pin-Out Diagram

Package outline not drawn to scale.



DIP

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–80	+80	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			400	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	*See Note (2)*	4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time	*See Note (3)*		500	ns
t_F	Input fall time	*See Note (3)*		500	ns

DC Operating Conditions V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
See Note (4) V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –8 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –4 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 8 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND No load, *See Note (5)*		10	150	µA
I_{CC2}	V_{CC} supply current (non-turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (6)*		3	5 (15)	mA
I_{CC3}	V_{CC} supply current (turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (6)*		18	30 (40)	mA

Capacitance *See Note (7)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		10	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Timing Parameters			EP320-1		EP320-2		EP320		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	*Note (8)*	Unit
t_{PD1}	Input to non-registered output	C1 = 50 pF		29		34		44	15	ns
t_{PD2}	I/O input to non-registered output			30		35		45	15	ns
t_{PZX}	Input to output enable			30		35		45	15	ns
t_{PXZ}	Input to output disable	C1 = 5 pF *Note (9)*		30		35		45	15	ns
t_{IO}	I/O input pad and buffer delay			1		1		1	0	ns

Synchronous Clock Mode

Timing Parameters			EP320-1		EP320-2		EP320		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	*Note (8)*	Unit
f_{MAX}	Maximum clock frequency	*Note (10)*	45.5		40		30.3		0	MHz
t_{SU}	Input setup time		22		25		33		15	ns
t_{H}	Input hold time		0		0		0		0	ns
t_{CH}	Clock high time		10		12		16		0	ns
t_{CL}	Clock low time		10		12		16		0	ns
t_{CO1}	Clock to output delay			17		20		25	0	ns
t_{CNT}	Minimum clock period			35		40		50	0	ns
f_{CNT}	Internal maximum frequency	*Note (6)*	28.6		25		20		0	MHz

2

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature versions.
(3) For all clocks: t_R and t_F = 250 ns (100 ns).
(4) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(5) When in non-turbo mode, an EPLD will automatically enter standby mode if logic transitions do not occur (approximately 100 ns after the last transition).
(6) Measured with a device programmed as an 8-bit counter.
(7) Capacitance measured at 25° C. Sample-tested only. Clock-pin capacitance for dedicated clock inputs only. Pin 11 (high-voltage pin during programming) has a maximum capacitance of 20 pF.
(8) See "Turbo Bit" in this data sheet.
(9) Sample-tested only for an output change of 500 mV.
(10) The f_{MAX} values represent the highest frequency for pipelined data.

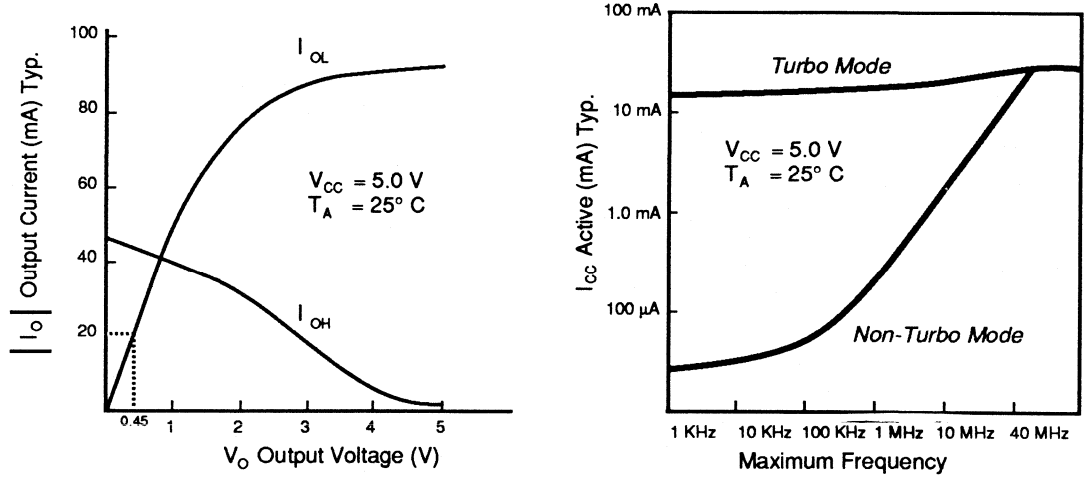
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP320-1, EP320-2, EP320
Industrial	(–40° C to 85° C)	EP320
Military	(–55° C to 125° C)	EP320

Note: Only military-temperature-range devices are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera's Marketing Department by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 11 shows output drive characteristics for EP320 I/O pins and typical supply current versus frequency for the EP320.

Figure 11. EP320 Output Drive Characteristics and I_{CC} vs. Frequency



EP600-Series EPLDs

High-Performance 16-Macrocell Devices

October 1990, ver. 1

Data Sheet

Features

- High-density replacement for TTL and 74HC with up to 600 gates
- EP630 and EP610 offer "zero power" (typically 20 µA standby)
- Very high speed (EP610A t_{PD} = 12 ns)
- Advanced CMOS EPROM technology to allow device erasure and reprogramming
- Asynchronous clocking of all registers or banked register operation from two synchronous clocks
- 16 macrocells with configurable I/O architecture, allowing up to 20 inputs and 16 outputs
- Individually programmable registers providing D, T, SR, or JK flip-flops with individual asynchronous Clear control
- 100% generically testable to provide 100% programming yield
- Programmable Security Bit for total protection of proprietary designs
- A+PLUS software support featuring schematic capture, Boolean equation, state machine, truth table, and netlist design entry methods
- Available in space-saving windowed ceramic and plastic 24-pin, 300-mil DIP and 28-pin J-lead packages, or plastic 24-pin, 300-mil SOIC packages
- Extensive third-party software and programming support

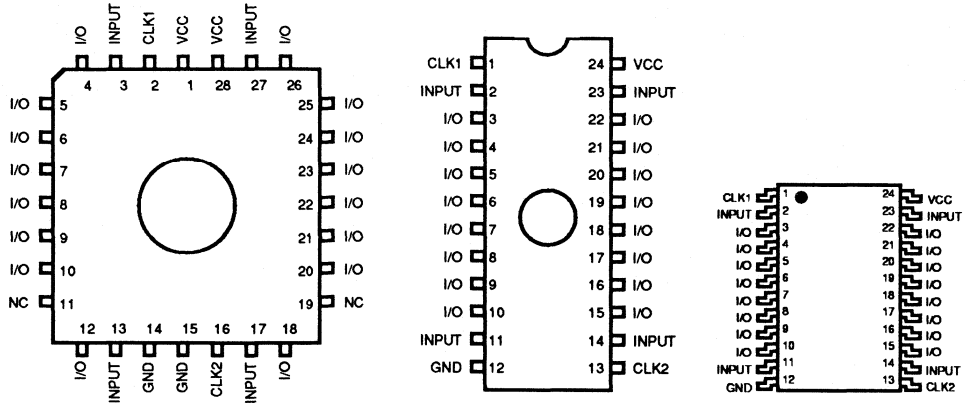
General Description

Altera's EP600-series Erasable Programmable Logic Devices (EPLDs) can implement up to 600 equivalent gates of SSI and MSI logic functions in space-saving windowed ceramic or one-time-programmable (OTP) 24-pin, 300-mil DIP and 28-pin J-lead (JLCC and PLCC) packages, or OTP plastic 24-pin, 300-mil SOIC packages. See Figure 1.

EP600-series EPLDs use sum-of-products logic that provides a programmable-AND/fixed-OR structure. These EPLDs accommodate combinatorial and sequential logic functions with up to 20 inputs and 16 outputs. Altera's proprietary programmable I/O architecture allows the designer to program output and feedback paths for combinatorial or registered operation in active-high and active-low modes.

EP600-series EPLDs can individually program D, T, SR, or JK flip-flop operation for each output without sacrificing product terms. In addition, each register can be individually clocked from any of the input or feedback paths in the AND array. These features make it possible to simultaneously implement a variety of logic functions.

Figure 1. Package Pin-Out Diagrams *Package outlines not drawn to scale.*



J-Lead DIP SOIC

J-Lead	DIP	SOIC
Ceramic/Plastic	Ceramic/Plastic	Plastic
EP610A EP630 EP610	EP610A EP630 EP610	EP610A EP630 EP610

The CMOS EPROM technology in EP600-series EPLDs can reduce active power consumption to less than 40% of the power required by equivalent bipolar devices, without losing speed. This reduced power consumption makes the EP600-series EPLDs highly desirable for a wide range of applications. Moreover, these EPLDs are 100% generically testable and can be erased with UV light. Designs and design modifications can be implemented quickly, eliminating the need for post-programming testing.

Logic is implemented with Altera's A+PLUS Development System, which supports schematic capture, Boolean equation, state machine, truth table, and netlist design entry methods. After the design is entered, A+PLUS automatically translates it into logic equations, performs Boolean minimization, and fits it into the EPLD. The device may then be programmed in seconds at the designer's desktop to create customized working silicon. In addition, extensive third-party support exists for design entry, design processing, and device programming.

EP600-Series EPLDs

The EP600 series includes the EP610A, EP630, and EP610 EPLDs. These EPLDs are JEDEC-file-compatible, allowing a single JEDEC file to be used for progamming any of the EPLDs.

EP610A

The EP610A is fastest member of the EP600 series. It has an input-to-non-registered-output delay (t_{PD}) of 12 ns, which is ideal for address decoding. The EP610A offers a 36% faster clock-to-output delay (t_{CO} = 6 ns) than a CMOS 22V10 and can easily integrate logic operating at today's faster system speeds. The EP610A is fabricated on an advanced 0.8-micron process, and supports 16-bit counter frequencies of up to 83 MHz.

EP630

The EP630 is fast and offers a low-power standby mode. This EPLD can implement a 16-bit counter at up to 83 MHz, and typically consumes 45 mA when operating at 1 MHz. It offers 60% more logic and 6 more flip-flops than the 22V10. It is fabricated on a 1-micron process, and is available with maximum t_{PD} values of 15 ns and 20 ns.

EP610

The EP610 combines high speed with low power. It can implement a 16-bit counter at up to 40 MHz, and typically consumes 32 mA when operating at 1 MHz. The EP610 is fabricated on a 1.2-micron process and is available in all temperature ranges. Both MIL-STD-883B-compliant and DESC-approved parts are available. The EP610 has maximum t_{PD} values of 25 ns, 30 ns, 35 ns, and 40 ns.

Functional Description

EP600-series EPLDs use CMOS EPROM technology to configure connections in a programmable-AND logic array. EPROM connections are also used to construct a highly flexible programmable I/O architecture that provides advanced functions for user-programmable logic.

EP600-series EPLDs have 4 dedicated data inputs, 2 synchronous clock inputs, and 16 I/O pins that can be configured for input, output, or bidirectional operation on a macrocell-by-macrocell basis.

Figure 2 shows the EP600-series macrocell. Each macrocell contains 10 product terms for the following functions: 8 product terms are dedicated to logic implementation; 1 product term is used for Clear control of the internal register; and 1 product term implements either Output Enable or an asynchronous Clock.

Figure 2. Logic Array Macrocell

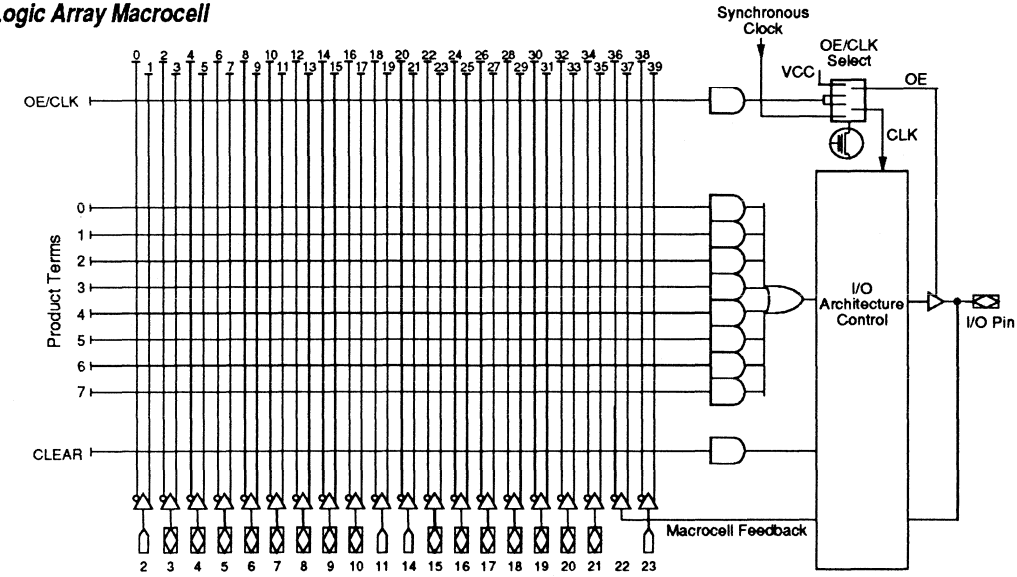
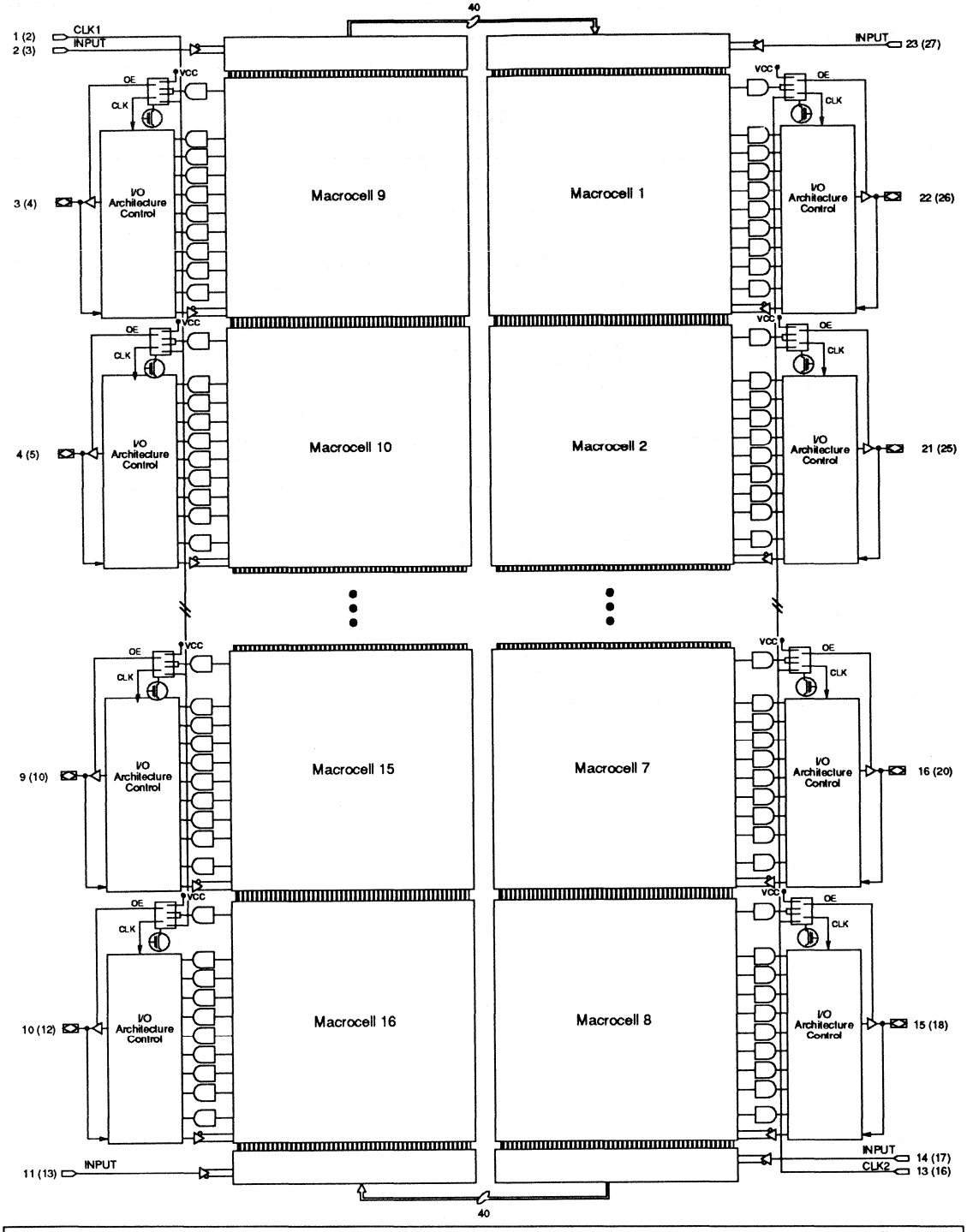


Figure 3 shows the complete block diagram of an EP600-series EPLD. The internal device architecture has a sum-of-products (AND/OR) structure. Inputs to the programmable AND array come from the true and complement signals of the 4 dedicated data inputs and 16 I/O feedback signals. The 40-input AND array has 160 product terms distributed among the 16 macrocells. Each product term represents a 40-input AND gate.

In the erased state, the true and complement of the AND-array inputs are connected to the product terms. An EPROM control cell is located at each intersection of an AND-array input and a product term. During programming, selected connections are opened, allowing any product

Figure 3. EP600-Series Block Diagram *Numbers in parentheses are for J-lead packages.*



term to be connected to a true or complement array input signal with the following results:

- If both the true and complement of an array input signal are connected, the output of the AND gate is a logic low.
- If both the true and complement of any array input signal are programmed "open," a logic "don't care" results for that input.
- If all inputs for a given product term are programmed "open," the output of the corresponding AND gate is a logic high.

Two dedicated clock inputs (which are not available in the AND array) provide the signals used for synchronous clocking of EP600-series internal registers. Each signal is positive-edge-triggered and has control over 8 registers: **CLK1** controls macrocells 9 to 16; **CLK2** controls macrocells 1 to 8. The programmable I/O architecture allows each of the 16 internal registers to have a synchronous or asynchronous Clock.

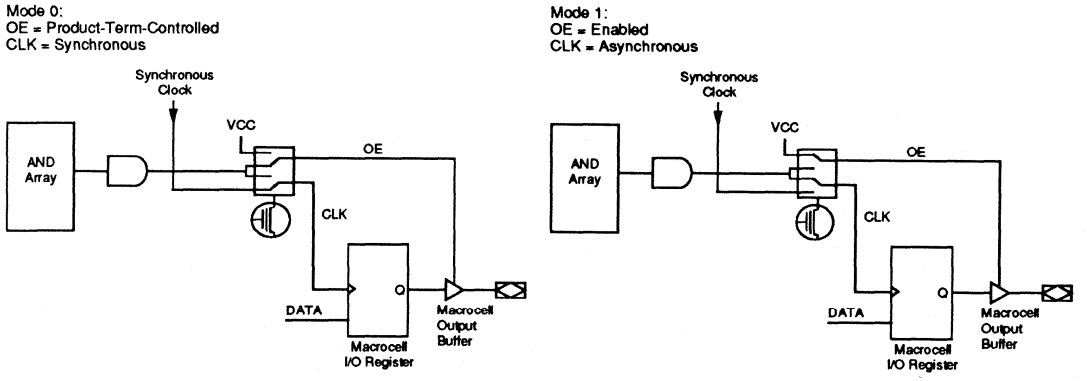
I/O Architecture

The EP600-series architecture provides each macrocell with over 50 programmable I/O configurations. Each macrocell can be configured for combinatorial or registered output, with programmable output polarity. One of four register types (D, T, JK, and SR) may be implemented in each macrocell without additional logic. I/O feedback selection can be programmed for registered or input feedback. The I/O architecture can also individually clock each internal register from any internal signal.

OE/CLK Selection

Figure 4 shows the two modes of operation provided by the OE/CLK Select multiplexer. This multiplexer, which is controlled by a single EPROM control bit, may be individually configured at each I/O pin.

Figure 4. OE/CLK Select Multiplexer



The register is clocked by the synchronous clock signal, which is common to seven other macrocells. The output is enabled by the logic from the product term.

The output is permanently enabled and the register is clocked by the product term, which allows gated clocks to be generated in EP600-series EPLDs.

In Mode 0, the tri-state output buffer is controlled by a single product term. If the output of the AND gate is high, then the output buffer is enabled. If the output is low, the output buffer has a high-impedance value. In this mode, the macrocell flip-flop is clocked by its synchronous Clock input signal (**CLK1** or **CLK2**). In the erased state, the OE/CLK Select multiplexer is configured to Mode 0.

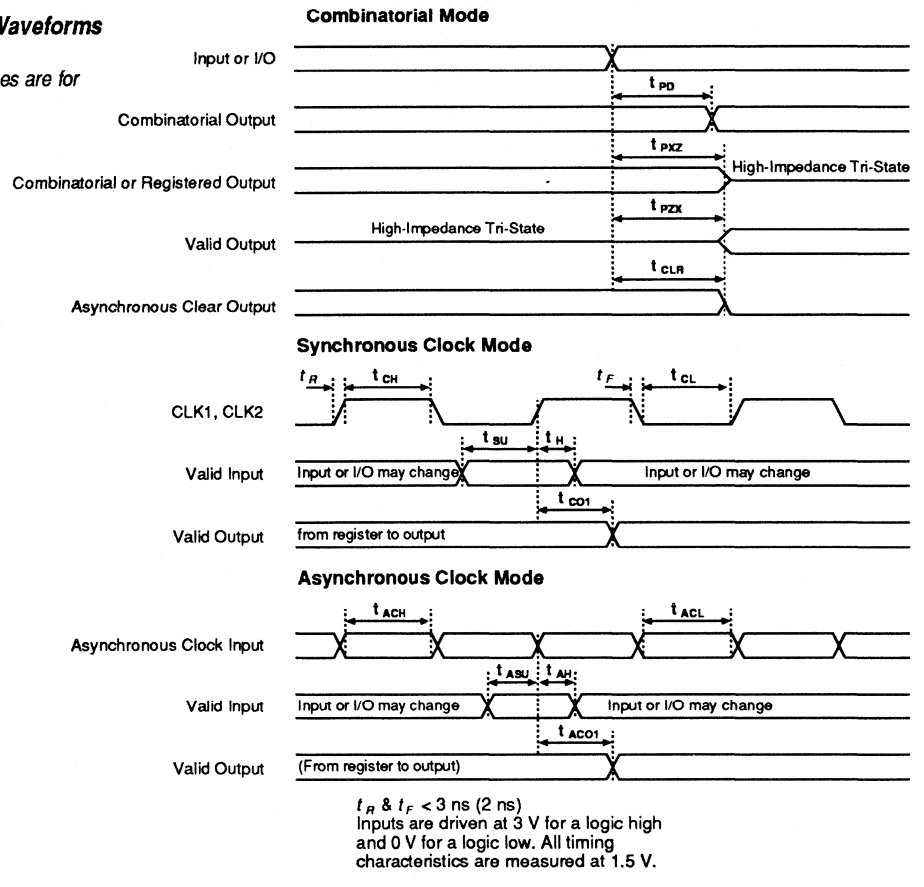
In Mode 1, the Output Enable buffer is always enabled, allowing the macrocell flip-flop to be triggered from an asynchronous Clock signal generated by the OE/CLK product term. This mode allows flip-flops to be individually clocked from any of the AND-array input signals. With true and complement signals in the AND array, the flip-flop can be configured to trigger on a rising or falling edge. This product-term-controlled clock configuration also allows implementation of gated clock structures.

Figure 5 shows waveforms for the following modes: combinatorial, synchronous Clock, and asynchronous Clock.

2

Figure 5. Switching Waveforms

Numbers in parentheses are for the EP610A EPLD.



t_R & t_F < 3 ns (2 ns)
Inputs are driven at 3 V for a logic high and 0 V for a logic low. All timing characteristics are measured at 1.5 V.

Output/ Feedback Selection

Output configurations available with EP600-series EPLDs are shown in Figure 6. Each macrocell can be individually configured with combinatorial output or with any of the four register outputs. All registers have an individual asynchronous Clear function controlled by a dedicated product term. When this product term is a logic high, the macrocell register is immediately loaded with a logic low. The Clear function is performed automatically during power-up.

The combinatorial configuration has eight product terms ORed together to generate the output signal. This configuration has the following characteristics:

- The Invert-Select EPROM bit controls output polarity.
- One product term controls the Output Enable buffer.
- The Feedback-Select multiplexer allows the user to choose I/O (pin) feedback or no feedback to the AND array.

The D or T register has eight product terms ORed together that are available to the register input. This configuration has the following characteristics:

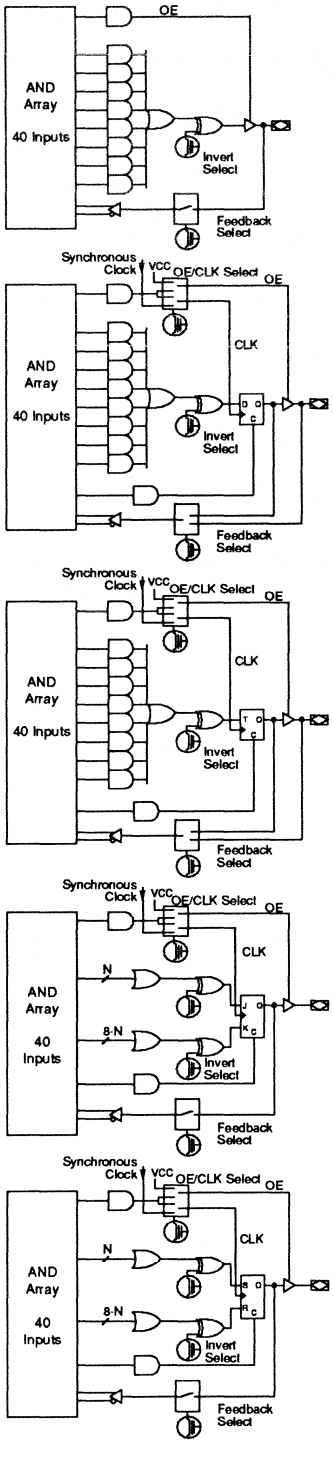
- The Invert Select EPROM bit controls output polarity.
- One product term controls asynchronous Clear.
- The OE/CLK Select multiplexer configures the mode of operation to Mode 0 or Mode 1.
- The Feedback Select multiplexer allows the user to choose registered feedback, I/O feedback, or no feedback to the AND array.

If the JK or SR register is selected, eight product terms are shared between two OR gates. The outputs of the OR gates feed the two primary register inputs. This configuration has the following characteristics:

- The A+PLUS Development System optimizes the allocation of product terms for each register input.
- One product term controls asynchronous Clear.
- The Invert Select EPROM bits control output polarity.
- The OE/CLK Select multiplexer configures the mode of operation to Mode 0 or Mode 1.
- The Feedback Select multiplexer allows the user to choose registered feedback or no feedback to the AND array.

Any I/O pin can be configured as a dedicated input by selecting no output with I/O feedback. In the erased state, the I/O architecture is configured for combinatorial active-low output with I/O feedback.

Figure 6. I/O Configurations



Combinatorial

I/O Selection

Output/Polarity	Feedback
Combinatorial/High	Pin, None
Combinatorial/Low	Pin, None
None	Pin

D Flip-Flop

I/O Selection

Output/Polarity	Feedback
D Register/High	D Register, Pin, None
D Register/Low	D Register, Pin, None
None	D Register
None	Pin

Function Table

D	Q_n	Q_{n+1}
L	L	L
L	H	L
H	L	H
H	H	H

T Flip-Flop

I/O Selection

Output/Polarity	Feedback
T Register/High	T Register, Pin, None
T Register/Low	T Register, Pin, None
None	T-Register
None	Pin

Function Table

T	Q_n	Q_{n+1}
L	L	L
L	H	H
H	L	H
H	H	L

JK Flip-Flop

I/O Selection

Output/Polarity	Feedback
JK Register/High	JK Register, None
JK Register/Low	JK Register, None
None	JK Register

Function Table

J	K	Q_n	Q_{n+1}
L	L	L	L
L	L	H	H
L	H	L	L
L	H	H	L
H	L	L	H
H	L	H	H
H	H	L	H
H	H	H	L

SR Flip-Flop

I/O Selection

Output/Polarity	Feedback
SR Register/High	SR Register, None
SR Register/Low	SR Register, None
None	SR Register

Function Table

S	R	Q_n	Q_{n+1}
L	L	L	L
L	L	H	H
L	H	L	L
L	H	H	L
H	L	L	H
H	L	H	H

2

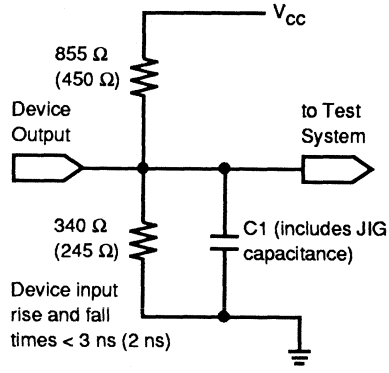
Functional Testing

EP600-series EPLDs are fully functionally tested and guaranteed through complete testing of each programmable EPROM bit and all internal logic elements. A 100% programming yield is ensured. This testing process eliminates problems associated with fuse-programmed circuits by allowing test programming patterns to be used and then erased. The ability to use application-independent, general-purpose tests, called generic testing, is unique to EPLDs. AC test measurements are performed under the conditions shown in Figure 7.

Figure 7. AC Test Conditions

Power supply transients can affect AC measurements. Simultaneous transitions of multiple outputs should be avoided for accurate measurement. Threshold tests must not be performed under AC conditions. Large-amplitude, fast-ground current transients normally occur as the device outputs discharge the load capacitances. When these transients flow through the parasitic inductance between the device ground pin and the test system ground, it can create significant reductions in observable input noise immunity.

Note: Numbers in parentheses are for the EP610A EPLD.



Design Security

EP600-series EPLDs contain a programmable design Security Bit that controls access to the data programmed into the device. If this feature is used, a proprietary design implemented in the EPLD cannot be copied or retrieved. This feature provides a high level of design security by making programmed data within EPROM cells invisible. The Security Bit, as well as all other program data, is reset by erasing the EPLD.

Turbo Bit

The EP610 and EP630 EPLDs contain a programmable Turbo Bit, set with the A+PLUS software, to control the automatic power-down feature that enables the low standby-power mode. When the Turbo Bit is programmed (Turbo = On), the low standby-power mode (I_{CC1}) is disabled, making the circuit less sensitive to V_{CC} noise transients created by the low-power mode power-up/power-down cycle. The typical I_{CC} vs. frequency data for both turbo and non-turbo mode is shown in each EPLD data sheet. All AC values are tested with the Turbo Bit programmed.

If the design requires low power operation, the Turbo Bit should be disabled (Turbo = Off). In this mode, some AC parameters may increase. To determine worst-case timing, values from the AC Non-Turbo Adder specifications must be added to the corresponding AC parameter.

Features

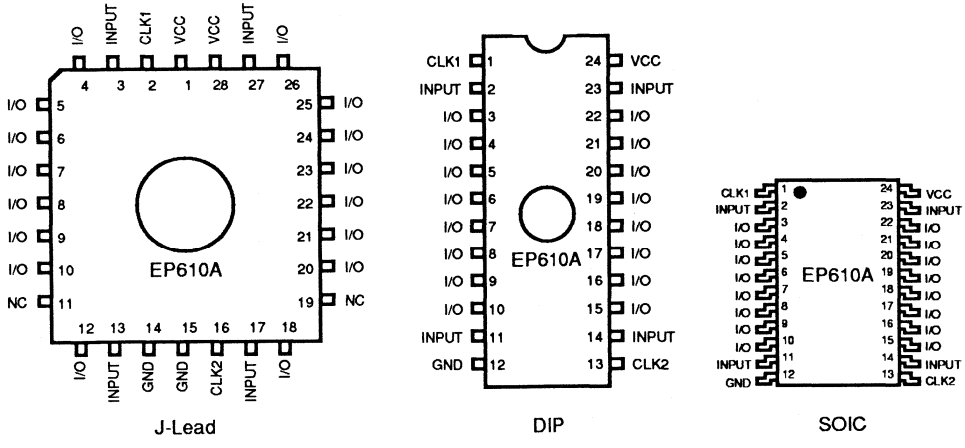
- High-performance 16-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 12 ns
  - Counter frequencies up to 83.3 MHz
  - Pipelined data rates up to 83.3 MHz
- Available in windowed ceramic and plastic one-time-programmable chip carrier packages
  - 24-pin DIP (ceramic and plastic)
  - 28-pin J-lead (ceramic and plastic)
  - 24-pin, 300-mil SOIC (plastic)
- Macrocells can be individually programmed as D, T, JK, or SR flip-flops, or for combinatorial operation.
- Programmable Clock option allows independent clocking of all registers.

Figure 8 shows the pin-outs for the EP610A EPLD.

2

Figure 8. EP610A Pin-Out Diagrams

Package outlines not drawn to scale.



Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–175	+175	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			1000	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75	5.25	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time			25	ns
t_F	Input fall time			25	ns

DC Operating Conditions *See Note (2)*

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 8 mA DC			0.5	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–40		+40	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND No load		90	130	mA
I_{CC3}	V_{CC} supply current	V_I = V_{CC} or GND, No load f = 1.0 MHz, *See Note (3)*		90	130	mA

Capacitance *See Note (4)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		8	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		8	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		16	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

			EP610A-10		EP610A-12		EP610A-15		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{PD1}	Input to non-registered output	C_1 = 35 pF		10		12		15	ns
t_{PD2}	I/O input to non-registered output			10		12		15	ns
t_{PZX}	Input to output enable			10		12		15	ns
t_{PXZ}	Input to output disable, *See Note (5)*	C_1 = 5 pF		10		12		15	ns
t_{CLR}	Asynchronous output clear time	C_1 = 35 pF		10		12		15	ns

Synchronous Clock Mode

			EP610A-10		EP610A-12		EP610A-15		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
f_{MAX}	Maximum frequency	*See Note (6)*	100		83.3		83.3		MHz
t_{SU}	Input setup time		8		8		10		ns
t_H	Input hold time		0		0		0		ns
t_{CH}	Clock high time		5		6		6		ns
t_{CL}	Clock low time		5		6		6		ns
t_{CO1}	Clock to output delay			6		6		8	ns
t_{CNT}	Minimum clock period			10		12		12	ns
f_{CNT}	Internal maximum frequency	*See Note (3)*	100		83.3		83.3		MHz

Asynchronous Clock Mode

			EP610A-10		EP610A-12		EP610A-15		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
f_{MAX}	Maximum frequency	*See Note (6)*	100		83.3		71.4		MHz
t_{ASU}	Input setup time		5		6		6		ns
t_{AH}	Input hold time		5		6		6		ns
t_{ACH}	Clock high time		5		6		7		ns
t_{ACL}	Clock low time		5		6		7		ns
t_{ACO1}	Clock to output delay			12		13		15	ns
t_{ACNT}	Minimum clock period			10		12		14	ns
f_{ACNT}	Internal maximum frequency	*See Note (3)*	100		83.3		71.4		MHz

Notes to tables:

(1) The minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(3) Measured with a device programmed as a 16-bit counter.
(4) Capacitance measured at 25° C. Sample-tested only. Clock-pin capacitance for dedicated clock inputs only. Pin 13 (high-voltage pin during programming) has a maximum capacitance of 50 pF.
(5) Sample-tested only for an output change of 500 mV.
(6) The f_{MAX} values represent the highest frequency for pipelined data.

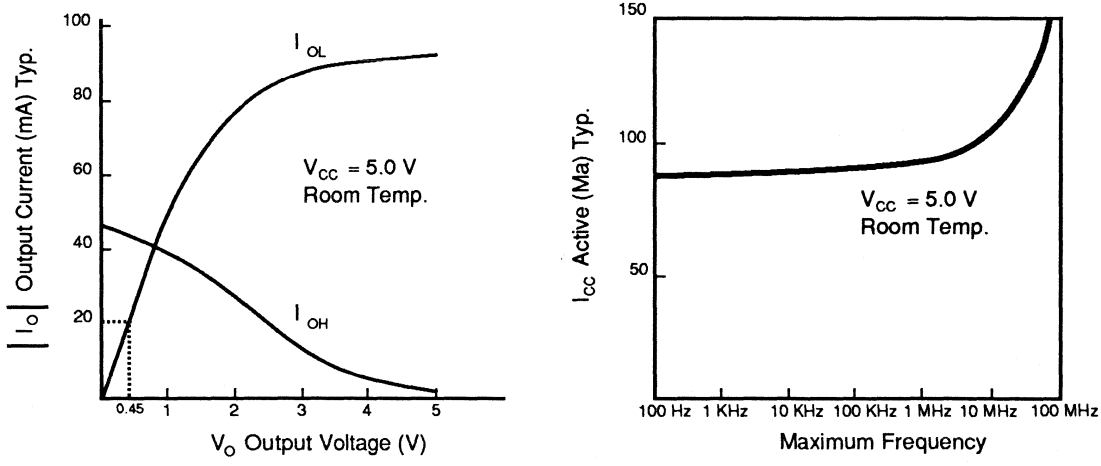
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP610A-12, EP610A-15
Industrial	(–40° C to 85° C)	Consult factory
Military	(–55° C to 125° C)	Consult factory

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera's Marketing Department by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 8 shows output drive characteristics for EP610A I/O pins and typical supply current versus frequency for the EP610A.

Figure 9. EP640 Output Drive Characteristics and I_{CC} vs. Frequency



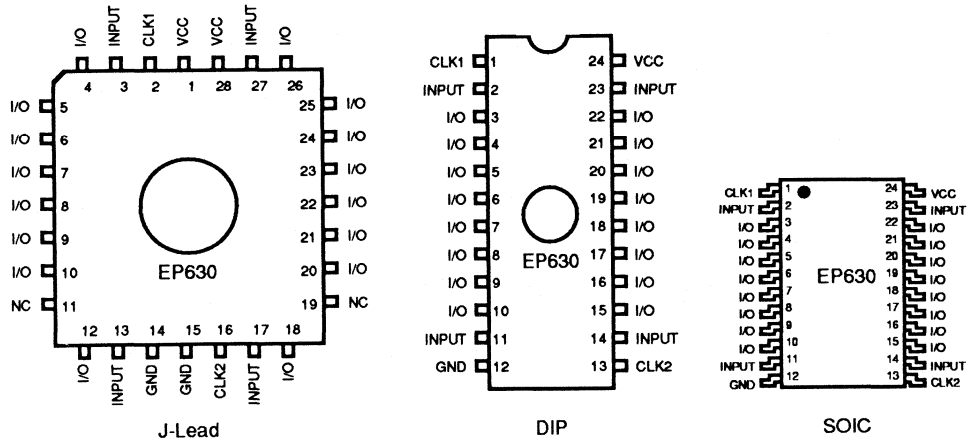
Features

- High-performance 16-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 15 ns
  - Counter frequencies up to 83 MHz
  - Pipelined data rates up to 83 MHz
- Very low power
  - I_{CC} = 5 mA (typical) for a 16-bit counter at 1 MHz
  - I_{CC} = 20 µA (typical) in standby mode
- Available in windowed ceramic and plastic one-time-programmable chip carrier packages
  - 24-pin DIP (ceramic and plastic)
  - 28-pin J-lead (ceramic and plastic)
  - 24-pin, 300-mil SOIC (plastic)
- Macrocells can be individually programmed as D, T, JK, or SR flip-flops, or for combinatorial operation.
- Programmable Clock option allows independent clocking at all registers.

Figure 10 shows pin-outs for the EP630 EPLD.

Figure 10. EP630 Pin-Out Diagrams

Package outlines not drawn to scale.



Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	14.0	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–175	+175	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			1000	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75	5.25	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time	*See Note (2)*		40	ns
t_F	Input fall time	*See Note (2)*		40	ns

DC Operating Conditions *See Note (3)*

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –2 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 4 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (non-turbo standby)	V_I = V_{CC} or GND I_O = 0, *See Note (4)*		20	150	µA
I_{CC2}	V_{CC} supply current (non-turbo mode)	V_I = V_{CC} or GND, No load, f = 1.0 MHz, *See Note (5)*		5	10	mA
I_{CC3}	V_{CC} supply current (turbo mode)	V_I = V_{CC} or GND, No load, f = 1.0 MHz, *See Note (5)*		45	90	mA

Capacitance *See Note (6)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		12	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		20	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

			EP630-15		EP630-20		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	*See Note (7)*	Unit
t_{PD1}	Input to non-registered output	C_1 = 35 pF		15		20	20	ns
t_{PD2}	I/O input to non-reg. output			17		22	20	ns
t_{PZX}	Input to output enable			15		20	20	ns
t_{PXZ}	Input to output disable	C_1 = 5 pF, *Note (8)*		15		20	20	ns
t_{CLR}	Asynchronous output clear time	C_1 = 35 pF		15		20	20	ns
t_{IO}	I/O input pad and buffer delay			2		2	0	ns

Synchronous Clock Mode

			EP630-15		EP630-20		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	*See Note (7)*	Unit
f_{MAX}	Maximum frequency	*See Note (9)*	83.3		62.5		0	MHz
t_{SU}	Input setup time		9		11		20	ns
t_H	Input hold time		0		0		0	ns
t_{CH}	Clock high time		6		8		0	ns
t_{CL}	Clock low time		6		8		0	ns
t_{CO1}	Clock to output delay			11		13	0	ns
t_{CNT}	Minimum clock period			12		16	0	ns
f_{CNT}	Internal maximum frequency	*See Note (5)*	83.3		62.5		0	MHz

Asynchronous Clock Mode

			EP630-15		EP630-20		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	*See Note (7)*	Unit
f_{MAX}	Maximum frequency	*See Note (9)*	71.4		55.5		0	MHz
t_{ASU}	Input setup time		6		8		20	ns
t_{AH}	Input hold time		6		8		0	ns
t_{ACH}	Clock high time		7		9		0	ns
t_{ACL}	Clock low time		7		9		0	ns
t_{ACO1}	Clock to output delay			15		20	20	ns
t_{ACNT}	Minimum clock period			14		18	0	ns
f_{ACNT}	Internal maximum frequency	*See Note (5)*	71.4		55.5		0	MHz

2

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) For all clocks: t_R and t_F = 20 ns.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) When in non-turbo mode, an EPLD will automatically enter standby mode if logic transitions do not occur (approximately 100 ns after the last transition). The non-turbo standby current specification (I_{CC1}) does not apply to the EP630-15 EPLD.
(5) Measured with a device programmed as a 16-bit counter.
(6) Capacitance measured at 25° C. Sample-tested only. Clock-pin capacitance for dedicated clock inputs only. Pin 13 (high-voltage pin during programming) has a maximum capacitance of 50 pF.
(7) See "Turbo Bit" earlier in this data sheet.
(8) Sample-tested only for an output change of 500 mV.
(9) The f_{MAX} values represent the highest frequency for pipelined data.

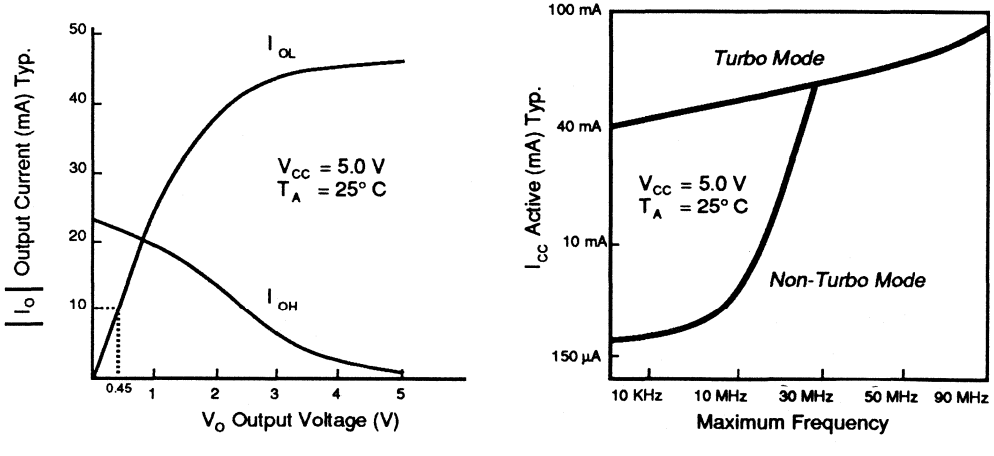
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP630-15, EP630-20
Industrial	(–40° C to 85° C)	Consult factory
Military	(–55° C to 125° C)	Consult factory

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera's Marketing Department by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 11 shows output drive characteristics for EP630 I/O pins and typical supply current versus frequency for the EP630.

Figure 11. EP630 Output Drive Characteristics and I_{CC} vs. Frequency



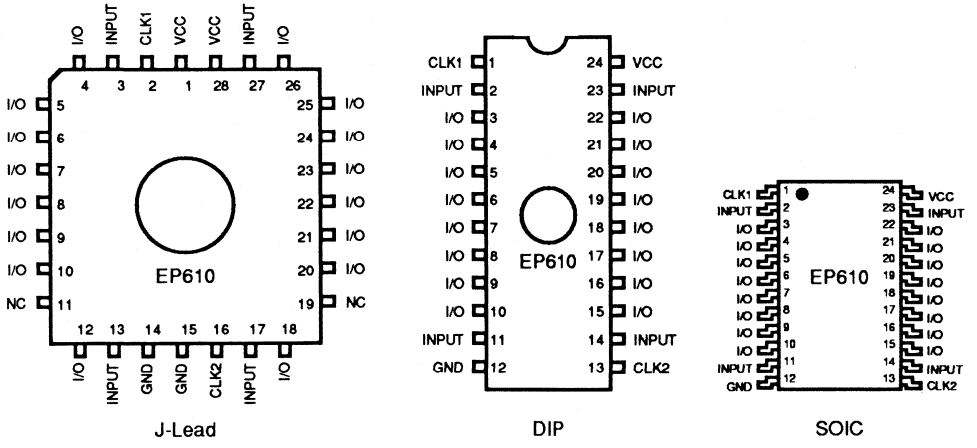
Features

- High-performance 16-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 25 ns
  - Counter frequencies up to 40 MHz
  - Pipelined data rates up to 47 MHz
- Very low power
  - I_{CC} = 3 mA (typical) for a 16-bit counter at 1 MHz
  - I_{CC} = 20 µA (typical) in standby mode
- Available in windowed ceramic and plastic one-time-programmable chip carrier packages
  - 24-pin DIP (ceramic and plastic)
  - 28-pin J-lead (ceramic and plastic)
  - 24-pin, 300-mil SOIC (plastic)
- Macrocell flip-flops can be individually programmed as D, T, JK, or SR, or for combinatorial operation.
- Programmable Clock option allows independent clocking at all registers.

Figure 12 shows the pin-outs for the EP610 EPLD.

Figure 12. EP610 Pin-Out Diagrams

Package outlines not drawn to scale.



Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–175	+175	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			1000	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	*See Note (2)*	4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time	*See Note (3)*		100 (50)	ns
t_F	Input fall time	*See Note (3)*		100 (50)	ns

DC Operating Conditions *See Note (4)*

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –2 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 4 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND No load, *See Note (5)*		20	150	µA
I_{CC2}	V_{CC} supply current (non-turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (6)*		3	10 (15)	mA
I_{CC3}	V_{CC} supply current (turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (6)*		32	60 (75)	mA

Capacitance *See Note (7)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		20	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		20	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Timing Parameters			EP610-25		EP610-30		EP610-35		Non-Turbo Adder	
Symbol	**Parameter**	**Conditions**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	*Note (8)*	**Unit**
t_{PD1}	Input to non-registered output	C1 = 35 pF		25		30		35	30	ns
t_{PD2}	I/O input to non-reg. output			27		32		37	30	ns
t_{PZX}	Input to output enable			25		30		35	30	ns
t_{PXZ}	Input to output disable	C1 = 5 pF, *Note (9)*		25		30		35	30	ns
t_{CLR}	Register clear delay	C1 = 35 pF		27		32		37	30	ns
t_{IO}	I/O input pad and buffer delay			2		2		2	0	ns

Synchronous Clock Mode

Timing Parameters			EP610-25		EP610-30		EP610-35		Non-Turbo Adder	
Symbol	**Parameter**	**Conditions**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	*Note (8)*	**Unit**
f_{MAX}	Max. frequency pipelined data	*Note (10)*	47.6		41.7		37.0		0	MHz
t_{SU}	Input setup time		21		24		27		30	ns
t_{H}	Input hold time		0		0		0		0	ns
t_{CH}	Clock high time		10		11		12		0	ns
t_{CL}	Clock low time		10		11		12		0	ns
t_{CO1}	Clock to output delay			15		17		20	0	ns
t_{CNT}	Minimum clock period			25		30		35	0	ns
f_{CNT}	Internal maximum frequency	*Note (6)*	40.0		33.3		28.6		0	MHz

Asynchronous Clock Mode

Timing Parameters			EP610-25		EP610-30		EP610-35		Non-Turbo Adder	
Symbol	**Parameter**	**Conditions**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	*Note (8)*	**Unit**
f_{MAX}	Max. frequency pipelined data	*Note (10)*	47.6		41.7		37.0		0	MHz
t_{ASU}	Input setup time		8		8		8		30	ns
t_{AH}	Input hold time		12		12		12		0	ns
t_{ACH}	Clock high time		10		11		12		0	ns
t_{ACL}	Clock low time		10		11		12		0	ns
t_{ACO1}	Clock to output delay			27		32		37	30	ns
t_{ACNT}	Minimum clock period			25		30		35	0	ns
f_{ACNT}	Internal maximum frequency	*Note (6)*	40.0		33.3		28.6		0	MHz

Notes to tables:

(1) The minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for to military and industrial temperature versions.
(3) For all clocks: t_R and t_F = 250 ns (100 ns).
(4) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(5) When in non-turbo mode, an EPLD will automatically enter standby mode if logic transitions do not occur (approximately 100 ns after the last transition). The non-turbo standby current specification (I_{CC1}) does not apply to the EP610-25 EPLD.
(6) Measured with a device programmed as a 16-bit counter.
(7) Capacitance measured at 25° C. Sample-tested only. Clock-pin capacitance for dedicated clock inputs only. Pin 13 (high-voltage pin during programming) has a maximum capacitance of 50 pF.
(8) See "Turbo Bit" earlier in this data sheet.
(9) Sample-tested only for an output change of 500 mV.
(10) The f_{MAX} values represent the highest frequency for pipelined data.

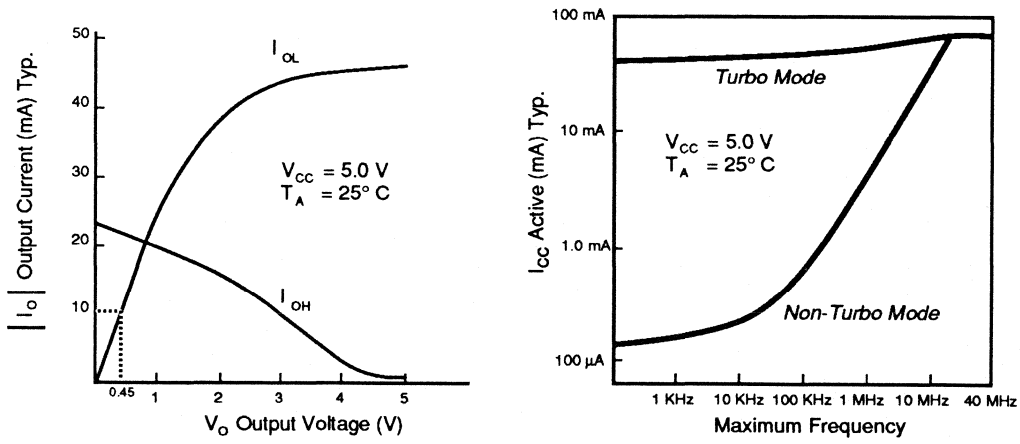
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP610-25, EP610-30, EP610-35
Industrial	(–40° C to 85° C)	EP610-30, EP610-35
Military	(–55° C to 125° C)	EP610-35

Note: Only military-temperature-range devices are listed here. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera's Marketing Department by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 13 shows the output drive characteristics for EP610 I/O pins and typical supply current versus frequency for the EP610.

Figure 13. EP610 Output Drive Characteristics and I_{CC} vs. Frequency



EP900-Series EPLDs

High-Performance 24-Macrocell Devices

October 1990, ver. 1 — **Data Sheet**

2

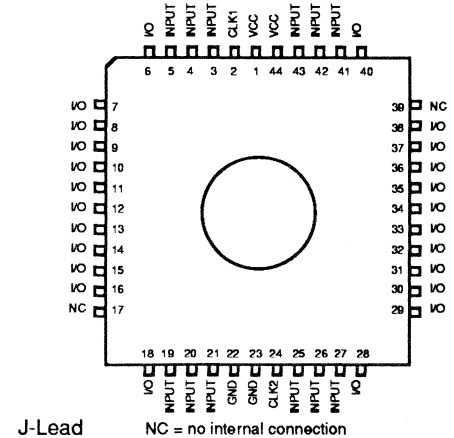
Features

- ❒ High-density replacement for TTL and 74HC with up to 900 gates
- ❒ "Zero power" (consumes only microamps in standby mode)
- ❒ High speed (EP910 t_{PD} = 30 ns)
- ❒ Advanced CMOS EPROM technology allowing devices to be erased and reprogrammed
- ❒ Asynchronous clocking of all registers or banked register operation from two synchronous clocks
- ❒ 24 macrocells with configurable I/O architecture, allowing up to 36 inputs and 24 outputs
- ❒ Individually programmable registers providing D, T, JK, or SR flip-flops with individual asynchronous Clear control
- ❒ 100% generically testable to provide 100% programming yield
- ❒ Programmable Security Bit for total protection of proprietary designs
- ❒ A+PLUS software support featuring schematic capture, Boolean equation, state machine, truth table, and netlist design entry methods
- ❒ Available in 40-pin, 600-mil DIP and 44-pin J-lead chip carriers
- ❒ Extensive third-party support

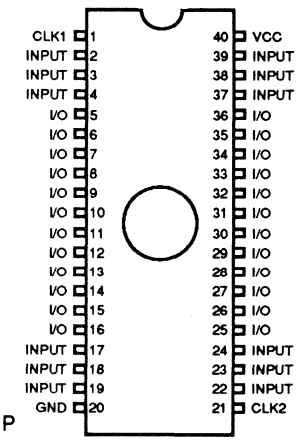
General Description

Altera's EP900-series Erasable Programmable Logic Devices (EPLDs) can implement up to 900 equivalent gates of SSI and MSI logic. These EPLDs are available in a windowed ceramic or one-time-programmable (OTP) plastic 40-pin DIP and a 44-pin J-lead chip carrier. See Figure 1.

Figure 1. Package Pin-Out Diagrams *Package outlines not drawn to scale.*



J-Lead NC = no internal connection



DIP

EP900-series EPLDs use sum-of-products logic that consists of a programmable-AND/fixed-OR structure. These EPLDs accommodate combinatorial and sequential logic functions with up to 36 inputs and 24 outputs.

Altera's proprietary programmable I/O architecture allows the designer to program output and feedback paths for combinatorial or registered operation in active-high and active-low modes.

EP900-series macrocells can be individually programmed for D, T, JK, or SR flip-flop operation, or configured for combinatorial operation. In addition, each register can be individually clocked from any of the input or feedback paths in the AND array. These features make it possible to simultaneously implement a variety of logic functions. For example, EP900-series EPLDs are ideal for integrating several 20- and 24-pin PAL devices.

The CMOS EPROM technology in EP900-series EPLDs can reduce power consumption to less than 20% of the power required by equivalent bipolar devices, without losing speed. This reduced power consumption makes EP900-series EPLDs desirable for a wide range of applications. Moreover, these EPLDs are 100% generically testable and can be erased with UV light. Designs and design modifications can be implemented quickly, eliminating the need for post-programming testing.

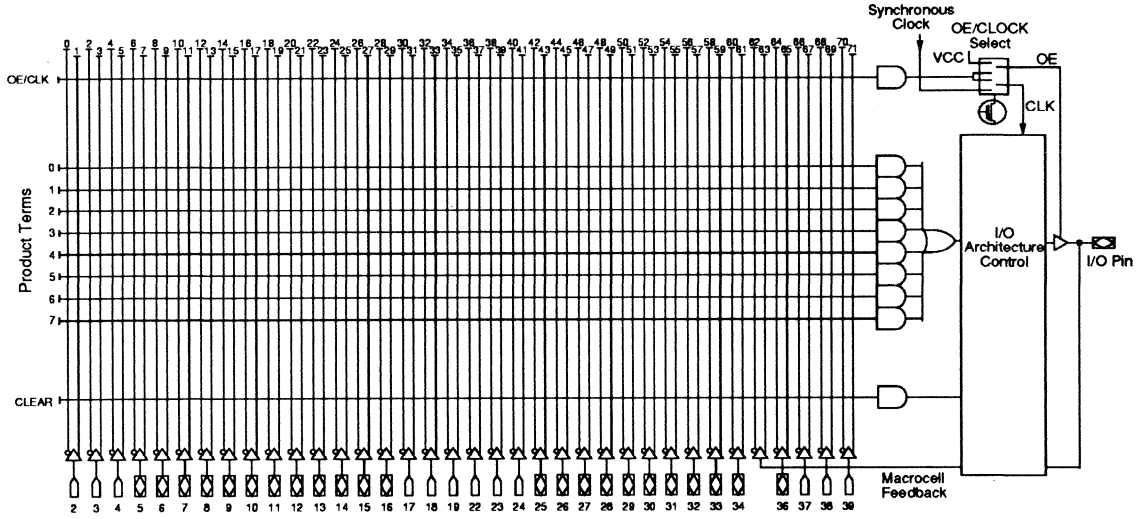
Logic is implemented with Altera's A+PLUS Development System, which supports schematic capture, Boolean equation, state machine, truth table, and netlist design entry methods. After the design is entered, A+PLUS automatically translates it into logic equations, performs Boolean minimization, and fits it into the EPLD. The device may then be programmed in minutes at the designer's desktop to create customized working silicon. In addition, extensive third-party support exists for design entry, design processing, and device programming.

Functional Description

EP900-series EPLDs use CMOS EPROM technology to configure connections in a programmable-AND logic array. EPROM connections are also used to construct a highly flexible programmable I/O architecture that provides advanced functions for user-programmable logic.

EP900-series EPLDs have 12 dedicated data inputs, 2 synchronous clock inputs, and 24 I/O pins that can be individually configured for input, output, or bidirectional operation. Figure 2 shows the EP900-series macrocell. Each macrocell contains 10 product terms for the following functions: 8 product terms are dedicated to logic implementation; 1 product term is used for asynchronous Clear control of the internal register; and 1 product term implements either Output Enable or an asynchronous Clock.

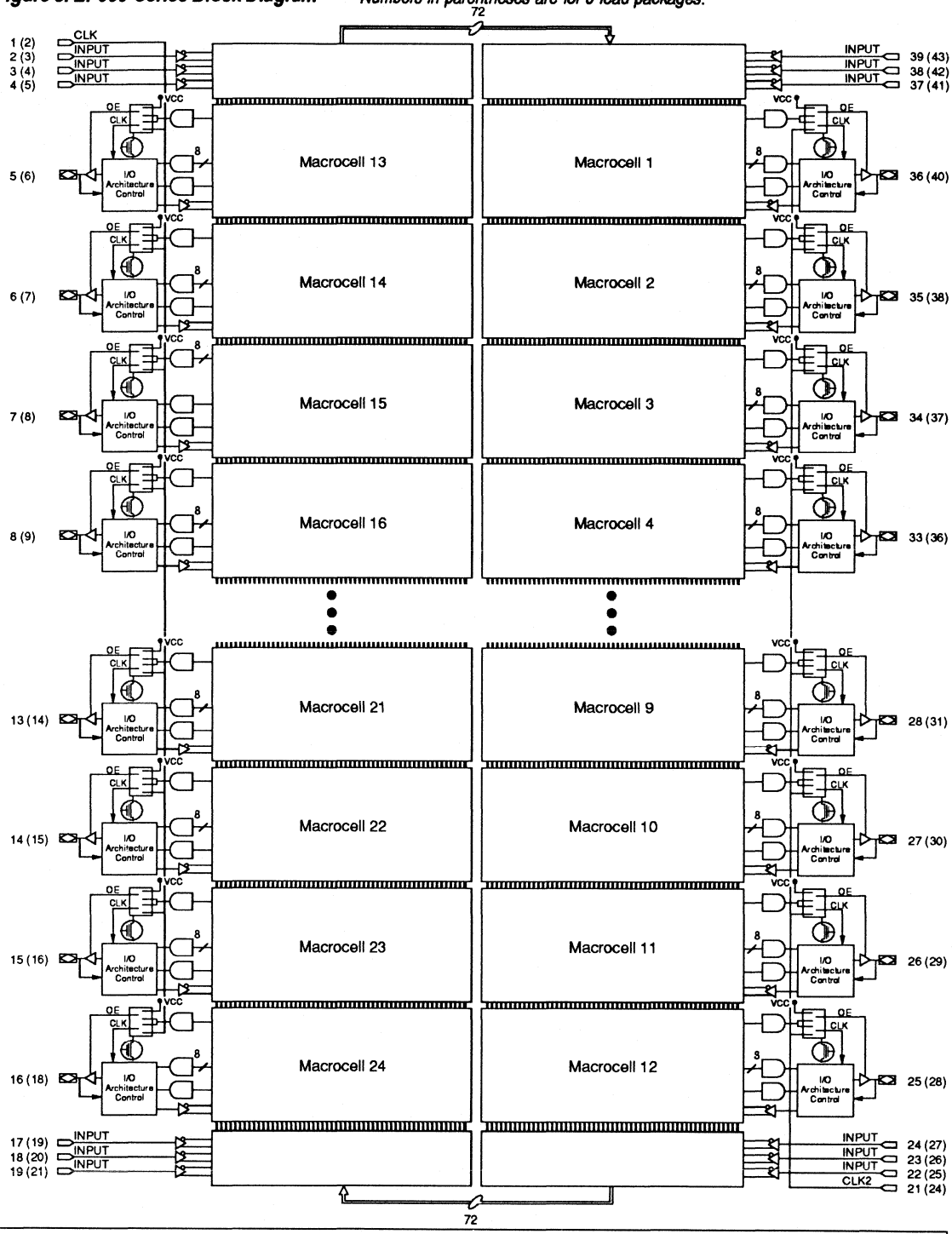
Figure 2. Logic Array Macrocell



The block diagram of an EP900-series EPLD is shown in Figure 3. The internal device architecture has a sum-of-products (AND-OR) structure. Inputs to the programmable AND array come from the true and complement signals of the 12 dedicated data inputs and the 24 I/O feedback signals. The 72-input AND array has 240 product terms that are distributed equally among the 24 macrocells. Each EP900-series product term represents a 72-input AND gate.

In the erased state, the true and complement of the AND-array inputs are connected to the product terms. An EPROM control cell is located at each intersection of an AND-array input and a product term. During the programming process, selected connections are opened, allowing any

Figure 3. EP900-Series Block Diagram *Numbers in parentheses are for J-lead packages.*



product term to be connected to the true or complement of an array input signal with the following results:

- If both the true and complement of an array input signal are connected, the output of the AND gate is a logical low.
- If both the true and complement of any array input signal are programmed "open," a logical "don't care" results for that input.
- If all 72 inputs for a given product term are programmed "open," the output of the corresponding AND gate is a logical high.

Two dedicated clock inputs (which are not available in the AND array) provide the signals used for synchronous clocking of EP900-series internal registers. Each signal is positive-edge-triggered and has control over 12 registers. **CLK1** controls macrocells 13 to 24; **CLK2** controls macrocells 1 to 12. The programmable I/O architecture allows each of the 24 internal registers to have a synchronous or asynchronous Clock mode.

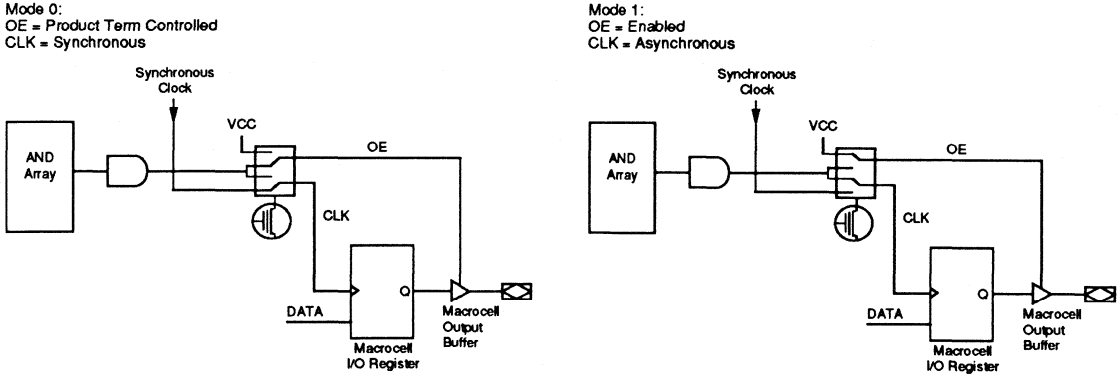
I/O Architecture

EP900-series architecture provides each macrocell with over 50 programmable I/O configurations. Each macrocell can be configured for combinatorial or registered output, with programmable output polarity. Four register types (D, T, JK, and SR) may be implemented in each macrocell without requiring additional logic. I/O feedback selection can be programmed for registered or input feedback. The I/O architecture can also individually clock each internal register from any internal signal.

OE/CLK Selection

Figure 4 shows the two modes of operation provided by the OE/CLK-Select multiplexer. This multiplexer, controlled by a single EPROM control bit, may be individually configured at each I/O pin.

Figure 4. OE/CLK Select Multiplexer



The register is clocked by the synchronous clock signal, which is common to seven other macrocells. The output is enabled by the logic from the product term.

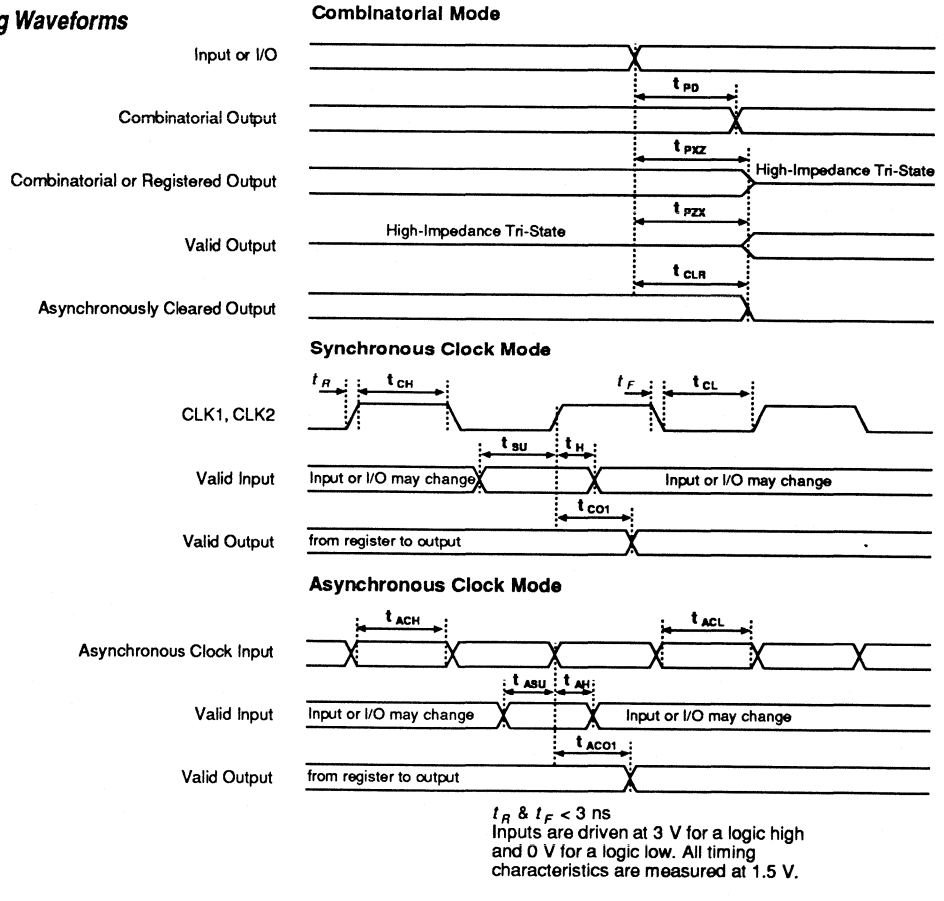
The output is permanently enabled and the register is clocked by the product term, which allows gated clocks to be generated in EP900-series EPLDs.

In Mode 0, the tri-state output buffer is controlled by a single product term. If the output of the AND gate is high, the output buffer is enabled. If the output is low, the output buffer has a high-impedance value. In this mode, the macrocell flip-flop is clocked by its synchronous Clock input signal (**CLK1** or **CLK2**). In the erased state, the OE/CLK Select multiplexer is configured to Mode 0.

In Mode 1, the Output Enable buffer is always enabled, so the macrocell flip-flop can be triggered from an asynchronous Clock signal generated by the OE/CLK product term. This mode allows flip-flops to be individually clocked from any of the 72 AND-array input signals. With both true and complement signals in the AND array, the flip-flop can be configured to trigger on a rising or falling edge. This product-term-controlled clock configuration also allows implementation of gated clock structures.

Figure 5 shows waveforms for the following modes: combinatorial, synchronous Clock, and asynchronous Clock.

Figure 5. Switching Waveforms



t_R & t_F < 3 ns
Inputs are driven at 3 V for a logic high and 0 V for a logic low. All timing characteristics are measured at 1.5 V.

Output/ Feedback Selection

Output configurations available with EP900-series EPLDs are shown in Figure 6. Each macrocell can be individually configured with combinatorial output or any of the four register outputs. All registers have an individual asynchronous Clear function controlled by a dedicated product term. When this product term is a logic high, the macrocell register is immediately loaded with a logic low. The Clear function is performed automatically during power-up.

The combinatorial configuration has eight product terms ORed together to generate the output signal. This configuration has the following characteristics:

- The Invert Select EPROM bit controls output polarity.
- One product term controls the Output Enable buffer.
- The Feedback Select multiplexer allows the user to choose I/O (pin) feedback or no feedback to the AND array.

The D or T register configuration has eight product terms ORed together that are available to the register input. This configuration has the following characteristics:

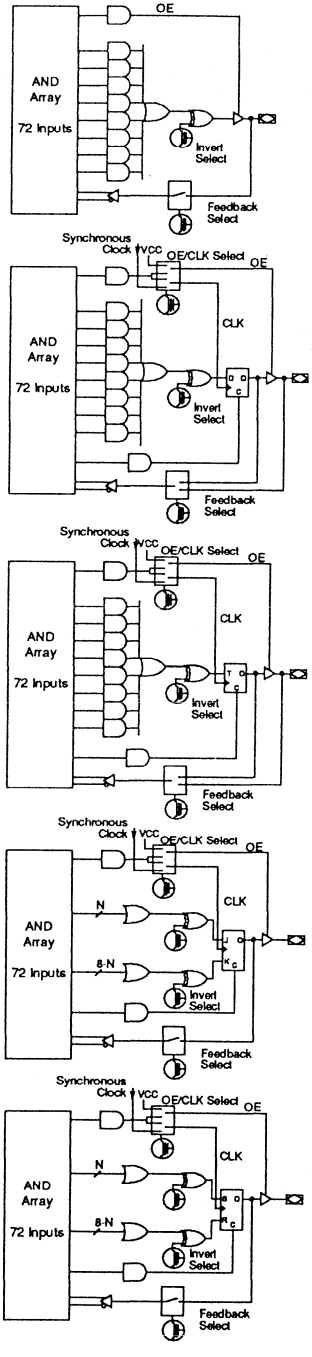
- The Invert Select EPROM bit controls output polarity.
- One product term controls asynchronous Clear.
- The OE/CLK Select multiplexer configures the mode of operation to Mode 0 or Mode 1.
- The Feedback Select multiplexer allows the user to choose registered feedback, I/O feedback, or no feedback to the AND array.

If the JK or SR register is selected, eight product terms are shared between two OR gates. The outputs of the OR gates feed two primary register inputs. This configuration has the following characteristics:

- The A+PLUS Development System optimizes the allocation of product terms for each register input.
- One product term controls asynchronous Clear.
- The Invert Select EPROM bits control output polarity.
- The OE/CLK Select multiplexer configures the operation mode to Mode 0 or Mode 1.
- The Feedback Select multiplexer allows the user to choose registered feedback or no feedback to the AND array.

Any I/O pin can be configured as a dedicated input by selecting no output with I/O feedback. In the erased state, the I/O architecture is configured for combinatorial active-low output with I/O feedback.

Figure 6. I/O Configurations



Combinatorial

I/O Selection

Output/Polarity	Feedback
Combinatorial/High Combinatorial/Low None	Pin, None Pin, None Pin

D Flip-Flop

I/O Selection

Output/Polarity	Feedback
D Register/High D Register/Low None None	D Register, Pin, None D Register, Pin, None D Register Pin

Function Table

D	Q_n	Q_{n+1}
L	L	L
L	H	L
H	L	H
H	H	H

T Flip-Flop

I/O Selection

Output/Polarity	Feedback
T Register/High T Register/Low None None	T Register, Pin, None T Register, Pin, None T-Register Pin

Function Table

T	Q_n	Q_{n+1}
L	L	L
L	H	H
H	L	H
H	H	L

JK Flip-Flop

I/O Selection

Output/Polarity	Feedback
JK Register/High JK Register/Low None	JK Register, None JK Register, None JK Register

Function Table

J	K	Q_n	Q_{n+1}
L	L	L	L
L	L	H	H
L	H	L	L
L	H	H	L
H	L	L	H
H	L	H	H
H	H	L	H
H	H	H	L

SR Flip-Flop

I/O Selection

Output/Polarity	Feedback
SR Register/High SR Register/Low None	SR Register, None SR Register, None SR Register

Function Table

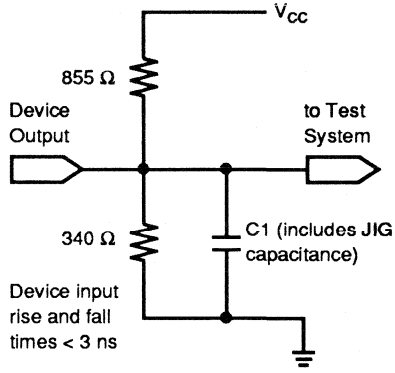
S	R	Q_n	Q_{n+1}
L	L	L	L
L	L	H	H
L	H	L	L
L	H	H	L
H	L	L	H
H	L	H	H

Functional Testing

EP900-series EPLDs are fully functionally tested and guaranteed through complete testing of each programmable EPROM bit and all internal logic elements. A 100% programming yield is ensured. This testing process eliminates traditional problems associated with fuse-programmed circuits by allowing test programming patterns to be used and then erased. This ability to use application-independent, general-purpose tests, called generic testing is unique to EPLDs. AC test measurements are performed under the conditions shown in Figure 7.

Figure 7. AC Test Conditions

Power supply transients can affect AC measurements. Simultaneous transitions of multiple outputs should be avoided for accurate measurement. Threshold tests must not be performed under AC conditions. Large-amplitude, fast-ground current transients normally occur as the device outputs discharge the load capacitances. When these transients flow through the parasitic inductance between the device ground pin and the test system ground, it can create significant reductions in observable input noise immunity.



Design Security

EP900-series EPLDs contain a programmable Security Bit that controls access to the data programmed into the device. If this feature is used, a proprietary design implemented in the device cannot be copied or retrieved. This feature provides a high level of design security by making programmed data within EPROM cells invisible. The Security Bit, as well as other program data, is reset by erasing the EPLD.

Turbo Bit

EP900-series EPLDs contain a programmable Turbo Bit, set with A+PLUS software, to control the automatic power-down feature that enables the low standby-power mode (I_{CC1}). When the Turbo Bit is programmed (Turbo = On), the low standby-power mode is disabled, making the circuit less sensitive to V_{CC} noise transients created by the low-power mode power-up/power-down cycle. Typical I_{CC} vs. frequency data for turbo and non-turbo mode is shown in each EPLD data sheet. All AC values are tested with the Turbo Bit programmed.

If the design requires low-power operation, the Turbo Bit should be disabled (Turbo = Off). In this mode, some AC parameters may increase. To determine worst-case timing, values from the AC Non-Turbo Adder specifications must be added to the corresponding AC parameter.

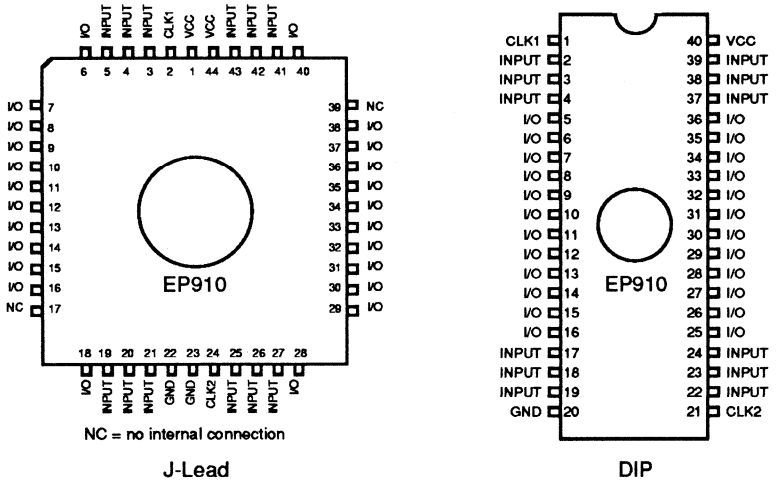
Features

- High-performance 24-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 30 ns
  - Counter frequencies up to 33 MHz
  - Pipelined data rates up to 41 MHz
- Very low power
  - I_{CC} = 6 mA (typical) for a 24-bit counter at 1 MHz
  - I_{CC} = 20 µA (typical) in standby mode
- Available in windowed ceramic and plastic one-time-programmable chip carrier packages
  - 40-pin DIP
  - 44-pin J-lead
- Macrocell flip-flops can be individually programmed as D, T, JK, or SR flip-flops, or for combinatorial operation.
- Programmable Clock option allows independent clocking of all registers.

Figure 8 shows the pin-outs for the EP910 EPLD.

Figure 8. EP910 Pin-Out Diagrams

Package outlines not drawn to scale.



Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–250	+250	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			1200	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	*See Note (2)*	4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time	*See Note (3)*		100 (50)	ns
t_F	Input fall time	*See Note (3)*		100 (50)	ns

DC Operating Conditions *See Note (4)*

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –2 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 4 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND No load, *See Note (5)*		20	150	µA
I_{CC2}	V_{CC} supply current (non-turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (6)*		6	20	mA
I_{CC3}	V_{CC} supply current (turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (6)*		45	80 (100)	mA

Capacitance *See Note (7)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		20	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		20	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Timing Parameters			EP910-30		EP910-35		EP910-40		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	*Note (8)*	Unit
t_{PD1}	Input to non-reg. output	C1 = 35 pF		30		35		40	30	ns
t_{PD2}	I/O input to non-reg. output			33		38		43	30	ns
t_{PZX}	Input to output enable			30		35		40	30	ns
t_{PXZ}	Input to output disable	C1 = 5 pF, *Note (9)*		30		35		40	30	ns
t_{CLR}	Register clear delay	C1 = 35 pF		33		38		43	30	ns
t_{IO}	I/O input pad and buffer delay			3		3		3	0	ns

Synchronous Clock Mode

Timing Parameters			EP910-30		EP910-35		EP910-40		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	*Note (8)*	Unit
f_{MAX}	Maximum clock frequency	*Note (10)*	41.7		37.0		32.3		0	MHz
t_{SU}	Input setup time		24		27		31		30	ns
t_{H}	Input hold time		0		0		0		0	ns
t_{CH}	Clock high time		12		13		15		0	ns
t_{CL}	Clock low time		12		13		15		0	ns
t_{CO1}	Clock to output delay			18		21		24	0	ns
t_{CNT}	Minimum clock period			30		35		40	0	ns
f_{CNT}	Internal maximum frequency	*Note (6)*	33.3		28.6		25.0		0	MHz

Asynchronous Clock Mode

Timing Parameters			EP910-30		EP910-35		EP910-40		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	*Note (8)*	Unit
f_{MAX}	Maximum clock frequency	*Note (10)*	33.3		31.3		29.4		0	MHz
t_{ASU}	Input setup time		10		10		10		30	ns
t_{AH}	Input hold time		15		15		15		0	ns
t_{ACH}	Clock high time		15		16		17		0	ns
t_{ACL}	Clock low time		15		16		17		0	ns
t_{AC01}	Clock to output delay			33		38		43	30	ns
t_{ACNT}	Minimum clock period			30		35		40	0	ns
f_{ACNT}	Internal maximum frequency	*Note (6)*	33.3		28.6		25.0		0	MHz

Notes to tables:

(1) The minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature versions.
(3) For all clocks: t_R and t_F = 100 ns (50 ns).
(4) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(5) When in non-turbo mode, an EPLD will automatically enter standby mode if logic transitions do not occur (approximately 100 ns after the last transition). The non-turbo standby current specification (I_{CC1}) does not apply to the EP910-30 EPLD.
(6) Measured with a device programmed as a 24-bit counter.
(7) Capacitance measured at 25° C. Sample-tested only. Clock-pin capacitance for dedicated clock inputs only. Pin 21 (high-voltage pin during programming) has a maximum capacitance of 60 pF.
(8) See "Turbo Bit" in this data sheet.
(9) Sample-tested only for an output change of 500 mV.
(10) The f_{MAX} values represent the highest frequency for pipelined data.

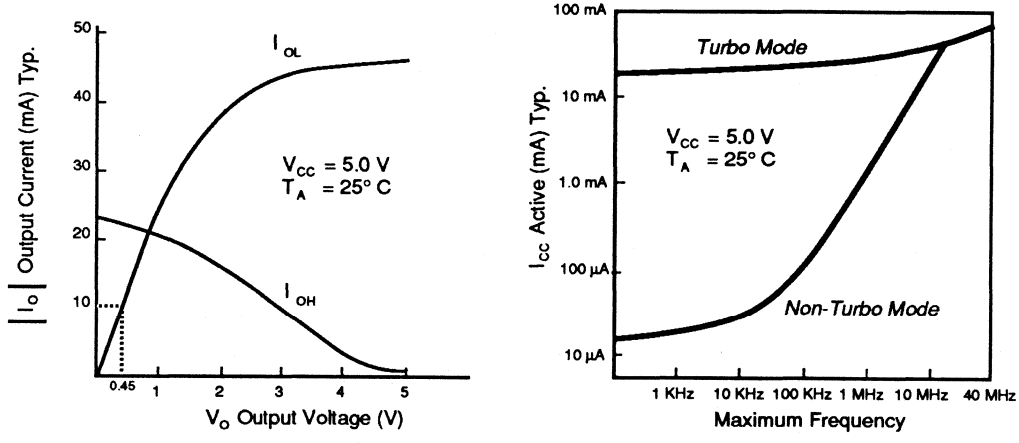
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP910-30, EP910-35, EP910-40
Industrial	(–40° C to 85° C)	EP910-35, EP910-40
Military	(–55° C to 125° C)	EP910-40

Note: Only military-temperature-range devices are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera's Marketing Department by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 9 shows the output drive characteristics for EP910 I/O pins and typical supply current versus frequency for the EP910.

Figure 9. EP910 Output Drive Characteristics and I_{CC} vs. Frequency



EP1800-Series EPLDs

High-Performance 48-Macrocell Devices

October 1990, ver. 1 — **Data Sheet**

Features

- Erasable, user-configurable LSI circuit capable of implementing up to 2,100 equivalent gates of conventional and custom logic
- "Zero power" (typically 35 µA standby)
- High speed (EP1830 t_{PD} = 20 ns)
- 48 macrocells with configurable I/O architecture allowing up to 64 inputs and 48 outputs
- Programmable clock option allowing independent clocking of registers
- Individually programmable registers providing D, T, JK, and SR flip-flops with individual asynchronous Clear control
- Accepts popular TTL SSI- and MSI-based macrofunction design inputs
- TTL/CMOS I/O compatibility and full military capability
- 100% generically testable to provide 100% programming yield
- A+PLUS software support featuring schematic capture, Boolean equation, state machine, truth table, and netlist design entry methods
- Available in 68-pin windowed ceramic and plastic one-time-programmable J-lead and windowed ceramic PGA packages
- Extensive third-party support

2

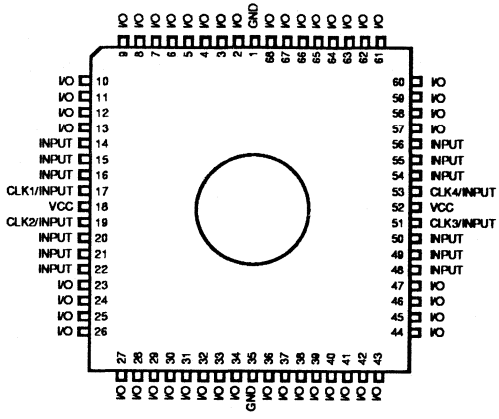
General Description

The EP1800-series Erasable Programmable Logic Devices (EPLDs) offer LSI density, TTL-equivalent speed, and low power consumption. Each EPLD can replace 20 to 30 SSI and MSI packages. These EPLDs are available in 68-pin windowed ceramic and plastic one-time-programmable J-lead and windowed ceramic Pin Grid Array (PGA) packages. See Figure 1.

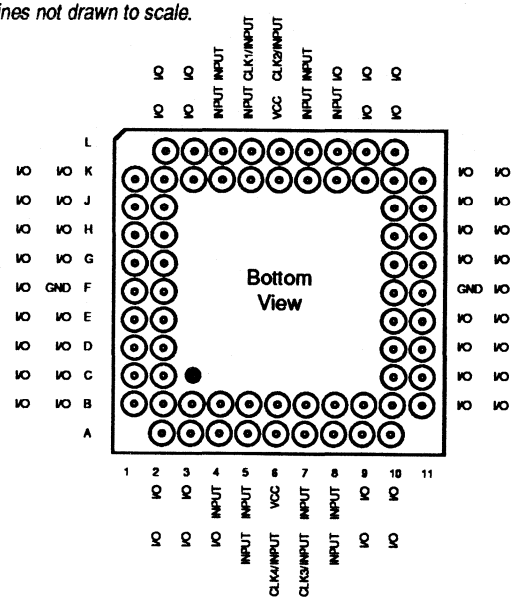
EP1800-series EPLDs are designed as LSI replacements for traditional low-power Schottky TTL logic circuits and low-density Programmable Logic Devices (PLDs). The speed and density of these EPLDs enable them to implement high-performance, complex functions, such as dedicated peripheral controllers and intelligent support chips. IC count and power requirements are considerably reduced with EP1800-series EPLDs, thus minimizing the total size and system cost and significantly increasing reliability.

EP1800-series architecture enables the designer to easily integrate designs with conventional TTL SSI and MSI building blocks. Logic is implemented with Altera's A+PLUS Development System, which supports schematic capture, state machine, Boolean equation, and netlist design entry methods and includes a library of standard TTL functions to facilitate integration. A+PLUS also provides a library of optimized gate and flip-flop elements. The A+PLUS Design Processor (ADP) automatically minimizes and

Figure 1. Package Pin-Out Diagrams *Package outlines not drawn to scale.*



J-Lead



PGA

J-Lead	PGA
Ceramic/Plastic	Ceramic
EP1830 EP1810	EP1830 EP1810

optimizes the design for the EP1800-series architecture, then fits it into the EPLD. A+PLUS, available for IBM PS/2, IBM PC-AT, and compatible computers, also includes the software required to program EPLDs.

EP1800-series EPLDs use EPROM technology to configure logic connections. User-defined logic functions may be constructed by selectively programming EPROM cells within the EPLD. This technology also allows 100% generic testing at the factory. Since these devices can be erased with UV light, design changes are not costly or time consuming, and post-programming testing is not required.

EP1800-Series EPLDs

The EP1800 series includes the EP1830 and EP1810 EPLDs. These EPLDs are JEDEC-file-compatible, allowing a single JEDEC file to be used for programming either EPLD.

EP1830

The EP1830 is the fastest member of the EP1800 series. Its speed and logic density makes it ideal for replacing multiple high-speed PALs. The EP1830 is fabricated on a 1.0-micron process and is available with maximum t_{PD} values of 20 ns, 25 ns, and 30 ns. The EP1830 can run four 12-bit counters at up to 50 MHz.

EP1810

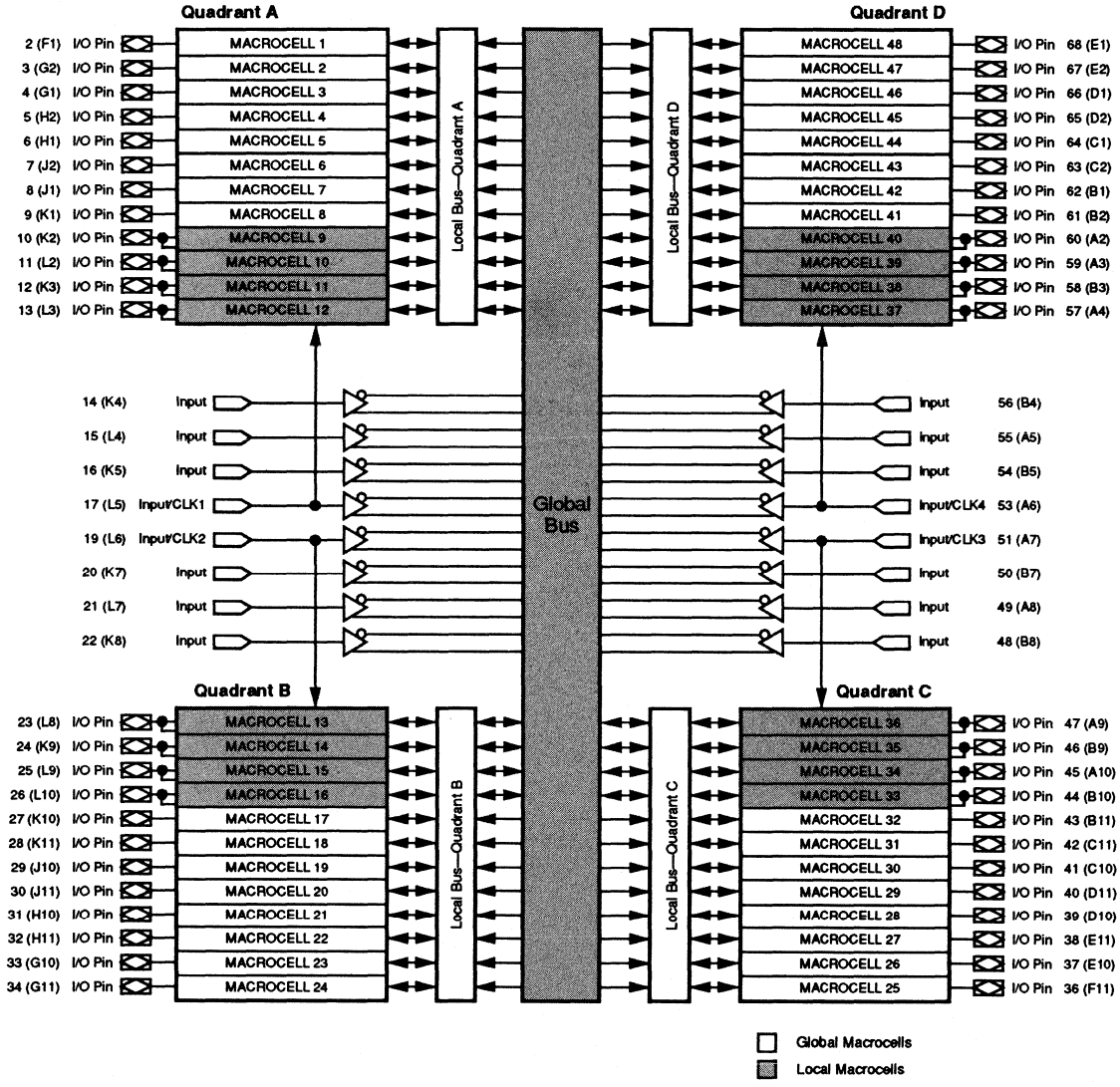
The EP1810 combines speed with low power. It is fabricated on a 1.2-micron process and is available with maximum t_{PD} values of 35 ns, 40 ns, and 45 ns. The EP1810 can implement four 12-bit counters at up to 28.6 MHz, and typically consumes less than 100 mA when operating at 1 MHz. Both MIL-STD-883B-compliant and DESC-approved parts are available.

Functional Description

EP1800-series EPLDs use CMOS EPROM cells to configure internal logic functions. The architecture is 100% user-configurable, allowing a variety of independent logic functions to be accommodated.

EP1800-series EPLDs have 16 dedicated data inputs, 4 of which may be used as system clock inputs. The 48 I/O pins may be individually configured for input, output, or bidirectional data flow. See Figure 2.

Figure 2. EP1800-Series Block Diagram *Numbers in parentheses are for PGA packages.*



Macrocells

The EP1800-series architecture consists of 48 macrocells that implement all logic (see Figure 3). Each macrocell consists of a logic array, a tri-state I/O buffer, and a selectable register element that can be programmed for D, T, JK, or SR operation, or bypassed entirely for purely combinatorial functions. All combinatorial logic (e.g., exclusive-OR, NAND, NOR, AND, OR, and NOT gates) are implemented within the logic array. Each macrocell is equivalent to as many as 40 two-input NAND gates.

Figure 3. Macrocell Components

Each EP1800-series macrocell consists of a logic array for gated logic, a flip-flop for data storage, and a tri-state I/O buffer that enables input, output, or bidirectional data flow.

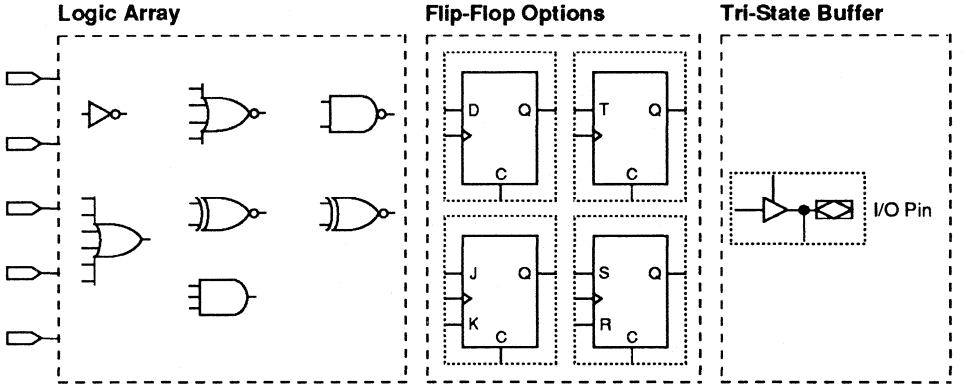
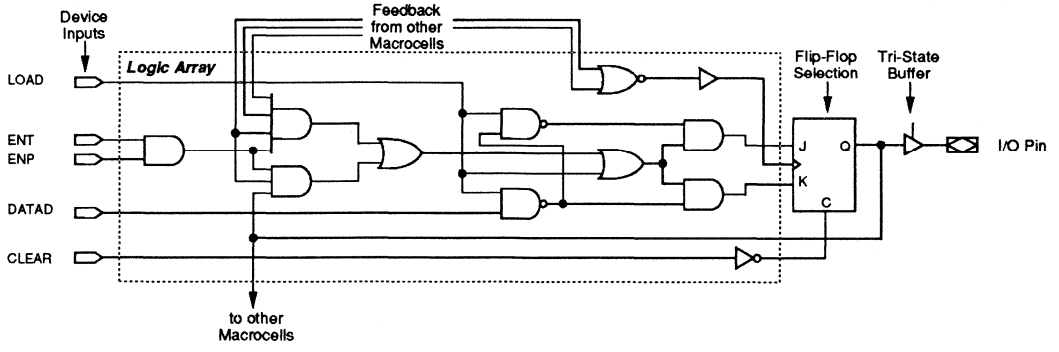


Figure 4 illustrates a simple logic function that can be implemented in a single macrocell. This function implements all combinatorial logic in the logic array, uses a JK flip-flop, and permanently enables the tri-state buffer.

Figure 4. Sample Circuit

This figure shows a typical logic function implemented in a single macrocell. Each EP1800-series macrocell can accommodate up to 40 equivalent gates.



EP1800-series EPLDs have 4 identical quadrants, each containing 12 macrocells. Internal bus structures in these EPLDs feed input signals into the macrocells. Macrocell outputs drive the external pins and internal buses.

Of the 48 macrocells, 36 are local (see Figure 5) and 16 are global macrocells (see Figure 6). Local macrocells offer a multiplexed feedback path (with pin or macrocell feedback) and drive the local bus in their quadrant. Global macrocells feature two dedicated feedback paths: one feeds the local bus; the other feeds the global bus. This process, called "dual feedback," allows global macrocells to implement buried logic functions while the associated I/O pin is used as an input. Dual feedback ensures maximum I/O flexibility.

Figure 5. Local Macrocell

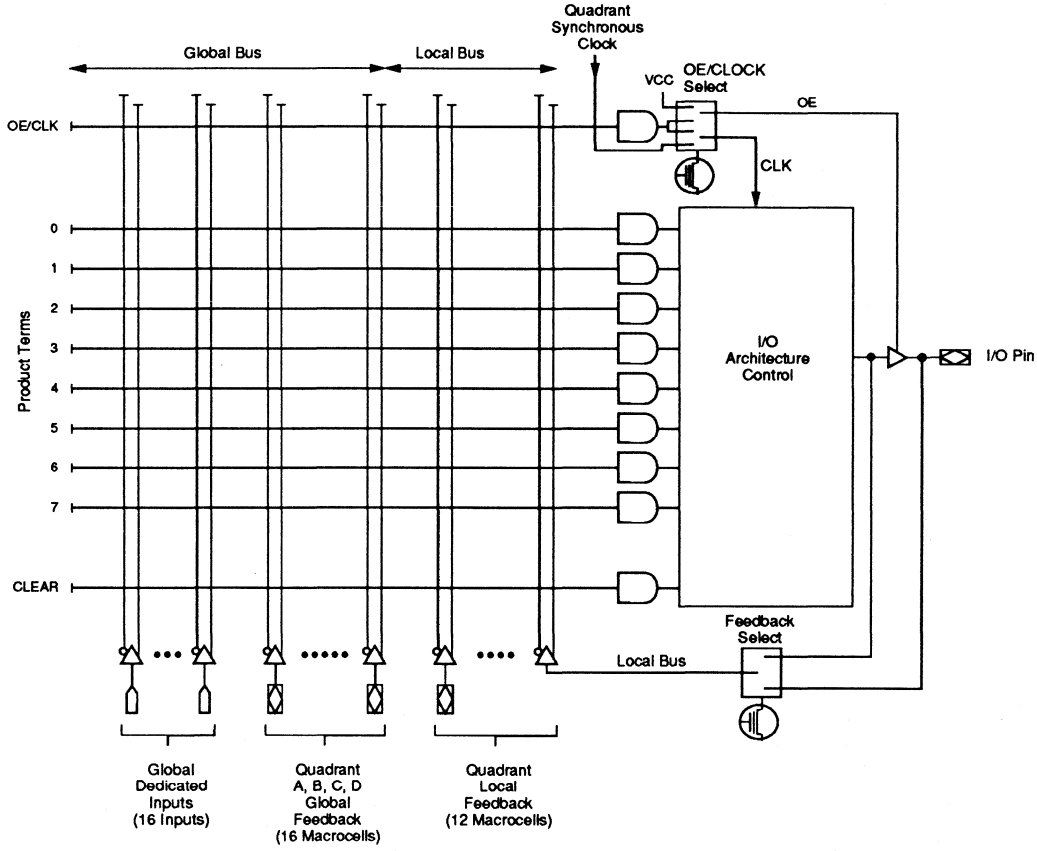
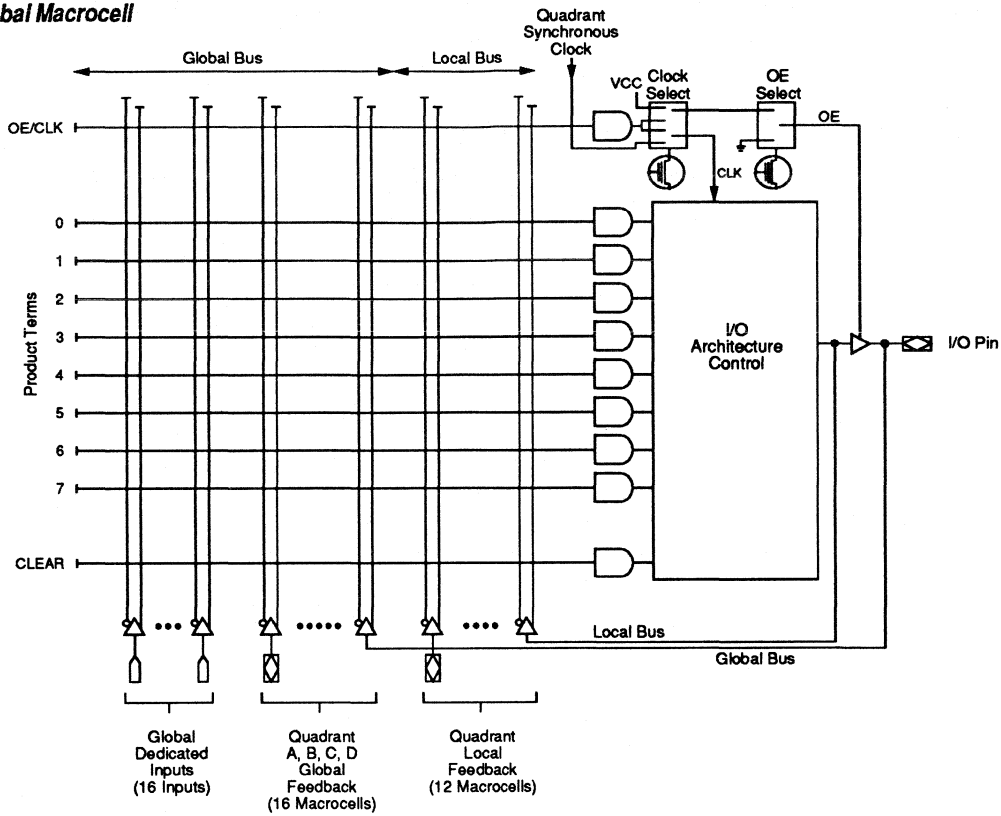


Figure 6. Global Macrocell



Both global and local macrocells have the same timing characteristics. Delay paths are shown in Figure 7. Switching waveforms for EP1800-series EPLDs are shown in Figure 8.

Figure 7. Macrocell Delay Paths *If the register is bypassed, the delay between the logic array and the output buffer is zero.*

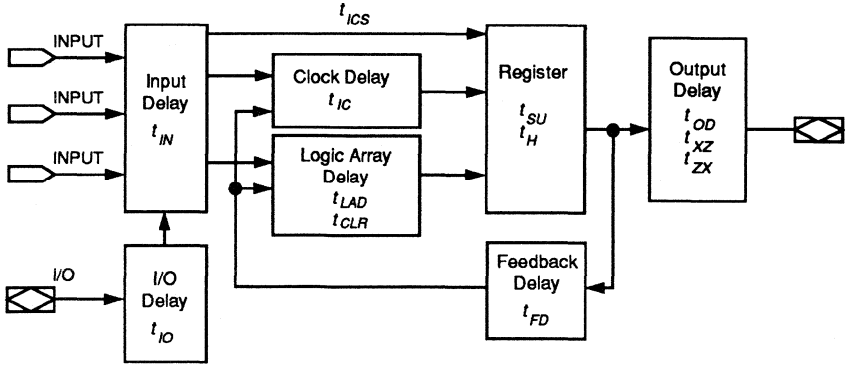
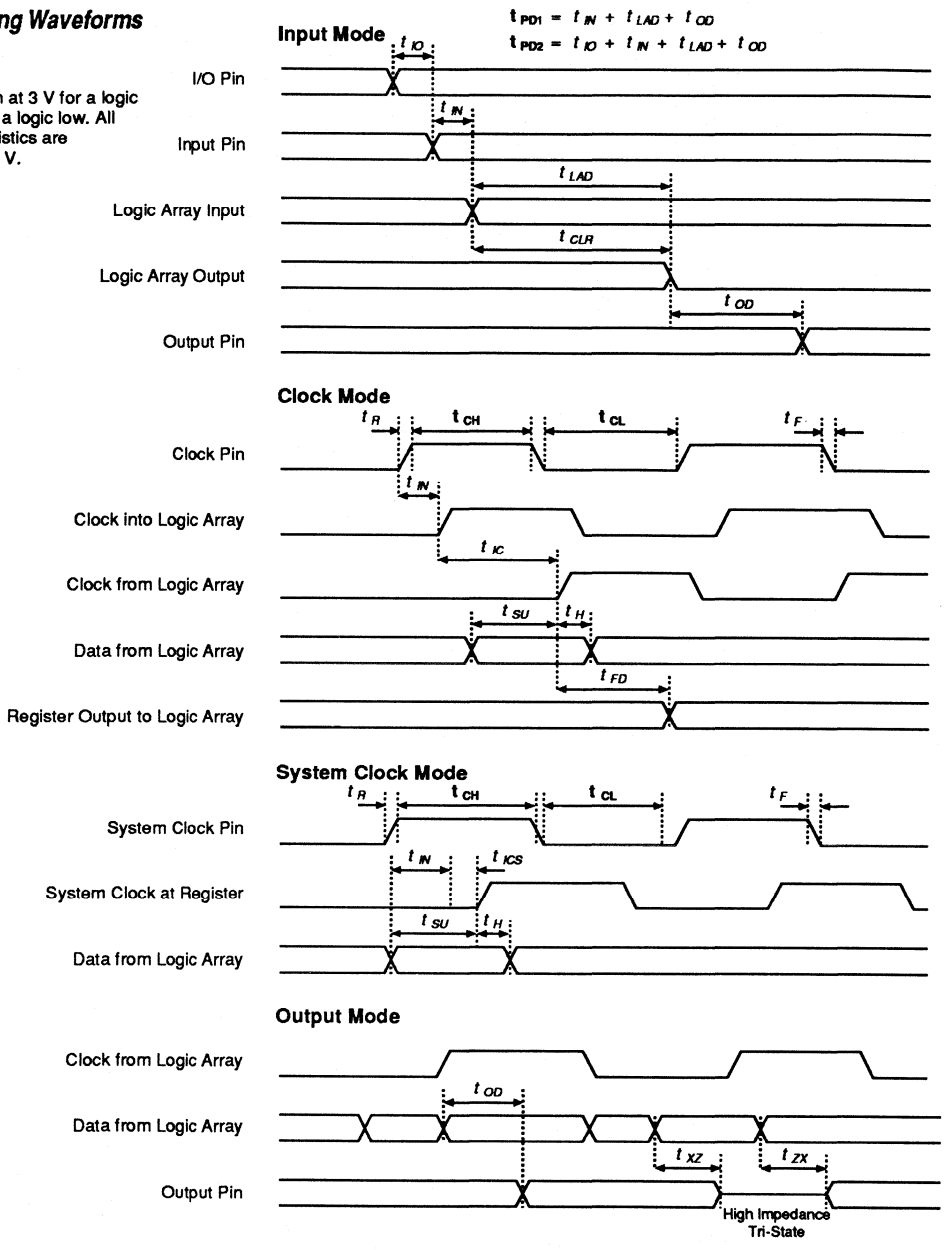


Figure 8. Switching Waveforms

t_R & t_F < 3 ns
Inputs are driven at 3 V for a logic high and 0 V for a logic low. All timing characteristics are measured at 1.5 V.



Clock Options

Each internal flip-flop in EP1800-series EPLDs can be clocked independently or in user-defined groups. Each internal register may select its clock source from a dedicated system clock pin or a product term within the macrocell. Any input or internal logic function can be used as a clock.

Product-term clock signals allow flip-flops to be configured for positive- or negative-edge-triggered operation.

Four dedicated system clocks (**CLK1** to **CLK4**) also provide synchronous or asynchronous clock signals to the flip-flops. One system clock is located in each quadrant; each clock is connected directly to an EP1800-series external pin. Synchronous clocks provide clock-to-output delay times that are faster than internally generated clock signals. Asynchronous pin-driven clock signals are activated by inserting a clock buffer (**CLKB**) primitive between the clock pin and the flip-flop clock input. When system clocks are used, the flip-flops are triggered by the positive edge, i.e., data transitions occur on the rising edge of the clock.

Functional Testing

EP1800-series EPLDs are fully functionally tested and guaranteed through complete testing of each programmable EPROM bit and all internal logic elements. A 100% programming yield is ensured. These EPLDs allow test programs to be used and then erased during early stages of production. The ability to use application-independent, general-purpose tests—called generic testing—is unique to EPLDs. The EPLDs also contain on-board test circuitry that allows verification of functions and AC specifications for one-time-programmable packages. AC test measurements are performed under the conditions shown in Figure 9.

Design Security

EP1800-series EPLDs contain a programmable Security Bit that controls access to the programmed information. If this Security Bit is used, a proprietary design implemented in the device cannot be copied or retrieved. Since this option makes programmed data within EPROM cells invisible, the designer has a high level of design security. The Security Bit, as well as all other program data, is reset by erasing the EPLD.

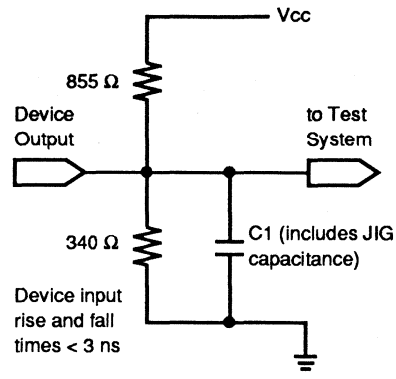
Turbo Bit

All EP1800-series EPLDs contain a Turbo Bit, set with the A+PLUS software, to control the automatic power-down feature that enables the low standby-power mode. When the Turbo Bit is programmed (Turbo = On), the low standby-power mode (I_{CC1}) is disabled, making the circuit less sensitive to V_{CC} noise transients from the non-turbo mode power-up/power-down cycle. The typical I_{CC} versus frequency data for turbo and non-turbo mode is shown in each EPLD data sheet. All AC values are tested with the Turbo Bit programmed.

If the design requires low-power operation, the Turbo Bit should be disabled (Turbo = Off). When operating in this mode, some AC parameters may increase. To determine worst-case timing, values given in the AC Non-Turbo Adder specifications must be added to the AC parameter.

Figure 9. AC Test Conditions

Power supply transients can affect AC measurements. Simultaneous transitions of multiple outputs should be avoided for accurate measurement. Threshold tests must not be performed under AC conditions. Large-amplitude, fast-ground current transients normally occur as the device outputs discharge the load capacitances. When these transients flow through the parasitic inductance between the device ground pin and the test system ground, it can create significant reductions in observable input noise immunity.



Features

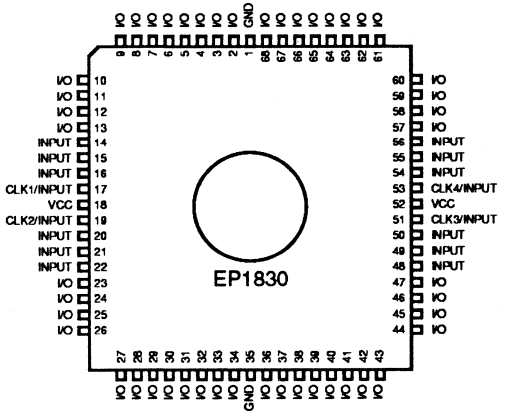
- High-performance 48-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 20 ns
  - Counter frequencies up to 50 MHz
  - Pipelined data rates up to 62.5 MHz
- Low power
  - I_{CC} = 20 mA (typical) for four 12-bit counters at 1 MHz
  - I_{CC} = 50 µA (typical) in standby mode
- Available in windowed ceramic and plastic one-time-programmable chip-carrier packages
  - 68-pin J-lead (ceramic and plastic)
  - 68-pin PGA (ceramic)
- Macrocell flip-flops can be individually programmed as D, T, JK, or SR flip-flops, or for combinatorial operation.
- Programmable Clock option allows independent clocking at all registers.

2

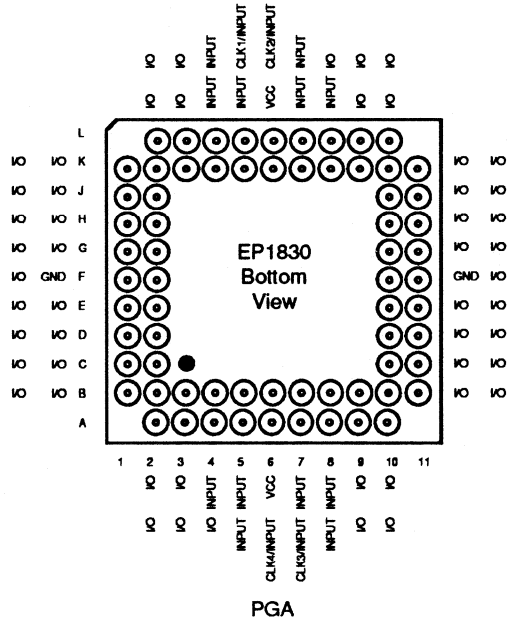
Figure 10 shows the pin-outs for the EP1830 EPLD.

Figure 10. EP1830 Pin-Out Diagrams

Package outlines not drawn to scale.



J-Lead



PGA

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	14.0	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–300	+300	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			1500	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75	5.25	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time	*See Note (2)*		50	ns
t_F	Input fall time	*See Note (2)*		50	ns

DC Operating Conditions *See Note (3)*

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –2 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 4 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND No load, *See Note (4)*		50	150	µA
I_{CC2}	V_{CC} supply current (non-turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (5)*		20	40	mA
I_{CC3}	V_{CC} supply current (turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (5)*		150	200	mA

Capacitance *See Note (6)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		20	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		25	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Timing Parameters			EP1830-20		EP1830-25		EP1830-30		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	*Note (7)*	Unit
t_{PD1}	Input to non-registered output	C1 = 35 pF		20		25		30	25	ns
t_{PD2}	I/O input to non-registered output	C1 = 35 pF		22		28		34	25	ns
t_{IN}	Input pad and buffer delay			5		7		7	0	ns
t_{IO}	I/O input pad and buffer delay			2		3		4	0	ns
t_{LAD}	Logic array delay			9		12		15	25	ns
t_{OD}	Output buffer and pad delay	C1 = 35 pF		6		6		8	0	ns
t_{ZX}	Output buffer enable	C1 = 35 pF		6		6		8	0	ns
t_{XZ}	Output buffer disable	C1 = 5 pF, *Note (8)*		6		6		8	0	ns
f_{MAX}	Maximum clock frequency	*See Note (9)*	62.5		50		41.7		0	MHz
t_{SU}	Register setup time		8		10		12		0	ns
t_{HS}	Register hold time (system clock)		0		0		0		0	ns
t_{H}	Register hold time		8		10		12		0	ns
t_{CH}	Clock high time		8		10		12		0	ns
t_{CL}	Clock low time		8		10		12		0	ns
t_{IC}	Clock delay			9		12		15	25	ns
t_{ICS}	System clock delay			4		5		7	0	ns
t_{FD}	Feedback delay			3		3		3	–25	ns
t_{CLR}	Register clear delay			9		12		15	25	ns
t_{CNT}	Minimum clock period (register output feedback to register input)			20		25		30	0	ns
f_{CNT}	Internal maximum frequency $1/t_{CNT}$	*See Note (5)*	50		40		33.3		0	MHz

2

Notes to tables:

(1) The minimum DC input is –0.3 V. During transitions, inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) For all clocks: t_R and t_F = 20 ns.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) When in non-turbo mode, an EPLD automatically enters standby mode if logic transitions do not occur (approximately 100 ns after the last transition). The non-turbo standby current specification (I_{CC1}) does not apply to the EP1830-20 EPLD.
(5) Measured with a device programmed as four 12-bit counters.
(6) Capacitance measured at 25° C. Sample-tested only. Clock–pin capacitance for dedicated clock inputs only. Pin 19 (high-voltage pin during programming) has a maximum capacitance of 160 pF.
(7) See "Turbo Bit" earlier in this data book.
(8) Sample-tested only for an output change of 500 mV.
(9) The f_{MAX} values represent the highest frequency for pipelined data.

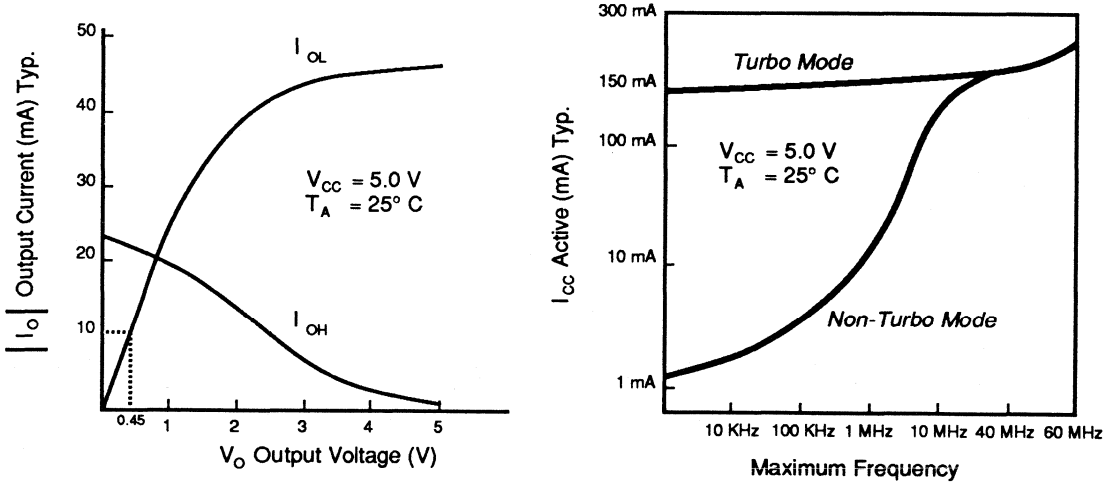
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP1830-20, EP1830-25, EP1830-30
Industrial	(–40° C to 85° C)	Consult factory
Military	(–55° C to 125° C)	Consult factory

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 11 shows output drive characteristics for EP1830 I/O pins and typical supply current versus frequency for the EP1830.

Figure 11. EP1830 Output Drive Characteristics and I_{CC} vs. Frequency



Features

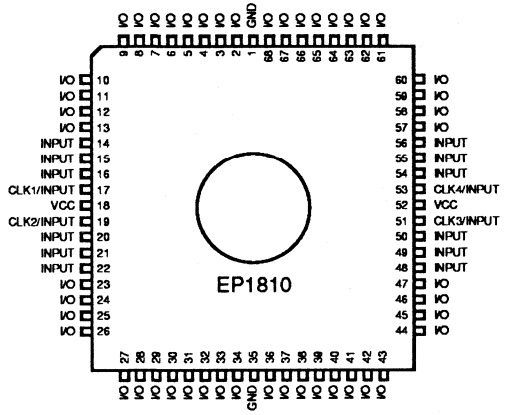
- High-performance 48-macrocell EPLD
  - Combinatorial speeds with t_{PD} = 35 ns
  - Counter frequencies up to 28.6 MHz
  - Pipelined data rates up to 40 MHz
- Low power
  - I_{CC} = 10 mA (typical) for four 12-bit counters at 1 MHz
  - I_{CC} = 35 µA (typical) in standby mode
- Available in windowed ceramic and plastic one-time-programmable packages
  - 68-pin J-lead (ceramic and plastic)
  - 68-pin PGA (ceramic)
- Macrocell flip-flops can be individually programmed as D, T, JK, or SR flip-flops, or for combinatorial operation.
- Programmable Clock option allows independent clocking of all registers.

2

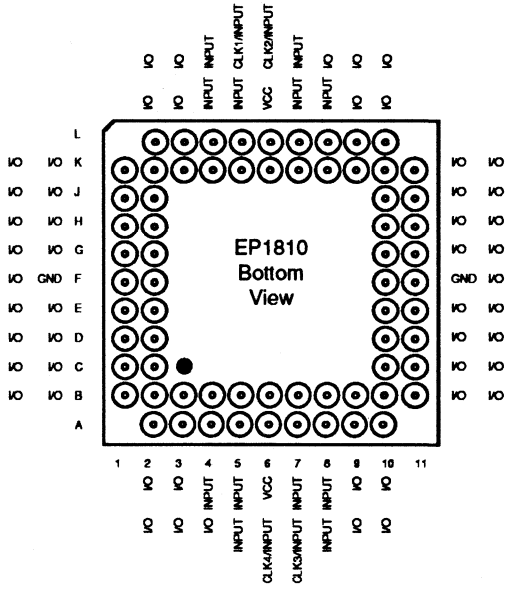
Figure 12 shows the pin-outs for the EP1810.

Figure 12. EP1810 Pin-Out Diagrams

Package outlines not drawn to scale.



J-Lead



PGA

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–300	+300	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			1500	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	*See Note (2)*	4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time			100 (50)	ns
t_F	Input fall time			100 (50)	ns

DC Operating Conditions V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
See Note (3) V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –2 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 4 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND No load, *See Note (4)*		35	150	µA
I_{CC2}	V_{CC} supply current (non-turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (5)*		10	30 (40)	mA
I_{CC3}	V_{CC} supply current (turbo mode)	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (5)*		100	180 (240)	mA

Capacitance *See Note (6)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		20	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		25	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Timing Parameters			EP1810-35		EP1810-40		EP1810-45		Non-Turbo Adder	
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	*Note (7)*	Unit
t_{PD1}	Input to non-registered output	C_1 = 35 pF		35		40		45	30	ns
t_{PD2}	I/O input to non-registered output			40		45		50	30	ns
t_{IN}	Input pad and buffer delay			7		7		7	0	ns
t_{IO}	I/O input pad and buffer delay			5		5		5	0	ns
t_{LAD}	Logic array delay			19		23		27	30	ns
t_{OD}	Output buffer and pad delay	C_1 = 35 pF		9		10		11	0	ns
t_{ZX}	Output buffer enable			9		10		11	0	ns
t_{XZ}	Output buffer disable	C_1 = 5 pF *See Note (8)*		9		10		11	0	ns
f_{MAX}	Maximum clock frequency	*See Note (9)*	40.0		35.7		33.3		0	MHz
t_{SU}	Register setup time		10		11		11		0	ns
t_{HS}	Register hold time (system clock)		0		0		0		0	ns
t_H	Register hold time		15		17		18		0	ns
t_{CH}	Clock high time		12		14		15		0	ns
t_{CL}	Clock low time		12		14		15		0	ns
t_{IC}	Clock delay			19		23		27	30	ns
t_{ICS}	System clock delay			4		6		8	0	ns
t_{FD}	Feedback delay			6		6		7	–30	ns
t_{CLR}	Register clear delay			24		28		32	30	ns
t_{CNT}	Minimum clock period (register output feedback to register input)			35		40		45	0	ns
f_{CNT}	Internal maximum frequency $1/t_{CNT}$	*See Note (5)*	28.6		25.0		22.2		0	MHz

2

Notes to tables:

(1) The minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for to military and industrial temperature versions.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) When in non-turbo mode, an EPLD will automatically enter standby mode if logic transitions do not occur (approximately 100 ns after the last transition). The non-turbo standby current specification (I_{CC1}) does not apply to the EP1810-35 EPLD.
(5) Measured with a device programmed as four 12-bit counters.
(6) Capacitance measured at 25° C. Sample-tested only. Clock-pin capacitance for dedicated clock inputs only. Pin 19 (high-voltage pin during programming) has a maximum capacitance of 160 pF.
(7) See "Turbo Bit" earlier in this data sheet.
(8) Sample-tested only for an output change of 500 mV.
(9) The f_{MAX} values represent the highest frequency for pipelined data.

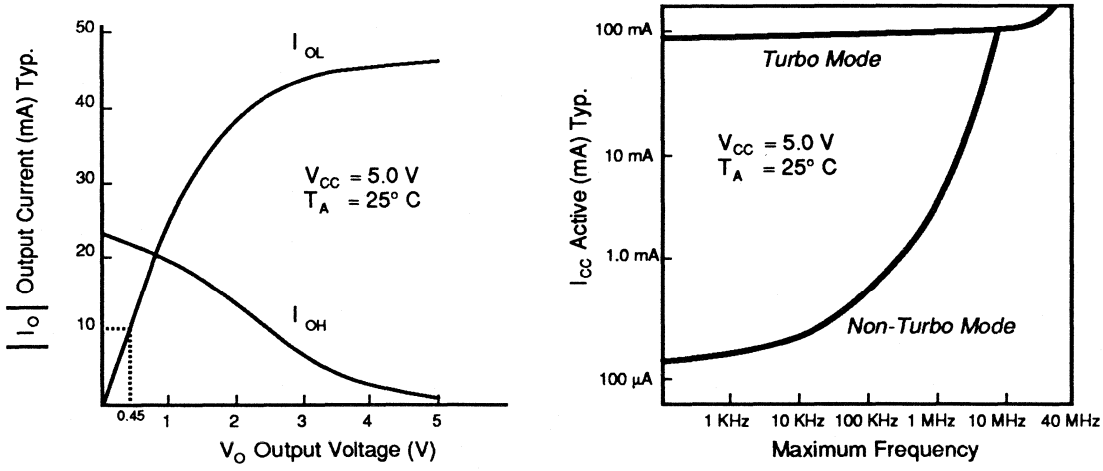
Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EP1810-35, EP1810-45
Industrial	(–40° C to 85° C)	EP1810-40, EP1810-45
Military	(–55° C to 125° C)	EP1810-45

Note: Only military-temperature-range devices are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera's Marketing Department by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 13 shows output drive characteristics for EP1810 I/O pins and typical supply current versus frequency for the EP1810.

Figure 13. EP1810 Output Drive Characteristics and I_{CC} vs. Frequency



PLS-SUPREME

Enhanced A+PLUS Programmable Logic Software

October 1990, ver. 1 — **Data Sheet**

Features

- High-level support for Altera's general-purpose EP-series EPLDs
- Multiple design entry methods
  - LogiCaps schematic capture
  - Boolean equation and netlist
  - State machine and truth table
- Complete symbol library of basic gates and over 120 TTL macrofunctions
- Support for user-defined macrofunctions with ADLIB software
- Fast and efficient design processing to ensure rapid design cycles
  - Elimination of unused gates
  - Automatic pin and part assignments
  - SALSA logic minimization
  - Device Fitter to optimize EPLD resources
- Functional simulation for quick design verification
  - Easy definition of inputs with state tables, vector patterns, or predefined patterns
  - State table or graphic waveform output formats (on-screen or hard-copy)
  - Access to buried nodes within the design
- Used with IBM PS/2, PC-AT, and compatible computers

2

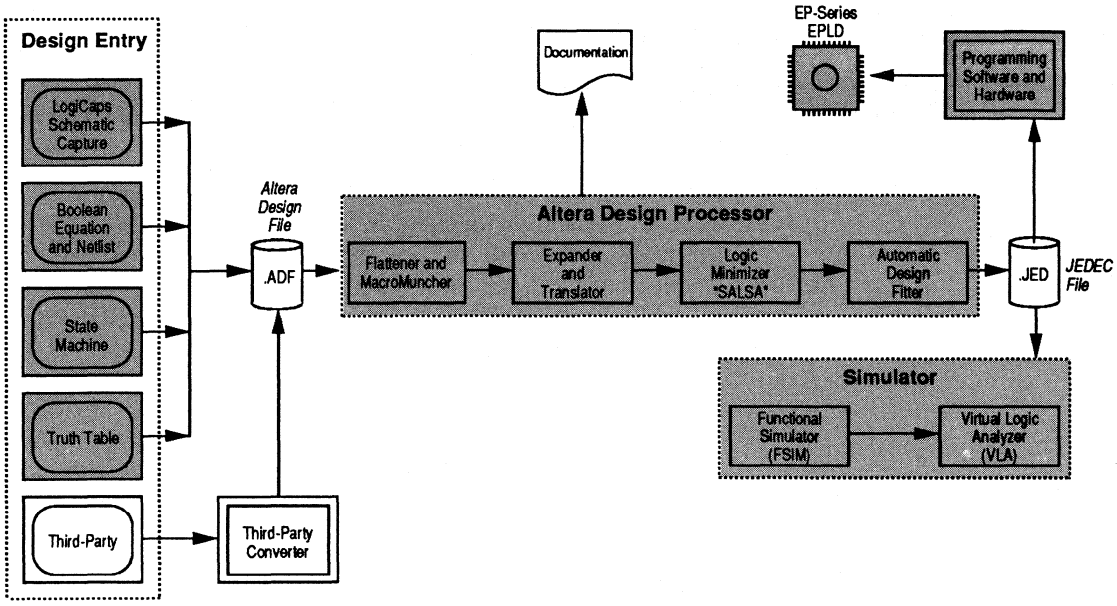
General Description

Altera's PLS-SUPREME (Enhanced A+PLUS Programmable Logic Software), shown in Figure 1, is a comprehensive CAE system for designing logic with the Altera EP-series (classic) EPLDs. PLS-SUPREME provides multiple design entry methods, including LogiCaps schematic capture, Boolean equation, state machine, truth table, and netlist design entry. These entry methods may be combined, allowing the designer to "mix and match" the entry methods that best suit each design. This package also interfaces with several third-party design entry formats. (Contact Altera Applications at 1 (800) 800-EPLD for more information.)

A+PLUS includes the Altera Design Processor (ADP), which consists of integrated modules that produce an industry-standard JEDEC file for programming the EPLD. The ADP implements logic minimization, automatic EPLD part selection, architecture optimization, and design fitting. In addition, the ADP produces documentation that shows minimized logic and EPLD utilization.

PLS-SUPREME also includes a functional simulator (FSIM) to verify designs, and LogicMap II software to program EPLDs. (Altera programming

Figure 1. A+PLUS Block Diagram



hardware is not included in PLS-SUPREME. Refer to the *PLCAD-SUPREME* or *PL-ASAP* data sheets for more information about products that include hardware.) PLS-SUPREME software runs on an IBM PS/2, PC/AT, or compatible computer and offers the most comprehensive support available for Altera's classic EPLDs.

Design Entry

A+PLUS provides the following design entry methods: LogiCaps schematic capture, Boolean equation, state machine, and truth table design entry.

LogiCaps Schematic Capture

Logic schematics are created with LogiCaps, which allows the user to quickly construct a wide range of logic circuits. LogiCaps provides two libraries that can be supplemented with libraries created by the user:

- The A+PLUS Primitive Library includes basic logic gates and flip-flops.
- The A+PLUS TTL MacroFunction Library has more than 120 TTL-equivalent macrofunctions, including counters, decoders, and comparators. This library also includes A+PLUS-specific macrofunctions that are optimized for the Altera EPLD architecture (see Table 1).
- User-defined libraries can be created easily with the Altera Design Librarian (ADLIB), allowing custom development of new macrofunction elements.

Table 1. A+PLUS Macrofunctions

Type	Available
Adders	7480, 7482, 7483, 74183, 8FADD
Comparators	7485, 74158, 74518, 8MCOMP
Converters	74184, 74185
Counters	7493, 74160, 74161, 74162, 74163, 74190, 74191, 74393, 74160T, 74161T, 74162T, 74163T, 74190T, 74191T, 74192T, 74193T, 8COUNT, 4COUNT, 16CUDSLR, UNICNT2, GRAY4
Decoders	7442, 7443, 7444, 7445, 7446, 7447, 7448, 7449, 74138, 74139, 74154, 74155, 74156
Flip-Flops	7470, 7471, 7472, 7473, 7474, 7476, 7478, 74173, 74174, 74175, 74273, 74374
Frequency Divider	FREQDIV
Latches	7475, 7477, 74116, 74259, 74279, 74373, NANDLTCH, NORLTCH
Multipliers	74261, MULT2, MULT24, MULT4
Multiplexers	74147, 74148, 74151, 74153, 74157, 74158, 74298, 21MUX
Parity Generators	74180, 74280
Shift Registers	7491, 7494, 7496, 7499, 74164, 74165, 74166, 74178, 74179, 74194, 74198, BARRELST, UNICNT2
SSI Functions	7400, 7402, 7404, 7408, 7410, 7411, 7420, 7421, 7427, 7430, 7432, 7486, INHB, CBUF
Storage Registers	7498, 74278
True/Comp Element	7487
ALU	74181

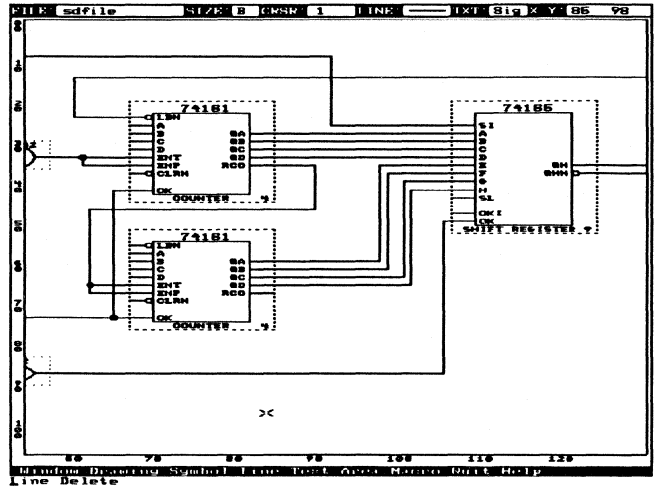
LogiCaps features easy-to-use mouse and keyboard command entry. Tag-and-drag editing with orthogonal rubberbanding, multiple zoom levels, and a dual-window display mode simplifies schematic entry. (See Figure 2.) Schematics can be printed on Epson FX- and LQ-series printers, and HP7475A, and 7585B, and 7495A drafting plotters. An Altera Design File (ADF) is generated when a design is saved and linked with other design files or processed directly with the Altera Design Processor (ADP) to generate a JEDEC file.

Boolean Equation Entry

The ADF syntax supports Boolean equation design entry and features free-form entry of all syntactical elements. Boolean equations need not be entered with a minimized sum-of-products form because the ADP automatically minimizes the equations before generating the JEDEC programming file. ADF-format versions of the 120 macrofunctions used with LogiCaps are also available. An ADF is created with any standard text editor (in non-document mode). Once a design has been entered, it can be linked with schematic or state machine files, or directly processed by A+PLUS.

Figure 2. Schematic Design Entry with LogiCaps

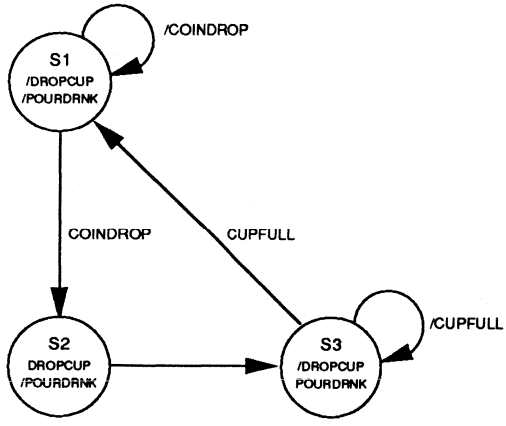
EPLD designs can be entered in LogiCaps with popular 7400-series symbols from the A+PLUS TTL MacroFunction Library.



State Machine and Truth Table Entry

State machine designs are entered in Altera's State Machine File (SMF) format (see Figure 3). This high-level language description features **IF–THEN** statements, **CASE** statements, and truth tables. Outputs of state machines may be defined conditionally or unconditionally, allowing flexible output structures to be merged with other portions of the design. SMF syntax also allows multiple state machines to be defined in a single file. Truth tables may also be used to specify random logic. Once a design has been entered, A+PLUS automatically generates state equations and transforms state machine descriptions, automatically choosing D or T registers for the state variables to ensure the most efficient use of EPLD resources.

Figure 3. State Machine Diagram and Partial State Machine File




```
MACHINE: dispenser
  CLOCK:    CLK
  STATES:   [DROPCUP   POURDRNK]
  S1        [   0          0    ]
  S2        [   1          0    ]
  S3        [   0          1    ]

S1:
  IF COINDROP THEN S2
  % No outputs %
S2:
  S3
S3:
  IF CUPFULL THEN S1
```

Design Processing

The Altera Design Processor (ADP) consists of a series of modules that automatically transforms the design into a JEDEC file used to program the EPLD. First, the design is "flattened" from high-level macrofunctions to low-level gate primitives. Next, the ADP analyzes the complete logic circuit and removes unused gates and flip-flops. This "MacroMunching" enables the designer to freely use high-level building blocks from the macrofunction library without worrying about optimizing the logic.

When the ADP detects macrofunctions with unconnected inputs, it assigns the following "intelligent" default values: active-high inputs default to GND, active-low inputs default to V_{CC}. Thus, the ADP enhances productivity by automatically performing some of the designer's "busy work."

The Translator module checks the design for logical completeness and consistency. For example, it ensures that no two logic function outputs are tied together and that all logic nodes have an origin. Also, if **AUTO** is entered as the EPLD name, the Translator automatically selects the EPLD best suited to the logic requirements of the design.

The Expander module expands the Boolean equations into sum-of-products form, checks for evidence of combinatorial feedback, and removes redundant factors from product terms.

Logic minimization of designs is performed by the Minimizer module. Minimization tools include Boolean minimization with SALSA (Speedy Altera Logic Simplifying Algorithm), which yields results that are superior to other heuristic reduction techniques. The Minimizer uses algorithms that select equations best represented by a complemented AND/OR function. This feature reduces product-term demands generated by complex logic functions. Moreover, for Altera EPLDs with programmable flip-flops, the Minimizer determines which type of flip-flop yields the most efficient solution and, if necessary, converts the architecture to D or T flip-flops.

The fully minimized design is then transferred to the Fitter module. The fitting routine relies on algorithms based on artificial intelligence software techniques to place and route the logic into the specified EPLD. If a pin assignment is specified, the Fitter matches the request. If no pin assignments are specified, the Fitter automatically finds the best fit for all pins.

Regardless of whether a fit is achieved, the Fitter generates a Utilization Report (**.RPT**) that documents macrocell and pin assignments, input and output pin names, and buried registers, as well as any unused resources. Figure 4 shows an excerpt from an **.RPT** file. At the end of design processing, the ADP generates an industry-standard JEDEC programming file.

Figure 4. Partial Utilization Report

The Utilization Report documents the EPLD resources that have been used.

```
ALTERA Design Processor Utilization Report
bevdis.rpt
@(#) FIT Version 7.0    7/2/90 23:56:49 39.16
 ***** Design implemented successfully

Your Name
Your Company
10/1/90
1.00
B
EP330
Beverage Dispenser Controller

Input files : bevdis.adf
ADP Options: Minimization = Yes,  Inversion Control = No,  LEF
Analysis = Yes

OPTIONS: TURBO = ON, SECURITY = OFF

                EP330
              - - - - -
     CLOCK -¦1      20¦- VCC
   CUPFULL -¦2      19¦- GND
  COINDROP -¦3      18¦- GND
       GND -¦4      17¦- GND
     RESET -¦5      16¦- GND
       GND -¦6      15¦- GND
    ENABLE -¦7      14¦- DROPCUP
       GND -¦8      13¦- STROBE
       GND -¦9      12¦- POURDRNK
       GND -¦10     11¦- GND
              - - - - -
```

Simulation

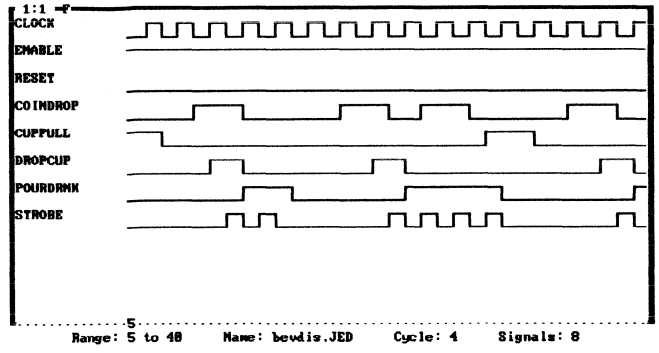
Included in the PLS-SUPREME package is a Functional Simulator (FSIM), which is a convenient and easy-to-use tool for testing design logic before it is committed to silicon. FSIM uses the JEDEC file generated by the ADP. Input logic values with state tables, vector patterns, or predefined patterns may be defined with any standard text editor. Design debugging is aided with the **BREAK**, **FORCE**, **SAVE**, and **RESTORE** commands; buried nodes can also be accessed. Interactive simulation results are displayed either graphically with the Virtual Logic Analyzer (VLA) or in a tabular format. The results may be printed in either format with an Epson or compatible printer. Figure 5 shows the output of the VLA. The designer can simulate interactively from the keyboard, or an optional Command File may be generated to perform batch-mode simulation.

Programming

LogicMap II programming software is included in the PLS-SUPREME package. The software uses the Altera Super Adaptive Programming (ASAP) algorithm, which significantly reduces the time required to program an EPLD. LogicMap II uses the JEDEC file created by A+PLUS and Altera programming hardware to program the target EPLD. LogicMap II also reads and produces JEDEC files from programmed EPLDs and verifies programmed devices.

Figure 5. Virtual Logic Analyzer Output

The VLA provides an interactive display of simulation results.



PLS-SUPREME Contents

- Floppy disks containing all programs and files for A+PLUS software for both PC-AT, PS/2, and compatible computers
  - Altera Design Processor
  - LogiCaps schematic capture
  - Functional Simulator (FSIM)
  - LogicMap II
- Documentation

Ordering Information

PLS-SUPREME

2

Notes:

Contents

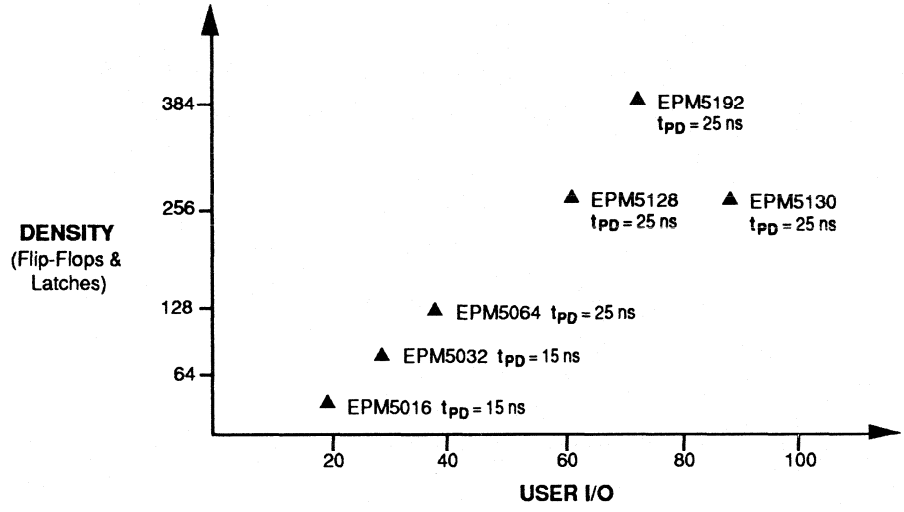
October 1990

Section 3 EPM5000-Series MAX EPLDs

2

EPM5000-Series MAX EPLDs

The Complete Industry-Standard Programmable Logic Family



- The Altera EPM5000-series Multiple Array MatriX (MAX) EPLDs offers the industry's most comprehensive family of programmable logic building blocks.
- Advanced MAX architecture provides the speed, ease of use, and familiarity of PAL devices with the density of programmable gate arrays.
- Modular family structure solves design tasks from fast 20-pin address decoders to 100-pin LSI custom peripherals.
- Non-volatile, reprogrammable EPROM technology aids prototype development.
- High sequential logic capacity provides up to 384 registers plus latches.
- Up to 66 product terms per output ensure efficient design of complex state machines.
- Exactly emulates all popular 7400-series functions to facilitate conversion of existing CMOS and TTL designs.
- Easily integrates multiple-package PAL and PLA designs.
- EPM5000-series EPLDs provide 15-ns combinatorial delays, counter frequencies up to 100 MHz, pipelined data rates up to 100 MHz, and high-complexity designs with true system-clock rates up to 66 MHz.
- A full selection of packages is provided, including DIP, SOIC, J-lead, PGA, and QFP formats in windowed ceramic and plastic one-time-programmable versions.
- Easily converts to custom-masked silicon for very-high-volume production.
- MAX EPLDs are supported with MAX+PLUS PC- and workstation-based design tools offering hierarchical schematic and Altera Hardware Description Language (AHDL) entry methods, an efficient logic synthesis-based compiler, and full timing simulator.
- Logic compilation and automatic place-and-route of 600- to 7,500-gate designs is performed in minutes.
- EDIF industry-standard workstation and third-party CAE tool interfaces are available.


ALTERA
EPM5032 SC
ALTERA
EPM5128 LC
ALTERA
EPM5192 JC
ALTERA
EPM5064 LC
ALTERA
EPM5130 QC
ALTERA
EPM5024 DC

EPM5016 to EPM5192

High-Speed, High-Density MAX EPLDs

October 1990, ver. 1 **Data Sheet**

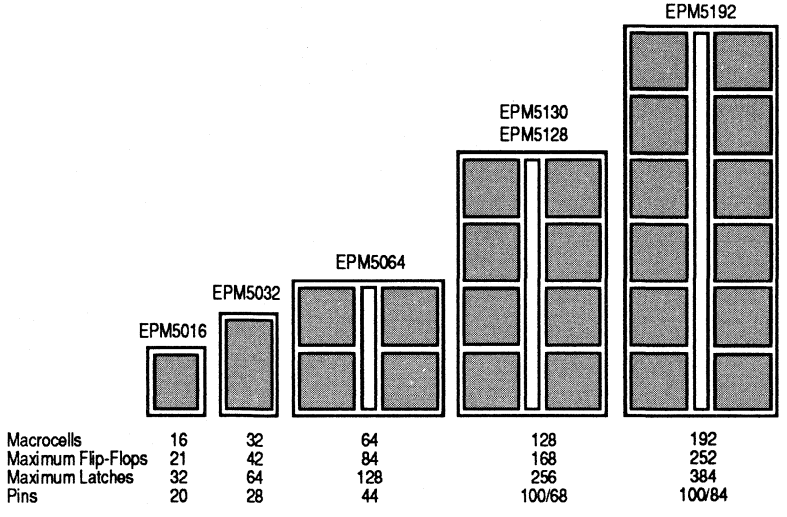
Features

- Complete family of CMOS EPLDs solves design tasks ranging from fast 20-pin address decoders to 100-pin LSI custom peripherals.
- The advanced MAX architecture combines the speed, ease of use, and familiarity of PAL devices with the density of programmable gate arrays.
- EP5000-series EPLDs provide 15-ns combinatorial delays, counter frequencies up to 100 MHz, pipelined data rates of 100 MHz, and high-complexity designs with true system clock rates up to 66 MHz.
- Available in a wide variety of packages, including DIP, SOIC, J-Lead, PGA, and QFP formats in windowed ceramic and plastic one-time-programmable versions.
- MAX+PLUS PC- and workstation-based development tools compile large designs in minutes.
- Industry-standard EDIF interfaces to workstation and third-party CAE tools are available.

3

Figure 1 shows the EPM5000-series modular architecture.

Figure 1. EPM5000-Series Modular Architecture



Family Highlights

- **Multiple Array MatriX (MAX) architecture solves speed, density, and design flexibility problems**
  - Advanced macrocell array provides registered, combinatorial, or flow-through latch operation.
  - Expander product-term array automatically provides additional combinatorial or registered logic.
  - Decoupled I/O block with dual feedback on I/O pins allows flexible pin utilization.
  - Programmable Interconnect Array provides automatic 100% routing in devices with multiple LABs.
  - Each macrocell supports synchronous or asynchronous operation of every macrocell, using single or multiple clocks per device.
- **EPM5000-Series Performance**
  - Pipelined data rates up to 100 MHz
  - Counters as fast as 100 MHz
  - t_{PD} performance from 15 ns to 25 ns
  - Advanced 0.8-micron CMOS EPROM technology
- **EPM5000-Series Logic Density**
  - 16- to 192-macrocell devices
  - 20- to 100-pin packages
  - 32 to 384 flip-flops and latches
  - More than 32 product terms on a single macrocell
  - Product-term expansion on any data or control path
- **MAX+PLUS Design Tools**
  - Design entry via unified, hierarchical schematic capture and Altera Hardware Description Language (AHDL)
  - Fast, automatic design processing with logic synthesis
  - Automatic device fitting, no hand editing needed
  - Hardware and software design verification tools
  - Compiles a 16-bit counter in 34 seconds on a 16-MHz 386 computer
- **EDIF interfaces to MAX+PLUS provide paths to Dazix, Valid Logic Systems, Mentor Graphics, and other workstation-based CAE tools.**

General Description

EPM5000-series Erasable Programmable Logic Devices (EPLDs) represent a revolutionary step in programmable logic: they combine innovative architecture and state-of-the-art process to offer optimum performance, logic density, flexibility, and the highest speeds and densities available in general-purpose reprogrammable logic. These EPLDs are high-speed, high-density replacements for SSI and MSI TTL and CMOS packages and conventional PLDs. For example, an EPM5192 replaces over 100 7400-series SSI and MSI TTL and CMOS packages, integrating complete subsystems into a single package, saving board area, and reducing power consumption.

These MAX EPLDs range in density from 16 to 192 macrocells. They are divided into two groups: higher-speed MAX EPLDs (EPM5016 and EPM5032) and higher-density MAX EPLDs (EPM5064, EPM5128, EPM5130, and EPM5192). The higher-speed MAX EPLDs achieve system clock frequencies of 66 MHz, and are capable of counter frequencies of 100 MHz.

Logic Array Blocks The EPM5016 and EPM5032 MAX EPLDs have a single Logic Array Block (LAB). The EPM5064, EPM5128, EPM5130, and EPM5192 MAX EPLDs contain multiple LABs. Each LAB contains a macrocell array, an expander product-term array, and a decoupled I/O block. Expander product terms (expanders) are unallocated, inverted product terms that can be used and shared by all macrocells in the LAB to create combinatorial and registered logic. Thus, expressions requiring up to 66 product terms can be implemented in a single macrocell. Signals in the higher-density devices are routed between multiple LABs by a Programmable Interconnect Array (PIA) that ensures 100% routability. This multiple array architecture enables EPM5000-series EPLDs to offer the speed of smaller arrays with the integration density of larger arrays.

Modular Architecture The modular architecture of MAX EPLDs provides integration solutions over a wide range of logic densities. Migration from one type of device to another is easy. For example, the EPM5128 and EPM5130 EPLDs have the same logic capacity, but have packages optimized to handle different I/O requirements. Over the entire family, a wide range of packaging options for both through-hole and surface-mount applications is available. Plastic one-time-programmable (OTP) packages are available for economical volume production.

Logic Design Entry Logic designs are created and programmed into EPM5000-series EPLDs with the MAX+PLUS Development System. MAX+PLUS is a complete CAE system offering hierarchical design entry tools, automatic design compilation and fitting, timing simulation, and device programming. The MAX+PLUS Compiler features advanced logic synthesis algorithms, allowing designs to be entered in a variety of high-level formats while ensuring the most efficient use of EPLD resources. The combination of a flexible architecture and advanced CAE tools ensures rapid design cycles so that a design may go from conception to completion in single day. Interfaces to third-party tools are also available to allow design entry and logic simulation on a variety of workstation platforms.

Functional Description

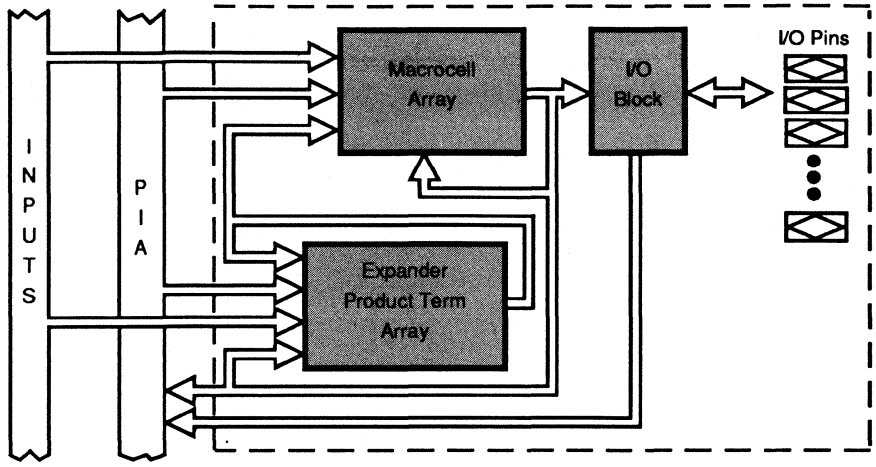
EPM5000-series EPLDs use CMOS EPROM cells to configure logic functions within the devices. MAX architecture is user-configurable to accommodate a variety of independent logic functions, and the EPLDs can be erased for quick and efficient iterations during design development and debug cycles.

Logic Array Block

EPM5000-series EPLDs contain from 1 to 12 Logic Array Blocks. Each LAB, shown in Figure 2, consists of a macrocell array, an expander product-term array, and an I/O control block. (The number of macrocells and expanders in the arrays varies with each device.) Macrocells are the primary resource for logic implementation, but if needed, expanders can be used to supplement the capabilities of any macrocell. The expander product-term array consists of a group of unallocated, inverted product terms. Flexible macrocells and allocatable expanders facilitate variable product-term designs

Figure 2. Logic Array Block

The LAB consists of a macrocell array, an expander product-term array, and a decoupled I/O block. The flexibility of the LAB ensures high speeds and efficient device utilization.



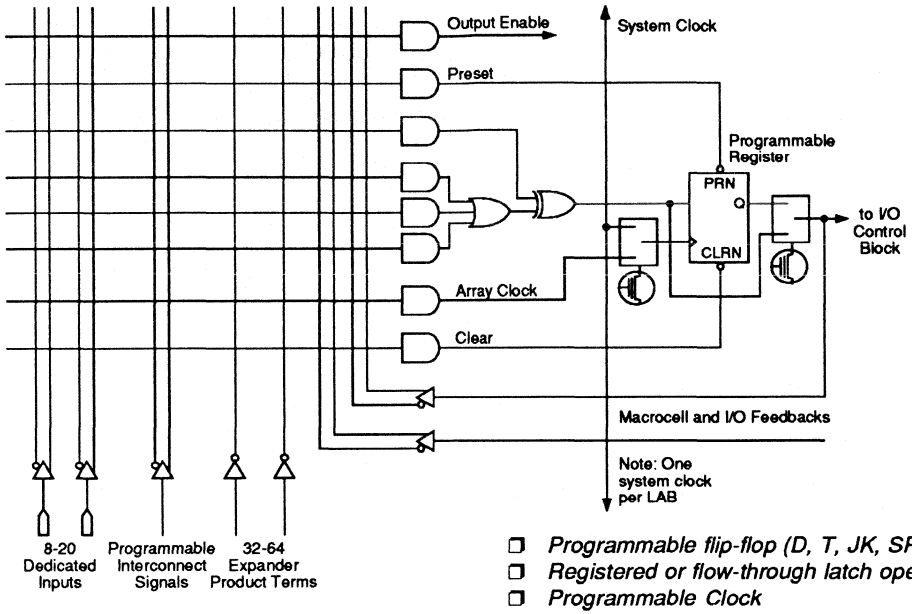
without the waste associated with fixed product-term architectures. Thus, PAL or PLA devices are easily integrated into MAX EPLDs. The outputs of the macrocells feed the decoupled I/O block, which consists of a group of programmable tri-state buffers and I/O pins. In the EPM5064, EPM5128, EPM5130, and EPM5192, multiple LABs are connected by a Programmable Interconnect Array (PIA).

Macrocells

The EPM5000-series macrocell, shown in Figure 3, consists of a programmable logic array and an independently configurable register. This register may be programmed for D, T, JK, or SR operation; or as a flow-through latch; or bypassed for purely combinatorial operation. Combinatorial logic is implemented in the programmable logic array, which consists of three product terms ORed together that feed one input of an **XOR** gate. The second input to the **XOR** gate is also controlled by a product term that makes it possible to implement active-high or active-low logic. The **XOR** gate is also used for complex **XOR** arithmetic logic functions and for De Morgan's inversion to reduce the number of product terms. The output of the **XOR** gate feeds the programmable register, or bypasses it for purely combinatorial operation. The logic array ensures high speed while eliminating inefficient, unused product terms. Also, expanders can be allocated to enhance the capability of the logic array.

Additional product terms, called secondary product terms, are used for Output Enable, Preset, Clear, and Clock logic. Preset and Clear product terms drive the active-low asynchronous Preset and asynchronous Clear inputs to the configurable flip-flop. The Clock product term allows each register to have an independent Clock and supports positive- and negative-

Figure 3. EPM5000-Series Macrocell



- *Programmable flip-flop (D, T, JK, SR)*
- *Registered or flow-through latch operation*
- *Programmable Clock*
- *Asynchronous Preset and Clear*

3

edge-triggered operation. Macrocells that drive an output pin may use the Output Enable product term to control the active-high tri-state buffer in the I/O control block. These secondary product terms allow 7400-series TTL functions to be emulated exactly.

The EPM5000-series macrocell configurability makes it possible to efficiently integrate complete subsystems into a single device. All macrocell outputs are globally routed within an LAB and also feed the PIA to provide efficient routing of signal-intensive designs.

Clock Options

Each LAB has two clocking modes: asynchronous and synchronous. During asynchronous clocking, each flip-flop is clocked by a product term. Thus, any input or internal logic may be used as a clock. Systems that require multiple clocks are easily integrated into EPM5000-series EPLDs. Asynchronous clocking also allows each flip-flop to be configured for positive- or negative- edge-triggered operation, giving the macrocell a high degree of flexibility.

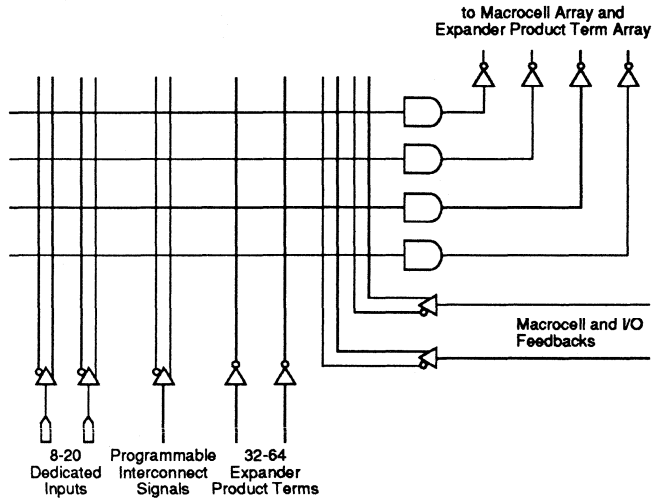
Synchronous clocking is provided by a dedicated system clock (**CLK**). This direct connection provides enhanced clock-to-output delay times. Since each LAB has one synchronous clock, all flip-flop clocks within it are positive-edge-triggered from the **CLK** pin. If the **CLK** pin is not used as a system clock, it may be used as a dedicated input.

Expander Product Terms

The expander product-term array (Figure 4) contains unallocated, inverted product terms that enhance the macrocell array. Expanders can be used and shared by all product terms in the LAB. Wherever extra logic is needed (including register control functions), expanders can be used to implement the logic. These expanders provide the flexibility to implement register-intensive and product-term-intensive designs for MAX EPLDs.

Figure 4. Expander Product Terms

Expander product terms are unallocated logic that can be used and shared by all macrocells in an LAB. Sharing allows efficient integration of complex combinatorial functions.



Expanders are fed by all signals in the LAB. One expander may feed all macrocells in the LAB or multiple product terms in the same macrocell. Since expanders also feed the secondary product terms of each macrocell, complex logic functions can be implemented without using another macrocell. Expanders can be cross-coupled to build additional flip-flops or latches.

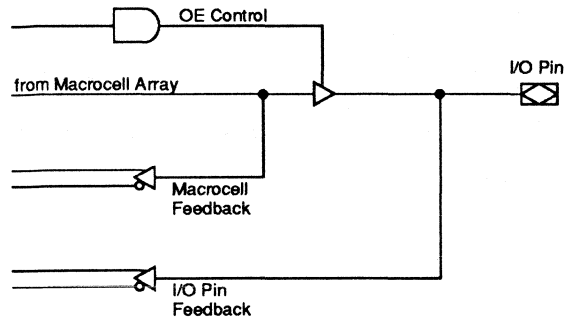
I/O Control Block

Each LAB has an I/O control block (Figure 5) that consists of a user-configurable I/O control function for each I/O pin. The I/O control block is fed by the macrocell array. The tri-state buffer is controlled by a dedicated macrocell product term, and drives the I/O pad.

Each MAX EPLD has dual feedback—one feedback path before and one after the tri-state buffer— for every I/O pin. The tri-state buffer decouples the I/O pins from the macrocells so that all registers within the LAB can be "buried." Thus, I/O pins can be configured as dedicated input, output, or bidirectional pins. In multiple-LAB MAX devices, I/O pins feed the PIA.

Figure 5. I/O Control Block

The decoupled I/O control block features dual feedback to maximize use of device pins.



Programmable Interconnect Array

The higher-density EPM5000-series devices (EPM5064, EPM5128, EPM5130, and EPM5192) use a Programmable Interconnect Array (PIA) to route signals between the various LABs. The PIA routes only the signals required for implementing logic in an LAB, and is fed by all macrocell feedbacks and all I/O pin feedbacks. Unlike channel routing in masked or programmable gate arrays—where routing delays are variable and path-dependent—the PIA has a fixed delay. Because the PIA eliminates skew between signals, timing performance is easy to predict.

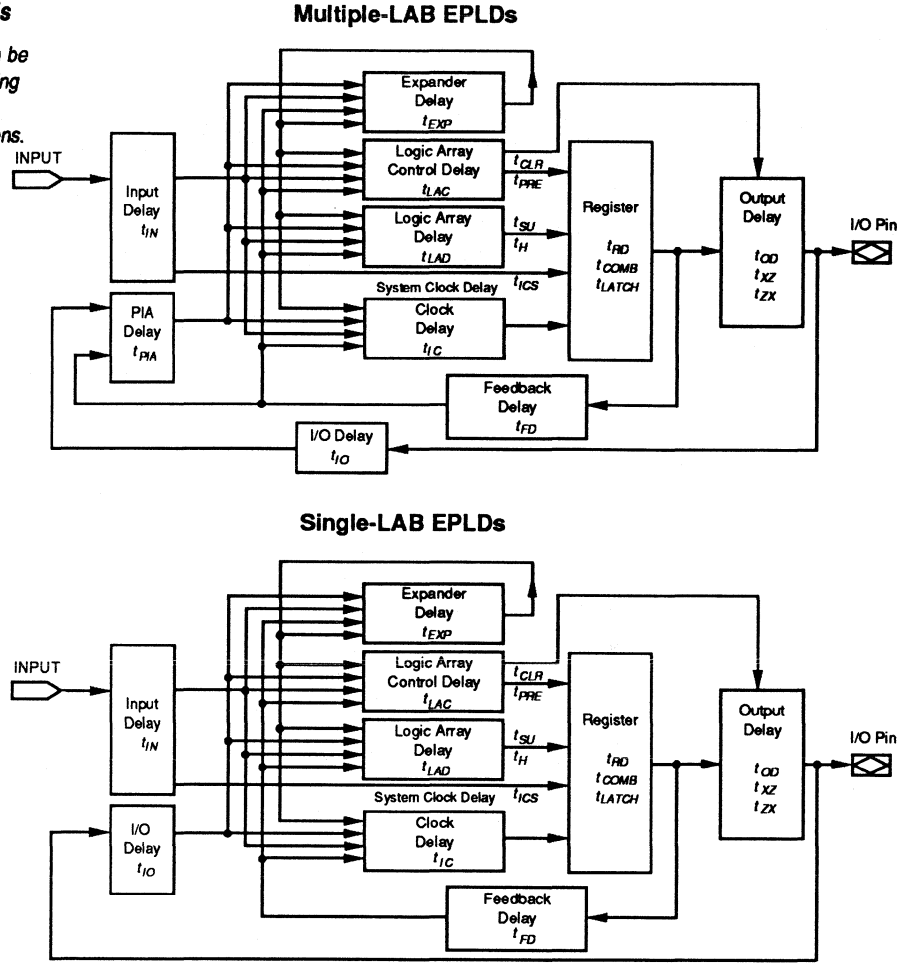
3

Timing Model

Timing within EPM5000-series EPLDs is easily determined with MAX+PLUS software or with the models shown in Figure 6. EPM5000-series EPLDs have fixed internal delays, that allow the user to determine the worst-case timing delays for any design. For complete timing information, MAX+PLUS software provides a timing simulator, a delay predictor, and a detailed timing analyzer.

Figure 6. Timing Models

Design performance can be predicted with these timing models and the device performance specifications.



The timing models shown in Figure 6 may be used together with the internal timing parameters for a particular EPLD to derive timing information. External timing parameters are derived from a sum of internal parameters and represent pin-to-pin timing delays. Figure 7 shows the internal timing waveforms for these devices. Refer to *Application Brief 75 (EPM5000-Series MAX EPLD Timing)* in this data book for further information.

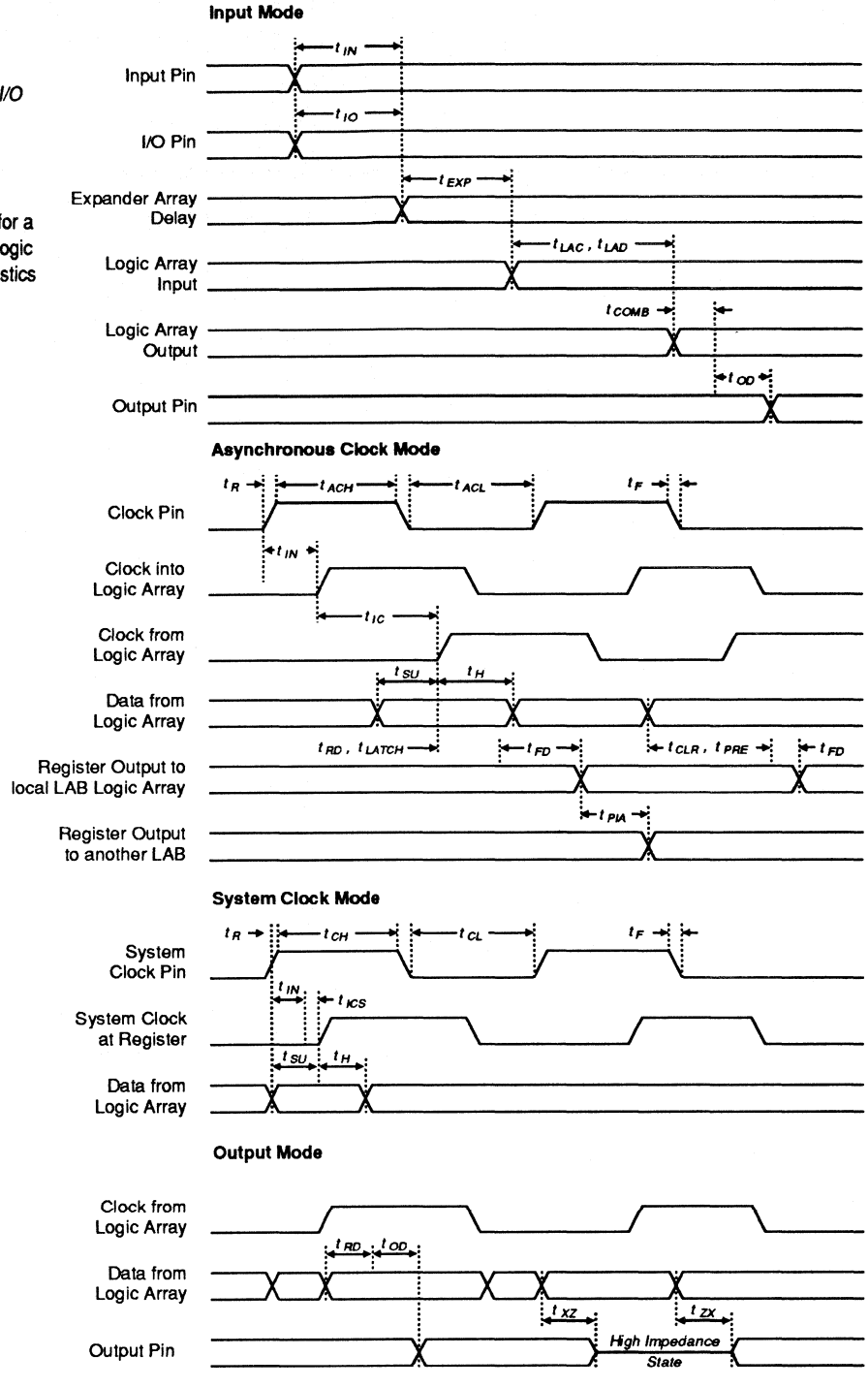
Design Security

MAX EPLDs contain a programmable Security Bit that controls access to the data programmed into the device. If this feature is used, a proprietary design implemented in the device cannot be copied or retrieved. This feature provides a high level of design security, since programmed data within EPROM cells is invisible. The Security Bit that controls this function, as well as all other program data, is reset by erasing the EPLD.

Figure 7. Switching Waveforms

In multiple LAB EPLDs, I/O pins used as inputs can traverse the PIA.

t_R & t_F < 3 ns.
Inputs are driven at 3 V for a logic high and 0 V for a logic low. All timing characteristics are measured at 1.5 V.

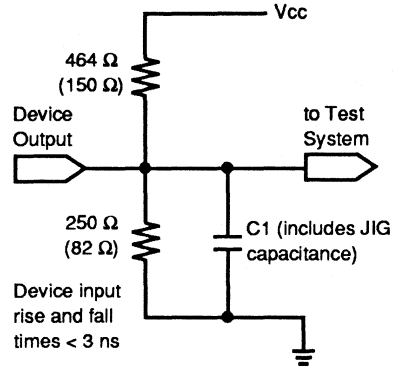


MAX EPLDs are fully functionally tested and guaranteed. Complete testing of each programmable EPROM bit and all internal logic elements ensures 100% programming yield. AC test measurements are performed under the conditions shown in Figure 8.

Figure 8. AC Test Conditions

Power supply transients can affect AC measurements. Simultaneous transitions of multiple outputs should be avoided for accurate measurement. Threshold tests should not be performed under AC conditions. Large-amplitude, fast-ground current transients normally occur as the device outputs discharge the load capacitances. When these transients flow through the parasitic inductance between the device ground pin and the test system ground, it can create significant reductions in observable input noise immunity.

Note: Numbers in parentheses are for the EPM5016.



Test programs may be used and then erased during early stages of the production flow. This facility to use application-independent, general-purpose tests is called generic testing and is unique among user-configurable logic devices. EPLDs also contain on-board logic test circuitry to allow verification of function and AC specifications once they are packaged in windowless packages.

MAX+PLUS Development System

The MAX+PLUS Development System is a unified CAE system for integrating designs into EPM5000-series MAX EPLDs. Designs can be entered as logic schematics with the Graphic Editor or as state machines, truth tables, and Boolean equations with the Altera Hardware Description Language (AHDL). Logic synthesis and minimization optimize the logic of a design. Design verification and timing analysis are performed with the Simulator or the delay prediction feature. Errors in a design are automatically located and highlighted in the schematic or text design file. Hosted on IBM PS/2, PC-AT, or compatible machines, and workstations (e.g., Apollo, Sun, IBM), MAX+PLUS gives the designer the tools to quickly and efficiently create complex logic designs. Further details about the MAX+PLUS Development System are available in the *PLS-MAX Data Sheet*.

Device Programming

EPM5000-series EPLDs may be programmed on an IBM PS/2, PC-AT or compatible computer with an Altera Logic Programmer card, the PLE3-12A Master Programming Unit, and an appropriate device adapter. These items are included in the complete PLDS-MAX Development System or may be purchased separately. EPM5000-series EPLDs may also be programmed with third-party hardware (see the *Third-Party Development & Programming Support Data Sheet* in this data book). Contact Altera or your programming equipment manufacturer for more information.

Features

- Fast 20-pin MAX single-LAB EPLD
  - Combinatorial speeds with t_{PD} = 15 ns
  - Counter frequencies up to 100 MHz
  - Pipelined data rates up to 100 MHz
- 16 individually configurable macrocells
- 32 expander product terms (expanders) that allow 34 product terms in a single macrocell
- Up to 21 flip-flops or 32 latches
- Up to 10 input latches that can be constructed with cross-coupled expanders
- 24-mA output drivers to allow direct interfacing to system buses
- Programmable I/O architecture allowing up to 16 inputs and 8 outputs
- Available in 20-pin windowed ceramic or plastic DIP, plastic J-lead (PLCC), and plastic 300-mil SOIC packages

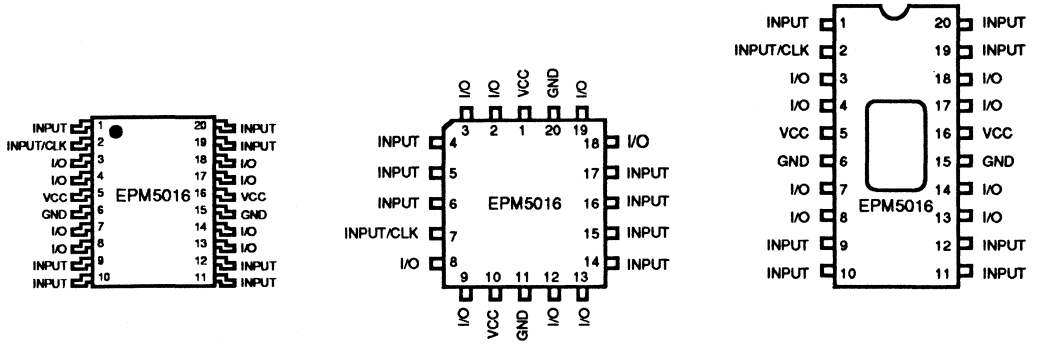
General Description

The Altera EPM5016 (Figure 9) is a Multiple Array MatriX (MAX) CMOS EPLD optimized for speed. It can integrate multiple SSI and MSI TTL and 74HC devices. In addition, it can replace any 20-pin PAL or PLA device with logic left over for further integration.

3

Figure 9. EPM5016 Pin-Out Diagrams

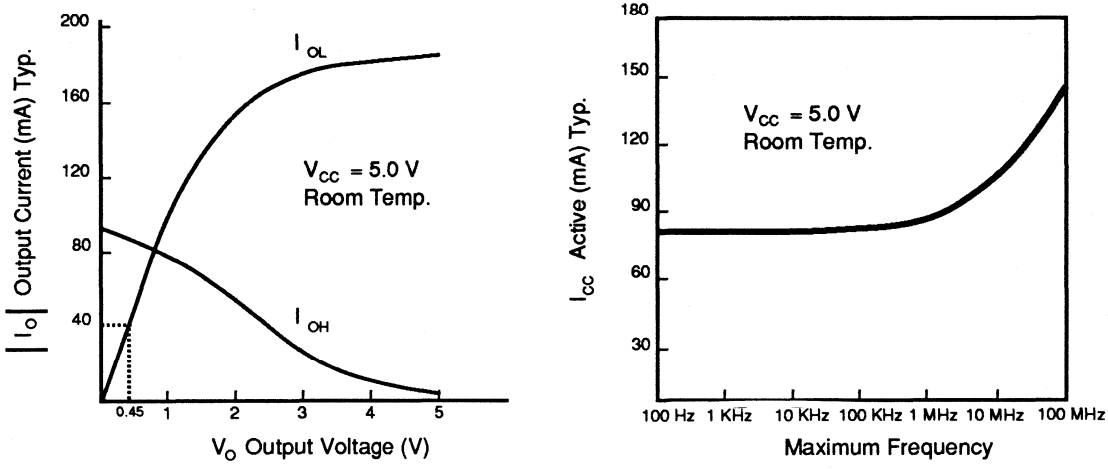
Package outlines not drawn to scale.



SOIC J-Lead DIP

Figure 10 shows output drive characteristics of EPM5016 I/O pins and typical supply current versus frequency for the EPM5016.

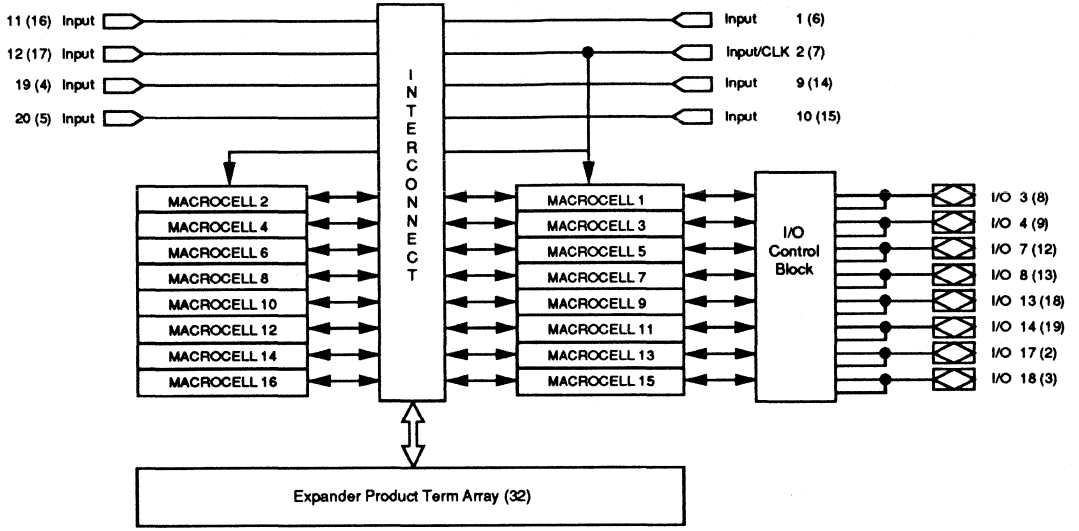
Figure 10. EPM5016 Output Drive Characteristics and I_{CC} vs. Frequency



The EPM5016, shown in Figure 11, contains 16 macrocells. The expander product-term array for the EPM5016 contains 32 expanders. The I/O control block contains 8 bidirectional I/O pins that can be configured for dedicated input, dedicated output, or bidirectional operation. All I/O pins feature dual feedback for maximum pin flexibility.

Figure 11. EPM5016 Block Diagram

The EPM5016 has 16 macrocells and 32 expanders.
Numbers in parentheses are for the PLCC package.



3

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current			200	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			1000	mW
T_{STG}	Storage temperature	No bias	–65	+150	°C
T_{AMB}	Ambient temperature	Under bias	–65	+135	°C
T_J	Junction temperature	Under bias		+150	°C

Recommended Operating Conditions *See Note (2)*

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	°C
T_A	Operating temperature	For industrial use	–40	+85	°C
T_C	Case temperature	For military use	–55	+125	°C
t_R	Input rise time			100	ns
t_F	Input fall time			100	ns

DC Operating Conditions *See Notes (2), (3), and (4)*

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –12 mA DC	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 24 mA DC			0.5	V
I_I	Input leakage current	$V_I = V_{CC}$ or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	$V_O = V_{CC}$ or GND	–40		+40	µA
I_{CC1}	V_{CC} supply current (standby)	$V_I = V_{CC}$ or GND		80	110 (150)	mA
I_{CC3}	V_{CC} supply current	$V_I = V_{CC}$ or GND No load, f = 1.0 MHz *See Note (5)*		85	115 (175)	mA

Capacitance

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		12	pF

AC Operating Conditions *See Note (4)*

External Timing Parameters			EPM5016-1		EPM5016-2		EPM5016		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{PD1}	Input to non-registered output	C1 = 35 pF		15		17		20	ns
t_{PD2}	I/O input to non-reg. output	C1 = 35 pF		15		17		20	ns
t_{SU}	Setup time		6		8		11		ns
t_{H}	Hold time		0		0		0		ns
t_{CO1}	Clock to output delay	C1 = 35 pF		9		11		13	ns
t_{ASU}	Asynchronous setup time		5		7		9		ns
t_{AH}	Asynchronous hold time		5		7		8		ns
t_{CH}	Clock high time		5		6		8		ns
t_{CL}	Clock low time		5		6		8		ns
t_{ACH}	Asynchronous clock high time		4		5		7		ns
t_{ACL}	Asynchronous clock low time		6		7		9		ns
t_{ACO1}	Asynch. clock to output delay	C1 = 35 pF		15		17		20	ns
t_{CNT}	Minimum clock period			10		12		16	ns
f_{CNT}	Internal maximum frequency		100		83.3		62.5		MHz
t_{ACNT}	Minimum asynch. clock period	*See Note (6)*		10		12		16	ns
f_{ACNT}	Max. internal asynch. frequency	*See Note (6)*	100		83.3		62.5		MHz
f_{MAX}	Max. frequency; pipelined data		100		83.3		62.5		MHz

For information on internal timing parameters, refer to *App. Brief 75 (EPM5000-Series MAX EPLD Timing).*

Internal Timing Parameters			EPM5016-1		EPM5016-2		EPM5016		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{IN}	Input pad and buffer delay			4		5		5	ns
t_{IO}	I/O input pad and buffer delay			4		5		5	ns
t_{EXP}	Expander array delay			5		8		10	ns
t_{LAD}	Logic array delay			6		7		9	ns
t_{LAC}	Logic control array delay			4		5		7	ns
t_{OD}	Output buffer and pad delay	C1 = 35 pF		4		4		5	ns
t_{ZX}	Output buffer enable delay			7		7		8	ns
t_{XZ}	Output buffer disable delay	C1 = 5 pF		7		7		8	ns
t_{SU}	Register setup time		2		5		8		ns
t_{LATCH}	Flow-through latch delay			1		1		1	ns
t_{RD}	Register delay			1		1		1	ns
t_{COMB}	Combinatorial delay			1		1		1	ns
t_{H}	Register hold time		6		8		9		ns
t_{IC}	Clock delay			6		6		8	ns
t_{ICS}	System clock delay			0		1		2	ns
t_{FD}	Feedback delay			1		1		1	ns
t_{PRE}	Register preset time			3		6		6	ns
t_{CLR}	Register clear time			3		6		6	ns

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods shorter than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature-range versions.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use.
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use.
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use.
(5) Measured with device programmed as a 16-bit counter.
(6) This parameter is measured with a positive-edge-triggered clock at the register. For negative-edge clocking, the t_{ACH} and t_{ACL} parameters must be swapped.

Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EPM5016-1, EPM5016-2, EPM5016
Industrial	(–40° C to 85° C)	EPM5016
Military	(–55° C to 125° C)	EPM5016

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Features

- Fast 28-pin DIP or J-lead single-LAB MAX EPLD
  - Combinatorial speeds with t_{PD} = 15 ns
  - Counter frequencies up to 76 MHz
  - Pipelined data rates up to 83 MHz
- 32 individually configurable macrocells
- 64 expander product terms (expanders) that allow 66 product terms on a single macrocell
- Up to 42 flip-flops or 64 latches
- Up to 21 input latches that can be constructed with cross-coupled expanders
- Programmable I/O architecture allowing up to 24 inputs and 16 outputs
- Available in 28-pin windowed ceramic or plastic one-time-programmable DIP and J-lead packages, as well as plastic 300-mil SOIC packages

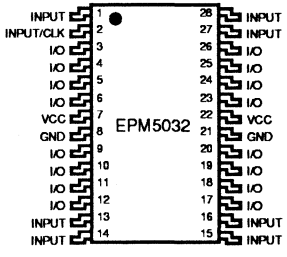
General Description

The Altera EPM5032 (Figure 12) is a Multiple Array MatriX (MAX) CMOS EPLD optimized for speed. It can integrate multiple SSI and MSI TTL and 74HC devices. In addition, it can replace multiple 20-pin PAL or PLA devices with logic left over for further integration.

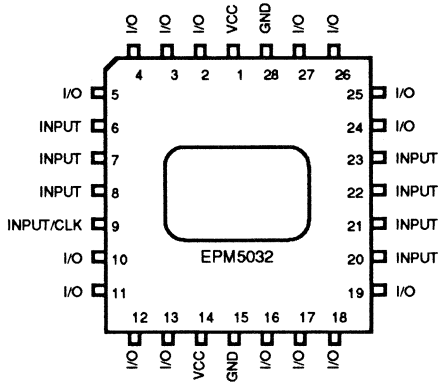
3

Figure 12. EPM5032 Pin-Out Diagrams

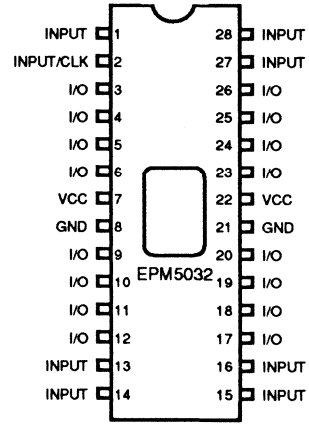
Package outlines not drawn to scale.



SOIC



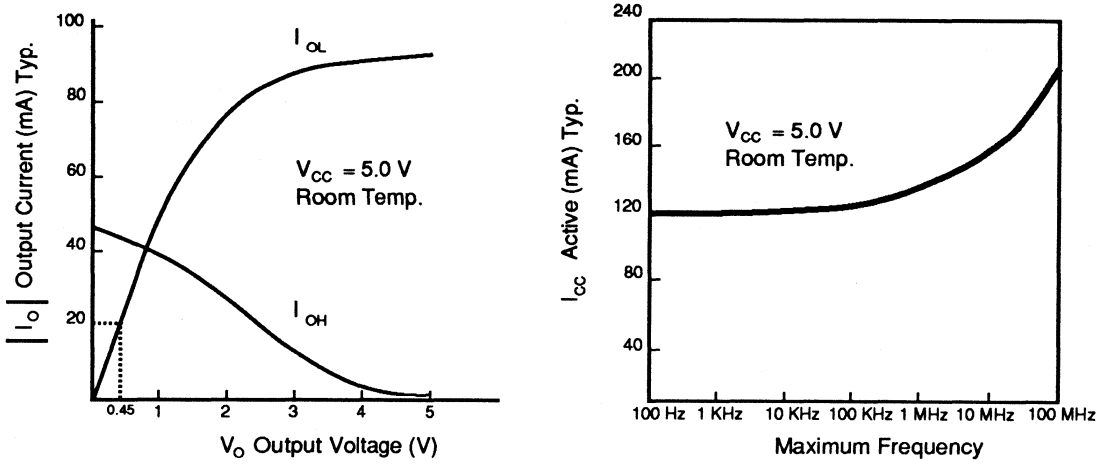
J-Lead



DIP

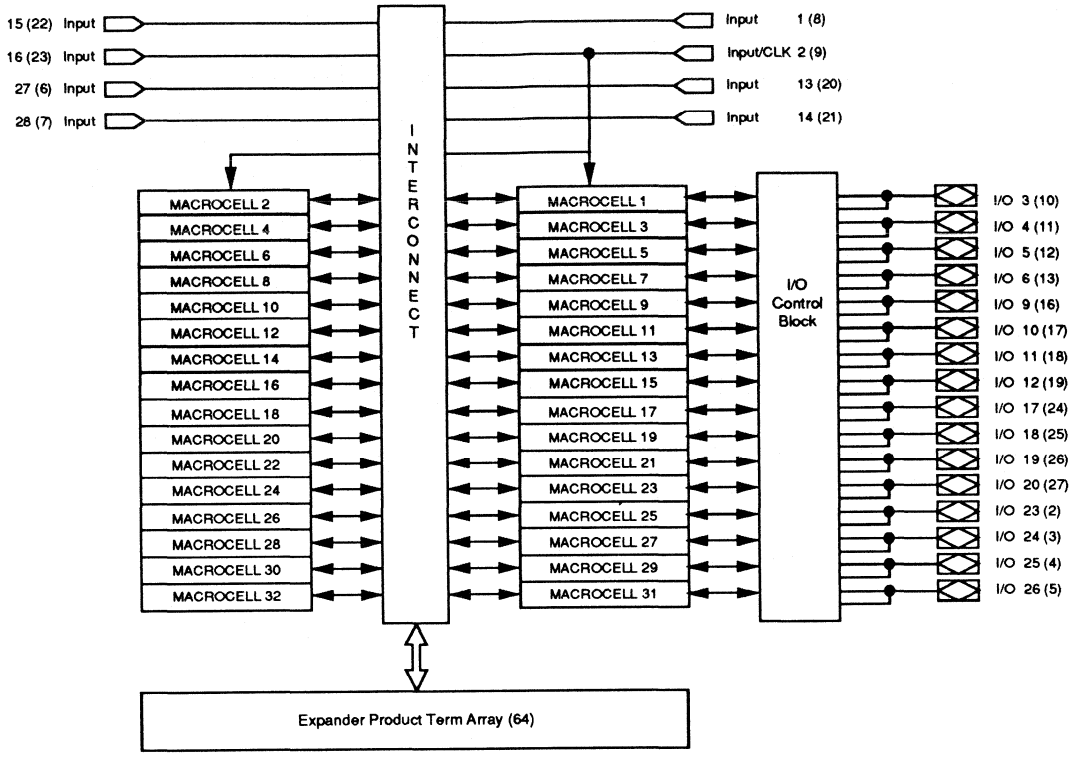
Figure 13 shows output drive characteristics of EPM5032 I/O pins and typical supply current versus frequency for the EPM5032.

Figure 13. EPM5032 Output Drive Characteristics and I_{CC} vs. Frequency



The EPM5032, shown in Figure 14, contains 32 macrocells. The EPM5032 expander product-term array contains 64 expanders. The I/O control block contains 16 bidirectional I/O pins that can be configured for dedicated input, dedicated output, or bidirectional operation. All I/O pins feature dual feedback for maximum pin flexibility.

Figure 14. EPM5032 Block Diagram *The EPM5032 has 32 macrocells and 64 expanders. Numbers in parentheses are for J-lead packages.*



3

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current			300	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			1500	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C
T_J	Junction temperature	Under bias		+150	° C

Recommended Operating Conditions *See Note (2)*

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time			100	ns
t_F	Input fall time			100	ns

DC Operating Conditions *See Notes (2), (3), (4)*

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 8 mA DC			0.45	V
I_I	Input leakage current	$V_I = V_{CC}$ or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	$V_O = V_{CC}$ or GND	–40		+40	µA
I_{CC1}	V_{CC} supply current (standby)	$V_I = V_{CC}$ or GND		120	150 (200)	mA
I_{CC3}	V_{CC} supply current	$V_I = V_{CC}$ or GND No load, f = 1.0 MHz *See Note (5)*		125	155 (225)	mA

Capacitance

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		12	pF

AC Operating Conditions *See Note (4)*

External Timing Parameters			EPM5032-1		EPM5032-2		EPM5032		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{PD1}	Input to non-registered output	C1 = 35 pF		15		20		25	ns
t_{PD2}	I/O input to non-reg. output	C1 = 35 pF		15		20		25	ns
t_{SU}	Setup time		9		12		15		ns
t_{H}	Hold time		0		0		0		ns
t_{CO1}	Clock to output delay	C1 = 35 pF		10		12		15	ns
t_{ASU}	Asynchronous setup time		7		9		12		ns
t_{AH}	Asynchronous hold time		7		9		12		ns
t_{CH}	Clock high time		6		7		8		ns
t_{CL}	Clock low time		6		7		8		ns
t_{ACH}	Asynchronous clock high time		6		7		9		ns
t_{ACL}	Asynchronous clock low time		7		9		11		ns
t_{ACO1}	Asynch. clock to output delay	C1 = 35 pF		15		20		25	ns
t_{CNT}	Minimum clock period			13		16		20	ns
f_{CNT}	Internal maximum frequency		76.9		62.5		50		MHz
t_{ACNT}	Min. asynch. clock period	*See Note (6)*		13		16		20	ns
f_{ACNT}	Max. internal asynch. frequency	*See Note (6)*	76.9		62.5		50		MHz
f_{MAX}	Max. frequency; pipelined data		83.3		71.4		62.5		MHz

For information on internal timing parameters, refer to *App. Brief 75 (EPM5000-Series MAX EPLD Timing)*.

Internal Timing Parameters			EPM5032-1		EPM5032-2		EPM5032		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{IN}	Input pad and buffer delay			4		5		7	ns
t_{IO}	I/O input pad and buffer delay			4		5		7	ns
t_{EXP}	Expander array delay			8		10		15	ns
t_{LAD}	Logic array delay			6		9		10	ns
t_{LAC}	Logic control array delay			4		7		7	ns
t_{OD}	Output buffer and pad delay	C1 = 35 pF		4		5		5	ns
t_{ZX}	Output buffer enable delay			7		8		11	ns
t_{XZ}	Output buffer disable delay	C1 = 5 pF		7		8		11	ns
t_{SU}	Register setup time		5		5		8		ns
t_{LATCH}	Flow-through latch delay			1		1		3	ns
t_{RD}	Register delay			1		1		1	ns
t_{COMB}	Combinatorial delay			1		1		3	ns
t_{H}	Register hold time		6		9		12		ns
t_{IC}	Clock delay			6		8		10	ns
t_{ICS}	System clock delay			1		2		3	ns
t_{FD}	Feedback delay			1		1		1	ns
t_{PRE}	Register preset time			5		6		9	ns
t_{CLR}	Register clear time			5		6		9	ns

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods shorter than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature versions.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) V_{CC} = 5 V ± 5%; T_A = 0° C to 70° C for commercial use.
V_{CC} = 5 V ± 10%; T_A = –40° C to 85° C for industrial use.
V_{CC} = 5 V ± 10%; T_C = –55° C to 125° C for military use.
(5) Measured with device programmed as a 32-bit counter.
(6) This parameter is measured with a positive-edge-triggered clock at the register. For negative-edge clocking, the t_{ACH} and t_{ACL} parameters must be swapped.

Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EPM5032-1, EPM5032-2, EPM5032
Industrial	(–40° C to 85° C)	EPM5032
Military	(–55° C to 125° C)	EPM5032

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Features

- High-density 64-macrocell general-purpose MAX EPLD
- 128 shareable expander product terms providing flexible logic expansion
  - Over 32 product terms in a single macrocell
  - 64 additional latches provided by cross-coupled expanders
- Multiple-LAB MAX architecture with t_{PD} = 25 ns, counter frequencies up to 50 MHz, and pipelined data rates up to 62.5 MHz
- Programmable I/O architecture allowing up to 36 inputs and 28 outputs
- 44-pin J-lead package that easily integrates 10 standard PALs in 1/2 square inch of board space; windowed ceramic or plastic one-time-programmable packages for volume production

General Description

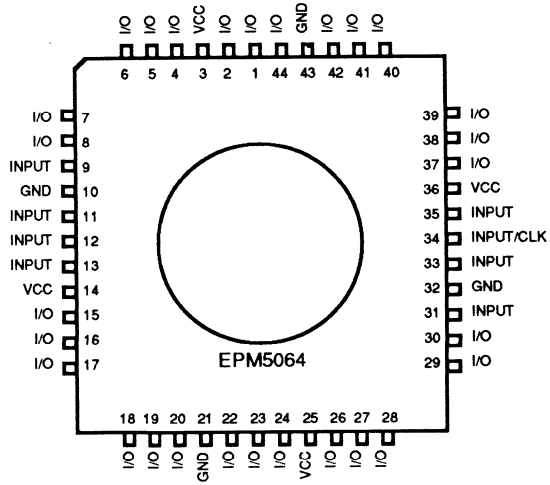
The Altera EPM5064 is a user-configurable, high-performance MAX EPLD that serves as a high-density replacement for 7400-series SSI and MSI TTL and CMOS logic. In addition, it can integrate multiple 20- and 24-pin low-density PLDs. For example, the EPM5064 can integrate the logic contained in over 10 standard 20-pin PALs.

Figure 15 shows the package pin-out for the EPM5064 J-lead package. This package occupies only 1/2 square inch of board space.

3

Figure 15. EPM5064 Pin-Out Diagram

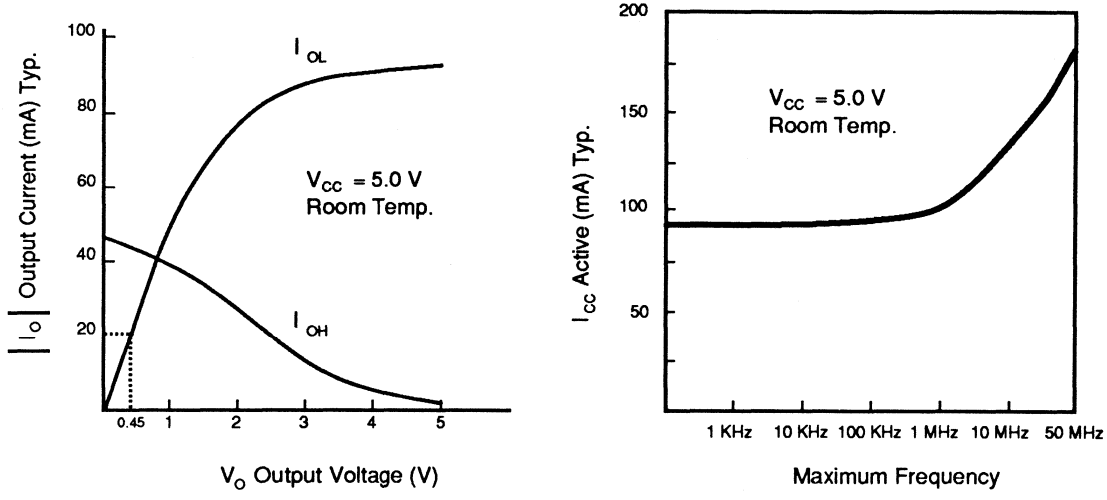
Package outline not drawn to scale.



J-Lead

Figure 16 shows output drive characteristics of EPM5064 I/O pins and typical supply current versus frequency for the EPM5064. The high integration density of the EPM5064 often greatly reduces system power requirements.

Figure 16. EPM5064 Output Drive Characteristics and I_{CC} vs. Frequency

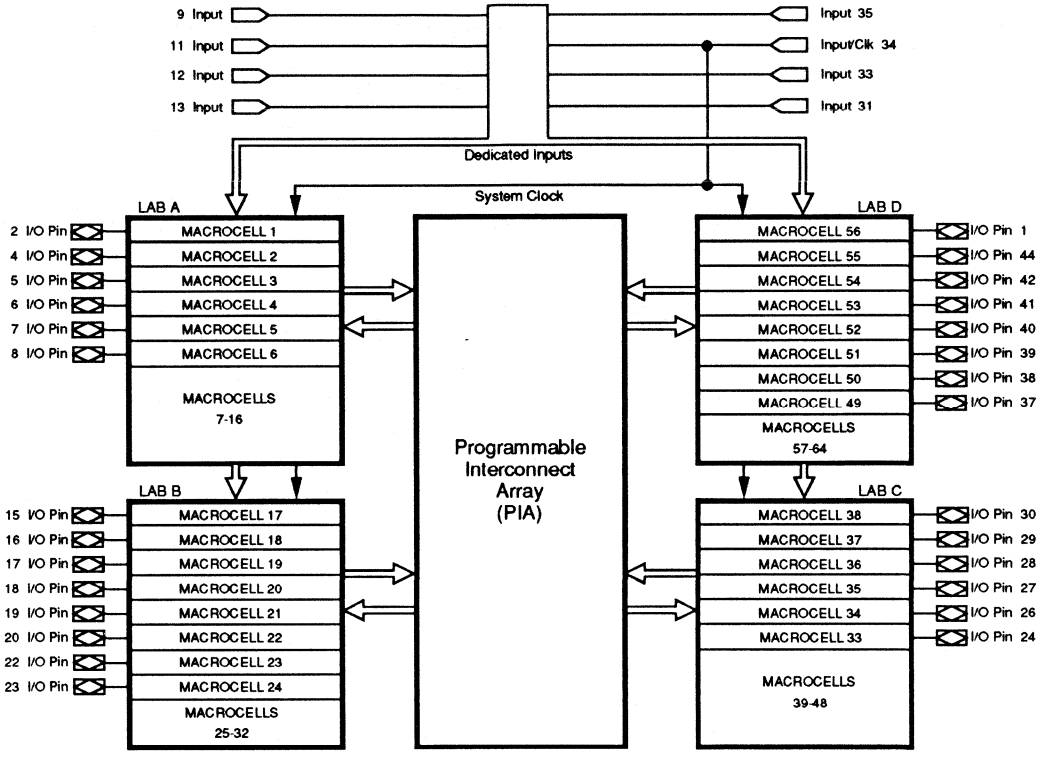


The EPM5064, shown in Figure 17, consists of 64 macrocells equally divided into 4 Logic Array Blocks (LABs) that each contain 16 macrocells. Each LAB also contains 32 expander product terms. The flexibility of the LABs allows easy integration of any common PLD.

The EPM5064 has 8 dedicated input pins, one of which may be used as a synchronous system clock that provides enhanced clock-to-output delays. The device has 28 I/O pins that can be configured for input, output, or bidirectional data flow. The I/O pins feature dual-feedback to allow any macrocell to be buried. Two of the LABs have 8 I/O pins (ensuring high speed for 8-bit bus functions) and the other two LABs have 6 I/O pins.

Figure 17. EPM5064 Block Diagram

The EPM5064 has 64 macrocells divided into 4 Logic Array Blocks.



3

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current			400	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			2000	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C
T_J	Junction temperature	Under bias		+150	° C

Recommended Operating Conditions *See Note (2)*

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time			100	ns
t_F	Input fall time			100	ns

DC Operating Conditions *See Notes (2), (3), (4)*

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 8 mA DC			0.45	V
I_I	Input leakage current	$V_I = V_{CC}$ or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	$V_O = V_{CC}$ or GND	–40		+40	µA
I_{CC1}	V_{CC} supply current (standby)	$V_I = V_{CC}$ or GND		90	125 (200)	mA
I_{CC3}	V_{CC} supply current	$V_I = V_{CC}$ or GND No load, f = 1.0 MHz *See Note (5)*		95	135 (225)	mA

Capacitance

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF

AC Operating Conditions

See Note (4)

External Timing Parameters			EPM5064-1		EPM5064-2		EPM5064		
Symbol	**Parameter**	**Conditions**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Unit**
t_{PD1}	Input to non-registered output	C1 = 35 pF		25		30		35	ns
t_{PD2}	I/O input to non-reg. output	C1 = 35 pF		40		45		55	ns
t_{SU}	Setup time		15		20		25		ns
t_{H}	Hold time		0		0		0		ns
t_{CO1}	Clock to output delay	C1 = 35 pF		14		16		20	ns
t_{ASU}	Asynchronous setup time		5		6		8		ns
t_{AH}	Asynchronous hold time		6		8		10		ns
t_{CH}	Clock high time		8		10		12.5		ns
t_{CL}	Clock low time		8		10		12.5		ns
t_{ACH}	Asynchronous clock high time		11		14		16		ns
t_{ACL}	Asynchronous clock low time		9		11		14		ns
t_{ACO1}	Asynch. clock to output delay	C1 = 35 pF		25		30		35	ns
t_{CNT}	Minimum clock period			20		25		30	ns
f_{CNT}	Internal maximum frequency		50		40		33.3		MHz
t_{ACNT}	Minimum asynch. clock period	*See Note (6)*		20		25		30	ns
f_{ACNT}	Max. internal asynch. frequency	*See Note (6)*	50		40		33.3		MHz
f_{MAX}	Max. frequency; pipelined data		62.5		50		40		MHz

For information on internal timing parameters, refer to *App. Brief 75 (EPM5000-Series MAX EPLD Timing)*.

Internal Timing Parameters			EPM5064-1		EPM5064-2		EPM5064		
Symbol	**Parameter**	**Conditions**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Unit**
t_{IN}	Input pad and buffer delay			5		7		9	ns
t_{IO}	I/O input pad and buffer delay			6		6		9	ns
t_{EXP}	Expander array delay			12		14		20	ns
t_{LAD}	Logic array delay			12		14		16	ns
t_{LAC}	Logic control array delay			10		12		13	ns
t_{OD}	Output buffer and pad delay	C1 = 35 pF		5		5		6	ns
t_{ZX}	Output buffer enable delay			10		11		13	ns
t_{XZ}	Output buffer disable delay	C1 = 5 pF		10		11		13	ns
t_{SU}	Register setup time		6		8		10		ns
t_{LATCH}	Flow-through latch delay			3		4		4	ns
t_{RD}	Register delay			1		2		2	ns
t_{COMB}	Combinatorial delay			3		4		4	ns
t_{H}	Register hold time		6		8		12		ns
t_{IC}	Clock delay			14		16		18	ns
t_{ICS}	System clock delay			2		2		3	ns
t_{FD}	Feedback delay			1		1		2	ns
t_{PRE}	Register preset time			5		6		7	ns
t_{CLR}	Register clear time			5		6		7	ns
t_{PIA}	Progr. Interconn. Array delay			14		16		20	ns

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature-range versions.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use.
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use.
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use.
(5) Measured with device programmed as a 16-bit counter in each LAB.
(6) This parameter is measured with a positive-edge-triggered clock at the register. For negative-edge clocking, the t_{ACH} and t_{ACL} parameters must be swapped.

Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EPM5064-1, EPM5064-2, EPM5064
Industrial	(–40° C to 85° C)	EPM5064
Military	(–55° C to 125° C)	EPM5064

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Features

- High-density 128-macrocell general-purpose MAX EPLD
- 256 shareable expander product terms that allow over 32 product terms in a single macrocell
- High-speed multiple-LAB architecture
  - t_{PD} as fast as 25 ns
  - Counter frequencies up to 50 MHz
  - Pipelined data rates up to 62.5 MHz
- Programmable I/O architecture allowing up to 60 inputs or 52 outputs
- Available in 68-pin windowed ceramic or plastic one-time-programmable J-lead packages and in 68-pin windowed ceramic PGA packages

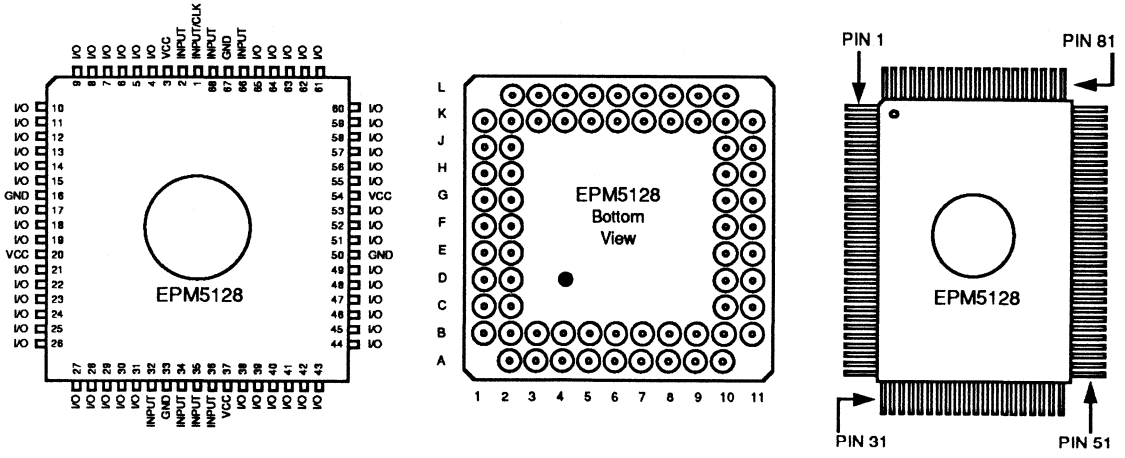
General Description

The Altera EPM5128 is a user-configurable, high-performance MAX EPLD that provides a high-density replacement for 7400-series SSI and MSI TTL and CMOS logic. (For example, a 74161 counter uses only 3% of the EPM5128.) The EPM5128 can replace over 60 TTL MSI and SSI components and integrate multiple 20- and 24-pin low-density PLDs. Figure 18 shows the J-lead and PGA package diagrams for the EPM5128.

3

Figure 18. EPM5128 Pin-Out Diagrams

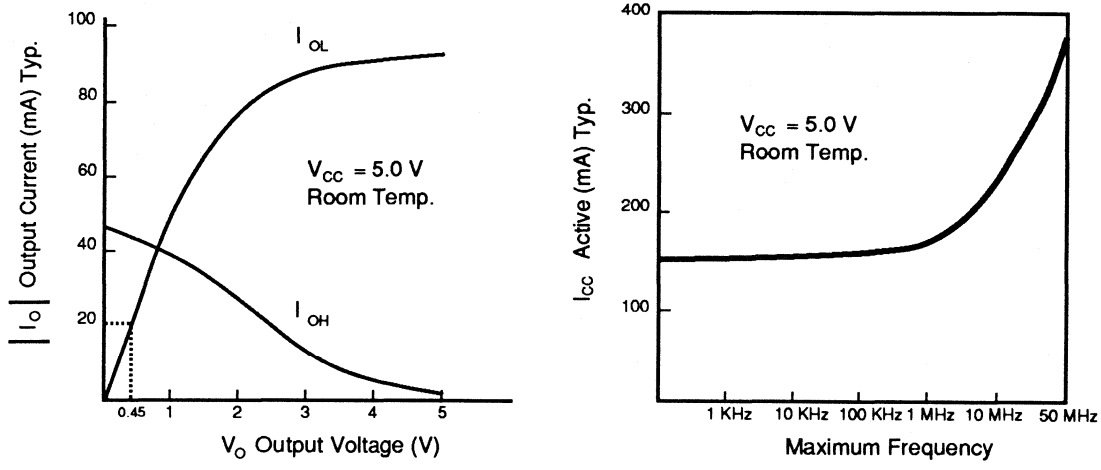
A quad flat pack (QFP) package is under development. Contact Altera Marketing for information. See Table 1 in this data sheet for PGA package pin-outs. Package outlines not drawn to scale.



J-Lead PGA QFP

Figure 19 shows output drive characteristics of EPM5128 I/O pins and typical supply current versus frequency for the EPM5128.

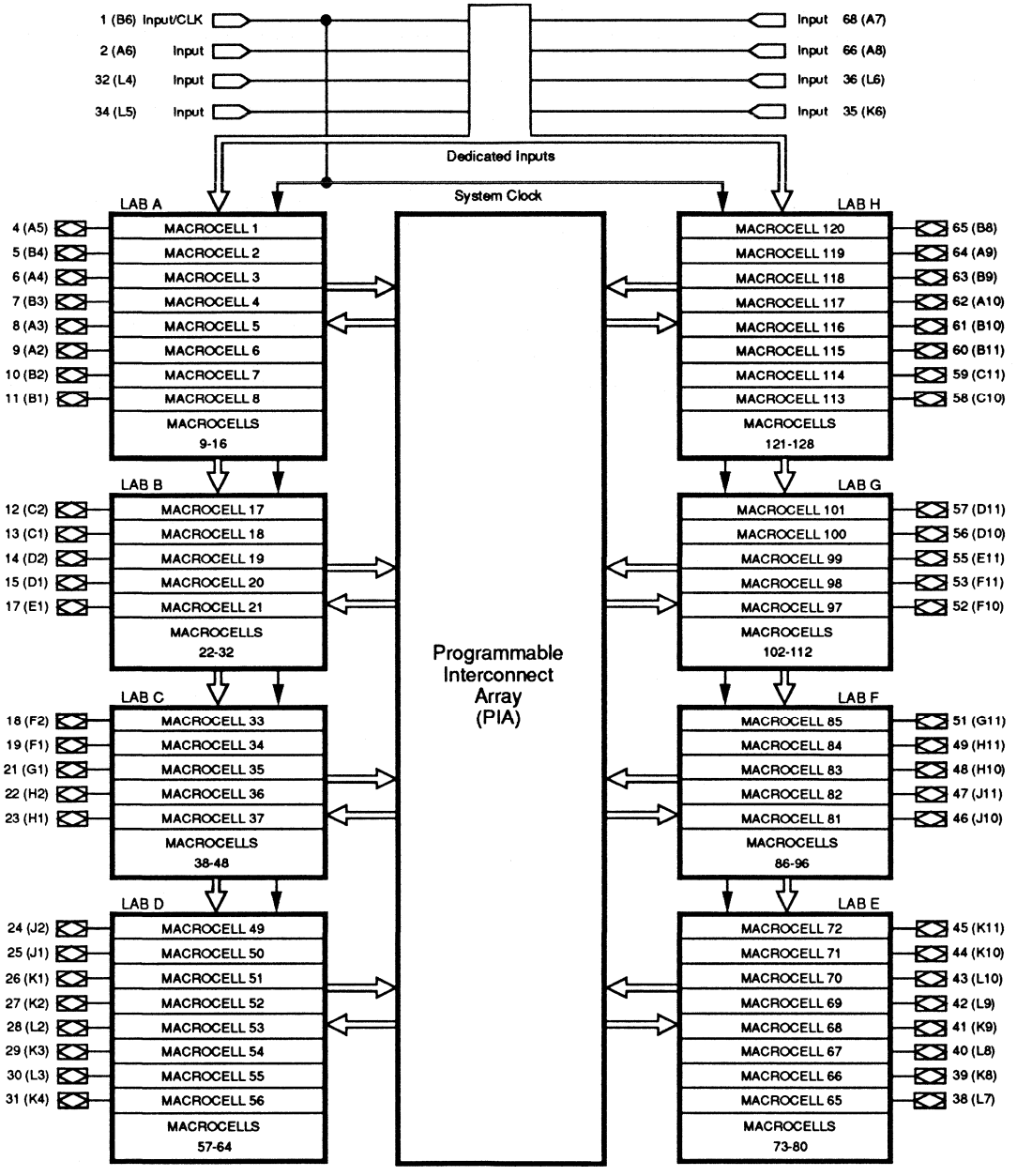
Figure 19. EPM5128 Output Drive Characteristics and I_{CC} vs. Frequency



The EPM5128 consists of 128 macrocells equally divided into 8 Logic Array Blocks (LABs) that each contain 16 macrocells (see Figure 20). Each LAB also contains 32 expander product terms. The EPM5128 has 8 dedicated input pins, one of which may be used as a synchronous system clock. The EPM5128 contains 52 I/O pins that can be configured for input, output, or bidirectional data flow. Four of the LABs have 8 I/O pins, and the other 4 have 5 I/O pins.

Figure 20. EPM5128 Block Diagram

Numbers in parentheses are for PGA packages.



Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current			500	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			2500	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C
T_J	Junction temperature	Under bias		+150	° C

Recommended Operating Conditions *See Note (2)*

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time			100	ns
t_F	Input fall time			100	ns

DC Operating Conditions *See Note (2), (3), (4)*

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 8 mA DC			0.45	V
I_I	Input leakage current	$V_I = V_{CC}$ or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	$V_O = V_{CC}$ or GND	–40		+40	µA
I_{CC1}	V_{CC} supply current (standby)	$V_I = V_{CC}$ or GND		150	225 (300)	mA
I_{CC3}	V_{CC} supply current	$V_I = V_{CC}$ or GND No load, f = 1.0 MHz *See Note (5)*		155	250 (350)	mA

Capacitance

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF

AC Operating Conditions *See Note (4)*

External Timing Parameters			EPM5128-1		EPM5128-2		EPM5128		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{PD1}	Input to non-registered output	C1 = 35 pF		25		30		35	ns
t_{PD2}	I/O input to non-reg. output	C1 = 35 pF		40		45		55	ns
t_{SU}	Setup time		15		20		25		ns
t_{H}	Hold time		0		0		0		ns
t_{CO1}	Clock to output delay	C1 = 35 pF		14		16		20	ns
t_{ASU}	Asynchronous setup time		5		6		8		ns
t_{AH}	Asynchronous hold time		6		8		10		ns
t_{CH}	Clock high time		8		10		12.5		ns
t_{CL}	Clock low time		8		10		12.5		ns
t_{ACH}	Asynchronous clock high time		11		14		16		ns
t_{ACL}	Asynchronous clock low time		9		11		14		ns
t_{ACO1}	Asynch. clock to output delay	C1 = 35 pF		25		30		35	ns
t_{CNT}	Minimum clock period			20		25		30	ns
f_{CNT}	Internal maximum frequency		50		40		33.3		MHz
t_{ACNT}	Minimum asynch. clock period	*See Note (6)*		20		25		30	ns
f_{ACNT}	Max. internal asynch. frequency	*See Note (6)*	50		40		33.3		MHz
f_{MAX}	Max. frequency; pipelined data		62.5		50		40		MHz

For information on internal timing parameters, refer to *App. Brief 75 (EPM5000-Series MAX EPLD Timing)*.

Internal Timing Parameters			EPM5128-1		EPM5128-2		EPM5128		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{IN}	Input pad and buffer delay			5		7		9	ns
t_{IO}	I/O input pad and buffer delay			6		6		9	ns
t_{EXP}	Expander array delay			12		14		20	ns
t_{LAD}	Logic array delay			12		14		16	ns
t_{LAC}	Logic control array delay			10		12		13	ns
t_{OD}	Output buffer and pad delay	C1 = 35 pF		5		5		6	ns
t_{ZX}	Output buffer enable delay			10		11		13	ns
t_{XZ}	Output buffer disable delay	C1 = 5 pF		10		11		13	ns
t_{SU}	Register setup time		6		8		10		ns
t_{LATCH}	Flow-through latch delay			3		4		4	ns
t_{RD}	Register delay			1		2		2	ns
t_{COMB}	Combinatorial delay			3		4		4	ns
t_{H}	Register hold time		6		8		10		ns
t_{IC}	Clock delay			14		16		18	ns
t_{ICS}	System clock delay			2		2		3	ns
t_{FD}	Feedback delay			1		1		2	ns
t_{PRE}	Register preset time			5		6		7	ns
t_{CLR}	Register clear time			5		6		7	ns
t_{PIA}	Progr. Interconn. Array delay			14		16		20	ns

3

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature-range versions.
(3) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(4) V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use.
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use.
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use.
(5) Measured with device programmed as a 16-bit counter in each LAB.
(6) This parameter is measured with a positive-edge-triggered clock at the register. For negative-edge clocking, the t_{ACH} and t_{ACL} parameters must be swapped.

Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EPM5128-1, EPM5128-2, EPM5128
Industrial	(–40° C to 85° C)	EPM5128
Military	(–55° C to 125° C)	EPM5128

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Table 1 shows the pin-outs for the EPM5128 PGA package.

Table 1. EPM5128 PGA Pin-Outs

Pin	Function	Pin	Function	Pin	Function	Pin	Function
A2	I/O	B10	I/O	G1	I/O	K7	VCC
A3	I/O	B11	I/O	G2	VCC	K8	I/O
A4	I/O	C1	I/O	G10	GND	K9	I/O
A5	I/O	C2	I/O	G11	I/O	K10	I/O
A6	Input	C10	I/O	H1	I/O	K11	I/O
A7	Input	C11	I/O	H2	I/O	L2	I/O
A8	Input	D1	I/O	H10	I/O	L3	I/O
A9	I/O	D2	I/O	H11	I/O	L4	Input
A10	I/O	D10	I/O	J1	I/O	L5	Input
B1	I/O	D11	I/O	J2	I/O	L6	Input
B2	I/O	E1	I/O	J10	I/O	L7	I/O
B3	I/O	E2	GND	J11	I/O	L8	I/O
B4	I/O	E10	VCC	K1	I/O	L9	I/O
B5	VCC	E11	I/O	K2	I/O	L10	I/O
B6	Input/CLK	F1	I/O	K3	I/O		
B7	GND	F2	I/O	K4	I/O		
B8	I/O	F10	I/O	K5	GND		
B9	I/O	F11	I/O	K6	Input		

Features

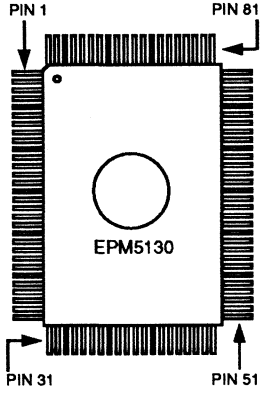
- High-density 128-macrocell general-purpose MAX EPLD
- 128 macrocells optimized for pin-intensive applications, easily integrating over 60 TTL MSI and SSI components
- High pin count for 16- or 32-bit data paths
- 256 shareable expander product terms
  - More than 32 product terms in a single macrocell
  - 128 additional latches provided by cross-coupling expanders
  - All inputs can be latched without using macrocells
- 20 high-speed dedicated inputs for fast latching of 16-bit functions
- Multiple LAB architecture ensuring high speeds
  - t_{PD} as fast as 25 ns
  - Counter frequencies up to 50 MHz
  - Pipelined data rates up to 62.5 MHz
- Synchronous clocking providing fast clock-to-output delays for bus-oriented functions
- Programmable I/O architecture that allows up to 84 inputs or 64 outputs in a windowed ceramic PGA or QFP or plastic QFP package

3

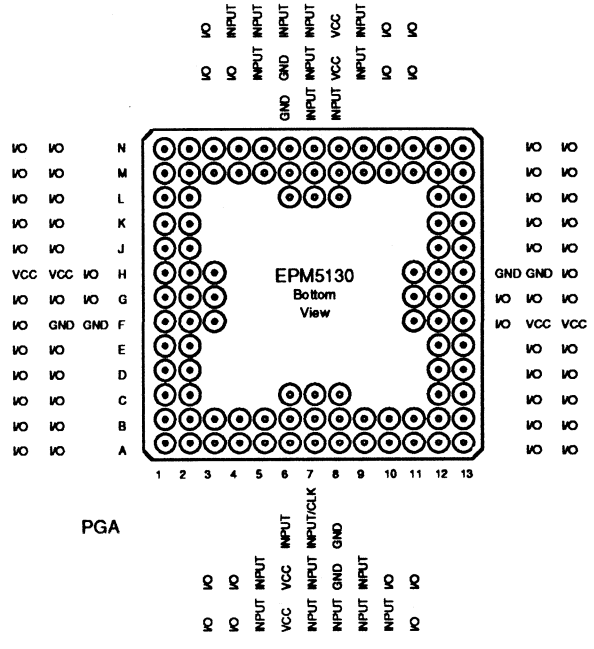
General Description

The Altera EPM5130 is a user-configurable, high-performance MAX EPLD that is optimized for pin-intensive designs. It provides a high-density replacement for 7400-series SSI and MSI TTL and CMOS logic. Package pin-out diagrams for the EPM5130 are shown in Figure 21.

Figure 21. EPM5130 Pin-Out Diagrams *See Table 2 in this data sheet for QFP pin-outs. Package outlines not drawn to scale.*



QFP



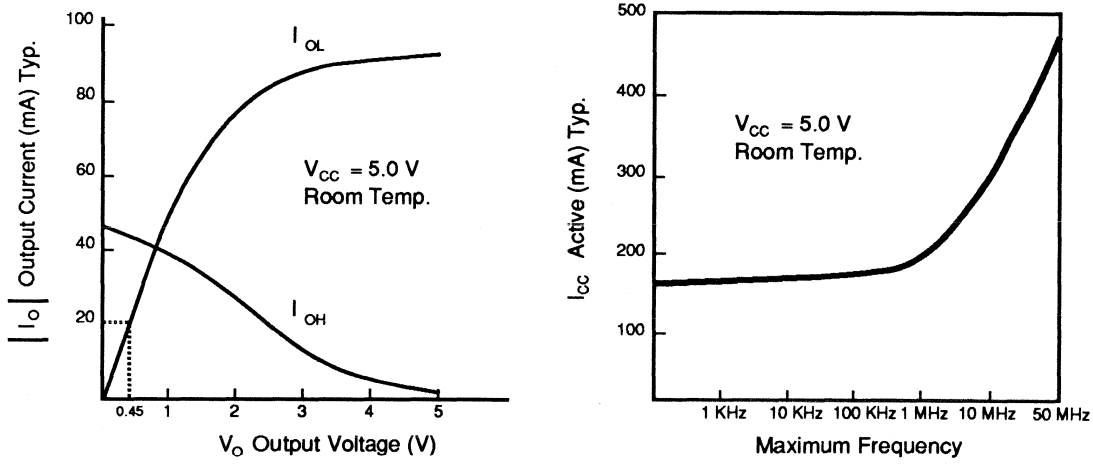
PGA

The EPM5130 EPLD is available in a windowed ceramic pin grid array (PGA) or 100-pin windowed ceramic or plastic one-time-programmable quad flat pack (QFP) package.

A single EPM5130 can quickly integrate multiple 20- and 24-pin low-density PLDs and high-pin-count subsystems, such as custom DMA controllers. In addition, it can handle a 32-bit data path application with enough I/O to allow the required control signals to be implemented.

Figure 22 shows output drive characteristics of EPM5130 I/O pins and typical supply current versus frequency for the EPM5130.

Figure 22. EPM5130 Output Drive Characteristics and I_{CC} vs. Frequency

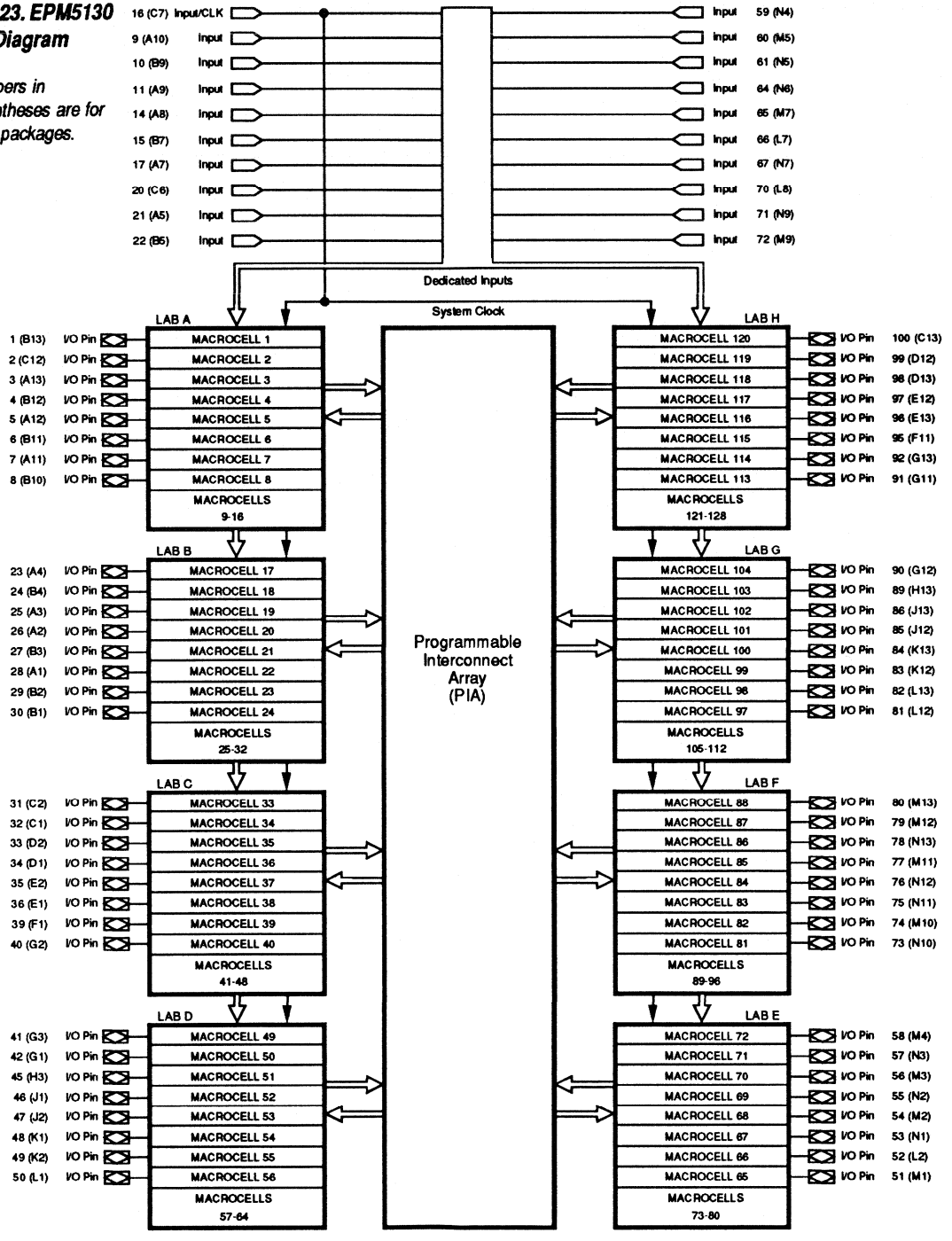


The EPM5130 consists of 128 macrocells equally divided into 8 Logic Array Blocks (LABs), each containing 16 macrocells and 32 expander product terms (see Figure 23). Expander product terms can be used and shared by all macrocells in the device to ensure efficient use of device resources. Because the LAB is very compact, the high speeds required by most I/O subsystems are maintained.

The EPM5130 has 20 dedicated input pins that allow high-speed input latching of 16-bit functions. One of these inputs can be configured as a synchronous system clock to provide enhanced clock-to-output delays for bus-oriented functions. The EPM5130 also has 64 I/O pins, 8 in each LAB, that can be configured for input, output, or bidirectional data flow. Dual feedback on the I/O pins provides the most efficient use of device pin resources.

Figure 23. EPM5130 Block Diagram

Numbers in parentheses are for PGA packages.



3

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current			500	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			2500	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C
T_J	Junction temperature	Under bias		+150	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75	5.25	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time			100	ns
t_F	Input fall time			100	ns

DC Operating Conditions V_{CC} = 5 V ± 5 %, T_A = 0° C to 70° C for commercial use, *See Note (2)*

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 8 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–40		+40	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND		175	250	mA
I_{CC3}	V_{CC} supply current	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (3)*		180	275	mA

Capacitance

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF

AC Operating Conditions V_{CC} = 5 V ± 5 %, T_A = 0° C to 70° C for commercial use

External Timing Parameters			EPM5130-1		EPM5130-2		EPM5130		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{PD1}	Input to non-registered output	C1 = 35 pF		25		30		35	ns
t_{PD2}	I/O input to non-reg. output	C1 = 35 pF		40		45		55	ns
t_{SU}	Setup time		15		20		25		ns
t_{H}	Hold time		0		0		0		ns
t_{CO1}	Clock to output delay	C1 = 35 pF		14		16		20	ns
t_{ASU}	Asynchronous setup time		5		6		8		ns
t_{AH}	Asynchronous hold time		6		8		10		ns
t_{CH}	Clock high time		8		10		12.5		ns
t_{CL}	Clock low time		8		10		12.5		ns
t_{ACH}	Asynchronous clock high time		11		14		16		ns
t_{ACL}	Asynchronous clock low time		9		11		14		ns
t_{ACO1}	Asynch. clock to output delay	C1 = 35 pF		25		30		35	ns
t_{CNT}	Minimum clock period			20		25		30	ns
f_{CNT}	Internal maximum frequency		50		40		33.3		MHz
t_{ACNT}	Minimum asynch. clock period	*See Note (4)*		20		25		30	ns
f_{ACNT}	Max. internal asynch. frequency	*See Note (4)*	50		40		33.3		MHz
f_{MAX}	Max. frequency; pipelined data		62.5		50		40		MHz

For information on internal timing parameters, refer to *App. Brief 75 (EPM5000-Series MAX EPLD Timing)*.

Internal Timing Parameters			EPM5130-1		EPM5130-2		EPM5130		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
t_{IN}	Input pad and buffer delay			5		7		9	ns
t_{IO}	I/O input pad and buffer delay			6		6		9	ns
t_{EXP}	Expander array delay			12		14		20	ns
t_{LAD}	Logic array delay			12		14		16	ns
t_{LAC}	Logic control array delay			10		12		13	ns
t_{OD}	Output buffer and pad delay	C1 = 35 pF		5		5		6	ns
t_{ZX}	Output buffer enable delay			10		11		13	ns
t_{XZ}	Output buffer disable delay	C1 = 5 pF		10		11		13	ns
t_{SU}	Register setup time		6		8		10		ns
t_{LATCH}	Flow-through latch delay			3		4		4	ns
t_{RD}	Register delay			1		2		2	ns
t_{COMB}	Combinatorial delay			3		4		4	ns
t_{H}	Register hold time		6		8		10		ns
t_{IC}	Clock delay			14		16		18	ns
t_{ICS}	System clock delay			2		2		3	ns
t_{FD}	Feedback delay			1		1		2	ns
t_{PRE}	Register preset time			5		6		7	ns
t_{CLR}	Register clear time			5		6		7	ns
t_{PIA}	Progr. Interconn. Array delay			14		16		20	ns

3

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(3) Measured with device programmed as a 16-bit counter in each LAB.
(4) This parameter is measured with a positive-edge-triggered clock at the register. For negative-edge clocking, the t_{ACH} and t_{ACL} parameters must be swapped.

Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EPM5130-1, EPM5130-2, EPM5130
Industrial	(–40° C to 85° C)	Consult factory
Military	(–55° C to 125° C)	Consult factory

Note: Only military-temperature-range EPLDs are listed above. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Pin-outs for the quad flat pack (QFP) are shown in Table 2.

Table 2. EPM5130 QFP Pin-Outs

Pin	Function	Pin	Function	Pin	Function	Pin	Function
1	I/O	26	I/O	51	I/O	76	I/O
2	I/O	27	I/O	52	I/O	77	I/O
3	I/O	28	I/O	53	I/O	78	I/O
4	I/O	29	I/O	54	I/O	79	I/O
5	I/O	30	I/O	55	I/O	80	I/O
6	I/O	31	I/O	56	I/O	81	I/O
7	I/O	32	I/O	57	I/O	82	I/O
8	I/O	33	I/O	58	I/O	83	I/O
9	Input	34	I/O	59	Input	84	I/O
10	Input	35	I/O	60	Input	85	I/O
11	Input	36	I/O	61	Input	86	I/O
12	GND	37	GND	62	GND	87	GND
13	GND	38	GND	63	GND	88	GND
14	Input	39	I/O	64	Input	89	I/O
15	Input	40	I/O	65	Input	90	I/O
16	Input/CLK	41	I/O	66	Input	91	I/O
17	Input	42	I/O	67	Input	92	I/O
18	V_{CC}	43	V_{CC}	68	V_{CC}	93	V_{CC}
19	V_{CC}	44	V_{CC}	69	V_{CC}	94	V_{CC}
20	Input	45	I/O	70	Input	95	I/O
21	Input	46	I/O	71	Input	96	I/O
22	Input	47	I/O	72	Input	97	I/O
23	I/O	48	I/O	73	I/O	98	I/O
24	I/O	49	I/O	74	I/O	99	I/O
25	I/O	50	I/O	75	I/O	100	I/O

Features

- 192 macrocells for easy replacement of over 100 TTL devices and for integration of complete logic boards into a single package
- 384 shareable expander product terms that offer flexibility for register and combinatorial logic expansion
- Multiple LAB architecture that ensures high speeds
  - t_{PD} as fast as 25 ns
  - Counter frequencies up to 50 MHz
  - Pipelined data rates up to 62.5 MHz
- Programmable I/O architecture allowing up to 72 inputs or 64 outputs, and I/O tri-state buffers that facilitate connections to system buses
- Available in 84-pin windowed ceramic and plastic one-time-programmable J-lead packages; 84-pin windowed ceramic PGA package; and 100-pin windowed ceramic and plastic one-time-programmable QFP packages

General Description

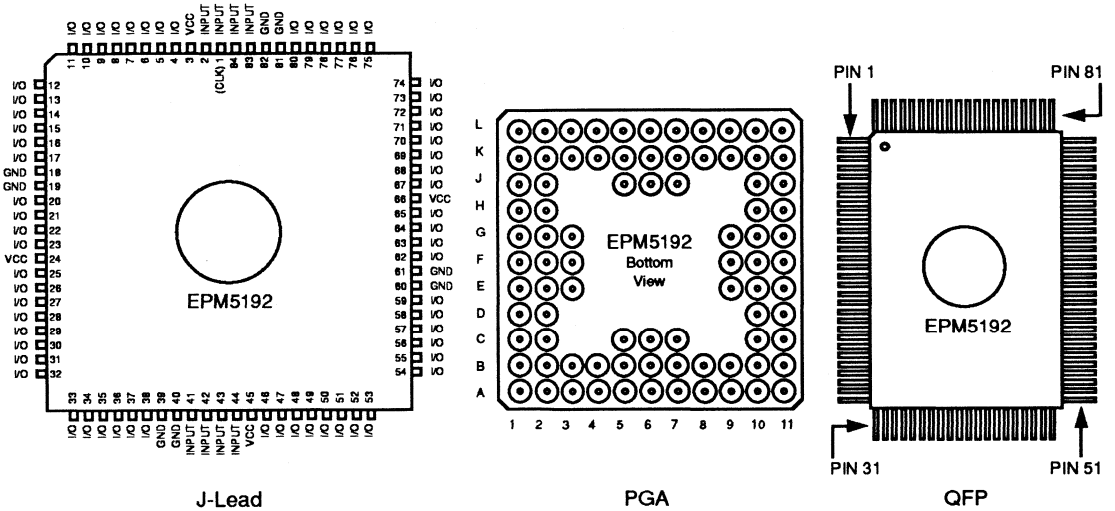
Altera's EPM5192 is a user-configurable, high-performance MAX EPLD that provides high-density replacement for 7400-series SSI and MSI TTL and CMOS logic. It is available in J-lead ceramic (JLCC) and plastic (PLCC), ceramic pin-grid array (PGA), and ceramic and plastic quad flat pack (QFP) packages (see Figure 24).

3

Because windowed packages are erasable, they may be used for quick and efficient system prototyping. On the other hand, plastic one-time-programmable (OTP) packages provide a low-cost solution for volume production.

Figure 24. EPM5192 Pin-Out Diagrams

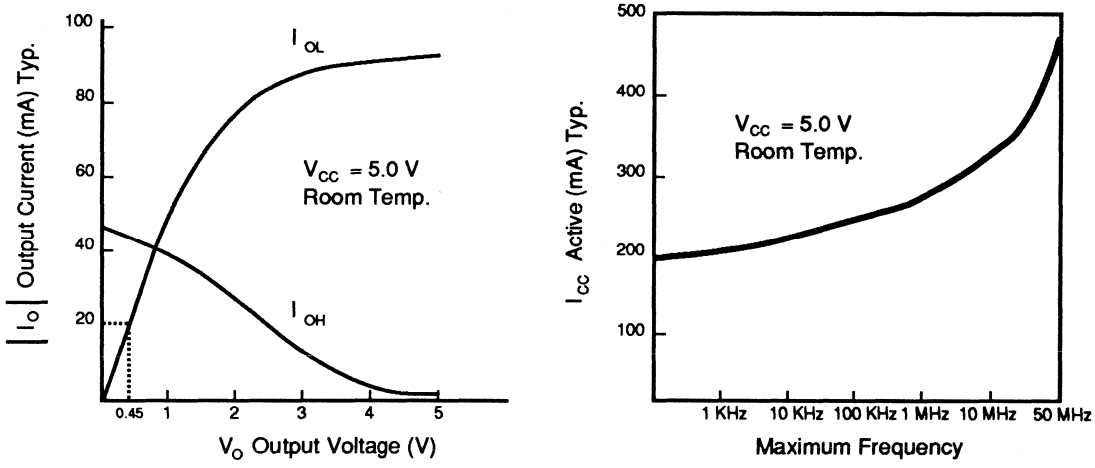
Tables 3 and 4 in this data sheet list the PGA and QFP package pin-outs. Package outlines not drawn to scale.



The EPM5192 can replace over 100 TTL SSI and MSI components and integrate the logic contained in over 20 22V10 devices. In addition, it accommodates other low-density PLDs of all sizes. These features allow the EPM5192 to easily integrate complete systems into a single device.

Figure 25 shows output drive characteristics of EPM5192 I/O pins and typical supply current versus frequency for the EPM5192.

Figure 25. EPM5192 Output Drive Characteristics and I_{CC} vs. Frequency

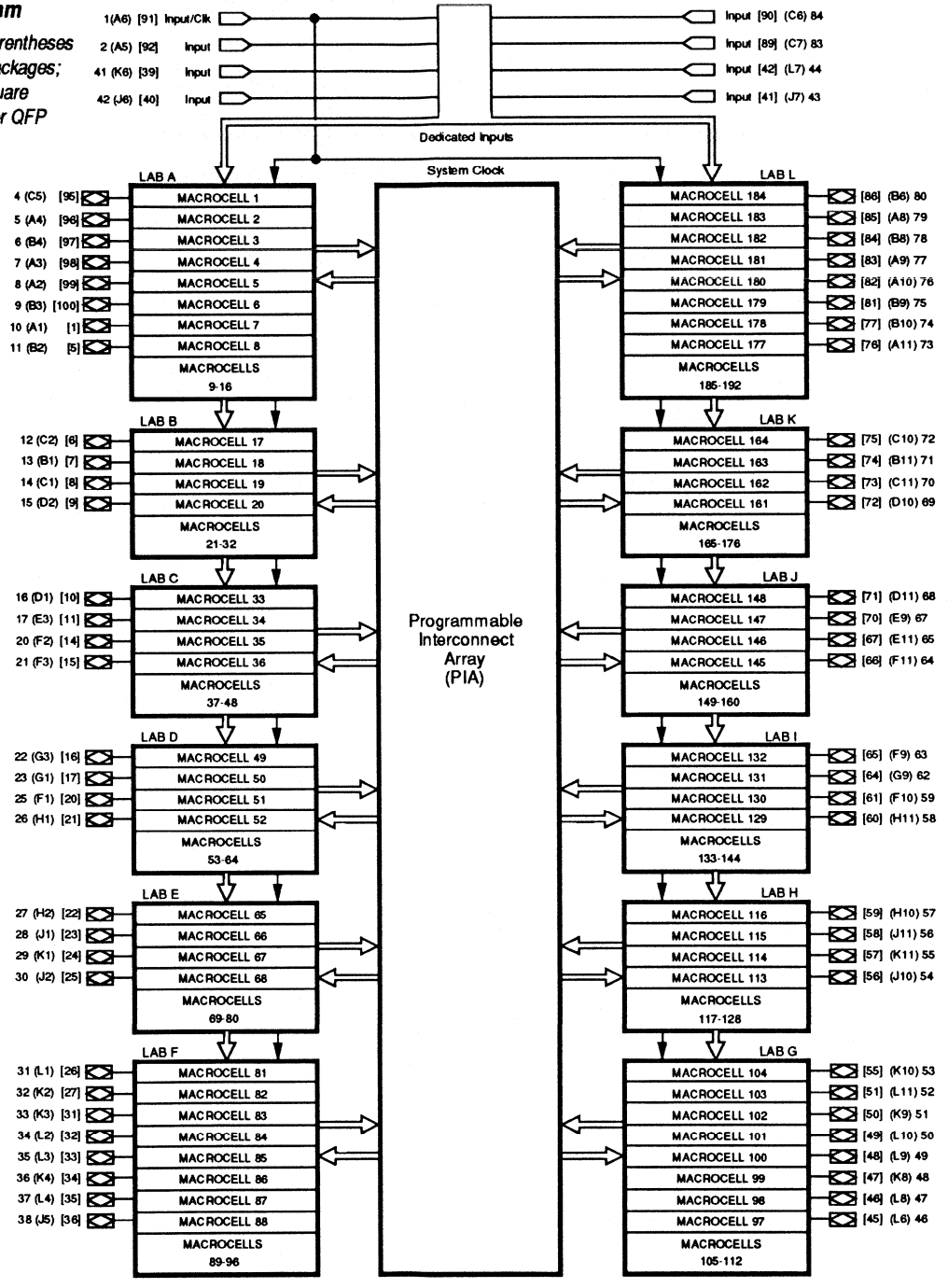


The EPM5192 consists of 192 macrocells equally divided into 12 Logic Array Blocks (LABs) that each contain 16 macrocells and 32 expander product terms (see Figure 26). Because each LAB is very compact, high performance is maintained and device resources are used efficiently.

The EPM5192 has 8 dedicated input pins, one of which can be used as a system clock. The EPM5192 can mix synchronous and asynchronous clocking in a single device, facilitating easy integration of multiple subsystems. It also has 64 I/O pins that can be configured for input, output, or bidirectional data flow, providing an interface to high-speed, bus-oriented applications.

Figure 26. EPM5192 Block Diagram

Numbers in parentheses are for PGA packages; numbers in square brackets are for QFP packages.



3

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	13.5	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current			500	mA
I_{OUT}	DC output current, per pin		–25	25	mA
P_D	Power dissipation			2500	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–65	+135	° C
T_J	Junction temperature	Under bias		+150	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage		4.75	5.25	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	+70	° C
T_A	Operating temperature	For industrial use	–40	+85	° C
T_C	Case temperature	For military use	–55	+125	° C
t_R	Input rise time			100	ns
t_F	Input fall time			100	ns

DC Operating Conditions *See Notes (2), (3)*

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –4 mA DC	2.4			V
V_{OL}	Low-level output voltage	I_{OL} = 8 mA DC			0.45	V
I_I	Input leakage current	V_I = V_{CC} or GND	–10		+10	µA
I_{OZ}	Tri-state output off-state current	V_O = V_{CC} or GND	–40		+40	µA
I_{CC1}	V_{CC} supply current (standby)	V_I = V_{CC} or GND		250	360	mA
I_{CC3}	V_{CC} supply current	V_I = V_{CC} or GND No load, f = 1.0 MHz *See Note (4)*		270	380	mA

Capacitance

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		20	pF

AC Operating Conditions *See Note (3)*

External Timing Parameters			EPM5192-1		EPM5192-2		EPM5192		
Symbol	**Parameter**	**Conditions**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Unit**
t_{PD1}	Input to non-registered output	C1 = 35 pF		25		30		35	ns
t_{PD2}	I/O input to non-reg. output	C1 = 35 pF		40		45		55	ns
t_{SU}	Setup time		15		20		25		ns
t_{H}	Hold time		0		0		0		ns
t_{CO1}	Clock to output delay	C1 = 35 pF		14		16		20	ns
t_{ASU}	Asynchronous setup time		5		6		8		ns
t_{AH}	Asynchronous hold time		6		8		10		ns
t_{CH}	Clock high time		8		10		12.5		ns
t_{CL}	Clock low time		8		10		12.5		ns
t_{ACH}	Asynchronous clock high time		11		14		16		ns
t_{ACL}	Asynchronous clock low time		9		11		14		ns
t_{ACO1}	Asynch. clock to output delay	C1 = 35 pF		25		30		35	ns
t_{CNT}	Minimum clock period			20		25		30	ns
f_{CNT}	Internal maximum frequency		50		40		33.3		MHz
t_{ACNT}	Min. asynch. clock period	*See Note (5)*		20		25		30	ns
f_{ACNT}	Min. internal asynch. frequ.	*See Note (5)*	50		40		33.3		MHz
f_{MAX}	Max. frequency; pipelined data		62.5		50		40		MHz

For information on internal timing parameters, refer to *App. Brief 75 (EPM5000-Series MAX EPLD Timing)*.

Internal Timing Parameters			EPM5192-1		EPM5192-2		EPM5192		
Symbol	**Parameter**	**Conditions**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Unit**
t_{IN}	Input pad and buffer delay			5		7		9	ns
t_{IO}	I/O input pad and buffer delay			6		6		9	ns
t_{EXP}	Expander array delay			12		14		20	ns
t_{LAD}	Logic array delay			12		14		16	ns
t_{LAC}	Logic control array delay			10		12		13	ns
t_{OD}	Output buffer and pad delay	C1 = 35 pF		5		5		6	ns
t_{ZX}	Output buffer enable delay			10		11		13	ns
t_{XZ}	Output buffer disable delay	C1 = 5 pF		10		11		13	ns
t_{SU}	Register setup time		6		8		10		ns
t_{LATCH}	Flow-through latch delay			3		4		4	ns
t_{RD}	Register delay			1		2		2	ns
t_{COMB}	Combinatorial delay			3		4		4	ns
t_{H}	Register hold time		6		8		10		ns
t_{IC}	Clock delay			14		16		18	ns
t_{ICS}	System clock delay			2		2		3	ns
t_{FD}	Feedback delay			1		1		2	ns
t_{PRE}	Register preset time			5		6		7	ns
t_{CLR}	Register clear time			5		6		7	ns
t_{PIA}	Progr. Interconn. Array delay			14		16		20	ns

3

Notes to tables:

(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Typical values are for T_A = 25° C and V_{CC} = 5 V.
(3) V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use.
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use.
V_{CC} = 5 V ± 10%, T_A = –55° C to 125° C for military use.
(4) Measured with device programmed as a 16-bit counter in each LAB.
(5) This parameter is measured with a positive-edge-triggered clock at the register. For negative-edge clocking, the t_{ACH} and t_{ACL} parameters must be swapped.

Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EPM5192-1, EPM5192-2, EPM5192
Industrial	(–40° C to 85° C)	Consult factory
Military	(–55° C to 125° C)	Consult factory

Note: Only military-temperature-range EPLDs are listed here. MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing by calling 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Table 3 shows the pin-outs for the EPM5192 PGA package.

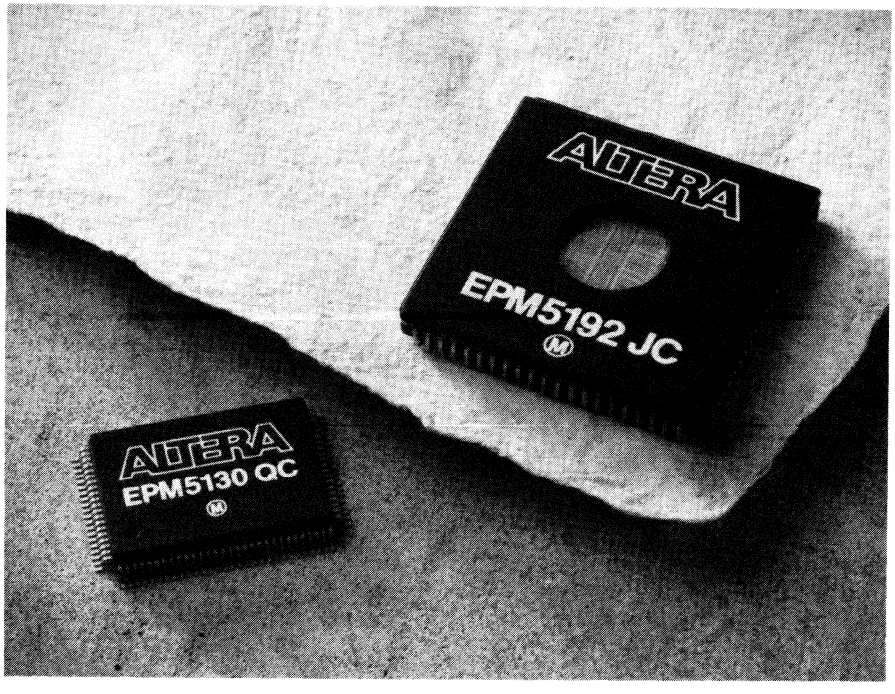
Table 3. EPM5192 PGA Pin-Outs

Pin	Function	Pin	Function	Pin	Function	Pin	Function
A1	I/O	C2	I/O	G2	VCC	K8	I/O
A2	I/O	C5	I/O	G3	I/O	K9	I/O
A3	I/O	C6	Input	G9	I/O	K10	I/O
A4	I/O	C7	Input	G10	GND	K11	I/O
A5	Input	C10	I/O	G11	GND	L1	I/O
A6	Input/CLK	C11	I/O	H1	I/O	L2	I/O
A7	GND	D1	I/O	H2	I/O	L3	I/O
A8	I/O	D2	I/O	H10	I/O	L4	I/O
A9	I/O	D10	I/O	H11	I/O	L5	GND
A10	I/O	D11	I/O	J1	I/O	L6	I/O
A11	I/O	E1	GND	J2	I/O	L7	Input
B1	I/O	E2	GND	J5	I/O	L8	I/O
B2	I/O	E3	I/O	J6	Input	L9	I/O
B3	I/O	E9	I/O	J7	Input	L10	I/O
B4	I/O	E10	VCC	J10	I/O	L11	I/O
B5	VCC	E11	I/O	J11	I/O		
B6	I/O	F1	I/O	K1	I/O		
B7	GND	F2	I/O	K2	I/O		
B8	I/O	F3	I/O	K3	I/O		
B9	I/O	F9	I/O	K4	I/O		
B10	I/O	F10	I/O	K5	GND		
B11	I/O	F11	I/O	K6	Input		
C1	I/O	G1	I/O	K7	VCC		

Table 4 shows the pin-outs for the EPM5192 QFP package (n.c. indicates "not connected").

Table 4. EPM5192 QFP Pin-Outs

Pin	Function	Pin	Function	Pin	Function	Pin	Function
1	I/O	26	I/O	51	I/O	76	I/O
2	n.c.	27	I/O	52	n.c.	77	I/O
3	n.c.	28	n.c.	53	n.c.	78	n.c.
4	n.c.	29	n.c.	54	n.c.	79	n.c.
5	I/O	30	n.c.	55	I/O	80	n.c.
6	I/O	31	I/O	56	I/O	81	I/O
7	I/O	32	I/O	57	I/O	82	I/O
8	I/O	33	I/O	58	I/O	83	I/O
9	I/O	34	I/O	59	I/O	84	I/O
10	I/O	35	I/O	60	I/O	85	I/O
11	I/O	36	I/O	61	I/O	86	I/O
12	GND	37	GND	62	GND	87	GND
13	GND	38	GND	63	GND	88	GND
14	I/O	39	Input	64	I/O	89	Input
15	I/O	40	Input	65	I/O	90	Input
16	I/O	41	Input	66	I/O	91	Input/CLK
17	I/O	42	Input	67	I/O	92	Input
18	n.c.	43	VCC	68	n.c.	93	VCC
19	VCC	44	n.c.	69	VCC	94	n.c.
20	I/O	45	I/O	70	I/O	95	I/O
21	I/O	46	I/O	71	I/O	96	I/O
22	I/O	47	I/O	72	I/O	97	I/O
23	I/O	48	I/O	73	I/O	98	I/O
24	I/O	49	I/O	74	I/O	99	I/O
25	I/O	50	I/O	75	I/O	100	I/O


ALTERA
EPM5192 JC
ALTERA
EPM5130 QC

PLS-MAX

MAX+PLUS Programmable Logic Software

October 1990, ver. 2 — **Data Sheet**

Features

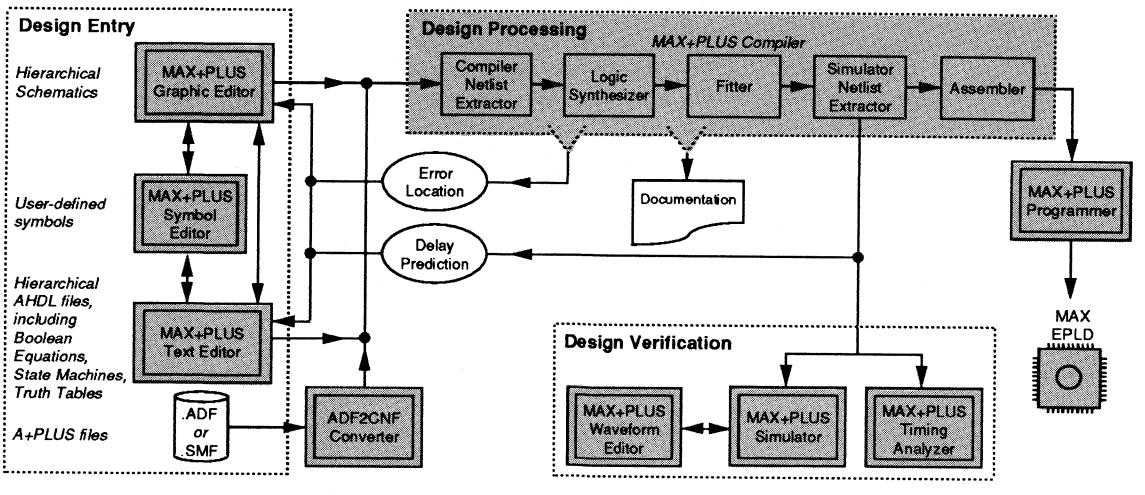
- Software support for MAX (Multiple Array MatriX) EPLDs
- Hierarchical design entry methods for both graphic and text designs
  - Multi-level schematics and hardware language descriptions
  - Over 340 7400-series TTL and bus macrofunctions optimized for MAX architecture
  - Altera Hardware Description Language (AHDL) for state machines, Boolean equations, truth tables, arithmetic and relational operations
  - Delay prediction and timing analysis for graphic and text designs
- Logic synthesis and minimization for quick and efficient processing
- Compiler that compiles a 100% utilized EPM5128 in only 10 minutes
- Automatic error location for schematics and AHDL text files
- Interactive Simulator with probe assignments for internal nodes
- Waveform Editor for entering and editing waveforms and viewing simulation results
- Used with IBM PS/2, PC-AT, or compatible machines
- EDIF industry-standard workstation and third-party interfaces available separately

3

General Description

The Altera PLS-MAX (MAX+PLUS Programmable Logic Software) package, shown in Figure 1, is a unified CAE system for designing logic with Altera's MAX family of EPLDs. PLS-MAX includes design entry, design processing, timing simulation, and device programming support. It runs on IBM PS/2,

Figure 1. MAX+PLUS Design Framework



PC-AT, and compatible computers and provides sophisticated tools to quickly and efficiently create and verify complex logic designs.

PLS-MAX is a software-only package. PLDS-MAX (MAX+PLUS Programmable Logic Development System) includes PLS-MAX software, all hardware required to program MAX EPLDs, and an extended software warranty (see the *PLDS-MAX Data Sheet* in this data book).

Designs may be entered with a variety of design entry methods. MAX+PLUS supports hierarchical entry of both Graphic Design Files (GDFs) with the MAX+PLUS Graphic Editor, and Text Design Files (TDFs) in the Altera Hardware Description Language (AHDL) with the MAX+PLUS Text Editor. The Graphic Editor offers advanced features such as multiple hierarchy levels, symbol editing, and an extensive library of 7400-series devices and basic SSI gates. AHDL designs may be mixed into any level of the hierarchy or used stand-alone. AHDL is tailored especially for EPLD designs and includes support for complex Boolean and arithmetic functions, relational comparisons, state machines with automatic state variable assignment, truth tables, and function calls.

MAX+PLUS also includes the sophisticated MAX+PLUS Compiler, which synthesizes and optimizes designs in minutes. The Compiler uses advanced logic synthesis and minimization techniques together with heuristic fitting rules to efficiently place designs into MAX EPLDs. The Compiler creates a programming file that the MAX+PLUS Programmer uses to program MAX EPLDs with standard Altera programming hardware.

Simulations are performed with a powerful, event-driven timing simulator. The MAX+PLUS Simulator interactively displays timing results in the MAX+PLUS Waveform Editor. Hardcopy table and waveform output is available. With the Waveform Editor, input vector waveforms may be entered, modified, grouped, and ungrouped. The Waveform Editor can also compare simulation runs and highlight the differences between them.

MAX+PLUS also provides features such as automatic error location and delay prediction. If a design contains an error in a schematic or an AHDL text file, MAX+PLUS flags the error *and* takes the user to the location of the error in the original schematic or text file. Propagation delays of critical paths can also be determined from within both the Graphic and Text Editors with the delay prediction feature. After the source and destination nodes are tagged, the shortest and longest timing delays are calculated.

MAX+PLUS software provides a seamless design framework that uses a consistent graphical user interface throughout. This framework simplifies all stages of the design cycle: design entry, processing, verification, and programming. In addition, MAX+PLUS offers extensive on-line help.

Design Entry

MAX+PLUS offers both graphic and text design entry methods. GDFs are entered with the MAX+PLUS Graphic Editor; Boolean equations, state machines, and truth tables are entered in the Altera Hardware Description Language (AHDL) with the MAX+PLUS Text Editor. The ability to freely mix graphic and text files at all levels of the design hierarchy, and to use a top-down or bottom-up design method, makes design entry simple and versatile. As the designer traverses the hierarchy, the Text Editor is automatically invoked for text files, and the Graphic Editor is invoked for schematics.

Once the user saves a text or graphic file, the Graphic Editor automatically generates a symbol for this file. This symbol, and the design it represents, can then be used as a subdesign in a higher-level schematic or in another design.

MAX+PLUS also accepts third-party netlists from OrCAD, Viewlogic Systems, and Data I/O (ABEL, FutureNet-DASH), as well as existing EPLD designs implemented with Altera's and Texas Instruments' A+PLUS, and Intel's iPLDS or iPLDS II systems.

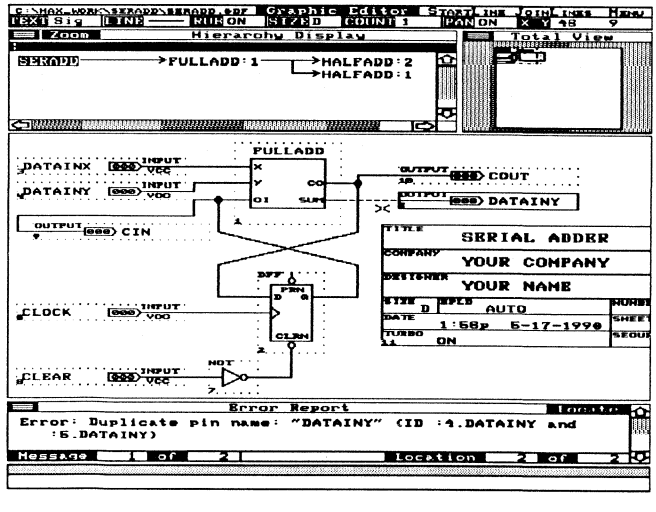
MAX+PLUS Graphic Editor

The MAX+PLUS Graphic Editor (Figure 2) provides a mouse-driven, multi-windowed environment in which commands are entered with pop-up menus or simple keystrokes. The Hierarchy Display window lists all schematics used in a design. The designer navigates the hierarchy by placing the cursor on the name of the design to be opened and clicking a mouse button. The Total View window shows the entire design. By clicking inside this window, the main work window is moved to the corresponding area of the schematic. The Error Report window lists all warnings and errors in the compiled design; selecting an error with the mouse highlights the problem node and symbol. A design is entered in the main work window, which can be enlarged by closing the auxiliary windows.

3

Figure 2. MAX+PLUS Graphic Editor

The Graphic Editor provides a multi-windowed, menu-driven environment. Auxiliary windows can be closed to increase work space.



When entering a design, the user can choose from a library of over 300 7400-series and special-purpose macrofunctions that are all optimized for MAX architecture. In addition, the designer can create custom functions that can be used in any MAX+PLUS design.

Tag-and-drag editing is used to move individual symbols or entire areas. Lines stay connected with orthogonal rubberbanding. Designs are printed on an Epson FX-compatible printer; HP7475A, 7485B, and 7495A plotters; or a Houston Instruments 695-compatible plotter.

MAX+PLUS Symbol Editor

The MAX+PLUS Symbol Editor enables the designer to create or modify a custom symbol representing a GDF or TDF. It is also possible to modify input and output pin placement on an automatically generated symbol.

A symbol represents a lower-level design, described by a GDF or TDF. The lower-level design can be displayed with a single command that invokes the Graphic Editor for schematics or the Text Editor for AHDL designs.

MAX+PLUS AHDL

The Altera Hardware Description Language (AHDL) is a high-level, modular language used to create logic designs for MAX EPLDs. It is completely integrated into MAX+PLUS, so AHDL files may be created, edited, compiled, simulated, and programmed from within MAX+PLUS.

AHDL supports state machines, truth tables, and Boolean equations, as well as arithmetic and relational operations. It is hierarchical, so that frequently used functions such as TTL and bus macrofunctions can be incorporated into a design. it also supports complex arithmetic and relational operations—such as addition, subtraction, equality, and magnitude comparisons—with the automatically generated logic functions. Standard Boolean functions, including AND, OR, NAND, NOR, XOR, and XNOR are also included. Groups are fully supported so operations can be performed on groups as well as on single variables. AHDL also allows the designer to specify the location of resources (e.g., latches, flip-flops, and pins) within MAX EPLDs. Together, these features enable complex designs to be implemented in a concise, high-level description (see Figure 3).

Figure 3. AHDL (Part 1 of 2)

AHDL allows complex arithmetic and relational operators to be described in a few lines.

```
TITLE "Timed Add and Compare function.";

DESIGN IS "add_cmp" DEVICE IS "EPM5128-2";

FUNCTION 74161 (LDN,A,B,C,D,ENT,ENP,CLRN,CLK) RETURNS (QA,QB,QC,QD,RCO);

SUBDESIGN add_cmp (
                a[7..0],          % inputs for adder/comparator          %
                b[7..0],
                cmpen,
                clock,reset       :INPUT;

                result[7..0],
                elapse[3..0],
                equal,
                less_than,
                grtr_than,
                done              :OUTPUT;
```

Figure 3. AHDL (Part 2 of 2)

```
)
VARIABLE
        timer           : 74161;    % timer is 74161 counter                %
        register[7..0]  : DFF;      % register is an octal FF                %
        flag            : NODE;

BEGIN
        result[] = register[];      % set up accumulate register             %
        register[].clrn = reset;
        register[].clk = clock;
        register[] = a[] + b[];     % this is the actual addition            %
        flag = (register[] != 0); % set flag high if register is not empty %
        done = flag;
        timer.enp = cmpen & flag;  % connect inputs for timer (74161)       %
        timer.clk = clock;
        timer.clrn = reset;

% elapse is the number of clock cycles it takes to do add                    %

        elapse[3..0] = (timer.QA,timer.QB,timer.QC,timer.QD);
        equal = ( a[] == b[]);      % the comparator section                 %
        less_than = (a[] < b[]);
        grtr_than = (a[] > b[]);

END;
```

3

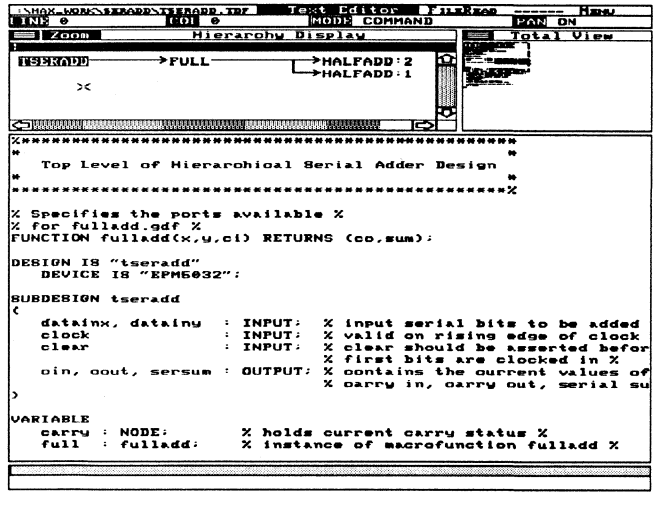
MAX+PLUS Text Editor

The MAX+PLUS Text Editor, shown in Figure 4, enables the user to view and edit text files within the MAX+PLUS environment. Any ASCII text file, including AHDL Text Design Files (TDFs), Vector Files, Table Files, and Report Files, may be viewed and edited in the Text Editor.

The Text Editor parallels the Graphic Editor's structure with Hierarchy Display and Total View windows for moving through hierarchy levels and around the design. It also provides automatic error location. If an error is found in a TDF during compilation, the Text Editor is automatically invoked and the line of AHDL code where the error occurred is highlighted.

Figure 4. MAX+PLUS Text Editor

The Text Editor and AHDL offer such features as hierarchical design entry and automatic error location.



Macrofunction Library

The MAX+PLUS TTL MacroFunction Library contains the most commonly used 7400-series devices such as counters, decoders, encoders, shift registers, flip-flops, latches, and multipliers, as well as special bus macrofunctions, all of which increase design productivity. The flexible architecture of MAX EPLDs (which includes asynchronous Preset and Clear) ensures that true TTL device emulation is achieved. Altera has also created special-purpose bus macrofunctions for designs that use buses. All macrofunctions have been optimized to provide the best speed and part utilization. Table 1 lists some of the macrofunctions currently available.

Table 1. Sampling of TTL MacroFunctions

TTL Macrofunctions:

Adders:	8FADD, 7480, 7482, 7483, 74183, 74283, 74385
ALU:	74181, 74182, 74381, 74382
AND-OR Gates:	7452
Comparators:	8MCOMP, 7485, 74518, 74684, 74686, 74688
Code Converters:	74184, 74185
Counters:	4COUNT, 8COUNT, 16CUDSLR, GRAY4, UNICNT, 7493, 74160, 74161, 74162, 74163, 74190, 74191, 74192, 74193, 74393...
Decoders:	7442, 7443, 7444, 7445, 7446, 7447, 7448, 7449, 74138, 74139, 74154, 74155, 74156...
Encoders:	74147, 74148
Frequency Divider:	FREQDIV, 7456, 7457
Latches:	INPLTCH, NANDLTCH, NORLTCH, 7475, 7477, 74116, 74259, 74279, 74373...
Multipliers:	MULT2, MULT4, MULT24, 74261...
Multiplexers:	21MUX, 74151, 74153, 74157, 74158, 74298...
Parity Generators:	74180, 74280
Registers:	7470, 7471, 7472, 7473, 7474, 7476, 7478, 74173, 74174, 74175, 74178, 74273, 74374...
Shift Registers:	BARRELST, 7491, 7494 ,7496, 7499, 74164, 74165, 74166, 74179, 74194, 74198...
SSI Gates:	CBUF, INHB, 7400, 7402, 7404, 7408, 7410, 7411, 7420, 7421, 7427, 7430, 7432, 7486...
Storage Elements:	7498, 74278
True/Comp Elements:	7487, 74265

Bus Macrofunctions:

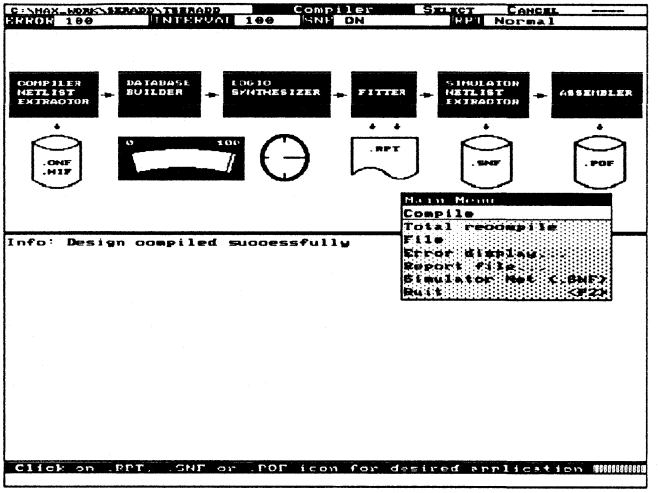
Adder:	8FADDB
Buffers:	74240B, 74241B, 74244B
Comparators:	8MCOMPB, 74518B
Counter:	16CUDSRB
Latches:	74373B, 74841B, 74842B
Multiplexers:	74151B
Multipliers:	MULT4B
Parity Generators:	74180B, 74280
Registers:	74174B, 74273B, 74374B, 74821B, 74822B, 74823B, 74824B, 74825B, 74826B
Shift Registers:	BARRLSTB, 74164B, 74165B

Design Processing

The MAX+PLUS Compiler processes designs in minutes (see Figure 5). It offers several options that speed the processing and analysis of a design. For example, the user can specify the degree of detail of the Report File, as well as the maximum number of errors to be detected before processing halts. The user may also select whether to extract a netlist file for simulation.

Figure 5. MAX+PLUS Compiler

The Compiler uses minimization, logic synthesis, and heuristic fitting algorithms to place designs into MAX EPLDs.



The Compiler compiles a design in increments. If a design has been compiled previously, only the new portion is compiled to reduce compilation time. This "Make" facility is an automatic feature of the Compiler.

The first module of the Compiler, the Compiler Netlist Extractor, extracts a netlist from each file. At this time, design rules are checked for any errors. If errors are found, the Graphic Editor or Text Editor is invoked after the compilation, depending on whether the error occurred in a GDF or TDF. The Error Report window displays the error and its location. The Compiler Netlist Extractor also generates a Hierarchy Interconnect File (HIF) that describes the hierarchy of the total design. This information is used by the Database Builder, which flattens the hierarchical design, examines design logic, and checks for schematic boundary connectivity and syntax errors.

The Logic Synthesizer module translates and optimizes the user-defined logic for the MAX architecture. The design is first minimized with SALSA (Speedy Altera Logic Simplification Algorithm). Any unused logic is automatically removed. This module uses expert system synthesis rules to factor and map logic within the multi-level MAX architecture. It then chooses the approach that ensures the most efficient use of silicon resources.

The next module, the Fitter, uses heuristic rules to place the synthesized design into the chosen MAX EPLD. For MAX devices with a Programmable Interconnect Array (PIA), the Fitter also routes the signals across this interconnect structure, so the designer doesn't have to worry about placement and routing issues. The Fitter issues a Report File (**.RPT**) that shows how the design is implemented and which resources in the EPLD are unused. The designer can then determine how much additional logic may be placed in the EPLD.

Next, the Simulator Netlist Extractor optionally generates a Simulator Netlist File (**.SNF**) that is used to perform timing simulation. Finally, the Assembler creates a Programmer Object File (**.POF**) from the compiled design that is used by the MAX+PLUS Programmer to program the target EPLD.

The advanced synthesis and minimization techniques employed by the Compiler allow designs to be placed within the MAX architecture in a matter of minutes. For example, a 16-bit counter/shift register compiles in less than 1 minute on a 16-MHz 386-based PC. The Compiler is equally efficient when compiling complex designs. For example, 5 serially linked multiplier/adder circuits that use 100% of the macrocells and 95% of all expanders in an EPM5128 take only 10 minutes to compile on a 20-MHz 386-based PC.

Delay Prediction and Probes

MAX+PLUS includes powerful analysis tools to verify and analyze the completed design. Delay prediction is performed interactively in the Graphic Editor, Text Editor, or Simulator.

The delay prediction feature provides instant feedback on the timing of the processed design. After selecting the start point and end point of a path, the designer may determine the shortest and longest propagation delays of speed-critical paths.

In addition, the designer can use probes to mark internal nodes in a design. A probe is entered in a Graphic Editor schematic by selecting any node, entering a command, and then assigning a unique name to define the probe. This name is then used in the Graphic Editor, Simulator, and Waveform Editor to identify the node.

MAX+PLUS Simulator

The designer defines input stimuli with a straightforward vector input language, or draws waveforms directly in the Waveform Editor.

The Simulator uses the Simulator Netlist File (SNF) extracted from the compiled design to perform timing simulation with 0.1-ns resolution. Simulator commands are provided to halt the simulation based on user-defined conditions; to force and group nodes; and to detect glitches, setup and hold violations, and unwanted oscillation. For example, if flip-flop

setup or hold times have been violated, the Simulator warns the user. Or, if a pulse is shorter than the minimum pulse width specified, or if a node oscillates for longer than the specified time, the Simulator issues a warning.

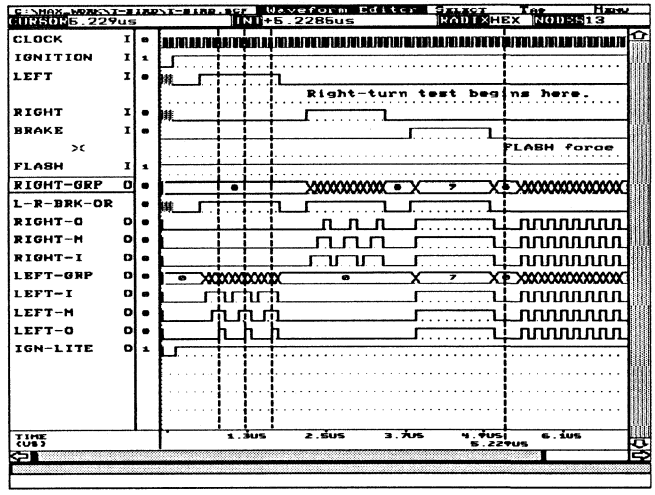
A Command File is used for batch operation, or commands may be entered interactively.

MAX+PLUS Waveform Editor

The MAX+PLUS Waveform Editor, shown in Figure 6, provides a mouse-driven environment for editing and viewing waveforms. It functions as a logic analyzer, enabling the user to observe simulation results. Simulated waveforms can be viewed and manipulated at multiple zoom levels. Nodes can be added, deleted, and combined into buses. Buses can contain up to 32 signals that are represented in binary, octal, decimal, or hexadecimal format. Logical operators can also be used on pairs of waveforms, so that waveforms can be inverted, ORed, ANDed, or XORed together.

Figure 6. MAX+PLUS Waveform Editor

With the Waveform Editor, input stimuli can be entered and modified, and Simulator outputs can be viewed and compared.



The Waveform Editor includes sophisticated editing features to define and modify input vectors. The designer uses the mouse and familiar commands to create and copy waveforms, to repeat waveform patterns, and to move and copy blocks of waveforms. For example, all or part of a waveform can be compressed to simulate an increase in clock frequency.

The Waveform Editor can also compare and highlight the differences between two different simulations. A user can simulate a design, observe and edit the results, and then resimulate the design; the Waveform Editor can then show the results superimposed on each other to highlight their differences.

Device Programming

PLS-MAX contains the MAX+PLUS Programmer software for programming and verifying the MAX-family EPLDs. (Programming hardware and several device adapters are provided with PLDS-MAX. See the *PLDS-MAX Data Sheet* in this data book for details.) The MAX+PLUS programming software drives the PC-AT or PS/2 add-on card and uses standard Altera programming hardware. If the Security Bit of the device is off, the designer can also read the contents of a MAX device and use this information to program additional EPLDs.

MAX+PLUS Timing Analyzer (MTA)

The MAX+PLUS Timing Analyzer (MTA) provides user-configurable reports that help the designer to analyze critical delay paths, setup and hold times, and overall system performance of any MAX EPLD design. Critical paths identified by these reports can be displayed and highlighted.

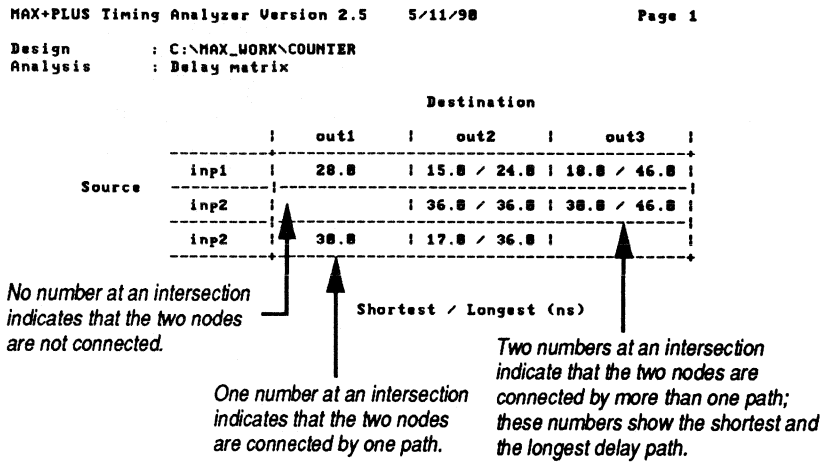
The MTA calculates timing delays between multiple source and destination nodes and creates a connection matrix that gives the shortest and longest delay paths between all specified source and destination nodes (Figure 7). The MTA also displays the detailed paths and delays between specified sources and destinations.

The setup/hold option provides information on setup and hold requirements at the device pins for all pins that feed the **D**, **CLK**, or **ENABLE** inputs of flip-flops and latches. Critical source nodes can be specified individually, or for all pins can be calculated. This information is then displayed in a table with one set of setup and hold times per flip-flop or latch.

The MTA also allows the user to print a complete list of all accessible nodes in a design, i.e., all nodes that can be displayed during simulation or delay prediction.

All MTA functions can be executed in batch mode with an MTA command file, so the user can specify all information needed to configure the output.

Figure 7. Delay Analysis Matrix



3

SNF2GDF Converter

The Simulator Netlist File-to-Graphic Design File (SNF2GDF) Converter converts the MAX+PLUS Simulator Netlist File (SNF) into logic schematics that contain basic gates and flip-flop elements. It uses the SNF's delay and connection information to create a series of schematics that are fully annotated with propagation delay and setup and hold information at each logic gate. Certain speed paths of a design can be specified for conversion, so the user can graphically analyze only those paths that are considered critical. See Figures 8 and 9.

If the Altera Hardware Description Language (AHDL) is used, SNF2GDF shows how the high-level description has been synthesized and placed into the MAX architecture.

PLS-MAX Contents

- ❒ Floppy disks containing all programs and files for MAX+PLUS software for both PC-AT and PS/2 and compatible computers
- ❒ Documentation

Ordering Information

PLS-MAX (supports both PC-AT and PS/2 formats)

Refer to the *PC System Requirements Data Sheet* in this data book for information on system requirements and sample system configurations.

Figure 8. Original Schematic File

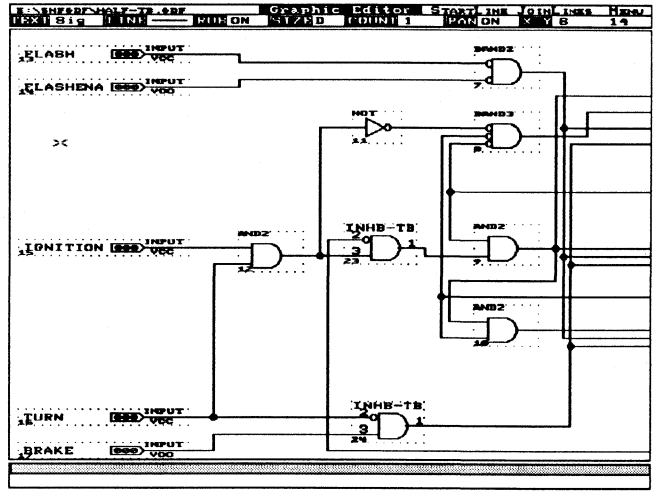
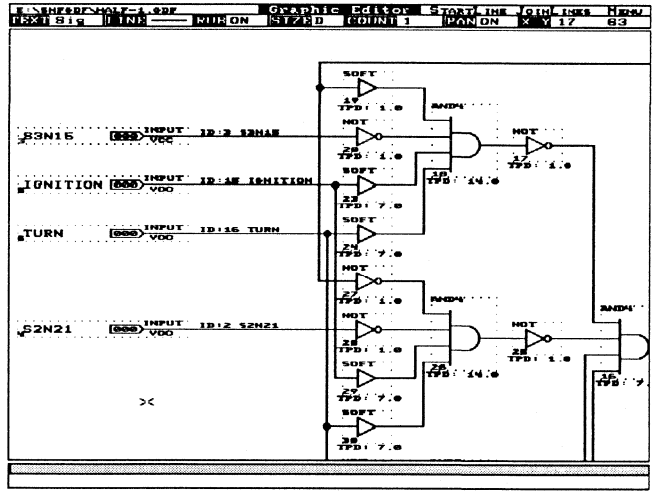


Figure 9. Schematic Converted and Annotated with SNF2GDF

This screen capture shows the schematic for the compiled and converted design, which displays delay information and the results of logic synthesis.



Contents

October 1990

Section 4 EPM7000-Series MAX EPLDs

4

EPM7000-Series Devices

High-Performance, High-Pin-Count MAX EPLDs

October 1990, ver. 1

Data Sheet

Features

Advance Information

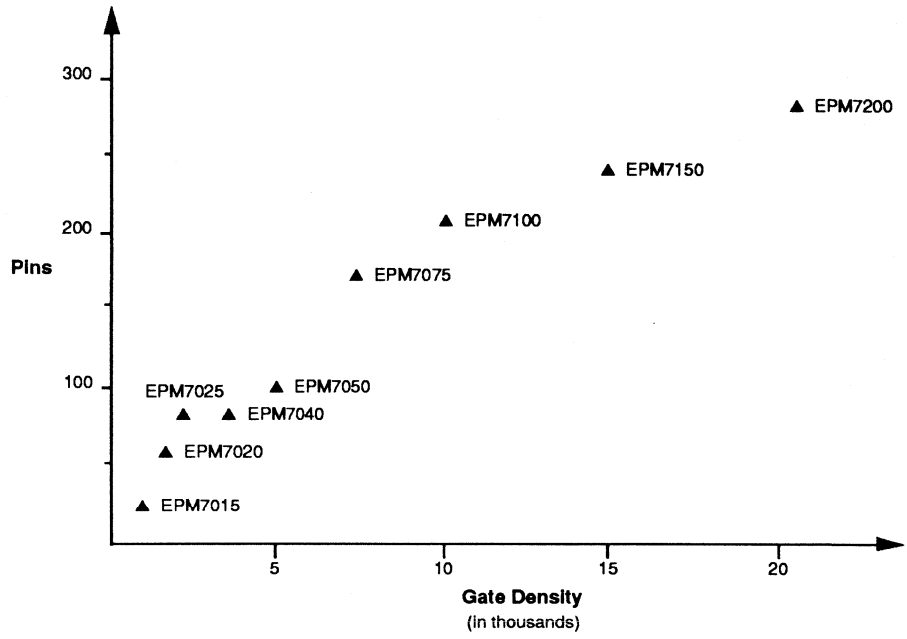
- High-density, high-speed CMOS EPLDs with second-generation Multiple Array MatriX (MAX) architecture
- Advanced 0.8-micron double-metal CMOS EPROM technology
- Complete EPLD series ranging from 1,500 to 20,000 gates
- Fast, 15-ns pin-to-pin logic delays with 70-MHz true system-clock frequency (including interconnect)
- Programmable "power saver" mode
- 44 to 208 pins available in PLCC, PGA, and QFP packages
- User-defined I/O options for support of bus-interface functions
- Enhanced Programmable Interconnect Array (PIA) that provides a fixed delay from any internal source to any destination within the EPLD
- Advanced macrocell to efficiently place logic for optimum speed and density
- Programmable registers providing D, T, JK, or SR flip-flops with individual Clear, Preset, and Clock controls
- High pin-to-logic ratio for I/O-intensive data path applications and 32-bit microprocessor support logic
- Full software support for PC and workstation platforms (including Apollo, Sun, and IBM) with Altera's software development systems
  - Hierarchical schematic capture with over 340 TTL and custom macrofunctions
  - Altera Hardware Description Language (AHDL) for Boolean equation, state machine, and truth table design entry
  - Waveform entry
  - Logic synthesis and minimization
  - Device fitting within minutes
  - Full timing simulation
  - Automatic multi-chip partitioning and simulation
- EDIF netlist interface for additional schematic capture and simulation support

4

General Description

The EPM7000 series—Altera's next generation of erasable, high-density, high-performance MAX EPLDs—provides a variety of solutions for general-purpose logic integration. Ranging in gate density from 1,500 to 20,000 gates and supplied in packages from 44 to over 250 pins, the EPM7000 series offers up to 4 times the logic density and more than 3 times the system clock speed of Field Programmable Gate Arrays (FPGAs). See Figure 1.

Figure 1. EPM7000-Series EPLDs

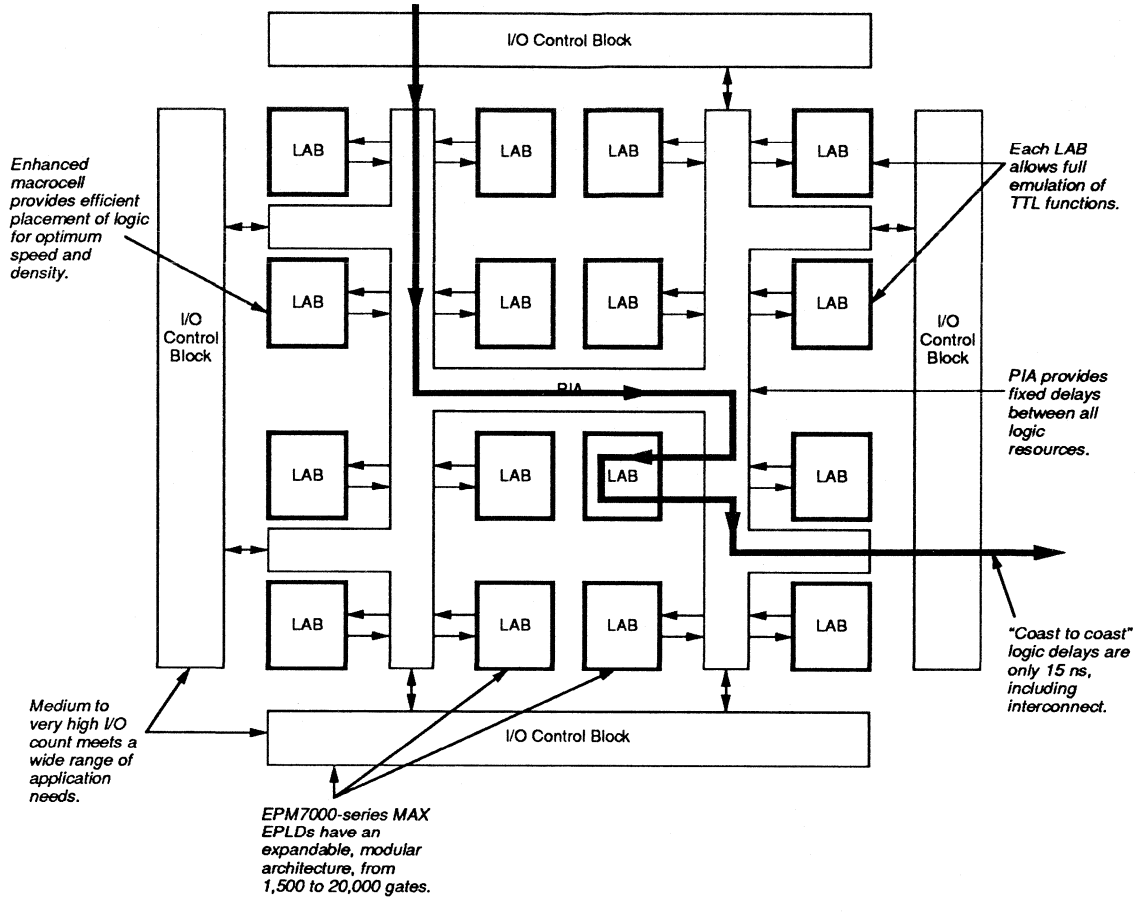


EPM7000-series MAX EPLDs are based on a logic matrix architecture that consist of modular Logic Array Blocks (LABs) connected with a Programmable Interconnect Array (PIA). See Figure 2. The PIA provides a connection path with a small fixed delay between all internal signal sources and logic destinations. It has been carefully optimized so that device performance can be accurately predicted early in the design phase, and speed penalties for design changes can be avoided.

Macrocells within the LAB are optimized for efficient placement of logic resources on the basis of speed and density requirements. These enhanced macrocells support both combinatorial and registered functions and allow 100% TTL emulation. Register options (D, T, JK, SR) and programmable Clock control may be individually configured for each macrocell. The EPM7000-series macrocell, together with a fast PIA, provides true system clock rates of 70+ MHz, even with complex logic functions.

Logic designs are implemented in EPM7000-series EPLDs with Altera's PC- and workstation-based development systems. Designs can be entered using hierarchical schematic capture with TTL macrofunctions, as well as using Boolean equations and state machines with the Altera Hardware Description Language (AHDL). Waveform design entry is also supported. Interfaces to third-party tools are available to allow design entry and logic simulation on a variety of workstation platforms.

Figure 2. EPM7000-Series Block Diagram



A powerful compiler minimizes and synthesizes the design, then automatically fits it into the most appropriate EPLD. If the design is too large, the compiler automatically partitions the logic into two or more EPLDs. The design may be verified with an integrated simulator, a full AC timing simulator, and with an interactive waveform editor that speeds waveform creation and debugging. Since the design is processed in minutes, several iterations can be completed in a single day.

Notes:

Contents

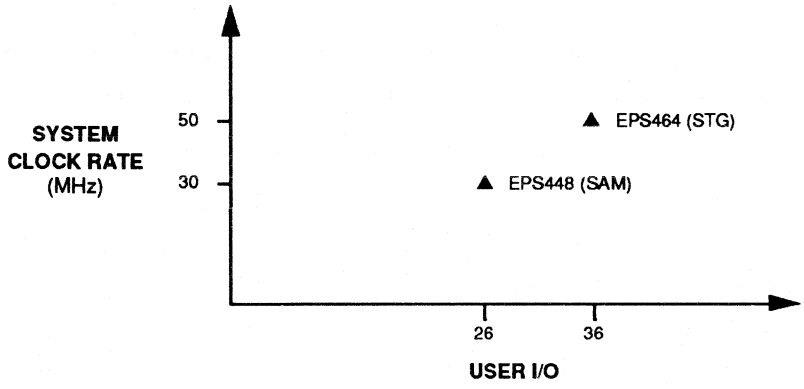
October 1990

Section 5 EPS-Series SAM & STG EPLDs

5

EPS-Series EPLDs

Synchronous State Machine and Waveform Generation EPLDs



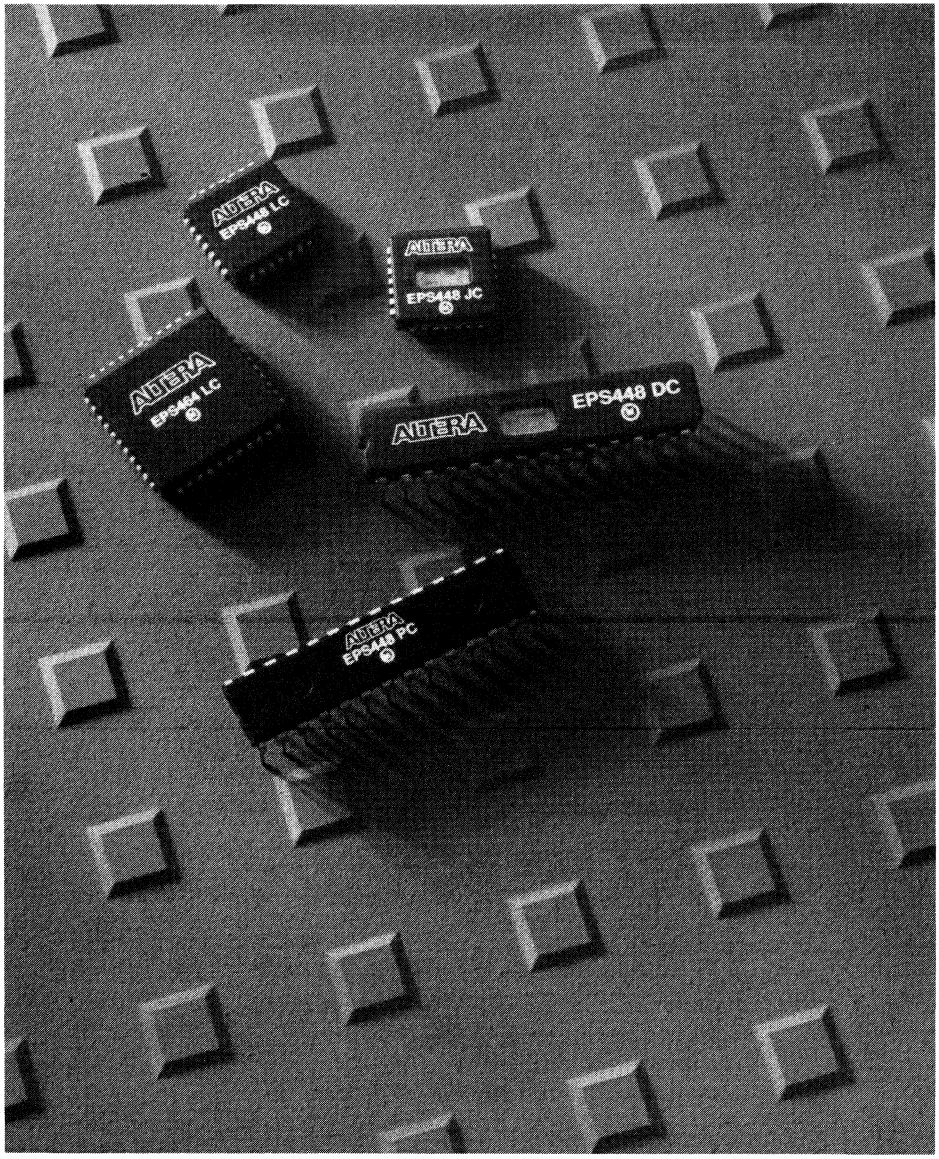
EPS448: Stand-Alone Microsequencer

- Provides efficient solutions for state machines, bus- and memory-control functions, graphics, and DSP algorithm controllers.
- On-chip reprogrammable microcode EPROM up to 448 words deep
- Prioritized, multiway branch control
- 15 × 8 stack for implementing subroutines, nested loops, branch control, and other iterative functions
- 8-bit loop counter for timing and delay loops
- 30-MHz clock frequency
- 28-pin, 300-mil DIP or JLCC/PLCC package
- Vertically and horizontally cascadable
- SAM+PLUS development software:
  - Altera State Machine Input Language (ASMILE)
  - Assembly Language (ASM)
  - SAM Design Processor (SDP)
  - Functional Simulator (SAMSIM)

EPS464: Synchronous Timing Generator

- Generates complex control timing waveforms for all types of imaging applications (CCD imagers, video displays, optical disks, etc.).
- Programmable architecture implements NTSC, PAL, and SECAM synchronization standards for TV/video applications.
- Powerful macrocell structure supports complex waveform and state machine designs.
- Programmable I/O supports up to 36 inputs and 32 outputs.
- "Quiet" outputs minimize output switching noise.
- 50-MHz clock frequency
- 44-pin JLCC/PLCC or 44-pin plastic QFP package options
- Advanced development software support:
  - Waveform design entry
  - Logic synthesis
  - Timing simulator

5


ALTERA
EPS448 LC
ALTERA
EPS448 JC
ALTERA
EPS448 LC
ALTERA
EPS448 DC
ALTERA
EPS448 PC

EPS448 SAM EPLD

Stand-Alone Microsequencer

October 1990, ver. 2 — **Data Sheet**

Features

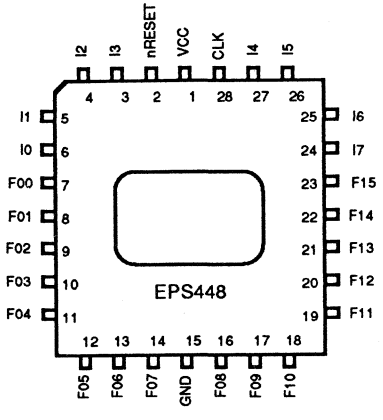
- User-configurable Stand-Alone Microsequencer (SAM) for implementing high-performance controllers
- On-chip reprogrammable microcode EPROM up to 448 words deep
- 15 × 8-bit stack
- Loop counter
- Prioritized multiway control branching
- 8 general-purpose branch-control inputs
- 16 general-purpose control outputs
- Cascadable to expand the number of outputs or states
- Low-power CMOS technology
- Footprint-efficient packages: 28-pin, 300-mil DIP or JLCC/PLCC
- 30-MHz clock frequency
- High-level software support with SAM+PLUS Development System:
  - Altera State Machine Input Language (ASMILE)
  - Assembly Language (ASM)
  - SAM Design Processor (SDP)
  - SAMSIM functional simulator

5

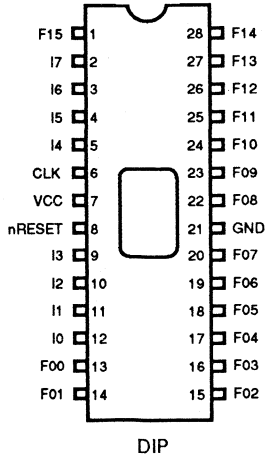
General Description

The EPS448 EPLD is a function-specific, user-configurable stand-alone microsequencer (SAM). The on-chip EPROM of each EPS448 device (up to 448 words) is integrated with branch-control logic, a pipeline register, a stack, and a loop counter. This generic microcoded architecture can efficiently implement a broad range of high-performance controllers, from state machines to waveform-generation applications.

Figure 1. EPS448 Pin-Out Diagrams



J-Lead



DIP

The EPS448 EPLD is available in 28-pin, 300-mil windowed ceramic dual in-line packages (DIPs) and in 28-pin ceramic J-lead chip carriers (JLCCs). One-time-programmable plastic J-lead (PLCC) versions of the EPS448 EPLD are also available for volume production. See Figure 1.

The 1.2-micron CMOS EPROM technology allows the EPS448 EPLD to operate at 30-MHz clock frequency while still benefitting from low CMOS power consumption. This technology also

facilitates 100% generic testability, which eliminates the need for post-programming testing.

Altera's SAM+PLUS software provides design entry, logic optimization and functional simulation for EPS448 designs. With SAM+PLUS, designs are entered in either state machine or microcoded format. The software automatically performs logic minimization and design fitting. The design can then simulate the design or program it directly to create customized working silicon. Programming takes only a few minutes with standard Altera programming hardware, LogicMap II software, and a PLED448 or PLEJ448 adapter. New users can purchase the complete PLDS-SAM Development System with programming hardware included; PLS-SAM is a software-only package for existing Altera systems.

Applications

Ideal EPS448 applications include programmable sequence generators (i.e., state machines), bus and memory control functions, graphics and DSP algorithm controllers, and other high-performance control logic. EPS448 devices can be cascaded horizontally for greater output capabilities and vertically for deeper microcode memory. See *Application Brief 65 (Vertical Cascading of EPS448 SAM EPLDs)* in this data book for more information.

EPS448 as a State Machine

EPS448 architecture easily implements synchronous state machines. The device's internal EPROM memory and pipeline register allow up to 448 unique states to be specified. Its branch-control logic allows single-clock, multiway branching based on the 8 inputs, the current device state, and the user-defined transition conditions. Design entry is simplified with the Altera State Machine Input Language (ASMILE) supported by SAM+PLUS software. This high-level language uses **IF–THEN** statements to define state transitions and truth tables to define or tri-state the outputs on a state-by-state basis.

EPS448 as a Microcoded Controller

EPS448 architecture provides several advanced features that make the device suitable for use as a complex microcoded controller. The EPS448 EPLD's 448-word on-chip EPROM is integrated with a microcode sequencer consisting of branch-control logic, a stack, and a loop counter. The branch-control logic—fed by the 8 general-purpose inputs, the counter, the stack, and the pipeline register—provides flexible, multiway microcode branch capability in a single clock, enhancing throughput beyond that of conventional controllers or sequencers.

For microcoded controllers, SAM+PLUS software offers the high-level Assembly Language (ASM) design entry format. This language consists of powerful instructions (i.e., opcodes) that easily implement conditional

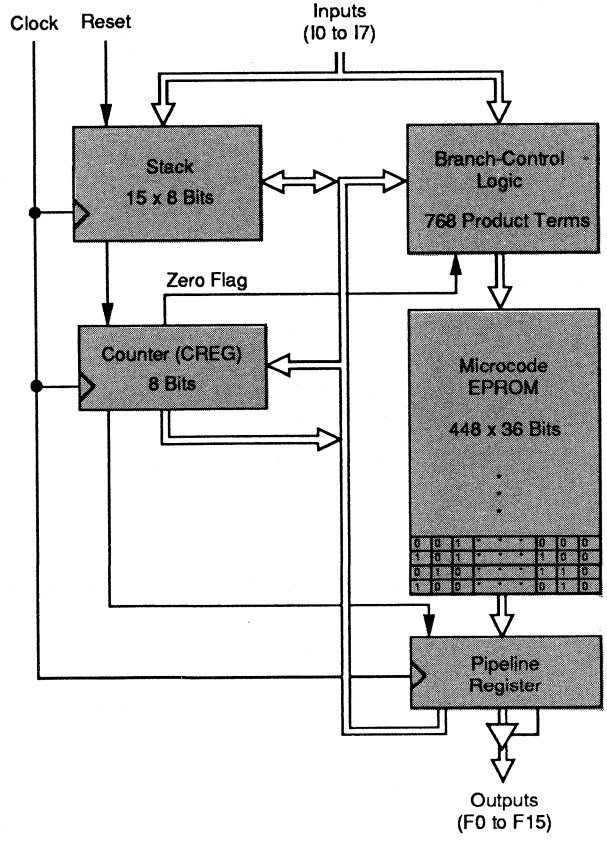
branches, subroutine calls, multi-level FOR-NEXT loops, and dispatch functions (i.e., branching to an externally specified address). For more information, see "Instruction Set" later in this data sheet.

Functional Description

As shown in Figure 2, the EPS448 EPLD consists of microcode EPROM, a 36-bit pipeline register, branch-control logic, a 15 × 8-bit stack, and an 8-bit loop counter.

The branch-control logic generates the address of the next state and applies it to the microcode memory. The outputs of the microcode memory represent user-defined outputs and internal control values associated with the next state. These new values are clocked into the pipeline register on the leading edge of the clock and become the current state. The new values in the pipeline register—along with the counter, stack, and inputs—are used by the branch-control logic to generate the new next-state address.

Figure 2. EPS448 Block Diagram



5

Microcode EPROM and Pipeline Register

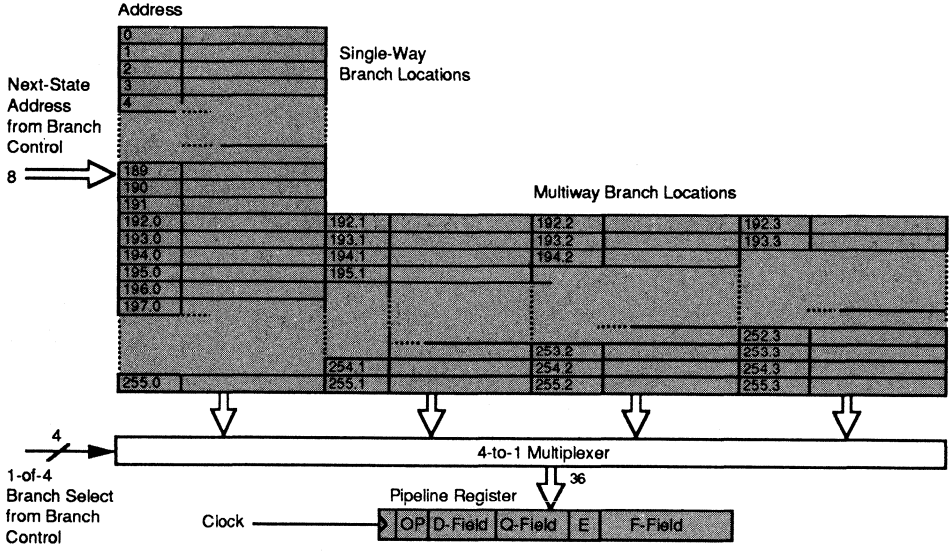
The microcode EPROM is organized into 448 36-bit words, each of which can be viewed as a single state location. Each of the 36 bits is divided into the following categories:

F-field (16 bits)	consists of user-defined outputs at device pins.
Q-field (8 bits)	provides the next-state address.
D-field (8 bits)	is a general-purpose field used either as a constant or as an alternative next-state address.
OP-field (3 bits)	contains the instruction (opcode).
E-field (1 bit)	enables or tri-states the device outputs.

As shown in Figure 3, the microcode memory is organized as 255 addresses. Addresses 0 through 191 contain a single 36-bit word, which is associated with the desired next state. This state information is clocked into the pipeline register on the rising edge of the clock, and the outputs become valid one clock-to-output delay (t_{CO}) later.

Addresses 192 through 255 access 4 unique 36-bit words, each of which corresponds to a different possible next state. (The extensions .0, .1, .2, and .3 are added to the addresses to distinguish the four states.) These 64 addresses make up the multiway branch locations, and are used to perform single-clock, four-way branching. Whenever the next-state address falls within the multiway branch locations, the branch-control logic makes the necessary 1-of-4 selection based on the next-state address and user-defined input conditions.

Figure 3. Microcode Memory and Pipeline Register

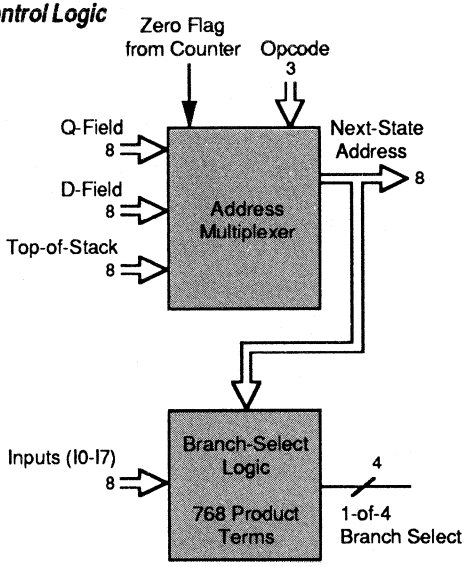


Branch-Control Logic Block

The branch-control logic is the key to the high-performance sequencing ability of the EPS448 EPLD. This block determines the next state to be clocked into the pipeline register, based on the current status of the pipeline register, the counter, the stack, and the eight input pins.

The branch-control logic is divided into two segments: the address multiplexer and the branch-select logic. See Figure 4.

Figure 4. Branch-Control Logic



The address multiplexer provides the next-state address to the microcode memory. The next-state address can come from the Q-field, the D-field, or the top-of-stack. The selection is based on the instruction in the pipeline register and the condition of the zero flag from the counter.

The branch-select logic is a programmable logic block with 768 product terms, 16 inputs, and 4 outputs. It is used to perform a 2-, 3-, or 4-way branch based on user-defined input conditions. When the next-state address falls within the multiway branch range of memory—i.e., any address greater than 191—the branch-select logic performs the necessary 1-of-4 selection. When the next-state address is less than 192, no selection is required and the branch-select logic is turned off.

The conditions controlling the multiway branch are defined by the user with a simple **IF-THEN-ELSE** format, as shown in the following example:

```
IF      (cond3)   THEN   select 201.3
ELSEIF  (cond2)   THEN   select 201.2
ELSEIF  (cond1)   THEN   select 201.1
ELSE                     select 201.0
```

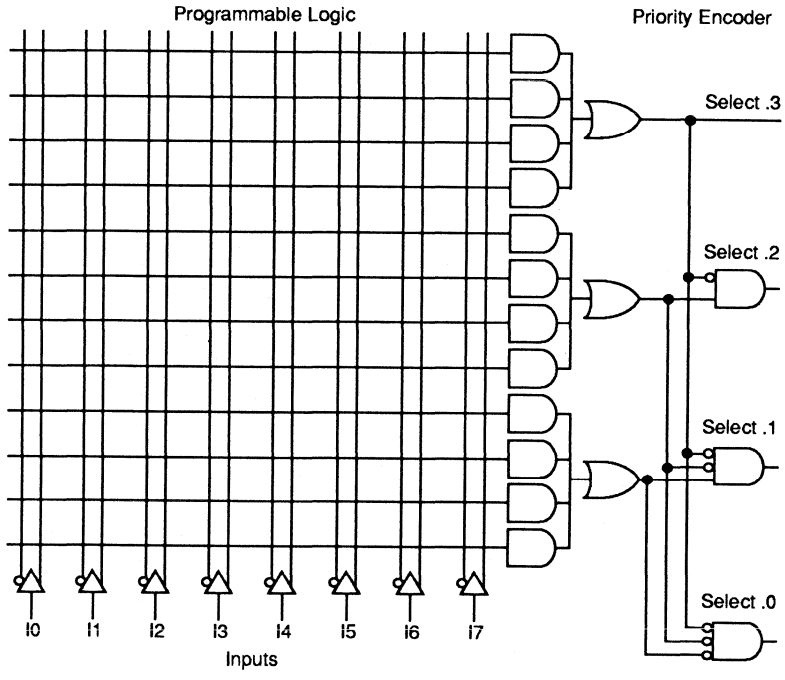
The conditions are prioritized so that if the first condition (i.e., **cond3**) is met, then microword **201.3** is selected and clocked into the pipeline register, regardless of the results of **cond2** and **cond1**. If none of the conditions is met, then microword **201.0** is clocked into the pipeline register.

The three conditional expressions are user-defined. They may contain any logical equation that is based on the inputs and can be reduced to four product terms, as shown in the following example:

```
  I1 * /I2 * /I4
+ I3 * /I4 * /I5 * /I6 * /I7
+ I0
+ I2 * /I4 * /I5
```

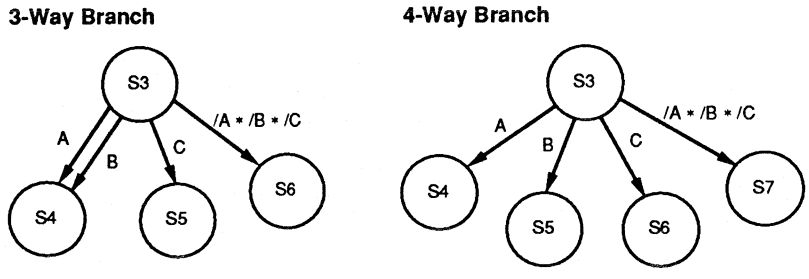
A unique set of 12 product terms is present in each of the 64 available multiway branch locations for a total of 768 product terms. See Figure 5.

Figure 5. Branch Logic in a Multiway Branch Location



The EPS448 EPLD is designed so that the number of available product terms is always sufficient for a design. Prioritization provides an effective product-term count of more than 12 per location. A tradeoff between the number of product terms and the number of possible branches can be made simply by placing identical state information in 2 locations, as shown in Figure 6.

Figure 6. Multiway Branching vs. Product-Term Needs

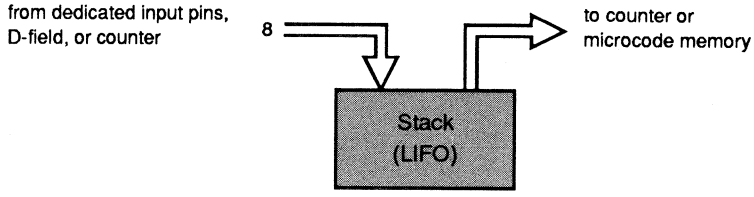


Stack

The EPS448 stack is a Last-In First-Out (LIFO) arrangement that consists of 15 8-bit words. The top of stack may be used as the next-state address or popped into the counter. Values may be pushed onto the stack from either the D-field in the pipeline register or from the counter. Thus subroutines, nested loops, and other iterative structures may be efficiently implemented. The logic levels on the 8 dedicated input pins may also be pushed onto the stack to allow external address specification in a dispatch function or to externally load the counter. See Figure 7.

The pushing or popping of the stack occurs on the leading edge of the clock. The stack is "zero-filled" so that a pop from an empty stack will reset all 8 bits to zero. On the other hand, a push to an already full stack will write over the top-of-stack, leaving the other 14 values unchanged.

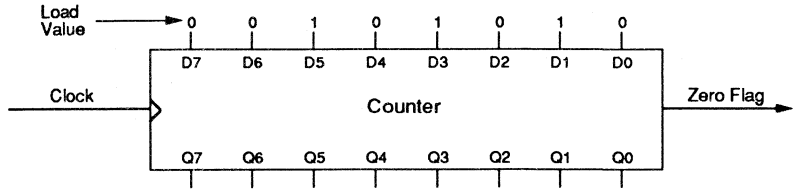
Figure 7. Stack



Loop Counter

The EPS448 EPLD contains an eight-bit loop counter called count register (CREG), which is useful for controlling timing loops and determining branch-control functions. The CREG is a down counter that may be loaded directly from the D-field of the pipeline register or from the top-of-stack. The value of the CREG may be saved and restored by pushing and popping it to and from the stack. See Figure 8.

Figure 8. Loop Counter



The CREG is loaded or decremented on the leading edge of the clock. It will not decrement once it reaches zero, thereby preventing roll-over. A zero flag indicates when the counter has reached zero. This flag is used with the **LOOPNZ** command to control program flow. (See "Instruction Set" later in this data sheet.) Single-instruction delay loops are easily constructed, and nested loops or delays of arbitrary length may be generated in combination with the stack.

Output Enable Control

Each microcode word contains an OE bit (i.e., the E-field) that enables the outputs when E = 1, and causes high impedance when E = 0. This bit is accessible through instruction set commands provided with SAM+PLUS software. This output-enable capability allows EPS448 EPLDs to be vertically cascaded to increase the number of states.

nRESET Pin

The **nRESET** pin acts as a master reset for the EPS448 EPLD, causing it to empty the stack, clear the counter, and load the microword at address 0 into the pipeline register. The **nRESET** signal is useful for system reset or for synchronizing several EPS448 devices that are cascaded vertically or horizontally.

The **nRESET** signal must be held low for at least three rising clock edges to reset the EPS448 EPLD. An **nRESET** of one rising clock edge causes the EPS448 device to enter into a supervisor mode; and an **nRESET** of two clock edges leads to an undefined state.

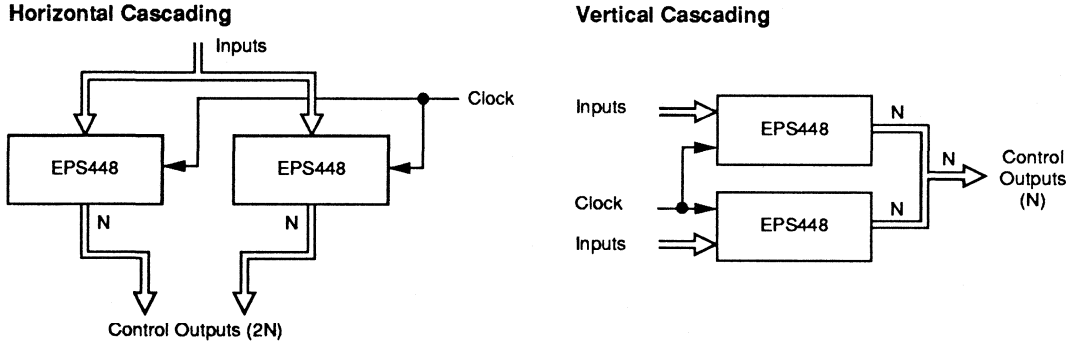
The outputs of the startup address (00 Hex) appear at the pins when the fourth clock edge after **nRESET** goes low, and are maintained until the third clock edge after **nRESET** returns to high.

When the EPS448 EPLD is operating in noisy environments, a glitch on the **nRESET** pin during one setup cycle (t_{SUR}) before the clock edge might initiate a supervisor mode. To prevent this effect, capacitor of at least 0.1 µF should be connected from the **nRESET** input to ground.

Horizontal and Vertical Cascading

EPS448 EPLDs, like memory- and bit-slice devices, can be cascaded to provide greater functionality (Figure 9). If an application requires more output lines, two or more EPS448 devices can be cascaded horizontally. Likewise, if an application requires more states, two or more EPS448 EPLDs can be cascaded vertically. In either case, no speed penalty is incurred. The designer can also simultaneously cascade EPS448 devices horizontally and vertically. Designs with horizontal cascading are fully supported by the SAM+PLUS development software. However, vertical cascading requires the designer to make certain tradeoffs to split the design. Refer to *Application Brief 65 (Vertical Cascading of EPS448 SAM EPLDs)* for more information.

Figure 9. Horizontal and Vertical Cascading

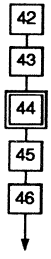


Instruction Set

The instruction set used to enter designs for the EPS448 EPLD consists of a compact assortment of powerful commands that allows efficient implementation of multiway branching, subroutines, nested FOR-NEXT loops, and dispatch functions. These instructions are used only with ASM design entry.

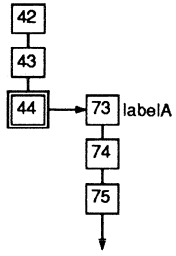
Each command in the instruction set is described and illustrated here. In the following descriptions, labelA and labelB represent arbitrary labels located in the ASM file. These symbolic labels are converted by the SAM+PLUS software into 8-bit absolute addresses. (SAM+PLUS allows the designer to use the high-level Assembly Language without worrying about the actual values that are placed in the various fields.) The parameter constant is any 8-bit number (0 to 255 decimal, 0 to FF hexadecimal) that represents an address, a mask, or a constant.

For simplicity, it is assumed that the sample destination labels in the following descriptions are not in the multiway branch block. See "Multiway Branching" later in this data sheet for more details about this capability.



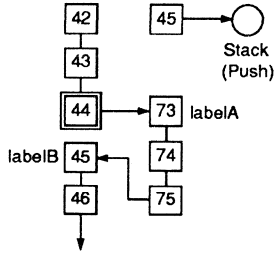
CONTINUE

This command causes execution to continue with the next sequential instruction in the ASM file. In this example, the current address is 44, and **CONTINUE** instructs SAM+PLUS to go to address 45 in the ASM file.



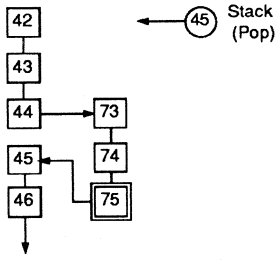
JUMP labelA

This instruction causes execution to branch to the indicated location. In this example, address 44 contains the instruction **JUMP labelA**; labelA is located at address 73. The next instruction will come from labelA.



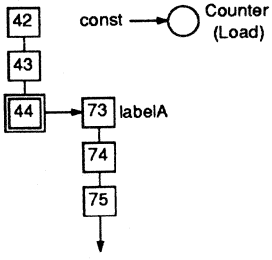
CALL labelA RETURNTO labelB

This instruction pushes the address of labelB onto the stack and makes labelA the next-state address. **CALL labelA** without the **RETURNTO** command makes labelB default to the next instruction in the ASM file. In this example, the address location 44 contains the instruction **CALL labelA**; labelA is located at address 73. The instruction pushes the address of the next instruction (45) onto the stack and causes the next instruction to come from address 73. The **RETURN** instruction at address 75 returns the execution to address 45. The **CALL** command is typically used to call a subroutine.



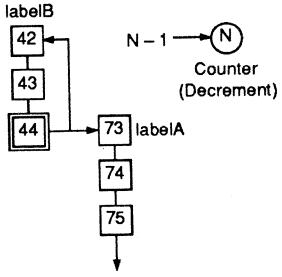
RETURN

This command causes the address of the next instruction to come from the top-of-stack and pops that value off the stack. In this example, the instruction at address 44 calls the subroutine at address 73 and pushes the value 45 onto the stack. The **RETURN** instruction at address 75 pops the value 45 out of the top-of-stack and causes execution to continue with address 45. **RETURN** is most frequently used to return from a subroutine.



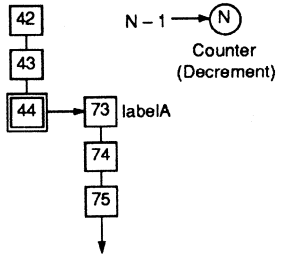
LOADC constant GOTO labelA

This command loads the counter with the specified value and then executes the instruction at labelA. If **GOTO** is not included in the instruction, labelA defaults to the next instruction in the ASM file. In this example, the instruction **LOADC 173D GOTO labelA** is located at address 44. This means that the decimal value 173 is loaded into the counter and the next state comes from labelA at address 73. **LOADC** is typically used to load the counter before entering a FOR-NEXT loop or a wait-state generator.



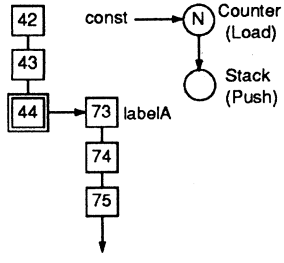
LOOPNZ labelB ONZERO labelA

This instruction jumps to one of two addresses based on the value of the zero flag, and decrements the counter if it is not already zero. If it is zero (i.e., zero flag = 1), the next instruction comes from labelA. If it is not zero (i.e., zero flag = 0), the next instruction comes from labelB. If the **ONZERO** instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **LOOPNZ labelB ONZERO labelA**, where labelB is located at address 42 and labelA at address 73. If the counter is not at zero, the instruction at address 42 is executed and the counter is decremented. If the counter is already at zero, the instruction at address 73 is executed and the counter remains at zero. **LOOPNZ** is typically used to implement FOR-NEXT loops.



DECNZ GOTO labelA

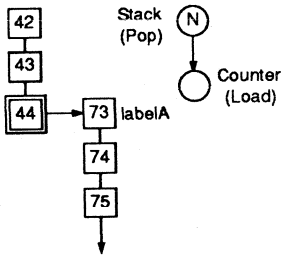
This command decrements the counter if it is not zero and then jumps to the instruction specified at labelA. If **GOTO** is not included in the instruction, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **DECNZ GOTO labelA**, where labelA is located at address 73. The counter is decremented if it is not zero and the next instruction comes from address 73. **DECNZ** is typically used to conditionally decrement the counter.



PUSHLOADC constant GOTO labelA

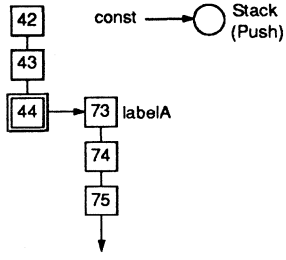
This instruction pushes the current value of the counter onto the stack, loads a new value into the counter, and jumps to labelA. If the **GOTO** instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **PUSHLOADC 153D GOTO labelA**, where labelA is located at address 73. The value in the counter is pushed onto the stack, the decimal value 153 is loaded into the counter, and the next instruction comes from address 73. **PUSHLOADC** is useful for implementing FOR-NEXT loops.

5



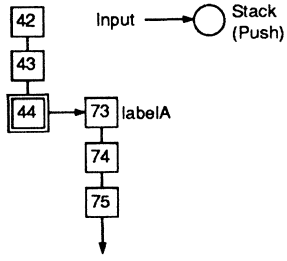
POPC GOTO labelA

This command pops the top-of-stack into the counter and jumps to labelA. If the **GOTO** instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **POPC GOTO labelA**, where labelA is located at address 73. The current value at the top-of-stack is removed from the stack (i.e., popped) and loaded into the counter. The next instruction comes from address 73. **POPC** is typically used with the **PUSHLOADC** instruction to implement nested FOR-NEXT loops.



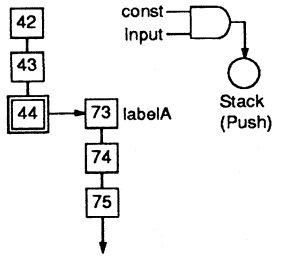
PUSH constant GOTO labelA

This command pushes the value of the constant onto the stack and jumps to labelA. If the **GOTO** instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **PUSH 34D GOTO labelA**, where labelA is located at address 73. The decimal value 34 is pushed onto the stack and the next instruction comes from address 73. **PUSH** is typically used to store a value on the stack.



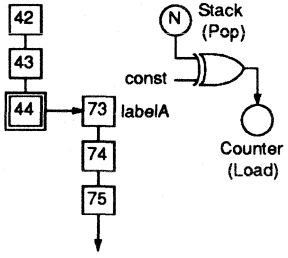
PUSHI GOTO labelA

This instruction pushes the eight inputs (I7 to I0) onto the stack. If the **GOTO** instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **PUSHI GOTO labelA**, where labelA is located at address 73. At the leading edge of the clock, the eight inputs are pushed onto the stack. Typically, address 73 would have a **RETURN** instruction that would cause execution to jump to the address represented by the recently pushed input pins, implementing a dispatch function. This instruction can also be used to load the counter with an externally specified variable. To do so in this example, address 73 would have a **POPC** instruction.



ANDPUSHI constant GOTO labelA

This command pushes the eight inputs (I7 to I0) onto the stack. It is identical to the **PUSHI GOTO labelA** command, except that the inputs are first bit-wise ANDed with a constant to allow the making of irrelevant inputs. If the **GOTO** instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **ANDPUSHI GOTO labelA**, where labelA is located at address 73. At the leading edge of the clock, the eight inputs are masked with the constant and pushed onto the stack. The next instruction comes from address 73. **ANDPUSHI** is an advanced instruction typically used to branch to an externally specified resource or to externally load the counter.



POPXORC constant GOTO labelA

This instruction pops the top-of-stack, bit-wise XORs it with a constant, loads the results into the counter, and jumps to labelA. If the **GOTO** instruction is not included, labelA defaults to the next instruction in the ASM file. In this example, the instruction at address 44 is **POPXORC 25D GOTO labelA**, where labelA is located at address 73. The top-of-stack is popped off the stack, XORed with the decimal 25, and the result is loaded into the counter. The next state comes from address 73. **POPXORC** is an advanced instruction typically used to compare the inputs against a known value and then branch on the basis of the result.

Table 1 summarizes the effects of each instruction on the address multiplexer, the stack, and the counter.

Table 1. Instruction Set Summary

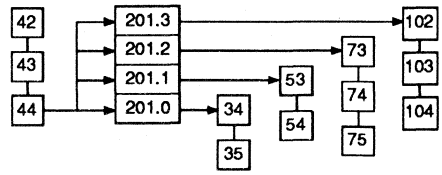
Instruction	Definition	Next-State Address	Effect on Stack	Effect on Counter
CONTINUE	Continue with next instruction	labelA	None	Hold
JUMP	Jump to a label	labelA	None	Hold
CALL	Call subroutine	labelA	labelB	Hold
RETURN	Return from subroutine	labelA	Pop	Hold
LOADC	Load CREG	labelA	None	Constant
LOOPNZ	Loop/decrement on non-zero	labelA or labelB	None	Decrement
DECNZ	Decrement CREG on non-zero	labelA	None	Decrement
PUSHLOADC	Push CREG to stack and load CREG	labelA	CREG	Constant
POPC	Pop stack to CREG	labelA	Pop	Stack
PUSH	Push constant to stack	labelA	Push	Hold
PUSHI	Push inputs to stack	labelA	Inputs	Hold
ANDPUSHI	Push masked inputs to stack	labelA	Inputs * Constant	Hold
POPXORC	XOR stack with constant and send result to CREG	labelA	Pop	Stack XOR constant

5

Multiway Branching

Multiway branching provides an added dimension to the capabilities of the instruction set. For example, a **JUMP labelA** to an address within the multiway branch block forces the branch-select logic to decide which of the four words to send to the pipeline register. This selection is based on user-defined functions of the inputs. See Figure 10.

Figure 10. Jumping to a Multiway Branch Address



Any of the 13 available commands can be enhanced with multiway branching. For example, location 44 in Figure 10 can be a **CALL** to a subroutine, and address 201 can contain the starting instruction for 4 unique subroutines. The routine that is actually executed depends on the user-defined condition of the inputs. The following ASM code can be used to implement this example:

```
44D:   [Output Spec] CALL labelA;
201D:  IF       cond1    THEN [out 1]  JUMP 102D;
         ELSEIF cond2    THEN [out 2]  JUMP 73D;
         ELSEIF cond3    THEN [out 3]  JUMP 53D;
         ELSE                 [out 4]  JUMP 34D;
```

Design Security

The EPS448 EPLD contains a programmable design Security Bit that controls access to the data programmed into the EPLD. If this Security Bit is used, a proprietary design implemented in the EPLD cannot be copied or retrieved. It provides a high level of design control because programmed data within EPROM cells is invisible. The Security Bit, along with all other program data, is reset by erasing the EPLD.

Functional Testing

The EPS448 EPLD is fully functionally tested and guaranteed through complete testing of each programmable EPROM bit and all internal logic elements, thus ensuring 100% programming yield. Figure 11 shows EPS448 AC test conditions.

Since the EPS448 EPLD is erasable, Altera can use and then erase test programs during early stages of production flow. This ability to use application-independent, general-purpose tests is called generic testing and is unique among user-defined LSI logic devices. EPS448 EPLDs also contain on-board test circuitry to allow verification of function and AC specifications after they are packaged in windowless packages.

Figure 11. AC Test Conditions

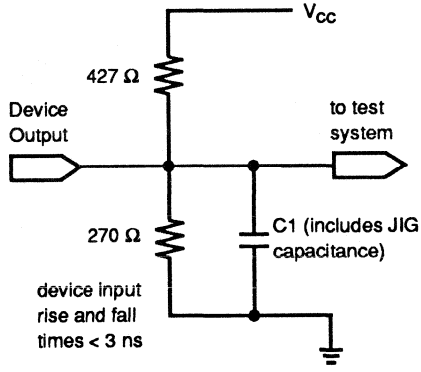
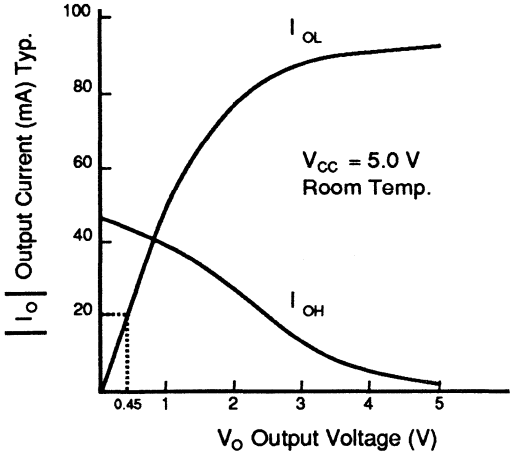
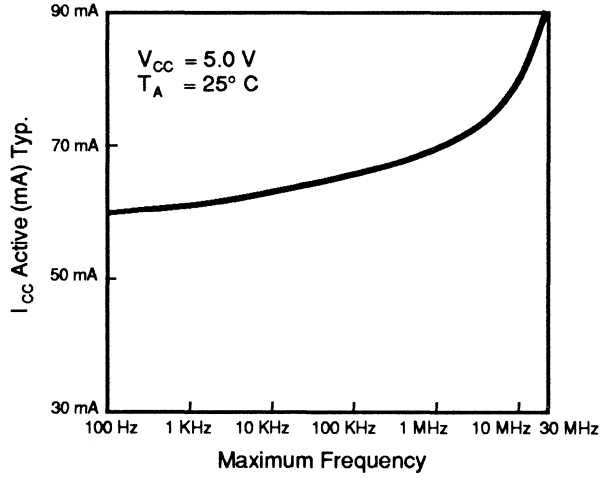


Figure 12 shows output drive characteristics for EPS448 I/O pins and typical supply current versus frequency for the EPS448 EPLD.

Figure 12. Output Drive Characteristics and I_{CC} vs. Frequency





5

Absolute Maximum Ratings Note: See *Operating Requirements for EPLDs* in this data book.

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	With respect to GND	–2.0	7.0	V
V_{PP}	Programming supply voltage	*See Note (1)*	–2.0	14.0	V
V_I	DC input voltage		–2.0	7.0	V
I_{MAX}	DC V_{CC} or GND current		–250	+250	mA
I_{OUT}	DC output current, per pin		–25	+25	mA
P_D	Power dissipation			1200	mW
T_{STG}	Storage temperature	No bias	–65	+150	° C
T_{AMB}	Ambient temperature	Under bias	–10	+85	° C

Recommended Operating Conditions

Symbol	Parameter	Conditions	Min	Max	Unit
V_{CC}	Supply voltage	*See Note (2)*	4.75 (4.5)	5.25 (5.5)	V
V_I	Input voltage		0	V_{CC}	V
V_O	Output voltage		0	V_{CC}	V
T_A	Operating temperature	For commercial use	0	70	° C
T_A	Operating temperature	For industrial use	–40	85	° C
T_C	Case temperature	For military use	–55	125	° C
t_R	Input rise time			500 (100)	ns
t_F	Input fall time			500 (100)	ns

DC Operating Conditions *See Note (2)*

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

Symbol	Parameter	Conditions	Min	Typ	Max	Unit
V_{IH}	High-level input voltage		2.0		V_{CC} + 0.3	V
V_{IL}	Low-level input voltage		–0.3		0.8	V
V_{OH}	High-level TTL output voltage	I_{OH} = –8 mA DC	2.4			V
V_{OH}	High-level CMOS output voltage	I_{OH} = –4 mA DC	3.84			V
V_{OL}	Low-level output voltage	I_{OL} = 8 (4) mA DC			0.45	V
I_I	Input leakage current	$V_I = V_{CC}$ or GND, *Note (3)*	–10		+10	µA
I_{OZ}	Tri-state output off-state current	$V_O = V_{CC}$ or GND	–10		+10	µA
I_{CC1}	V_{CC} supply current (standby)	$V_I = V_{CC}$ or GND, *See Note (4), Note (5)*		60	95 (120)	mA
I_{CC3}	V_{CC} supply current (active)	No load, 50% duty cycle, f = 20 MHz, *See Note (4)*		90	140 (200)	mA

Capacitance *See Note (6)*

Symbol	Parameter	Conditions	Min	Max	Unit
C_{IN}	Input capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{OUT}	Output capacitance	V_{OUT} = 0 V, f = 1.0 MHz		15	pF
C_{CLK}	Clock pin capacitance	V_{IN} = 0 V, f = 1.0 MHz		10	pF
C_{RST}	nRESET pin capacitance			75	pF

AC Operating Conditions

V_{CC} = 5 V ± 5%, T_A = 0° C to 70° C for commercial use
V_{CC} = 5 V ± 10%, T_A = –40° C to 85° C for industrial use
V_{CC} = 5 V ± 10%, T_C = –55° C to 125° C for military use

			EPS448-30		EPS448-25		EPS448-20		
Symbol	Parameter	Conditions	Min	Max	Min	Max	Min	Max	Unit
f_{CYC}	Maximum frequency	C1 = 35 pF	30		25		20		MHz
t_{CYC}	Maximum clock cycle			33.3		40		50	ns
t_S	Input setup time		16.5		20		22		ns
t_H	Input hold time		0		0		0		ns
t_{CO}	Clock to output delay	C1 = 35 pF		16.5		20		22	ns
t_{CZ}	Clock to output disable or enable			16.5		20		22	ns
t_{CL}	Minimum clock low time		11		12		15		ns
t_{CH}	Minimum clock high time		11		12		15		ns
t_{SUR}	nRESET setup time		16.5		18		18		ns
t_{HR}	nRESET hold time		5		5		5		ns

Notes to tables:
(1) Minimum DC input is –0.3 V. During transitions, the inputs may undershoot to –2.0 V or overshoot to 7.0 V for periods less than 20 ns under no-load conditions.
(2) Numbers in parentheses are for military and industrial temperature versions.
(3) For 1.0 < V_I < 3.8, the nRESET pin will source up to 200 µA.
(4) Typical values are for T_A = 25° C, V_{CC} = 5 V.
(5) This condition applies when the present state is a single-way branch location.
(6) Capacitance is measured at 25° C. Sample-tested only.

Product Availability

Grade		Availability
Commercial	(0° C to 70° C)	EPS448-30, EPS448-25, EPS448-20
Industrial	(–40° C to 85° C)	EPS448-20
Military	(–55° C to 125° C)	EPS448-20

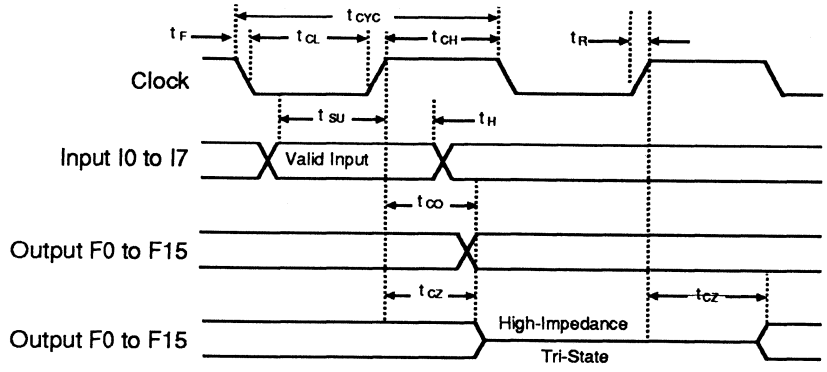
Note: Only military temperature-range EPLDs are listed above. MIL-STD-883B-compliant product specifications are provided in Military Product Drawings (MPDs), available from Altera Marketing at 1 (800) SOS-EPLD. These MPDs should be used to prepare Source Control Drawings (SCDs). See *Military Products* in this data book.

Figure 13 shows EPS448 timing and reset timing waveforms.

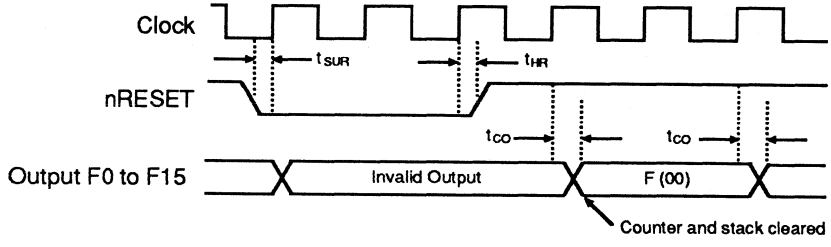
Figure 13. EPS448 Waveforms

If nRESET is held low for more than three clock edges, then the outputs associated with the boot address (00 Hex) will remain at the pins until the third clock after nRESET goes high.

Timing Waveforms



Reset Timing Waveforms



EPS464 STG EPLD

Synchronous Timing Generator

October 1990, ver. 1 **Data Sheet**

Features

Advance Information

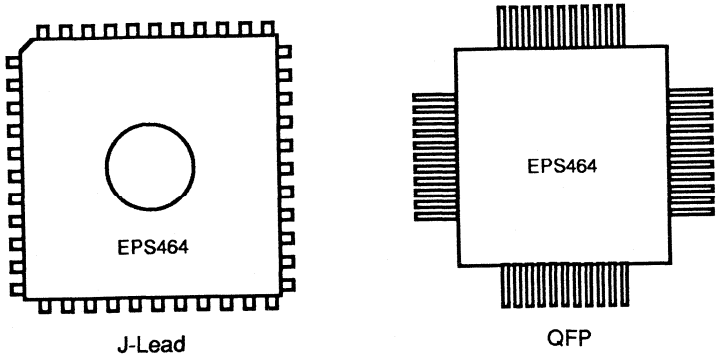
- High-performance Synchronous Timing Generator (STG) EPLD is ideally suited for custom waveform and state machine designs.
- Generates complex control timing waveforms for all types of imaging and display applications: CCD imagers, video displays, optical disks.
- Programmable architecture implements NTSC, PAL, and SECAM synchronization standards for TV/video applications.
- High-performance 50-MHz clock frequency
- Programmable I/O supports up to 36 inputs and 32 outputs
- "Quiet" output buffers and input buffers with 250-mV hysteresis for noise immunity and reliable operation
- Powerful macrocell structure
  - Modulo-*n* binary and Gray-code counters
  - Complex state machines
  - Multiple-product-term JK flip-flops for complex waveform generation
  - Phase comparator and clock oscillator functions
- Available in 44-pin, windowed ceramic JLCC, one-time-programmable PLCC, and plastic QFP packages
- Advanced software support featuring waveform design entry, Altera Hardware Description Language (AHDL), compilation, and simulation

5

General Description

The Altera EPS464 Erasable Programmable Logic Device (EPLD) provides an integrated solution for synchronous timing waveform-generation applications. Each of the EPS464 outputs can generate customized waveforms to meet a variety of different system requirements. Possible applications include TV/video synchronization signals (e.g., NTSC, PAL, SECAM, HDTV) as well as CCD timing controllers, high-performance state machines, and memory- and servo-controllers. The EPS464 EPLD is packaged in a 44-pin, windowed ceramic J-lead chip carrier (JLCC) a one-time-programmable plastic J-lead chip carrier (PLCC), or a 44-pin plastic quad flat pack (QFP) package. See Figure 1.

Figure 1. EPS464 Pin-Out Diagrams

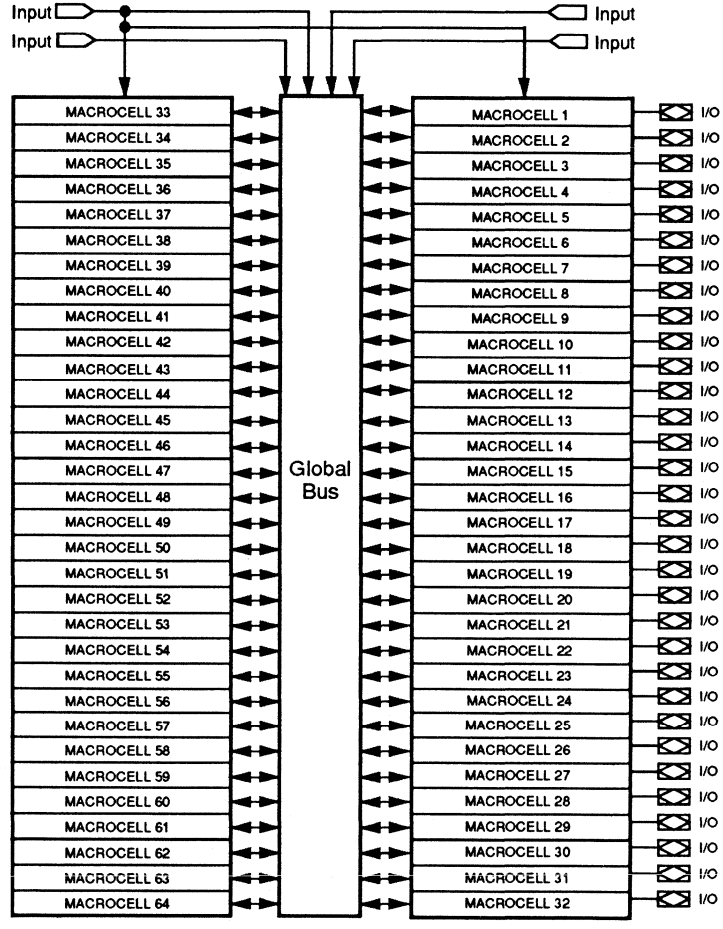


J-Lead QFP

The EPS464 EPLD contains 32 I/O pins that can be independently configured as dedicated outputs, dedicated inputs, or bidirectional pins. The EPS464 also contains 4 dedicated input pins, one of which may be programmed as a synchronous system clock.

The EPS464 EPLD contains 64 macrocells that are ideally suited for waveform-synthesis applications (Figure 2). The advanced macrocell structure of the EPS464 device allows integration of complex logic functions, with over 100 product terms available to any one macrocell. Each of the 64 internal flip-flops may be programmed for D, T, JK, or SR operation. JK and SR flip-flops are well suited for pattern-generation applications, since simple set and reset operations can be used to define the transitions of output waveforms. Each flip-flop can be clocked from either a fast system clock or a programmable asynchronous clock.

Figure 2. EPS464 Block Diagram



The EPS464 EPLD is programmed with Altera's development software. To simplify design entry for waveform-generation applications, a new graphical waveform entry method is available to describe the necessary timing waveforms. This waveform entry method is used with the Altera Hardware Description Language (AHDL), reducing the overall design time and allowing modifications to be made within minutes. The logic is then automatically synthesized to implement the function specified by the waveforms.

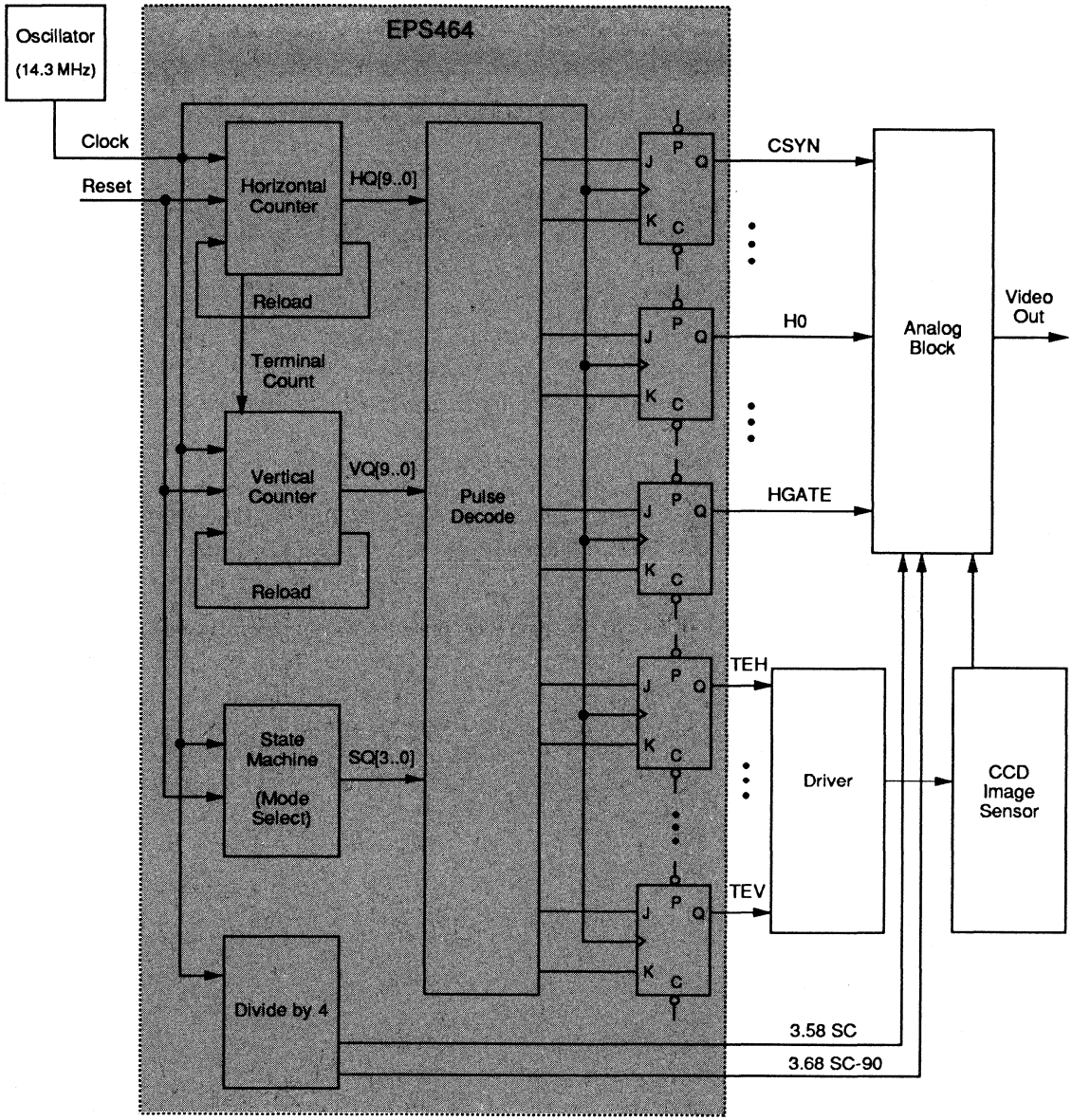
Principle of Operation

Waveform-generation and state machine applications can be efficiently implemented with the EPS464 device. Output waveforms are decoded from internal counters and, optionally, from internal state registers to set (i.e., perform low-to-high transition) or reset (perform high-to-low transition) the specified waveform.

Figure 3 shows a sample EPS464 design in which the EPS464 generates NTSC video-display waveforms and CCD timing control. Twenty of the EPS464 macrocells are configured for counters (10 bits horizontal and 10 bits vertical), and 4 macrocells are used to implement a state machine. The counters and state machine registers are then decoded by the EPS464 macrocells and connected to internal synchronous JK registers to set and reset desired output waveforms.

5

Figure 3. Typical EPS464 Application



PLS-SAM
SAM+PLUS Programmable Logic Software

October 1990, ver. 2 — **Data Sheet**

Features

- Development software for Altera's EPS448 Stand-Alone Microsequencer (SAM) EPLDs
- Altera State Machine Input Language (ASMILE)
- Assembly Language (ASM)
- User-definable macros
- SAM Design Processor (SDP) that generates industry-standard JEDEC files
- SAMSIM interactive functional simulator with Virtual Logic Analyzer (VLA) user interface
- Disassembler for examination of assembly code during simulation
- Full support for horizontal cascading of multiple EPS448 EPLDs
- Runs on IBM PC-AT, and PS/2 computers (and compatibles)
- Device programming with Altera programming hardware

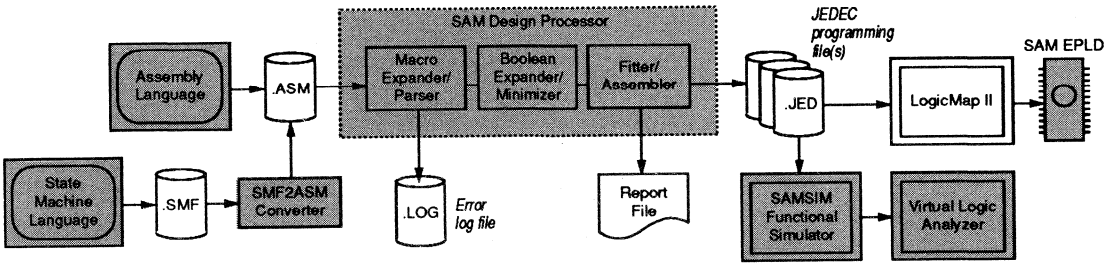
General Description

The Altera PLS-SAM (SAM+PLUS Programmable Logic Software) provides a complete software solution for implementing state machine and microcoded applications in Altera's EPS448 SAM EPLD. PLS-SAM is a comprehensive, easy-to-use system that includes state machine and assembly language design entry, design processing with the SAM Design Processor (SDP), and design debugging with SAMSIM. The EPS448 EPLD is programmed with Altera's LogicMap II software and programming hardware. See Figure 1. PLS-SAM is a software-only package. PLDS-SAM (Programmable Logic Development System) includes LogicMap II, programming hardware, and a software warranty (see the *PLDS-SAM: SAM+PLUS Programmable Logic Development System Data Sheet* for details).

The SDP accepts two forms of design entry—state machine and assembly language—and automatically generates an industry-standard JEDEC file

5

Figure 1. SAM+PLUS Block Diagram



for simulation and programming. SAMSIM is an interactive functional simulator created especially to verify state machine and microcoded designs implemented in EPS448 EPLDs.

Functional Description

Designs are entered in either the Altera State Machine Input Language (ASMILE) or the Altera Assembly Language (ASM). A standard text editor is used to create the input file with either method. If ASMILE is used, the State Machine File (SMF) is processed by a converter to produce an ASM file. The various modules of the SDP then process the ASM file.. The SDP produces three outputs: an industry-standard JEDEC file used to simulate and program the EPS448 EPLD, an error log file, and a utilization report file that shows how resources within the EPLD are used.

After the JEDEC file is created, the user can simulate the design with the SAMSIM functional simulator, which provides an interactive design-debugging environment. SAMSIM's Virtual Logic Analyzer (VLA) provides on-screen examination of input and output waveforms, and the disassembler converts object code back into the original ASM source code during simulation.

Horizontal cascading (i.e., using multiple EPS448 EPLDs to increase the number of outputs) is fully supported in design entry, processing, simulation, and programming. Multiple EPS448 EPLDs are listed in a single source file, but separate report and JEDEC files are created for each device.

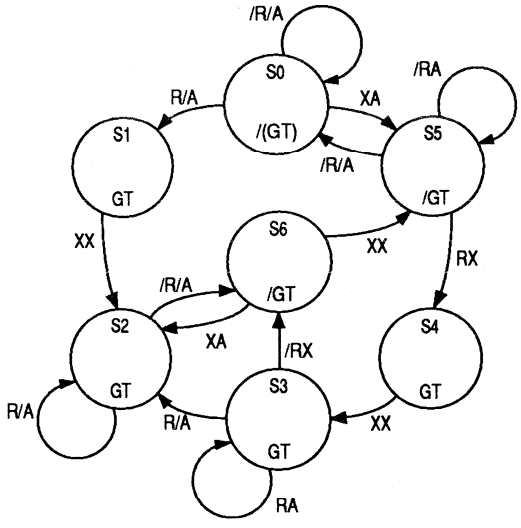
Finally, the EPS448 EPLD is programmed with LogicMap II software and programming hardware. Users who already have an Altera development system may use existing hardware together with the LogicMap II software and PLED448 or PLEJ448 adapters to program EPS448 EPLDs. For new users, PLDS-SAM includes all the programming hardware and software required to program the EPS448 EPLDs with a PC-AT, PS/2, or compatible system.

State Machine Design Entry

SAM+PLUS software supports high-level state machine design entry through ASMILE. A designer can use this language with any standard text editor to create a file describing a state machine. The State Machine File-to-Assembly Language file (SMF2ASM) Converter translates the SMF into an equivalent ASM file before sending it to the SDP.

ASMILE provides a simple yet comprehensive means of converting a conceptual state diagram into a simple text description. Figure 2 shows the state diagram for a 68020 bus arbiter. Each circle represents a state, the values within the circles represent the output values for that state, and the expressions adjacent to the arrows represent the conditional branches between states.

Figure 2. State Diagram for a 68020 Bus Arbiter



5

Figure 3 shows the ASMILE description of the state machine shown in Figure 2. The states and their respective outputs have been defined in the States Section with a truth table, and the transitions between states have been defined with simple **IF-THEN** constructs. Once this file is created, it can be submitted to the SDP without any further modifications.

Assembly Language Design Entry

Direct ASM design entry is also available for those who prefer to use EPS448 EPLDs for microcoded controller designs. This entry method provides access to the advanced features of the EPS448 device, including the on-chip stack and loop counter. Thirteen instructions directly control such functions as multiway branching, subroutines, nested FOR-NEXT loops, and dispatch calls (i.e., jumping to an externally specified address).

User-defined macros that allow users to define their own instruction mnemonics are also available, providing a higher-level design entry approach. Macros can also be used to define values for various output fields so that the designer does not have to work at the binary level.

Figure 4 shows an example of an ASM file in which macros have been used to define the seven new instructions **GOTOS0** through **GOTOS6**.

Figure 3. State Machine File

```
DESIGNER NAME
COMPANY NAME
10/1/90
68020 Bus Arbitration Controller for EPS448 SAM EPLD

PART:     EPS448
INPUTS:   REQUEST ACK
OUTPUTS:  GRANT TRISTATE
MACHINE:  BUSARBITER
CLOCK:    CLK

% The state table defines the outputs for each state %
STATES:   [  GRANT TRISTATE  ]
S0        [    0      0      ]
S1        [    1      1      ]
S2        [    1      1      ]
S3        [    1      1      ]
S4        [    1      1      ]
S5        [    0      1      ]
S6        [    0      1      ]

% Transition specifications %
S0:   IF REQUEST*/ACK THEN S1
      IF ACK THEN S5
      S0
S1:   S2
S2:   IF /REQUEST */ACK +ACK THEN S6
      S2          % IMPLIED ELSE %
S3:   IF /REQUEST THEN S6
      IF REQUEST*/ACK THEN S2
      S3
S4:   S3
S5:   IF /REQUEST*/ACK THEN S0
      IF REQUEST THEN S4
      S5
S6: S5

END$
```

Design Processor

The SDP takes an ASM file and creates an optimized JEDEC file for the target EPLD. This process includes the following steps:

- The user-defined macros are expanded.
- The design is parsed, and any syntax or connection errors are listed in an error log file.
- The Boolean expressions that define the transition conditions are minimized.
- The design is fitted into the EPS448 EPLD and an industry-standard JEDEC programming file is generated. A utilization report that shows how the design is implemented in the EPS448 EPLD is also created.

Figure 4. Assembly Language File

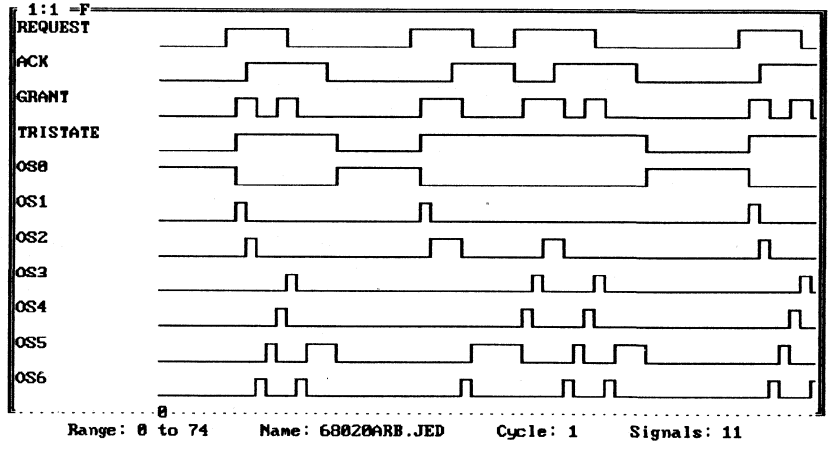
```
DESIGNER NAME
COMPANY NAME
10/1/90
68020 Bus Arbitration Controller for EPS448 SAM EPLD

PART:     EPS448
INPUTS:   REQUEST ACK
OUTPUTS:  GRANT TRISTATE

MACROS:
GOTOS0 = "[00] JUMP S0"
GOTOS1 = "[11] JUMP S1"
GOTOS2 = "[11] JUMP S2"
GOTOS3 = "[11] JUMP S3"
GOTOS4 = "[11] JUMP S4"
GOTOS5 = "[01] JUMP S5"
GOTOS6 = "[01] JUMP S6"

PROGRAM:
%    BUSARBITER    %
%    CLK           %
0D: GOTOS0;
S0: IF REQUEST*/ACK THEN GOTOS1;
    ELSEIF ACK THEN GOTOS5;
    ELSE GOTOS0;
S1:      GOTOS2;
S2: IF /REQUEST*/ACK+ACK THEN GOTOS6;
    ELSE GOTOS2;
S3: IF /REQUEST THEN GOTOS6;
    ELSEIF REQUEST*/ACK THEN GOTOS2;
    ELSE GOTOS3;
S4: GOTOS3;
S5: IF /REQUEST*/ACK THEN GOTOS0;
    ELSEIF REQUEST THEN GOTOS4;
    ELSE [01] JUMP S5;
S6: GOTOS5;

END$
```

SAMSIM Functional Simulator

Once a design has been processed, it can be simulated with the SAMSIM Functional Simulator. SAMSIM provides a comprehensive design debugging environment. The Virtual Logic Analyzer (VLA) displays the input and output waveforms interactively, providing multiple zoom levels, split screens, and differential time displays (see Figure 5). The internal state of the EPS448 EPLD, including the stack and counter, can be examined and modified. An online disassembler can convert the actual object code back into the original ASM source code.

5

Figure 5. Virtual Logic Analyzer Screen



Programming Hardware & Software

LogicMap II is the software used to program the EPS448 SAM EPLD. The software fully calibrates the programming environment and checks out the programming hardware (available in PLDS-SAM) when initiated. Programming hardware consists of a software-configured Logic Programmer card that occupies a half-card slot in the computer, a Master Programming Unit (PLE3-12A, and a programming adapter. LogicMap II works with this hardware to program and verify EPS448 EPLDs.

PLS-SAM Contents

PLS-SAM is provided for existing owners of Altera programming hardware. PLDS-SAM or PLDS-SAM/PS users receive all of the required programming hardware and PLS-SAM software (see the *PLDS-SAM Data Sheet* for more information).

- ❒ Floppy diskettes containing all programs and files for SAM+PLUS software for both PC-AT and PS/2 computers
  - Altera State Machine Input Language (ASMILE)
  - Assembly Language (ASM)
  - SAM Design Processor
  - SAMSIM Functional Simulator
  - LogicMap II
- ❒ Documentation

Ordering Information

PLS-SAM (supports both PC-AT and PS/2 formats)

Contents

October 1990

Section 6 EPB-Series EPLDs

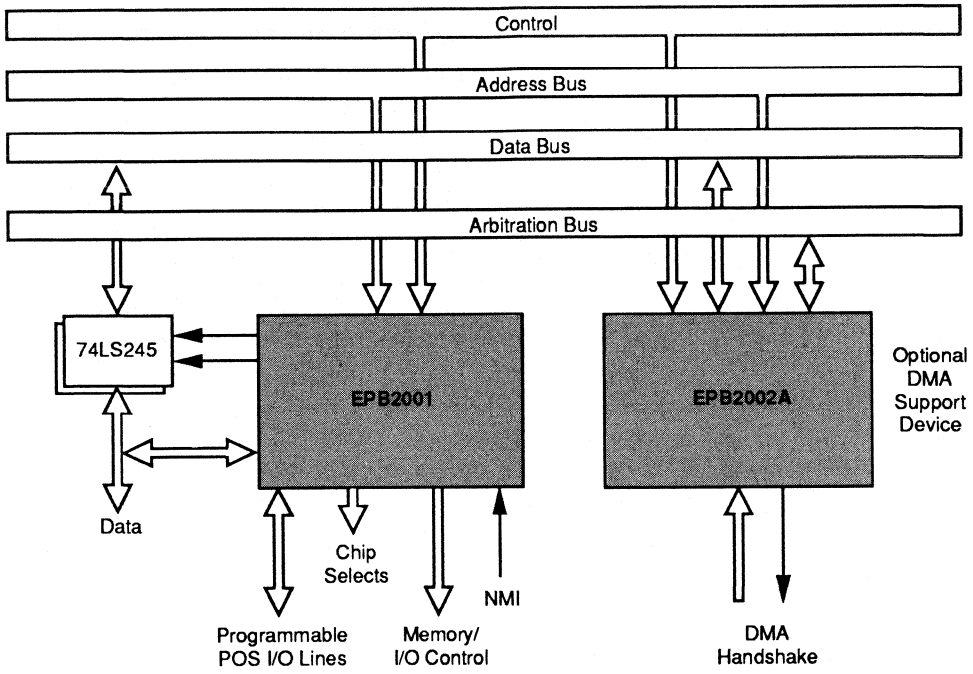
This section presents an overview of the EPB-Series EPLDs. Complete data sheets and application notes and briefs about EPB-Series EPLDs are available in the ***Micro Channel Adapter Handbook*** (April 1990).

6

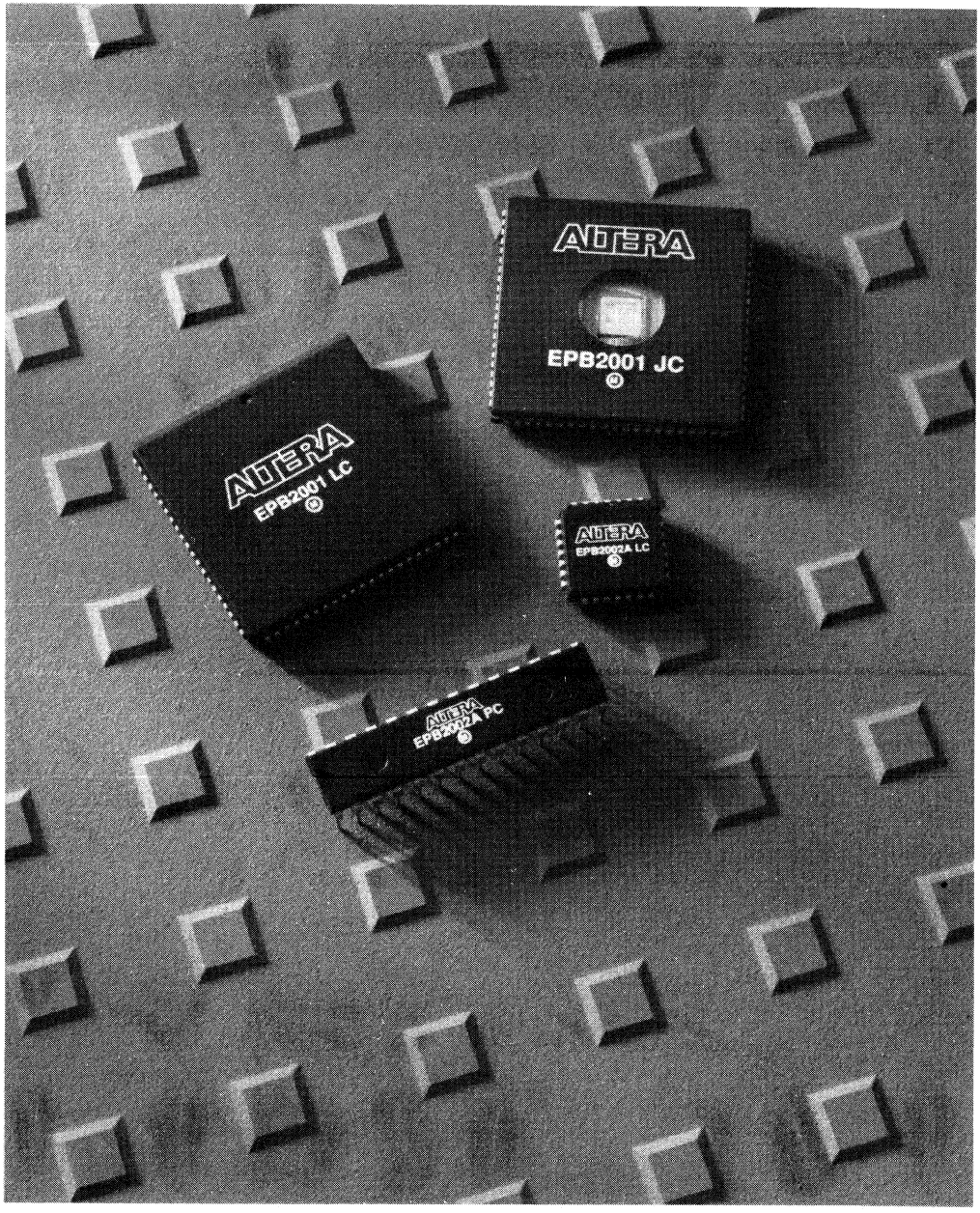
EPB-Series EPLDs

Altera User-Configurable Micro Channel Interface

Micro Channel Bus



- 100% Micro Channel-compatible architecture eliminates design debug problems and results in faster board design time.
- 30-mA power-supply current conserves limited board power for memory, I/O, and other essential ICs.
- 25-ns address decoding supports high-speed, zero "wait-state" data transfers.
- EPROM board ID POS registers eliminate extra ID registers.
- Programmable POS register I/O gives the designer a choice of POS bits accessible on board.
- 8 programmable chip-select outputs eliminate the need for extra address decoder PLDs and glue logic ICs.
- 24 Micro Channel address inputs support full address decoding from the Micro Channel bus.
- Multiple I/O or address decode ranges (up to 8 per chip-select output) provide multiple addressing options for the designer's board.
- 24-mA current drive outputs eliminate extra buffer ICs.
- Channel-check interrupt support enables the board to use bus Non-Maskable Interrupts for fast CPU interrupt response.
- Optional 28-pin EPB2002A EPLD provides DMA arbitration support.
- Altera's MCMap Development System simplifies Micro Channel design and eliminates design errors.


ALTERA
EPB2001 JC
ALTERA
EPB2001 LC
ALTERA
EPB2002A LC
ALTERA
EPB2002A PC

Contents

October 1990

Section 7 Operating Requirements for EPLDs

7

Operating Requirements for EPLDs

October 1990, ver. 1

Introduction

Altera EPLDs combine unique architectures with an advanced CMOS EPROM process that provides exceptional performance with low power. Like any high-performance CMOS process, systems must be designed with care to obtain maximum performance with minimum problems.

Operating Conditions

Operation of Altera EPLDs at conditions above those listed under "Absolute Maximum Ratings" in the EPLD data sheets may cause permanent damage to the devices. These ratings are stress ratings only. Functional operation of the device at these conditions or at any other conditions above those indicated in the operational sections of these data sheets is not implied. Exposure to absolute maximum ratings conditions for extended periods of time may affect device reliability. Altera EPLDs contain circuitry to protect device pins from high-static voltages or electric fields; however, precautions should be taken to avoid voltages higher than maximum-rated voltages.

For proper operation, input and output pins must be in the range GND < (V_{IN} or V_{OUT}) < V_{CC}. Unused inputs must be tied to V_{CC} or GND. Unused I/O pins should be tied to V_{CC} or GND, or left unconnected ("reserved"). Specific requirements are given in the EPLD pin-out in the Report File (utilization report) for a design. Each set of V_{CC} and GND pins must be connected directly at the device, with power supply decoupling capacitors of at least 0.2 µF connected between them. For effective decoupling, each V_{CC} pin should be separately decoupled to GND, directly at the device. Decoupling capacitors should have good frequency response, such as the response in monolithic-ceramic types.

Noise Precautions

If more than 12 EPLD output pins are switching simultaneously, precautions must be taken to minimize system noise. Certain board layouts can induce switching noise into the system from high-speed devices due to transmission-line effects and radiated coupling. These effects can be minimized by using printed circuit boards with embedded V_{CC} and GND planes. They can also be lessened by restricting trace length in a board to under eight inches. In cases where long board traces or highly capacitive loads are impossible to avoid, a small series resistance (10 to 30 Ω) usually lessens undershoot and overshoot voltages if they cause a problem in a particular printed circuit board layout.

Turbo Bit

Some EP-series EPLDs contain a programmable Turbo Bit, set with A+PLUS software, to control the automatic power-down feature that enables low-standby-power mode. When the Turbo Bit is programmed (Turbo = On),

the low standby power mode (I_{CC1}) is disabled, making the circuit less sensitive to V_{CC} noise transients created by the low-power mode power-up/power-down cycle. Typical I_{CC} versus frequency data for both turbo and non-turbo mode is given in each EPLD data sheet. All AC values are tested with the Turbo Bit programmed on.

If the design requires low-power operation, the Turbo Bit should be disabled (Turbo = Off). In this mode, some AC parameters may increase. To determine worst-case timing, values from the AC Non-Turbo Adder specifications in the EPLD data sheet must be added to the corresponding AC parameter.

Device Erasure

Altera EPLDs begin to erase when exposed to lights with wavelengths shorter than 4,000 Å. Since fluorescent lighting and sunlight fall into this range, opaque labels should be placed over the EPLD window to ensure long-term reliability. The recommended erasure procedure for EPLDs is exposure to UV light with a wavelength of 2,537 Å. Required erasure times assuming use of a lamp with a 12,000 $\mu W/cm^2$ power rating are given in the table below; some low-power erasers may take longer.

Part Number	Erasure Time
EP320, EP610, EP910, EP1810, EPS448	30 minutes
EP640, EPB2001, EPM5016, EPM5032, EPM5064, EPM5128, EPM5130, EPM5192	1 hour
EP330, EP630, EP1830	2 hours

Altera EPLDs may be damaged by long-term exposure to high-intensity UV light. Altera EPLDs may be erased and reprogrammed as often as necessary if the recommended erasure exposure levels are used.

ESD and Latch-Up Protection

EPLD input, I/O, and clock pins have been designed to resist the electrostatic discharge (ESD) and latch-up inherent in CMOS structures. Unless otherwise noted, each of the EPLD pins will withstand voltage energy levels exceeding 1,500 V, per method specified by MIL-STD-883C. The pins will not latch up for input voltages in the range $V_{SS} - 1$ V to $V_{CC} + 1$ V with currents up to 100 mA. During transitions, the inputs may undershoot to –2.0 V for periods less than 20 ns. Additionally, the programming pin is designed to resist latch-up to the 13.5 V maximum device limit.

Power Calculations

As with any CMOS device, power is a function of frequency and internal node switching. To obtain the most accurate power information, current consumption should be measured after the design is completed and the EPLD is placed in the system.

Conclusion

If the precautions given in this data sheet are followed during system and board design, Altera EPLDs should provide superior system performance and design flexibility, regardless of design size or production volume.

ALTERA

Contents

October 1990

Section 8 Development Products

8

PLDS-ENCORE

Complete Programmable Logic Development System

October 1990, ver. 1

Data Sheet

Contents

General Description

PLDS-ENCORE is Altera's most comprehensive EPLD development package. It supports design entry, logic optimization, and design verification for all Multiple Array MatriX (MAX), EP-series, and Stand-Alone Microsequencer (SAM) EPLDs.

MAX EPLD designs are implemented with PLS-MAX—MAX+PLUS Programmable Logic Software. Designs can be entered with any combination of hierarchical schematic files, and hierarchical text files

8

containing Boolean equations, state machines, and truth tables in the Altera Hardware Description Language (AHDL). Over 300 7400-series and special-purpose macrofunctions are available for design entry. MAX+PLUS also includes a fast and efficient design compiler, automatic error location, delay prediction, interactive timing simulation, timing analysis, and device programming applications.

EP-series EPLD designs are implemented with PLS-SUPREME (Enhanced A+PLUS Programmable Logic Software). PLS-SUPREME supports schematic capture, Boolean equation, state machine, truth table, and netlist design entry methods. It includes LogiCaps schematic capture, TTL MacroFunction Library, Altera Design Librarian (ADLIB), State Machine Entry, Altera Design Processor, Functional Simulator (FSIM), and LogicMap II software.

SAM EPLD designs are implemented with PLS-SAM (SAM+PLUS Programmable Logic Software). SAM+PLUS includes state machine and microcode design entry, the SAM Design Processor, and the SAMSIM Functional Simulator, providing an efficient logic development system for SAM EPLDs.

The PLDS-ENCORE Development System includes all necessary hardware—a Logic Programmer card, Master Programming Unit, and a range of programming adapters—to program EPLDs at the designer's desktop.

PLDS-ENCORE also includes PLAESW-PC, a 12-month renewable warranty that covers all PC-based Altera software, including PLS-MAX, PLS-SAM, and PLS-SUPREME. It provides automatic upgrades to each new version of Altera software and guarantees software support for new EPLDs as they are introduced. PLAESW-PC also provides a toll-free hotline and 24-hour modem interface to Altera's Electronic Bulletin Board Service.

Individual PLDS-ENCORE components can be purchased separately. However, PLDS-ENCORE provides a full range of EPLD logic development support at a significant savings compared with the cost of purchasing each individual software application separately.

See the individual data sheets for PLDS-ENCORE components (in this data book) for additional information on Altera software and hardware.

Ordering Information

PLDS-ENCORE	(for IBM PC-AT and compatibles)
PLDS-ENCORE/PS	(for IBM PS/2 Models 50, 60, 70, 80, and compatibles)

PLDS-MAX

MAX+PLUS Programmable Logic Development System

October 1990, ver. 1

Data Sheet

Contents

- PLS-MAX—MAX+PLUS Programmable Logic Software
- Software-controlled Logic Programmer interface card
- PLE3-12A—EPLD Master Programming Unit
- Programming Adapters:
  - PLED5016 DIP
  - PLED5032 DIP
  - PLEJ5064 J-lead
  - PLEJ5128 J-lead
- Sample EPLDs for evaluation
- PLAESW-PC—12-Month Software Warranty and Update Service

General Description

The Altera PLDS-MAX Development System is a unified CAE tool kit for implementing designs in the MAX (Multiple Array MatriX) family of EPLDs. PLDS-MAX, which includes MAX+PLUS software, provides a comprehensive range of design entry, design processing, timing simulation, and device programming capabilities.

PLDS-MAX allows MAX designs to be completed rapidly and efficiently. Designs can be entered with any combination of hierarchical schematic files created with the MAX+PLUS Graphic Editor, and hierarchical text files containing Boolean equations, state machines, or truth tables in the Altera Hardware Description Language (AHDL). The MAX+PLUS Compiler minimizes and synthesizes the design logic, and fits the design

into a targeted MAX EPLD. Processing is completed within minutes. The MAX+PLUS Simulator provides full interactive timing simulation. To simplify design verification, simulation inputs can be edited graphically in the MAX+PLUS Waveform Editor. (For more information on MAX+PLUS software, refer to *PLS-MAX: MAX+PLUS Programmable Logic Software Data Sheet*.)

PLDS-MAX hardware consists of a Master Programming Unit, Logic Programmer card, programming adapters, and a variety of device samples. The MAX+PLUS Programmer uses this hardware and the programming file created by the Compiler to translate design outputs into working MAX EPLDs.

PLAESW-PC, a 12-month renewable warranty that covers all PC-based Altera software, is included in PLDS-MAX. It provides automatic upgrades to each new version of Altera software and guarantees software support for new MAX EPLDs as they are introduced. PLAESW-PC also provides a toll-free hotline and 24-hour modem interface to Altera's Electronic Bulletin Board Service. (See *PLAESW-PC: Extended Software Warranty Data Sheet* for further details.)

The optional PLS-EDIF package provides an interface between MAX+PLUS and third-party CAE systems. For customers who already own Altera programming hardware, PLS-MAX (MAX+PLUS Programmable Logic Software) is available as a software-only enhancement to their current system.

Ordering Information

PLDS-MAX	(for IBM PC-AT and compatibles)
PLDS-MAX/PS	(for IBM PS/2 Models 50, 60, 70, 80, and compatibles)

PLCAD-SUPREME

Enhanced A+PLUS Programmable Logic Development System

October 1990, ver. 1

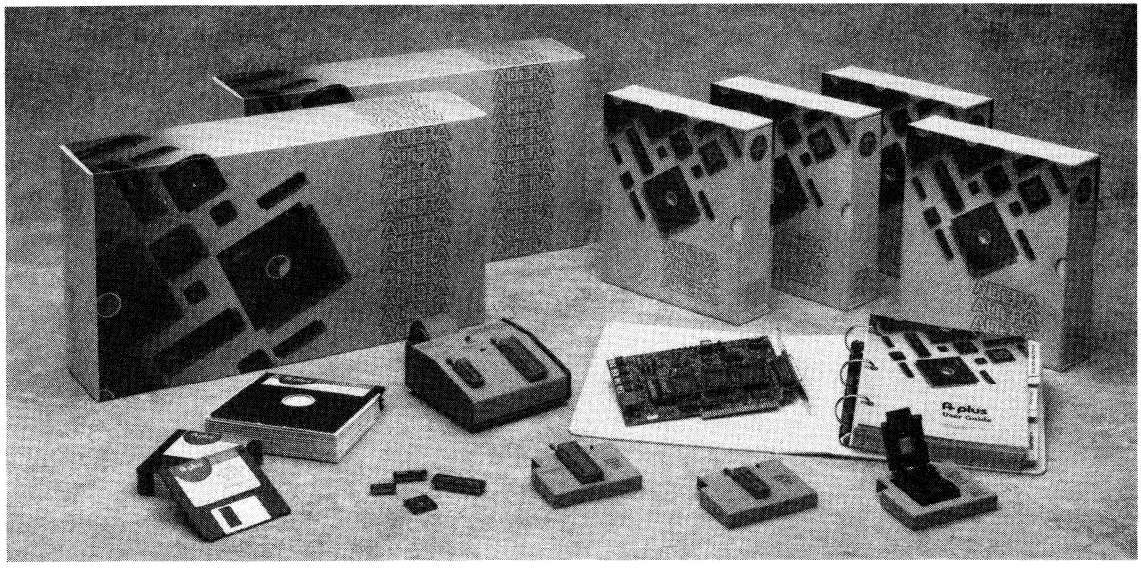
Data Sheet

Contents

- PLS-SUPREME—Enhanced A+PLUS Programmable Logic Software and Documentation
  - A+PLUS Programmable Logic Software
    - Altera Design Processor (ADP)
    - LogicMap II device programming software
  - LogiCaps schematic capture software
  - TTL MacroFunction Library
  - Altera Design Librarian (ADLIB) software
  - State Machine Entry software
  - Functional Simulator (FSIM) software
- Software-controlled Logic Programmer interface card
- PLE3-12A—EPLD Master Programming Unit
- PLED610 DIP programming adapter
- PLED910 DIP programming adapter
- PLEJ1810 J-lead programming adapter
- Sample EPLDs: EP320DC, EP610DC, EP910DC, EP1810JC
- PLAESW-PC—12-Month Software Warranty and Update Service

General Description

PLCAD-SUPREME provides basic A+PLUS software, additional software applications for design entry and design verification, and programming hardware that supports all Altera general-purpose EP-series EPLDs.



8

PLCAD-SUPREME offers four different methods of design entry: LogiCaps schematic capture, state machine, Boolean equation, and netlist. Designers can use over 100 functions from the A+PLUS TTL MacroFunction Library, and design their own macrofunctions with the Altera Design Librarian (ADLIB). The Altera Design Processor optimizes the design and generates a JEDEC file for EPLD programming. The Functional Simulator provides a convenient method for testing the logical operation of the compiled design. PLCAD-SUPREME also includes LogicMap II device programming software, a Master Programming Unit, a Logic Programmer card, and a variety of device samples and programming adapters. (For more information on A+PLUS software and enhancements, refer to *PLS-SUPREME: Enhanced A+PLUS Programmable Logic Software Data Sheet.*)

PLAESW-PC, a 12-month renewable warranty that covers all PC-based Altera software, is included in PLS-SUPREME. It provides automatic upgrades to each new version of Altera software and guarantees software support for new EPLDs as they are introduced. PLAESW-PC also provides a toll-free hotline and 24-hour modem interface to Altera's Electronic Bulletin Board Service. (See *PLAESW-PC: Extended Software Warranty Data Sheet* for further details.)

PLCAD-SUPREME provides a complete logic design capability for the full range of Altera EP-series EPLDs at a significant savings compared with the cost of purchasing the PLDS2 and PLS-SUPREME packages separately. For customers who already own Altera programming hardware, PLS-SUPREME (Enhanced A+PLUS Programmable Logic Software) is available as a software-only package.

Ordering Information

PLCAD-SUPREME	(for IBM PC-AT and compatibles)
PLCAD-SUPREME/PS	(for IBM PS/2 Models 50, 60, 70, 80, and compatibles)

PLDS2

Basic A+PLUS Programmable Logic Development System

October 1990, ver. 1

Data Sheet

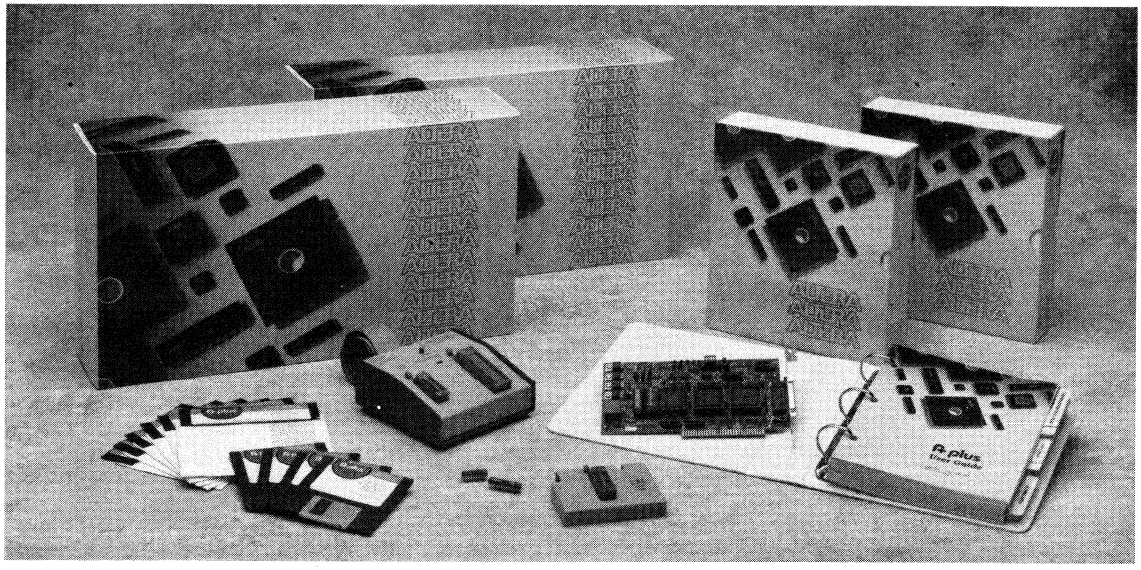
Contents

- ❒ A+PLUS Programmable Logic Software and Documentation
  - Altera Design Processor (ADP)
  - LogicMap II device programming software
- ❒ Software-controlled Logic Programmer interface card
- ❒ PLE3-12A—EPLD Master Programming Unit
- ❒ PLED610 DIP programming adapter
- ❒ EP320DC and EP610DC sample EPLDs
- ❒ PLAESW-PC—12-Month Software Warranty and Update Service

General Description

The Altera PLDS2 package is a Programmable Logic Development System for developing and implementing custom logic circuits with EP-series EPLDs. PLDS2 includes basic A+PLUS software, which supports netlist and Boolean equation design input methods. The Altera Design Processor (ADP) optimizes the input logic design and transforms it into an industry-standard JEDEC file used to program an EPLD.

The ADP performs logic minimization, automatic EPLD part selection, architecture optimization, and design fitting. The PLDS2 hardware includes an Altera Logic Programmer card—controlled by the LogicMap II software—and a Master Programming Unit used for device programming. The programming card fits into a computer expansion slot and connects



8

via ribbon cable to the PLE3-12A Master Programming Unit. The PLE3-12A unit programs Altera 20-pin EP300-series DIP devices directly; the PLED610 adapter included in the PLDS2 system programs EP600-series DIP devices.

PLDS2 also includes PLAESW-PC, a 12-month renewable warranty that covers all PC-based Altera software, including A+PLUS and LogicMap II. It provides automatic upgrades to each new version of Altera software and guarantees software support for new EPLDs as they are introduced. PLAESW-PC also provides a toll-free hotline and 24-hour modem interface to Altera's Electronic Bulletin Board Service. (See *PLAESW-PC: Extended Software Warranty Data Sheet* for further details.)

The LogiCaps schematic capture, TTL MacroFunction Library, Altera Design Librarian (ADLIB), State Machine Entry, and Functional Simulator (FSIM) enhancements to A+PLUS software are available in the PLCAD-SUPREME package, or the software-only PLS-SUPREME package.

Ordering Information

PLDS2	(for IBM PC-AT and compatibles)
PLDS2/PS	(for IBM PS/2 Models 50, 60, 70, 80, and compatibles)

PLDS-SAM

SAM+PLUS Programmable Logic Development System

October 1990, ver. 1

Data Sheet

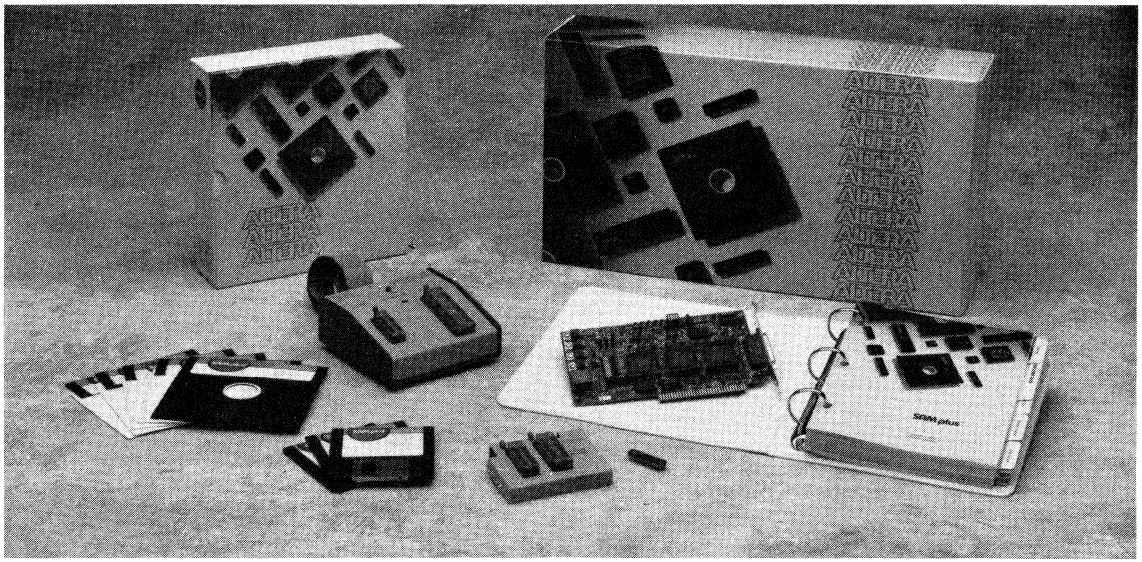
Contents

- PLS-SAM—SAM+PLUS Programmable Logic Software and Documentation
  - SAM Design Processor
  - SAMSIM Functional Simulator
  - LogicMap II device programming software
- Software-controlled Logic Programmer interface card
- PLE3-12A—EPLD Master Programming Unit
- PLED448 DIP programming adapter
- EPS448DC sample EPLD

General Description

Altera's PLDS-SAM Programmable Logic Development System provides a complete software and hardware solution for implementing state machine and microcode sequencer applications in the function-specific EPS448 Stand-Alone Microsequencer (SAM) EPLDs. PLDS-SAM is a comprehensive, easy-to-use system that includes design entry with SAM+PLUS, design debugging with SAMSIM, and device programming with LogicMap II software and standard Altera programming hardware.

The SAM+PLUS Design Processor accepts two forms of design entry, state machine and assembly language, and automatically generates an industry-standard JEDEC file for device programming. SAMSIM is an interactive



8

functional simulator for verifying state machine and microcoded designs implemented with SAM+PLUS. The PLDS-SAM programming hardware consists of a software-controlled Logic Programmer card, a Master Programming Unit, and an adapter for programming SAM EPLDs. (For more information on SAM+PLUS software, refer to *PLS-SAM: SAM+PLUS Programmable Logic Software Data Sheet* in this data book.)

PLDS-SAM also includes PLAESW-PC, a 12-month renewable warranty that covers all PC-based Altera software, including PLS-SAM. It provides automatic upgrades to each new version of Altera software and guarantees software support for new EPLDs as they are introduced. PLAESW-PC also includes a toll-free hotline and 24-hour modem interface to Altera's Electronic Bulletin Board Service. (See *PLAESW-PC: Extended Software Warranty Data Sheet* for further details.)

For customers who already own Altera programming hardware, SAM+PLUS Programmable Logic Software (PLS-SAM) is available as a software-only package.

Ordering Information

PLDS-SAM	(for IBM PC-AT and compatibles)
PLDS-SAM/PS	(for IBM PS/2 Models 50, 60, 70, 80, and compatibles)

PLDS-MCMAP

MCMap Programmable Logic Development System

October 1990, ver. 1

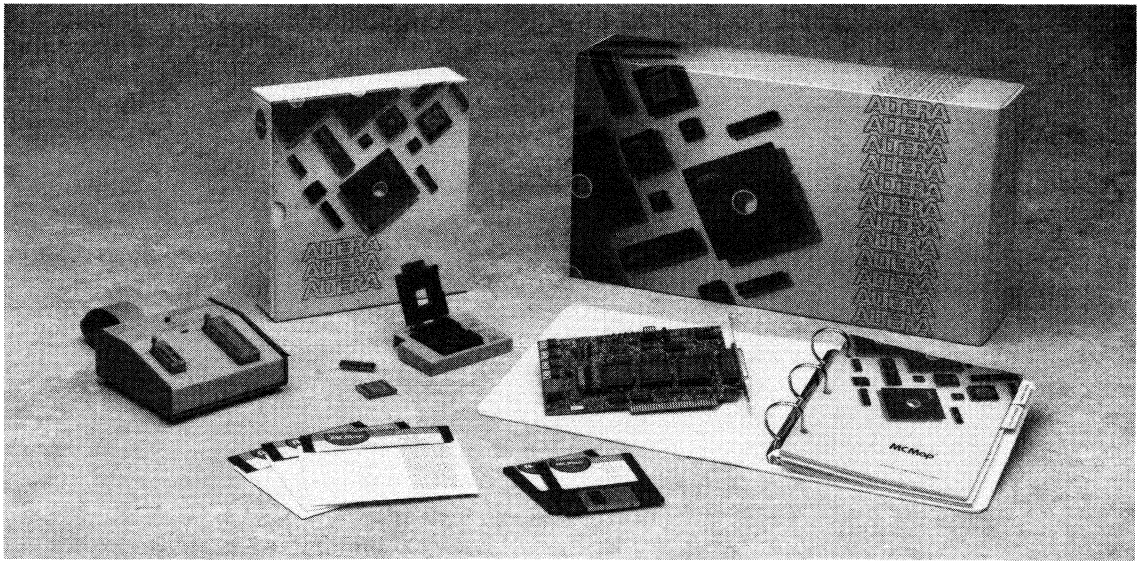
Data Sheet

Contents

- PLS-MCKIT—MCMap Programmable Logic Software and Documentation
  - MCMap design entry and processing software
  - LogicMap II device programming software
- Software-controlled Logic Programmer interface card
- PLE3-12A—EPLD Master Programming Unit
- PLEJ2001 J-lead programming adapter
- EPB2001JC and EPB2002PC sample EPLDs

General Description

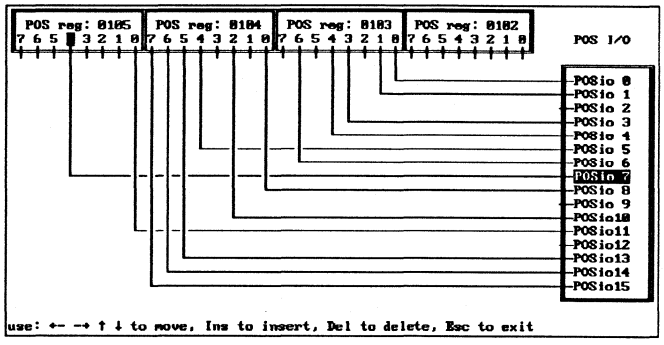
The Altera PLDS-MCMAP Programmable Logic Development System provides design entry, design processing, and device programming support for the EPB2001 EPLD. The EPB2001 provides—in a single chip—all essential interface functions required between a PS/2 add-on card and the IBM PS/2 Micro Channel bus.



MCMap software features interactive, table-driven design entry with real-time error checking, and automatically generated utilization reports. During design entry, MCMap software prompts the designer for information on the programmable portions of the design: board ID, chip-select ranges, POS register control for address remapping, and POS I/O crosspoint configuration. (See Figure 1.)

8

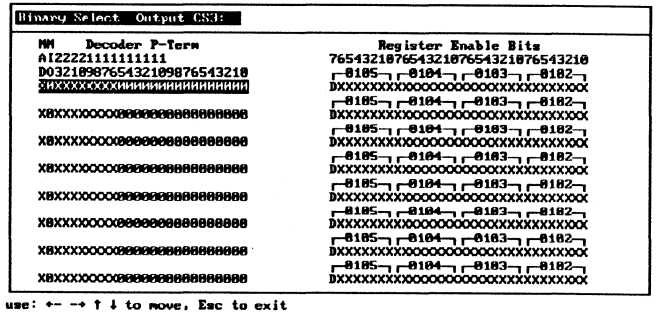
Figure 1. POS I/O Connections



Address decode chip-select ranges may be entered in either I/O, memory, or binary format (see Figure 2).

MCMap software automatically checks the validity of the design information and makes any necessary corrections to comply with the hardware in the EPB2001. The MCMap Compiler then transforms the design inputs into a JEDEC programming file. LogicMap II device programming software and standard Altera programming hardware—a Master Programming Unit, Logic Programmer card, and programming adapter—are included to program EPB2001 devices with design information.

Figure 2. Chip-Select Binary Decoder Outputs



PLDS-MCMAP also includes PLAESW-PC, a 12-month renewable warranty that provides automatic upgrades to each new version of MCMap software. (See *PLAESW-PC: Extended Software Warranty Data Sheet* for further details.)

For customers who already own Altera programming hardware, PLS-MCKIT (MCMap Programmable Logic Software) is available as a software-only enhancement to their current system.

Ordering Information

PLDS-MCMAP (for IBM PC-AT and compatibles)

(Contact Altera Marketing for information on systems containing PS/2-compatible hardware.)

Software Utility Programs

October 1990, ver. 2

Application Brief 73

Introduction

Altera provides a variety of software utility programs that complement the MAX+PLUS, A+PLUS, SAM+PLUS, and MCMap development systems.

All programs are available via Altera's Electronic Bulletin Board Service (BBS) from the EAU (Electronic Application Utilities) directory. The BBS telephone number is (408) 249-1100; operation of the BBS is described in this data book and Altera software manuals. These utility programs can also be obtained by contacting Altera Applications at 1 (800) 800-EPLD. Customers outside North America can obtain copies of these programs from their local Altera representative or distributor. All utility programs operate on an IBM PC-AT or compatible, and on IBM PS/2 Model 50 or higher computers with DOS version 3.1 or higher.

PAL2EPLD

(EAU002) The PAL2EPLD utility converts 20-pin PAL designs into EP320 or EP330 designs. It directly converts PAL JEDEC files into EP320/EP330 JEDEC files.

310-to-320/30 Converter

(EAU003) The EP310-to-EP320/EP330 JEDEC File Converter automatically converts EP310 JEDEC files to EP320/EP330-compatible JEDEC files.

LogiCaps Plotter Interface

(EAU004) An Altera customer has written an interface program between LogiCaps and Houston Instruments plotters. This EAU provides information on how to obtain the interface program.

JEDPACK

(EAU005) The JEDPACK utility compacts the size of JEDEC files, freeing up space on the computer's hard disk while retaining EPLD programming information. This utility is handy for archiving A+PLUS-, SAM+PLUS-, and MCMap-generated JEDEC files.

Address Decoder

(EAU006) The DECODER utility automatically generates Boolean equations for address decoding applications. The program accepts a user-specified address bus width with upper and lower address bounds. It generates equations that can be placed into a MAX+PLUS-compatible Text Design File (TDF) or an A+PLUS-compatible Altera Design File (ADF).

JEDSUM

(EAU007) The JEDSUM utility calculates the EPROM data checksum, the file transmission checksum, and the number of programmed architecture

bits contained in the JEDEC file for an EP-series, EPB-series, or SAM EPLD. The EPROM data checksum is often useful for documenting programming files.

AVEC

(EAU008) The AVEC utility adds functional test vectors to EP-series EPLD JEDEC files. AVEC translates the table output files generated by the A+PLUS Functional Simulator into JEDEC-standard test vectors. Third-party programmers (e.g., Data I/O 29B and UniSite 40 machines) have built-in hardware drivers that can apply these vectors to the programmed EPLD. Note, however, that Altera EPLDs are 100% generically tested before they leave the factory, so post-programming functional testing is not required.

BACKPIN

(EAU009) The BACKPIN utility extracts the pin assignments—assigned during design fitting—contained in an A+PLUS-generated JEDEC file and places them into the corresponding LogiCaps schematic drawing. If the Altera Design Processor (ADP) is set up to make pin assignments automatically, BACKPIN can then place the ADP's pin assignments back into the LogiCaps schematic drawing. The same pin assignments are then retained even if additional changes are made to the circuit design.

LEF2ABEL

(EAU012) The LEF2ABEL utility translates a Logic Equation File (LEF) generated by the Altera Design Processor (ADP) to ABEL format. A+PLUS users may thus take advantage of the ADP's SALSA Minimizer to generate an optimized ABEL input file.

PAL2ADF

(EAU013) The PAL2ADF utility converts PALASM 1 or 2 files to the A+PLUS- and MAX+PLUS-compatible ADF input format.

LCA2ADF

(EAU016) The LCA2ADF utility converts LCA design files for XC2000- and XC3000-series devices into the A+PLUS- and MAX+PLUS-compatible ADF format. The new target device may be a larger-size Altera EPLD, such as the EP1810, EP1830, EPM5064, EPM5128, EPM5130, or EPM5192.

LEF2AHDL

(EAU017) The LEF2AHDL utility converts an A+PLUS-generated Logic Equation File (LEF) to a MAX+PLUS-compatible Text Design File (TDF) in the Altera Hardware Description Language (AHDL).

PLD2EQN

(EAU018) The PLD2EQN utility converts JEDEC files from a variety of 20- and 24-pin PAL and GAL devices (e.g., 16V8, 20V8, 22V10, 16L8, 16R8, 23S8) into the A+PLUS-compatible ADF and MAX+PLUS-compatible AHDL Text Design File (TDF) formats. This utility allows users to combine multiple PALs and GALs into a single Altera EPLD.

ABEL2MAX

(EAU019) The ABEL2MAX utility converts ABEL version 4.0 design files (with the extension **.TT2**) into the MAX+PLUS-compatible Text Design File (TDF) format.

PC System Requirements

October 1990, ver. 1

Data Sheet

Introduction

All PC-based Altera Programmable Logic Development Systems, Programmable Logic Software, and Software Utility Programs can be installed in IBM PS/2 Model 50 or higher, PC-AT, or compatible computers.

Minimum System Configuration

- IBM PS/2 Model 50 or higher, PC-AT, or compatible computer
- DOS version 3.1 or higher
- 640 Kbytes of RAM
- *For MAX+PLUS only:* 1 Mbyte of expanded memory with version 3.2 or higher of the Lotus/Intel/Microsoft (LIM) Expanded Memory Specification
- VGA, EGA, or Hercules Monochrome display (CGA is also supported by A+PLUS, SAM+PLUS, and MCMap)
- 20-Mbyte hard disk
- 1.2-Mbyte 5 1/4-inch or 1.44-Mbyte 3 1/2-inch floppy disk drive
- 3-button serial-port mouse or 2-button Microsoft-compatible serial-port or bus mouse (plus a serial port for a serial-port mouse.)
- Empty card slot (full length) for programming card

Recommended System Configuration

- IBM PS/2 Model 70 or higher, or 20-MHz or higher 386-based computer
- DOS version 3.3
- 640 Kbytes of RAM
- VGA graphics display
- 40-Mbyte hard disk
- 3 Mbytes Expanded Memory with LIM 3.2-compatible driver
- 1.2-Mbyte 5 1/4-inch or 1.44-Mbyte 3 1/2-inch floppy disk drive
- Serial port and 3-button serial-port mouse
- Empty card slot (full length) for programming card

Sample Configurations

- Compaq 386-20 with 3 Mbytes of RAM, the CEMM Expanded Memory Manager, Compaq VGA display, and Mouse Systems 3-button serial-port mouse
- Wyse 386-16 with Intel Above Board 286, VEGA VGA card with NEC MultiSync II monitor, and Logitech C-7 serial-port mouse
- IBM PS/2 Model 80 with 4 Mbytes of RAM, DOS version 4.01 Expanded Memory Manager, Logitech Series 9 Mouse attached to pointing device port, and EGA display
- Everex 386-25 with extended memory configured as expanded memory, Paradise VGA card, VGA display, and 2-button Microsoft-compatible bus mouse

8

PLS-EDIF

Bidirectional EDIF Netlist Interface to MAX+PLUS Software

October 1990, ver. 2

Data Sheet

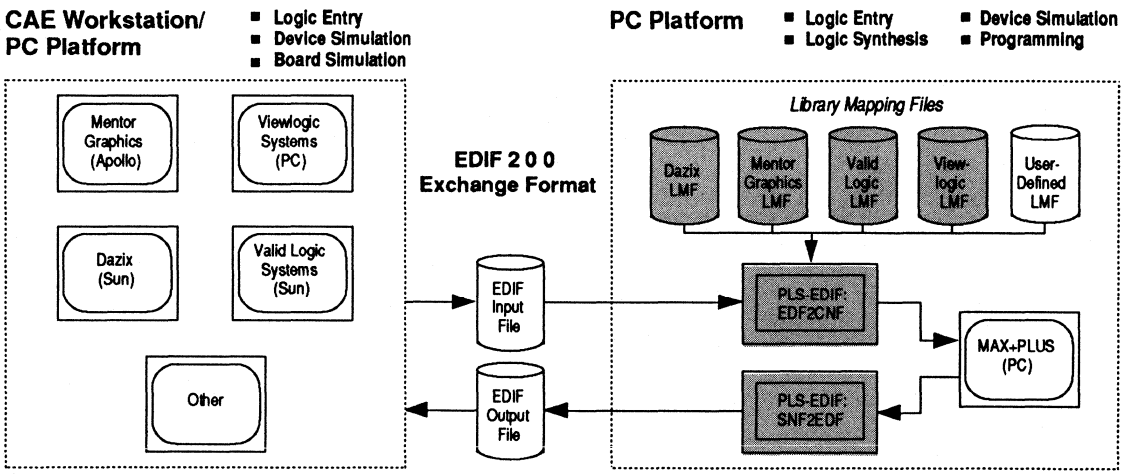
Features

- Provides a bidirectional netlist interface between MAX+PLUS and other major CAE software packages.
- Supports the industry-standard Electronic Design Interchange Format (EDIF) version 2 0 0.
- Allows MAX EPLD designs to be created with workstation CAE tools and transferred to MAX+PLUS for compilation; compiled designs can be returned to the workstation for device- or system-level simulation.
- Altera EDIF netlist reader imports EDIF netlists into MAX+PLUS.
- Altera-provided Library Mapping Files (LMFs) convert basic gate and many common TTL library functions from Dazix, Mentor Graphics, Valid Logic Systems, and Viewlogic Systems CAE tools to equivalent MAX+PLUS functions.
- Altera EDIF netlist writer produces post-synthesis logic and delay information used during device- or board-level simulation with popular CAE tools.
- Runs on IBM PS/2, PC-AT, or compatible machines.

General Description

The Altera PLS-EDIF tool kit is a bidirectional EDIF netlist interface between PC- or workstation-based CAE software packages and the Altera MAX+PLUS Programmable Logic Development System. See Figure 1.

Figure 1. PLS-EDIF Workstation Interface *Shading indicates items provided with PLS-EDIF.*



PLS-EDIF (Bidirectional EDIF Netlist Interface to MAX+PLUS Software) allows designers to enter and verify logic designs for Altera MAX EPLDs with third-party CAE tools. The EDIF 2 0 0 netlist exchange format provides a two-way bridge between MAX+PLUS and third-party schematic capture and simulation tools. PLS-EDIF runs on an IBM PS/2, PC-AT, or compatible computer.

Any CAE software package that produces EDIF 2 0 0 netlists can use the PLS-EDIF interface to MAX+PLUS. EDIF netlists are imported into MAX+PLUS with the EDIF Design File-to-Compiler Netlist File (EDF2CNF) Converter. Library Mapping Files (LMFs) are used with EDF2CNF to map library functions from third-party CAE tools to the MAX+PLUS library functions. LMFs are provided for Dazix, Mentor Graphics, Valid Logic, and Viewlogic software, but designers may create their own LMFs to map any CAE software library.

After a design is imported into MAX+PLUS, it is compiled with the sophisticated MAX+PLUS Compiler, which uses advanced logic synthesis and minimization techniques together with heuristic fitting rules to optimize the design for MAX EPLD architecture. MAX devices are then programmed with a Programmer Object File (POF) created by the MAX+PLUS Compiler and standard Altera or third-party programming hardware.

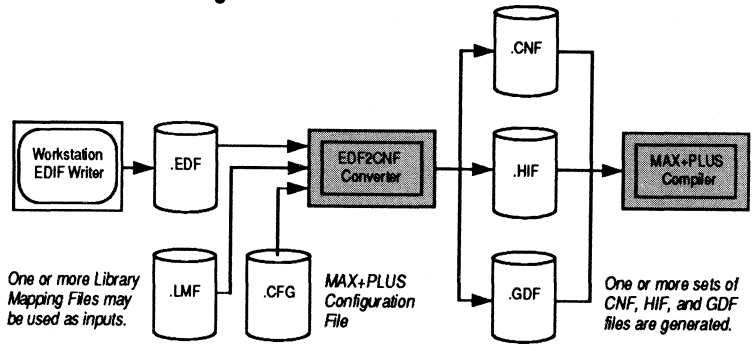
EDIF netlists can be exported from MAX+PLUS with the Simulator Netlist File-to-EDIF Design File (SNF2EDF) Converter. This converter generates an EDIF output file from a compiled MAX+PLUS design. The EDIF file contains the post-synthesis information used by CAE simulators to perform device- or board-level simulation.

PLS-EDIF provides an open environment that allows designers to use popular third-party CAE tools to create and simulate MAX EPLD designs. The designer can use a preferred workstation schematic capture package to enter a logic design, quickly convert it with EDF2CNF, and compile it with MAX+PLUS. Likewise, designs compiled in MAX+PLUS and converted with SNF2EDF can be transferred to a workstation for simulation. Together, the PLS-EDIF netlist reader and writer (EDF2CNF and SNF2EDF) allow MAX EPLD designs to be entered and simulated on the workstation platform of choice.

EDF2CNF Converter

The EDF2CNF Converter generates one or more MAX+PLUS Compiler Netlist Files (CNFs) from an EDIF file. For each CNF, a Hierarchy Interconnect File (HIF) and a Graphic Design File (GDF) are also created. See Figure 2. The CNF contains the logic and connectivity data for a design file, while the HIF defines the hierarchical connections between design files. The GDF is a token symbol that represents the actual design data in the CNF. This symbol—and the underlying logic—may be used in a logic schematic in the MAX+PLUS Graphic Editor.

Figure 2. EDF2CNF Block Diagram



EDF2CNF can convert any EDIF 2 0 0 netlist with the following characteristics:

- EDIF level **0**
- keyword level **0**
- view type **NETLIST**
- cell type **GENERIC**

EDF2CNF gives designers the flexibility to design logic solely with workstation CAE tools, or to mix design inputs from a variety of platforms and software packages. For example, a workstation CAE schematic converted with EDF2CNF may be combined with an Altera Hardware Description Language (AHDL) state machine in MAX+PLUS. Designers can choose the entry methods and platforms that best meet their needs.

Library Mapping Files (LMFs) are used with EDF2CNF to convert functions of workstation CAE tools to equivalent MAX+PLUS functions. This direct substitution is beneficial because MAX+PLUS functions are optimized for both logic utilization and performance in MAX EPLD designs.

Workstation Information

EDF2CNF has been specifically tested for use with the Dazix, Mentor Graphics, Valid Logic Systems, and Viewlogic Systems CAE software packages. LMFs for these products are also provided with the PLS-EDIF tool kit.

Dazix

To design logic and create an EDIF file with Dazix software, the following applications are required:

- ACE (Dazix graphics editor)
- DANCE and DRINK (Dazix compiler)
- ENW version 1.0 (Dazix EDIF netlist writer)

Table 1 lists the Dazix basic functions that are mapped to MAX+PLUS-compatible functions:

Table 1. Dazix Library Mapping File (Basic Functions)

Dazix Function	MAX+PLUS-Compatible Function
R#AND	AND# (#=2,3,4,5,6,7,8,9)
R#ANDD	BNOR# (#=2,3,4,5,6,7,8,9)
R#NAND	NAND# (#=2,3,4,6,7,8,9,13)
R#NANDD	BOR# (#=2,3,4,5,7,8,9,13)
R#NOR	NOR# (#=2,3,4,5)
R#NORD	BAND# (#=2,3)
R#OR	OR# (#=2,3,4,5)
R#ORD	BNAND# (#=2,3,4,5)
R1BUF	MCELL
R1INV	NOT
R1INVD	EXP
R1OCBUF	SCLK
R1OTBUF	TRIBUF
R1TINV	TRINOT
R2XNOR	XNOR
R2XOR	XOR
R3UAOI	1A2NOR2
R4AOI	2A2NOR2
R4OAI	2OR2NA2
R8AOI	4A2NOR4
R13TNAND	TNAND13
R13TNANDD	TBOR13
RDFLOP	DFF2
RDLATCH	RDLATCH
RJKFLOP	JKFF2

8

Mentor Graphics

To design logic and create an EDIF file with Mentor Graphics software, the following applications are required:

- NETED (Mentor Graphics graphics editor)
- EXPAND (Mentor Graphics schematic file translator)
- EDIFNET version 7.0 (Mentor Graphics EDIF netlist writer)

Table 2 lists the Mentor Graphics basic functions that are mapped to MAX+PLUS-compatible functions:

Table 2. Mentor Graphics Library Mapping File (Basic Functions)

Mentor Graphics Function	MAX+PLUS-Compatible Function	
AND#	AND#	(#=2,3,4,5,6)
BUF	SCLK	
DELAY	MCELL	
DFF	DFF2	
INV	NOT	
JKFF	JKFF2	
LATCH	MLATCH	
NAND	NAND#	(#=2,3,4,5 6,9)
NOR	NOR#	(#=2,3,4,6,8,16)
OR	OR#	(#=2,3,4,6,8)
XNOR2	XNOR	
XOR2	XOR	

Valid Logic Systems

To design logic and create an EDIF file with Valid Logic Systems software, the following applications are required:

- ValidGED (Valid Logic Systems graphics editor)
- ValidCompiler (Valid Logic Systems compiler)
- GEDIFNET (Valid Logic Systems EDIF netlist writer)

Table 3 lists the Valid Logic Systems basic functions that are mapped to MAX+PLUS-compatible functions:

Table 3. Valid Logic Systems Library Mapping File (Basic Functions)

Valid Logic Systems Function	MAX+PLUS-Compatible Function
INV	EXP
LS00	NAND2
LS02	NOR2
LS04	NOT
LS08	AND2
LS10	NAND3
LS11	AND3
LS20	NAND4
LS21	AND4
LS27	NOR3
LS28	NOR2
LS30	NAND8
LS32	OR2
LS37	NAND2
LS40	NAND4
LS74	DFF2
LS86	XOR
LS126	TRI
LS280	DFF2
LS386	XOR

8

Viewlogic Systems

To design logic and create an EDIF file with Viewlogic Systems software, the following applications are required:

- WorkView (Viewlogic Systems graphics editor)
- EDIFNET2 version 3.02 (Viewlogic Systems EDIF netlist writer)

Table 4 lists the Viewlogic Systems basic functions that are mapped to MAX+PLUS-compatible functions:

Table 4. Viewlogic Systems Library Mapping File (Basic Functions)

Viewlogic Systems Function	MAX+PLUS-Compatible Function	
AND#	AND#	(#=2,3,4,8)
ANDNOR22	2A2NOR2	
BUF	SOFT	
DAND#	DAND#	(#=2,3,4,8)
DELAY	MCELL	
DOR#	DOR#	(#=2,3,4,8)
DXOR#	DXOR#	(#=2,3,4,8)
JKFFRE	JKFFRE	
MUX41	MUX41	
NAND#	NAND#	(#=2,3,4,8)
NOR#	NOR#	(#=2,3,4,8)
NOT	NOT	
OR#	OR#	(#=2,3,4,8)
TRIAND#	TAND#	(#=2,3,4,8)
TRIBUF	TRIBUF	
TRINAND#	TNAND#	(#=2,3,4,8)
TRINOR#	TNOR#	(#=2,3,4,8)
TRINOT	TRINOT	
TRIOR#	TOR#	(#=2,3,4,8)
UBDEC38	DEC38	
UDFDL	UDFDL	
UJKFF	UJKFF	
XNOR2	XNOR	
XNOR#	XNOR#	(#=3,4,8)
XOR2	XOR	
XOR#	XOR#	(#=3,4,8)

LMF Support for TTL Macrofunctions

In addition to mapping the basic functions listed above, Altera-provided LMFs map various TTL macrofunctions from Dazix, Mentor Graphics, Valid Logic Systems, and Viewlogic Systems to their MAX+PLUS-compatible equivalents. See Table 5.

Table 5. TTL Function Mappings in Altera-Provided LMFs

MAX+PLUS	Dazix	Mentor Graphics	Valid Logic Systems	Viewlogic Systems
7442	LS42	74LS42	LS42	74LS42
DFF2	LS74	74LS74A	LS74	74LS74A
7483	LS83	74LS83A	LS83	74LS83A
7485	LS85	74LS85	LS85	74LS85
7491	LS91	74LS91	LS91	74LS91
7493	LS93	74LS93	LS93	74LS93
74138	LS138	74LS138	LS138	74LS138
74139	LS139	—	—	—
74139M	—	74LS139A	LS139	74LS139
74151	LS151	74LS151	LS151	74LS151
74153	—	74LS153	—	74LS153
74153M	LS153	—	LS153	—
74157	LS157	74LS157	—	74LS157
74157M	—	—	—	LS157
74160	LS160	74LS160A	LS160	74LS160A
74161	LS161	74LS161A	LS161	74LS161A
74162	LS162	74LS162A	LS162	74LS162A
74163	LS163	74LS163A	LS163	74LS163A
74164	LS164	74LS164	LS164	74LS164
74165	LS165	74LS165	LS165	74LS165
74174	LS174	74LS174	—	74LS174
74174M	—	—	LS174	—
74181	LS181	74LS181	LS181	74LS181
74190	LS190	74LS190	LS190	74LS190
74191	LS191	74LS191	LS191	74LS191
74194	LS194	74LS194A	LS194A	74LS194A
74273	LS273	74LS273	—	74LS273
74174M	—	—	LS273	—
74279MD	LS279	—	—	—
74279M		74LS279	LS279	74LS279
74280	LS280	74LS280	LS280	74LS280
74373	LS373	74LS373	—	74LS373
74373M	—	—	LS373	—
74374	LS374	74LS374	—	74LS374
74374M	—	—	LS374	—
74393M	LS393	74LS393	LS393	74LS393

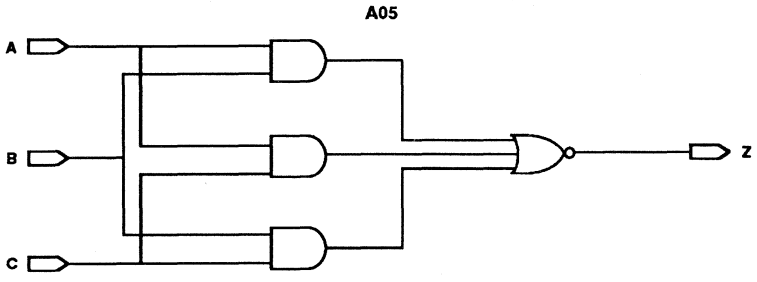
8

Custom Library Mapping Files

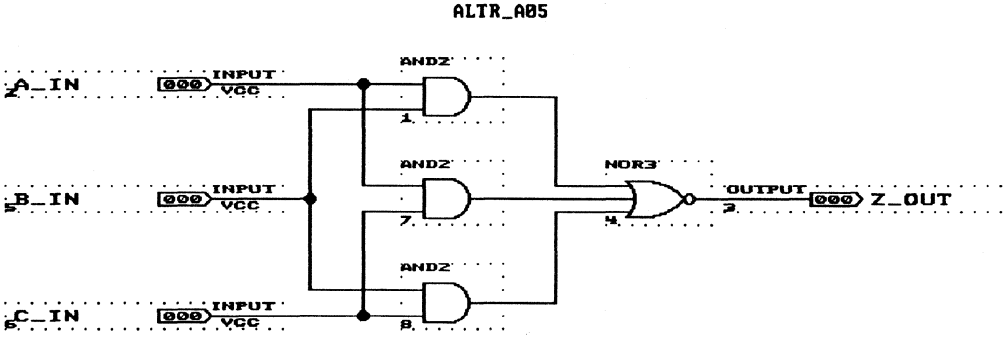
Designers can map their commonly used workstation functions to MAX+PLUS-compatible equivalents by modifying an LMF or creating a new one. If no equivalent function currently exists in MAX+PLUS, the designer can create the function with the MAX+PLUS Graphic Editor or Text Editor and then map it in an LMF. Figure 3 demonstrates this process.

Figure 3. Creating a Library Mapping File

Step 1: Select a workstation function for mapping.



Step 2: Design an equivalent circuit with the MAX+PLUS Graphic Editor.



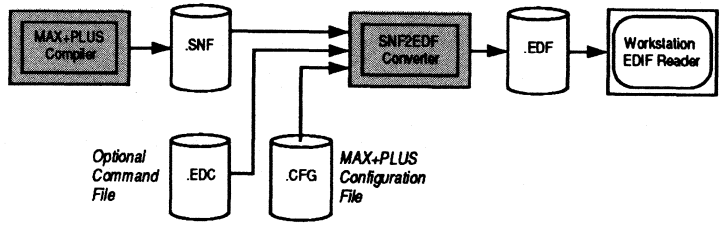
Step 3: Map the workstation function to the MAX+PLUS function in an LMF.

```
LIBRARY new_lib

%User Library Mapping File%

BEGIN
FUNCTION ALTR_A05 (A_IN, B_IN, C_IN)
RETURNS (Z_OUT)
FUNCTION "A05" ("A", "B", "C")
RETURNS ("Z")
END
```

SNF2EDF Converter

The SNF2EDF Converter creates an industry-standard level 0 EDIF file from a MAX+PLUS Simulator Netlist File (SNF). The SNF, which is optionally generated during compilation of a MAX EPLD design, contains all post-synthesis functional and delay information for the completed design. This design-specific information is also contained in the EDIF output file after conversion, so it can be integrated into a workstation environment for simulation. The user can customize the output EDIF file for various workstation environments with an optional command file that renames certain constructs, or changes the EDIF level or keyword level. See Figure 4.

Figure 4. SNF2EDF Block Diagram



The EDIF output file may have one of two formats. The first format expresses all delays with special EDIF property constructs. The second expresses combinatorial delays with port-delay constructs and registered delays as path-delay constructs—a format that is especially useful for behavioral simulators. Both formats are shown in the example in Figure 5.

Figure 5. EDIF File Formats

Format 1: Combinatorial delays expressed with property constructs

```
( instance xor2_5
   ( viewRef view1
      ( cellRef XOR2
   ( property TPD ( integer 20 ) ( unit TIME ) ) )
```

Format 2: Combinatorial delays expressed with port-delay constructs

```
( instance xor2_5
   ( viewRef view1
      ( cellRef XOR2
   ( portInstance &1
      ( portDelay
         ( derivation CALCULATED
         ( delay ( e 20 -10 ) ) ) ) ) )
```

8

System Requirements

- IBM PC-AT or compatible computers; IBM PS/2 models 50, 60, 70, 80
- DOS version 3.1 or higher
- 640 Kbytes of RAM
- 1 Mbyte of expanded memory compatible with version 3.2 or higher of the Lotus/Intel/Microsoft Expanded Memory Specification
- VGA, EGA, or Hercules Monochrome display
- 20-Mbyte hard disk drive
- 1.2-Mbyte 5 1/4-inch or 1.44-Mbyte 3 1/2-inch floppy disk drive
- MAX+PLUS version 2.01 or higher
- Workstation-to-PC network hardware and software with the ability to transfer ASCII files

PLS-EDIF Contents

- Floppy diskettes containing all PLS-EDIF programs and files for both PC-AT and PS/2 computers
  - EDF2CNF Converter
  - SNF2EDF Converter
  - Library Mapping Files for Dazix, Mentor Graphics, Valid Logic, and Viewlogic netlists
  - MAX+PLUS macrofunctions for Dazix, Mentor Graphics, Valid Logic Systems, and Viewlogic Systems libraries
  - Sample files
- Documentation

Ordering Information

PLS-EDIF (supports both PC-AT and PS/2 formats)

PLS-APOLLO

MAX+PLUS Programmable Logic Software for Apollo Computers

October1990, ver. 2 **Data Sheet**

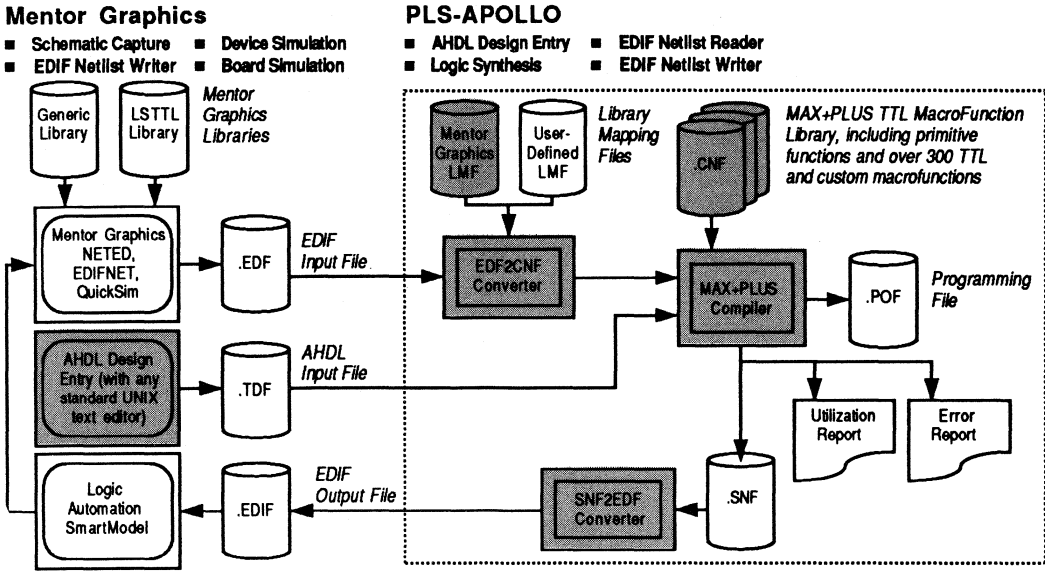
Features

- Software support for MAX (Multiple Array MatriX) EPLDs
- Runs on Apollo (Hewlett Packard) Series 3000, 3500, 4000, and 4500 computers with Domain/OS SR 10.1 operating system
- Supports hierarchical design entry for graphic and text designs
  - Schematic capture with Mentor Graphics' NETED
  - Altera Hardware Description Language (AHDL) supporting state machines, Boolean equations, truth tables, and arithmetic and relational operations
- Full Altera/Mentor Graphics cross-compatibility supplied via bidirectional EDIF 2 0 0 netlist interfaces
- MAX+PLUS Compiler provides logic synthesis and minimization for efficient device utilization
- Generates post-synthesis timing simulation data for use with Logic Automation's SmartModel and Mentor Graphics' QuickSim simulator
- Produces MAX EPLD programming files for use with an Altera PC-based programmer (PL-ASAP) or third-party programming hardware

General Description

The Altera PLS-APOLLO package brings the popular MAX+PLUS Development System to Apollo Series 3000, 3500, 4000, and 4500 computers (see Figure 1). PLS-APOLLO includes AHDL design entry, bidirectional

Figure 1. PLS-APOLLO Design Framework *Shading indicates items provided with PLS-APOLLO.*



EDIF netlist interfaces, Mentor Graphics library support, advanced logic synthesis, and design fitting in the Apollo computer environment. Together, PLS-APOLLO and Mentor Graphics software provide the tools to quickly and efficiently create, compile, and verify logic designs for MAX EPLDs.

Design Entry

PLS-APOLLO supports both schematic and textual design entry options. Hierarchical schematic designs are entered with the Mentor Graphics NETED schematic capture program. Hierarchical Text Design Files (TDFs) created in the Altera Hardware Description Language (AHDL) can be used separately or mixed with NETED schematic designs. AHDL is tailored especially for EPLD designs and supports complex Boolean and arithmetic functions, relational comparisons, multiple hierarchy levels, state machines with automatic state variable assignment, and truth tables. AHDL designs can also use over 300 primitives and TTL and custom macrofunctions from the MAX+PLUS TTL MacroFunction Library.

EDF2CNF Converter

NETED schematics are converted into EDIF 2 0 0 netlist files with the Mentor Graphics EDIFNET netlist writer. PLS-APOLLO's EDF2CNF Converter and Library Mapping File (LMF) then translate each EDIF netlist into a MAX+PLUS-compatible format. (See Figure 2.) Library symbols from the Mentor Graphics generic and LSTTL libraries are automatically mapped to corresponding primitive and TTL functions in the MAX+PLUS TTL MacroFunction Library. Table 1 shows the mappings provided in the LMF. PLS-APOLLO users can add to these libraries or build their own libraries by creating additional Logic Mapping Files. (See "Custom Library Mapping Files" later in this data sheet.)

Figure 2. EDF2CNF Block Diagram

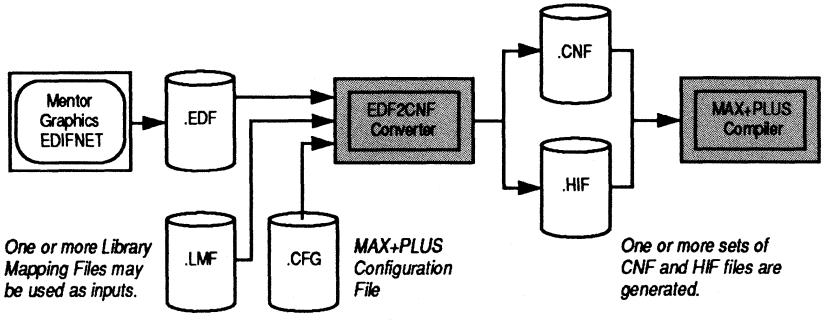


Table 1. Mentor Graphics Library Mapping File (Basic and TTL Functions)

Mentor Graphics Generic Function	MAX+PLUS Primitive Function *(1)*	
AND#	AND#	(#=2,3,4,5,6)
BUF	SCLK	
DELAY	MCELL	
DFF	DFF2	
INV	NOT	
JKFF	JKFF2	
LATCH	MLATCH	
NAND#	NAND#	(#=2,3,4,5 6,9)
NOR#	NOR#	(#=2,3,4,6,8,16)
OR#	OR#	(#=2,3,4,6,8)
XNOR2	XNOR	
XOR2	XOR	
Mentor Graphics LSTTL Function	**MAX+PLUS TTL Macrofunction** *(1)*	
74LS42	7442	
74LS74A	DFF2	
74LS83A	7483	
74LS85	7485	
74LS91	7491	
74LS93	7493	
74LS138	74138	
74LS139A	74139M	
74LS151	74151	
74LS153	74153	
74LS157	74157	
74LS160A	74160	
74LS161A	74161	
74LS162A	74162	
74LS163A	74163	
74LS164	74164	
74LS165	74165	
74LS174	74174	
74LS181	74181	
74LS190	74190	
74LS191	74191	
74LS194A	74194	
74LS273	74273	
74LS279	74279M	
74LS280	74280	
74LS373	74373	
74LS374	74374	
74LS393	74393M	

Note:

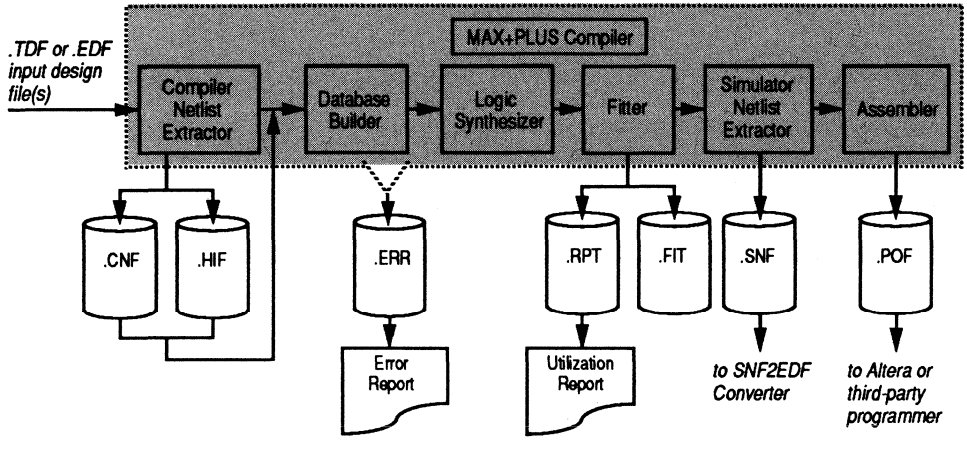
(1) Contact Altera Applications at 1 (800) 800-EPLD for the most up-to-date list of mappings.

8

MAX+PLUS Compiler

PLS-APOLLO includes the sophisticated MAX+PLUS Compiler, which synthesizes and optimizes designs for MAX EPLDs in minutes. (See Figure 3.) The Compiler uses advanced logic synthesis and minimization techniques together with heuristic fitting rules to efficiently place designs into MAX EPLDs.

Figure 3. MAX+PLUS Compiler



The Compiler offers several options to customize the processing and analysis of a design. The user can set the degree of detail of the Report File as well as the maximum number of errors to be detected before processing halts. The user can also choose whether to extract a netlist containing post-synthesis timing information.

Designs are compiled in increments, so that if a design has been compiled previously, only the new portion is extracted to save time. All errors encountered during design processing are documented in an Error File (**.ERR**).

The first module of the Compiler, the Compiler Netlist Extractor, extracts a Compiler Netlist File (**.CNF**) and a Hierarchy Interconnect File (**.HIF**) from each file. At this time, design rules are checked for any errors. The Database Builder module then converts the successfully extracted CNFs and HIFs into a database to be used by the Logic Synthesizer module.

The Logic Synthesizer translates and optimizes the user-defined logic for the MAX architecture. The design is first minimized with SALSA (Speedy Altera Logic Simplification Algorithm); unused logic is automatically removed. The Logic Synthesizer module uses expert system synthesis rules to factor and map logic within the multi-level MAX architecture, choosing the approach that ensures the most efficient use of silicon resources.

The next module, the Fitter, uses heuristic rules to optimally place the synthesized design into the chosen MAX EPLD. For MAX devices that have a Programmable Interconnect Array (PIA), the Fitter also routes the signals across the PIA, automatically handling all placement and routing issues. The Fitter issues two files: a Report File (**.RPT**) that shows how the design is implemented in the EPLD and whether any unused resources are available for additional logic; and a Fit File (**.FIT**) that preserves pin assignments for optional future use.

The Simulator Netlist Extractor module optionally creates a Simulator Netlist File (described later in this data sheet). Finally, the Assembler creates a Programmer Object File (**.POF**) from the compiled design. The MAX+PLUS Programmer uses this file with Altera or third-party hardware to program the target EPLD.

The advanced synthesis and minimization techniques used by the Compiler allow designs to be placed within the architecture in a matter of minutes. For example, a 16-bit counter/shift register compiles in less than 1 minute. The Compiler is equally efficient when compiling complex designs. For example, a series of 5 serially linked multiplier/adder circuits that uses 100% of the macrocells and 95% of all expanders in an EPM5128 takes only 10 minutes to compile.

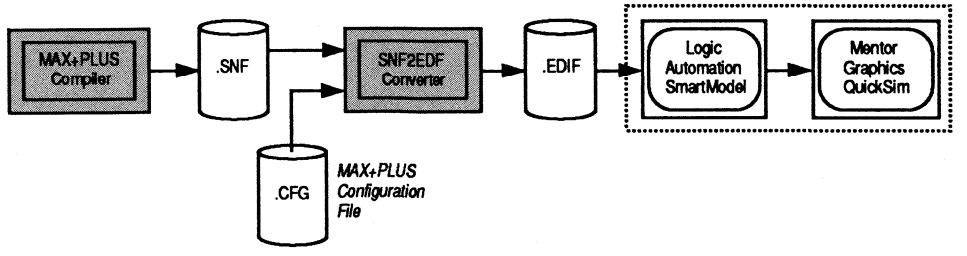
For more information on AHDL, the MAX+PLUS Compiler, and Altera programming hardware, refer to the *PLS-MAX: MAX+PLUS Programmable Logic Software* and *PL-ASAP: Altera Stand-Alone Programmer* data sheets in this data book.

SNF2EDF Converter

The Simulator Netlist Extractor of the MAX+PLUS Compiler optionally creates a Simulator Netlist File (**.SNF**) that contains post-synthesis logic and delay information for the compiled design. PLS-APOLLO's Simulator Netlist File-to-EDIF Design File (SNF2EDF) Converter can then generate an annotated EDIF 200 netlist output file with the same information. (See Figure 4.) The Logic Automation SmartModel for MAX EPLDs can convert this EDIF file into a behavioral model for use with the Mentor Graphics QuickSim simulator.

8

Figure 4. SNF2EDF Block Diagram

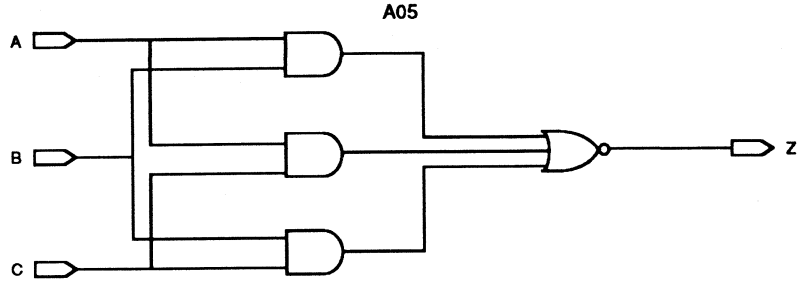


Custom Library Mapping Files

A designer can map his or her commonly used NETED functions to MAX+PLUS equivalents by creating a new LMF. If no equivalent function currently exists, the designer can create the function in an AHDL Text Design File, and then map it in an LMF. Figure 5 demonstrates this process.

Figure 5. Creating a Library Mapping File

Step 1: Select a Mentor Graphics function for mapping



Step 2: Design an equivalent circuit in the Altera Hardware Description Language (AHDL)

```
TITLE "ALTR_A05" ;
DESIGN IS "ALTR_A05" ;
SUBDESIGN ALTR_A05
(
   A_IN, B_IN, C_IN : INPUT ;
   Z_OUT : OUTPUT ;
)
VARIABLE
   X1, X2, X3 : NODE ;
BEGIN
   Z_OUT = !(X1 # X2 # X3) ;
   X1 = A_IN & B_IN ;
   X2 = A_IN & C_IN ;
   X3 = B_IN & C_IN ;
END ;
```

Step 3: Map the Mentor Graphics function to the AHDL function in an LMF

```
LIBRARY user_lib

% User Library Mapping File %

BEGIN
FUNCTION ALTR_A05 (A_IN, B_IN, C_IN)
RETURNS (Z_OUT)
FUNCTION "A05" ("A", "B", "C")
RETURNS ("Z")
END
```

System Requirements

- HP Apollo Series 3000, 3500, 4000, or 4500 computer with 5+ Mbytes of free disk space
- Domain/OS SR 10.1 operating system
- Quarter-inch cartridge (QIC-24, 9 track) 60-Mbyte tape drive
- Schematic capture and EDIF conversion software:
  - Mentor Graphics NETED version 7.0 (graphics editor)
  - Mentor Graphics EXPAND (schematic file translator)
  - Mentor Graphics EDIFNET version 7.0 (EDIF 2 0 0 netlist writer)
- EDIF conversion and simulation software:
  - Logic Automation SmartModel for MAX EPLDs
  - Mentor Graphics QuickSim version 7.0

PLS-APOLLO Contents

- Quarter-inch cartridge tape (QIC-24, 9 track) containing all PLS-APOLLO programs and files:
  - EDF2CNF Converter
  - Library Mapping Files for converting Mentor Graphics-generated EDIF 2 0 0 netlists
  - MAX+PLUS TTL MacroFunction Library
  - MAX+PLUS Compiler
  - SNF2EDF Converter
  - Sample files
- Documentation

Ordering Information

PLS-APOLLO

8

PL-ASAP

Altera Stand-Alone Programmer

October 1990, ver. 1

Data Sheet

Contents

- ❒ Software-controlled Logic Programmer interface card
- ❒ PLE3-12A—EPLD Master Programming Unit
- ❒ LogicMap II device programming software

Features

- ❒ Independent EPLD programming station
- ❒ Programming support for all Altera EPLDs
- ❒ Hardware extension to existing Altera systems
- ❒ Provides software for programming EP-series, EPB-series, and SAM devices; programming software for EPM-series MAX EPLDs is available via Altera's Electronic Bulletin Board Service
- ❒ Included in the price of PLDS-ENCORE, PLCAD-SUPREME, PLDS-SAM, PLDS-MCMAP, and PLDS2 systems

General Description

The Altera Stand-Alone Programmer, PL-ASAP, provides the hardware needed for programming Altera EPLDs. PL-ASAP is designed for customers who wish to program Altera devices at a "programming station" separate from their design station, or who require a hardware extension to existing Altera systems. It includes only the materials required for EPLD programming; software tools for design entry, processing, and verification are sold separately.

PL-ASAP includes the software-controlled LP5 or LP6 Logic Programmer card. The LP6 interfaces with IBM PC-AT or compatible computers; the LP5 interfaces with IBM PS/2 Model 50, 60, 70, 80, or compatible computers.

The PLE3-12A Master Programming Unit programs Altera 20-pin EP300-series DIP devices directly; all other Altera EPLDs can be programmed with optional plug-in adapters. The Logic Programmer card generates all programming waveforms and voltages, so the PLE3-12A requires no additional power supply.

PL-ASAP also includes LogicMap II device programming software, which uses the Altera hardware and the programming file created by A+PLUS, SAM+PLUS, and MCMap to translate design outputs into working EP-series, SAM, and EPB-series EPLDs, respectively. Software for programming EPM-series MAX EPLD designs is available from Altera's Electronic Bulletin Board Service or in the PLS-MAX software package.

Ordering Information

PL-ASAP	(for IBM PC-AT and compatibles)
PL-ASAP/PS	(for IBM PS/2 Models 50, 60, 70, 80, and compatibles)

PLE3-12A

EPLD Master Programming Unit

October 1990, ver. 1

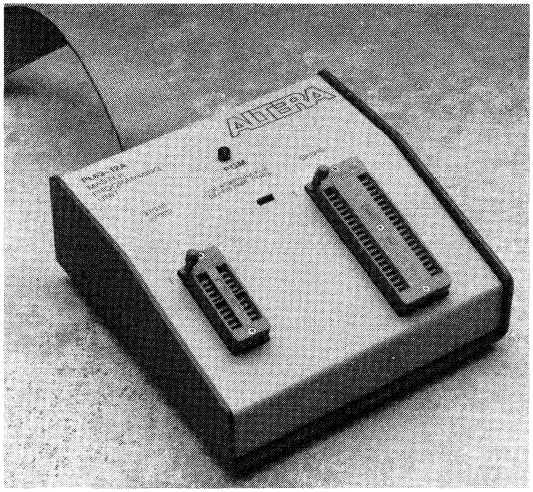
Data Sheet

Features

- Master Programming Unit for all Altera EPLDs
- Direct programming of Altera 20-pin EP310, EP320, and EP330 DIP EPLDs
- Support for all other Altera EPLDs via optional plug-in adapters
- Zero-insertion-force sockets for easy device insertion and extraction
- Indicator LED shows when unit is active
- Included in the price of PLDS-ENCORE, PLDS-MAX, PLDS-SAM, PLCAD-SUPREME, PLDS-MCMAP, PLDS2, and PL-ASAP packages

General Description

The Altera PLE3-12A Master Programming Unit is a hardware module that programs all Altera EPLDs. It contains both 20-pin (300 mil) and 40-pin (600 mil) zero-insertion-force sockets. The PLE3-12A directly supports the EP310, EP320, and EP330 EPLDs (DIP packages only), and serves as the base unit for programming all other Altera EPLDs. Programming adapters supporting each EPLD (DIP, J-lead, PGA, QFP, and SOIC packages) plug directly into the PLE3-12A module.



The PLE3-12A includes a 30-inch ribbon cable terminated with a 25-pin D-type connector that interfaces only with the Altera Logic Programmer card. Programming information is transmitted from the programming card (installed in any full expansion slot of the computer) through the ribbon cable to the PLE3-12A unit. A programming indicator LED on the PLE3-12A lights up when the unit is active. The Logic Programmer card generates all programming waveforms and voltages, so the PLE3-12A requires no additional power supply.

8

Ordering Information

PLE3-12A

PLED/J/G/S/Q

PLED, PLEJ, PLEG, PLES & PLEQ Programming Adapters

October 1990, ver. 1

Data Sheet

Features

- Programming adapters for all devices other than EP310, EP320, and EP330 DIP EPLDs
- Adapters plug directly into PLE3-12A Master Programming Unit
- Zero-insertion-force sockets for easy device insertion and extraction

General Description

The Altera PLED, PLEJ, PLEG, PLES, and PLEQ are socket adapters for programming Altera EPLDs that are not directly supported by the PLE3-12A Master Programming Unit.

Each adapter contains a zero-insertion-force dual in-line package (DIP), J-lead (JLCC and PLCC), pin-grid array (PGA), small-outline IC (SOIC), or quad flat pack (QFP) socket. The adapters plug directly into the PLE3-12A programming unit, which supplies programming waveforms and voltages to the adapters. Table 1 gives a complete list of available adapters.

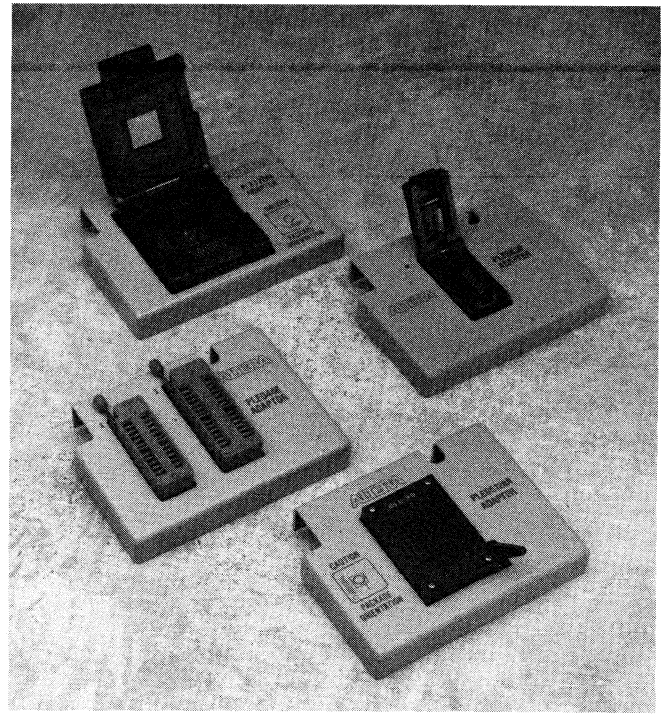


Table 1. EPLD Adapter Support

EPLD	Package	Part Number
EP330	DIP	PLED330
	J-Lead	PLEJ330
	SOIC	PLES330
EP600/610	DIP	PLED610
	J-Lead	PLEJ610
EP600/610/630/610A	DIP	PLED630
	J-lead	PLEJ630
	SOIC	PLES630
EP900/910	DIP	PLED910
	J-lead	PLEJ910
EP1800/1810	J-lead	PLEJ1810
	PGA	PLEG1810
EP1800/1810/1830	J-lead	PLEJ1830
	PGA	PLEG1830
EPS448	DIP	PLED448
	J-lead	PLEJ448
EPM5016	DIP	PLED5016
	J-lead	PLEJ5016
	SOIC	PLES5016
EPM5032	DIP	PLED5032
	J-lead	PLEJ5032
	SOIC	PLES5032
EPM5064	J-lead	PLEJ5064
EPM5128	J-lead	PLEJ5128
	PGA	PLEG5128
EPM5130	PGA	PLEG5130
	QFP	PLEQ5130
EPM5192	J-lead	PLEJ5192
	PGA	PLEG5192
	QFP	PLEQ5192
EPB2001	J-lead	PLEJ2001

8

Ordering Information

Order by the adapter part number shown in Table 1.

PLAESW-PC

Extended Software Warranty

October 1990, ver. 1 **Data Sheet**

Features

- ❒ Software and documentation update service for registered owners of Altera's PC-based development systems
- ❒ Access to Altera Applications telephone support hotline
- ❒ Discounts on additional software options
- ❒ Design assistance from Altera Applications via the Electronic Bulletin Board Service
- ❒ Access to Altera Electronic Application Utilities, Notes, and Briefs
- ❒ Included in the price of PLDS-ENCORE, PLDS-MAX, PLDS-SAM, PLCAD-SUPREME, and PLDS2 systems

General Description

PLAESW-PC is a software warranty and customer-support product that provides access to the latest EPLD development information. The Extended Software Warranty ensures that customers receive all new software and documentation when PC-based development systems are upgraded to provide new features and to support new EPLDs. Customer-support services include a telephone hotline to Altera Applications Engineers for assistance with design work. A 24-hour electronic bulletin board and design upload service is available via modem. PLAESW-PC also guarantees discounts on future Altera development software purchases.

Ordering Information

PLAESW-PC

Third-Party Development & Programming Support

October 1990, ver. 1

Data Sheet

Introduction

Altera recognizes the importance of third-party support tools and works closely with many third-party vendors to ensure high-quality support for EPLDs. Tables 1 through 4 provide an overview of third-party design entry tools, logic compilers, simulators, and device programmers. Current support (at the time of printing) is shown; contact the Altera Applications Department at 1 (800) 800-EPLD for the most up-to-date information.

Table 1. Third-Party Design Entry Tools

Company	Product	Design Entry Format	EPLDs Supported
Data I/O Corp.	FutureNet DASH ABEL	Schematic Equation	EP-series EP-series, EPM-series
ISDATA GmbH	LOG/iC	Equation	EP-series
Daisy/Cadnetix Inc. (Dazix)	ACE	Schematic	EPM-series *(1)*
Logical Devices, Inc.	CUPL	Equation	EP-series
OrCAD Systems Corp.	SDT III	Schematic	EP-series, EPM-series
Mentor Graphics Corp.	NETED	Schematic	EPM-series *(1)*
Valid Logic Systems, Inc.	ValidGED	Schematic	EPM-series *(1)*
Viewlogic Systems, Inc.	Workview	Schematic	EP-series, EPM-series *(1)*

Note:
(1) Information exchange occurs via EDIF 2 0 0 netlist format.

8

Table 2. Third-Party Logic Compilers

Company	Product	EPLDs Supported
Data I/O Corp.	ABEL	EP-series, EPM-series *(1)*
ISDATA GmbH	LOG/iC	EP-series, EPM-series
Logical Devices, Inc.	CUPL	EP-series
Minc Inc.	PLD/PGA Designer	EP-series EPM-series *(2)*

Notes:
(1) Directly supports EPM5016 and EPM5032. Multi-LAB devices (EPM5064, EPM5128, EPM5130, EPM5192) are supported by the Open-ABEL interface and the Altera-supplied ABEL2MAX Converter.
(2) Information exchange occurs via EDIF 2 0 0 netlist format.

Table 3. Third-Party Logic Simulators

Company	Product	EPLDs Supported
Aldec, Inc.	Susie	EP-series
Daisy/Cadnetix Inc. (Dazix)	DLS	EPM-series *(1)*
Cadence Design Systems, Inc.	Verilog	EPM-series
Logic Automation Inc.	SmartModel	EPM-series *(1)* EP-series
Mentor Graphics Corp.	QuickSim	EP-series *(2)*, EPM-series *(3)*
Quadtree Software Corp.	VHDL Models	EP-series
Valid Logic Systems, Inc.	ValidSIM	EPM-series *(1)*
Viewlogic Systems, Inc.	Viewsim	EP-series, EPM-series *(1)*

Notes:
(1) Information exchange occurs via EDIF 2 0 0 netlist format.
(2) Supported by Logic Automation SmartModel.
(3) Information exchange occurs via EDIF 2 0 0 netlist format with Logic Automation SmartModel.

Table 4. Third-Party Programming Hardware

Company	Product	EPLDs Supported
Data I/O Corp.	29B/LogicPak 2900 60 A/H UniSite 40	EP-series, EPM-series EP-series, EPM-series EP-series EP-series, EPM-series, EPS-series
DIGELEC INC.	UP-803	EP-series
Logical Devices, Inc.	ALLPRO	EP-series
Japan Macnics Corp.	PROMAC P3	EP-series
SMS GmbH	SPRINT Expert	EP-series, EPM-series
Stag Microsystems Ltd.	ZL30, ZL30A PPZ SYSTEM3000	EP-series EP-series EPM-series

Third-Party Vendors

Table 5 lists telephone numbers for third-party vendors. Altera strongly recommends direct contact with manufacturers for specific details on product features and availability. Altera can accept no responsibility for the suitability or accuracy of third-party development software or programming hardware.

Table 5. Manufacturers of Third-Party Tools

Company	Telephone Number
Aldec, Inc.	(805) 499-6867
Data I/O Corp.	(800) 247-5700
Daisy/Cadnetix Inc. (Dazix)	(415) 960-6777
DIGELEC INC.	(201) 493-2420
Cadence Design Systems, Inc.	(408) 943-1234
ISDATA GmbH	West Germany (49) 7 21/69 30 92
Logic Automation Inc.	(503) 690-6900
Logical Devices, Inc.	(800) 331-7766
OrCAD Systems Corp.	(503) 690-9881
Mentor Graphics Corp.	(503) 626-7000
Minc, Inc.	(719) 590-1155
Japan Macnics Corp.	Japan (81) 45 939 6150
Quadtree Software Corp.	(213) 597-5995
SMS GmbH	West Germany (49) 7522 5018
Stag Microsystems Ltd.	(408) 988-1118
Valid Logic Systems, Inc.	(408) 432-9400
Viewlogic Systems, Inc.	(800) 422-4660

8

Notes:

Contents

October 1990

Section 9 Military Products

9

Introduction

Tables 1 through 6 provide information on military temperature range, MIL-STD-883B-qualified, and DESC EPLDs. For detailed information, refer to individual data sheets in this book, or call Altera Military Marketing at 1 (800) SOS-EPLD. Outside the USA, dial (408) 984-2800.

MIL-STD-883-compliant product specifications are provided in military product drawings (MPDs) that are available on request from Altera Military Marketing. An MPD is prepared in accordance with the appropriate military specification format and should be used for preparing Source Control Drawings (SCDs). Refer to *Source Control Drawings* in this data book for an outline on SCD generation.

Table 1. EP-Series Classic Military-Temperature-Range EPLDs

EPLD	Package *(1)*	t_{PD1} (ns)	Notes
EP1830	J, G	30	*(2)*
EP1810	J, G	45	
EP910	D, J	40	
EP630	D, J	20	*(2)*
EP610	D, J	35	
EP330	D	15	*(2)*
EP320	D	45	

Table 2. EPM-Series MAX Military-Temperature-Range EPLDs

EPLD *(3)*	Package *(1)*	t_{PD1} (ns)	Notes
EPM5192	J, G	35	*(2)*
EPM5130	G, W	35	*(2)*
EPM5128	J, G	35	
EPM5064	J	35	
EPM5032	D, J	25	
EPM5016	D	20	

9

Table 3. EPS-Series SAM Military-Temperature-Range EPLDs

EPLD	Package *(1)*	f_{MAX}
EPS448	D, J	20 MHz

Table 4. EP-Series Classic MIL-STD-883B-Qualified EPLDs

EPLD	Package *(1)*	t_{PD1} (ns)	Altera Military Drawing	Notes
EP1810JMB	J	45	02D-00782	*(4)*
EP1810GM883B	G	45	02D-00782	
EP1800JMB	J	75	02D-00509	*(4)*
EP1800GM883B	G	75	02D-00205	
EP910DM883B	D	40	02D-00935	
EP910JMB	J	40	02D-00935	*(4)*
EP900DM883B	D	60	02D-00210	
EP900JMB	J	60	02D-00521	*(4)*
EP610DM883B	D	35	02D-00522	
EP610JM883BX	J	35	02D-00522	
EP600DM883B	D	55	02D-00194	
EP600JM883BX	J	55	02D-00194	
EP320DM883B	D	45	02D-00209	
EP310DM883B	D	50	02D-00179	

Table 5. EPM-Series MAX MIL-STD-883B-Qualified EPLDs

EPLD *(3)*	Package *(1)*	t_{PD1} (ns)	Altera Military Drawing	Notes
EPM5128JMB	J	35	02D-00827	*(4)*
EPM5128GM883B	G	35	02D-00827	
EPM5064JM883B	J	35	02D-00968	*(2)*
EPM5032DM883B	D	25	02D-00828	
EPM5032JM883B	J	25	02D-00828	
EPM5016DM883B	D	20	02D-00967	*(2)*

Table 6. EP-Series Classic DESC EPLDs

EPLD	Package *(1)*	t_{PD1} (ns)	DESC Order Number
EP1810JMB	J	45	8946901XX
EP1810GM883B	G	45	8946901YC
EP1800GM883B-1	G	90	8854901YC
EP1800GM883B	G	75	8854902YC
EP900DM883B	D	60	8854801QA
EP900JMB	J	60	8854801XX
EP610DM883B	D	35	8947601LX
EP610JM883BX	J	35	8947601XX
EP600DM883B	D	55	8686401LA
EP600JM883BX	J	55	8686401XX
EP310DM883B	D	50	8863501RA

Notes to tables:

(1) Package configurations:
- D: Windowed ceramic dual in-line package (CerDIP)
- J: Windowed ceramic J-lead chip carrier (JLCC)
- G: Windowed ceramic pin-grid array (PGA)
- W: Windowed ceramic quad flat pack (WQFP)

(2) This product is currently under development. Contact Altera Military Marketing for current information.

(3) Additional MAX EPLDs are still being qualified. Contact Altera Military Marketing for current information.

(4) This device is screened to MIL-STD-883 with a published deviation. Contact Altera Military Marketing for current information.

9

Source Control Drawings
for Military-Qualified EPLDs

October 1990, ver. 1

Data Sheet

General Description

Source Control Drawings (SCDs) are documents created by military subcontractors for documentation control. An SCD can be generated for commercial EPLDs that are or will be qualified for MIL-STD-883B.

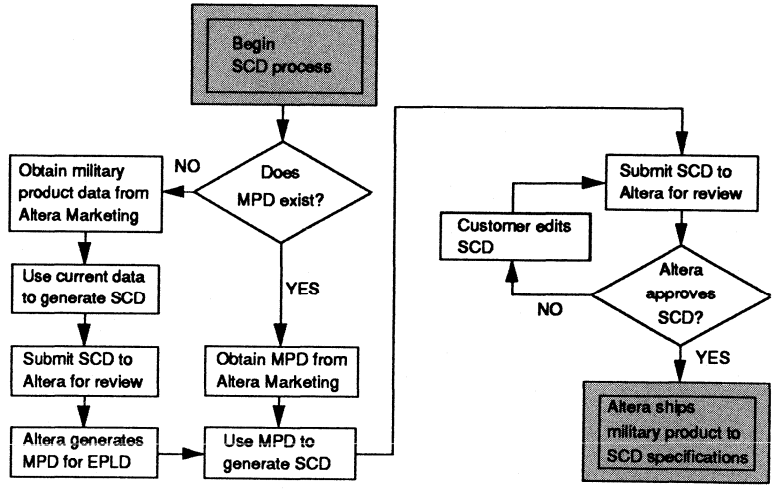
An SCD should be based on a Military Product Drawing (MPD) provided by Altera. When an MPD is not available, the user should contact Altera's Marketing Department for current data. An SCD generated from information in Altera's commercial data book is not acceptable.

Altera's MPDs contain information on the scope, reference documents, MIL-STD-883B requirements, quality assurance provisions, and preparation of delivery for EPLDs. Characteristics of device screening—such as burn-in testing, AC/DC electrical properties, timing waveforms, and package dimensions—are also detailed in MPDs. These specifications may differ from those for Altera's commercial EPLDs.

Altera will not approve an SCD from a military subcontractor until the described MPD requirements are met.

Figure 1 shows the process flow for generating an SCD for Altera's military products.

Figure 1. Generation of a Source Control Drawing



Total-Dose Radiation Hardness of Altera EPLDs

October 1990, ver. 2

Application Brief 51

Introduction

Altera has measured the EPROM process in a series of tests, both for the EP-series classic EPLDs and the EPM5000-series MAX EPLDs. This application brief summarizes the results of these tests.

EP-Series EPLDs

Tests for the EP-series EPLDs were performed under the following conditions:

- Tested in conformance with method 1019.2 of MIL-STD-883C
- Radiation hardness to 14 krad (Si) under worst-case conditions

For the classic EPLDs, the EP600DM was used as the vehicle for evaluation. Based on the failure mechanisms determined by the tests, results for this product are expected to be typical of all Altera EP-series EPLDs.

The EP600 functions properly at the worst-case military power-supply range (V_{CC} = 5.5 V) until total doses in excess of 10 krad (Si) are reached. Conditions that are not worst-case provide total-dose tolerance far in excess of 10 krad (Si).

The experiments were conducted with a cobalt-60 isotope source with a dose rate of 750 krad (Si) per minute (±10%). Test units were powered up with V_{CC} = 5.0 V and all other pins connected to ground, both during and after the irradiation, until measurements were made. All conditions of temperature control, dose rate, and electrical bias conformed to method 1019.2 of MIL-STD-883C.

Numerous key parameters were measured. The margin of programmed EPROM cells, as indicated by the maximum V_{CC} value for error-free verification and functional operation of the part, proved to be the most important. The following five parameters were monitored:

- Programmed cell margin (from V_{CC} maximum)
- Erased cell margin (from V_{CC} minimum)
- AC timing (from t_{PD})
- I_{CC} standby vs. input voltage level
- Output-high and output-low drive currents

9

These measurements made it possible to monitor the extent of EPROM-cell programming, and to detect changes in junction leakage and transistor threshold voltage.

Degradation of programmed EPROM cell margins was the first cause of failure in all experiments. The rate of this degradation—and consequently the total-dose radiation tolerance—was observed to be a strong function of irradiation. If the control (top) gate is biased at V_{CC} during the irradiation, the tendency for the cell to lose charge is greatly reduced. In this case, total-dose radiation tolerance observed was > 30 krad (Si) for 5.5 V operation, and in excess of 40 krad (Si) for 5.0 V operation. If, on the other hand, the control gate is grounded during the irradiation, the total-dose radiation tolerance is reduced to about 14 krad (Si) for 5.5 V operation, and to about 18 krad (Si) for 5.0 V operation. This behavior suggests that energetic holes created in the bulk silicon by the radiation and attracted toward the floating gate by its negative (programmed) charge are the dominant cause of cell charge loss.

When the control gate of these EPROM cells is grounded during irradiation, the worst-case result of about 14 krad (Si) for 5.5 V operation is the best estimate of the radiation tolerance of these products. Most radiation-tolerance experiments on EPROM technologies include control gates that are biased at V_{CC} during at least part of the irradiation.

EPM5000-Series EPLDs

Tests for the EPM5000-series MAX EPLDs were performed at radiation hardness to 6.5 krad (Si) under worst-case V_{CC}.

For the EPM5000 series, the EPM5032DC was used as the vehicle for evaluation. The tests were conducted with a cobalt-60 isotope source with a dose rate of 117 rad (SI)/s. All irradiation was performed at room temperature (+25° C, ±3), while device-under-test (DUT) V_{CC} was set to +5.5 V. Nine input pins were pulled to V_{CC} through 1-kΩ resistors and three were tied directly to GND. All output pins were pulled up to V_{CC} through 1-kΩ resistors, while the reserved I/O pins were left open. These specifications provided a realistic control-gate bias distribution for a typical design.

Total-dose-induced functional failure thresholds are highly sensitive to measurement voltage. Under the worst-case military power-supply range (V_{CC} = 5.5 V), the EPM5032 test devices function properly until total doses in excess of 6.5 krad (Si) are reached. For V_{CC} = 5.0 V, proper functioning occurs above 13.0 krad (Si) and 17.5 krad (Si) for V_{CC} = 4.5 V.

Timing parameters were measured using V_{CC} = 5.0 V. No significant change in maximum operating speed or propagation-delay times occurred at levels as high as 10 krad (Si).

Conclusion

The differences in the test results between the EP600 and EPM5032 indicate that actual device types used in a final design should be separately evaluated to verify total-dose effects. The evaluation should include a complete functional test at the expected operating supply-voltage extremes, as well as extensive parametric measurements. More complex devices may show greater increase in response to irradiation if additional features that have not been checked in these test devices—such as the Programmable Interconnect Array (PIA)—are included.

Programmed cell margin fell linearly with increasing total radiation dose. Therefore, to maximize radiation hardness, it is important to program an adequate cell margin into the EPLD before irradiation. Designers are advised to use Altera-approved programming systems that provide optimized cell programming.


ALTERA
EPM5032 DM
ALTERA
EPM5016 DM
ALTERA
EPM5064 JM
ALTERA
EPM5128 JM

Section 10 Application Notes & Briefs

Replacing 20-Pin PAL and GAL Devices with EP300-Series EPLDs

October 1990, ver. 3

Application Note 2

Introduction

Altera EP300-series Erasable Programmable Logic Devices (EPLDs) have an I/O architecture that is user-configurable, allowing them to be functionally and pin-to-pin compatible with PALs and GALs. These EPLDs have the following features:

- ❒ Highly flexible user-configurable I/O architecture
- ❒ "Zero power" (typically 10 to 40 µA standby)
- ❒ High speed (t_{PD} = 12 ns) with "quiet" outputs
- ❒ Directly replaces all common 20-pin PALs and GALs

EP300-series EPLDs can directly replace functions implemented with 20-pin PAL and GAL devices. The EP320 is optimized for low-power applications, typically consuming 10 µA when operating in standby mode. The EP330 EPLD offers very high speed while taking advantage of the low-power benefits of CMOS. Thus, EP300-series EPLDs offer significant power savings over conventional fuse-programmable bipolar PALs and CMOS GALs, and are also erasable and reprogrammable. These benefits, combined with a highly flexible architecture, enable a single EP300-series EPLD to replace a variety of other 20-pin Programmable Logic Devices (PLDs).

EP300-Series Functional Overview

EP320 and EP330 EPLDs, like PALs and GALs, implement sum-of-products logic with a programmable-AND/fixed-OR logic array. These EPLDs provide 10 dedicated inputs and 8 I/O pins. Each I/O pin can be independently configured for input, output, or bidirectional operation (see the *EP300-Series EPLDs Data Sheet* for more information).

The internal architecture of EP300-series EPLDs is divided into 8 macrocells, each of which implements logic with up to 8 product terms. An additional product term controls Output Enable. Each product term represents a 36-input AND gate, whose inputs come from the true and complement signals of the 10 input pins and 8 feedback paths. Altera's proprietary programmable I/O architecture allows the designer to program output and feedback paths for combinatorial or registered operation in active-high and active-low modes.

Pin 1 may be used to directly clock all registers, or as an additional input to the AND array if no clocking is required. The EP300-series flip-flops are positive-edge-triggered, i.e., data is registered on the rising edge of the clock signal. During chip power-up, all registers are automatically reset to zero.

PAL Compatibility

While PAL devices have fixed I/O architectures, the macrocells of EP300-series EPLDs are based on a flexible, user-configurable I/O structure that gives the designer freedom to configure macrocells for combinatorial or registered operation in active-high or active-low modes. Figure 1 shows the possible output modes, which can be configured on a macrocell-by-macrocell basis.

Figure 1. EP300-Series I/O Options

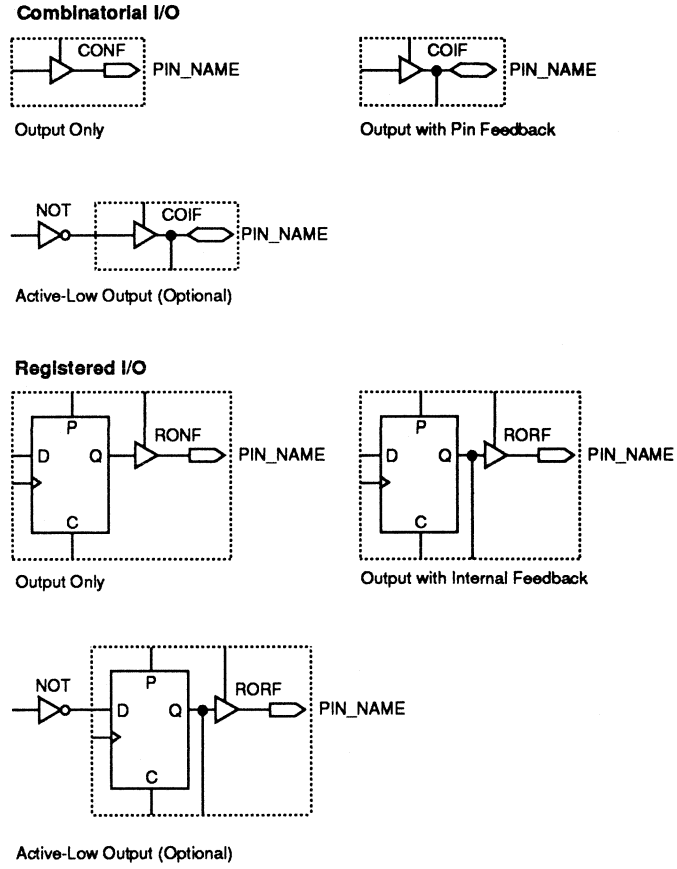


Table 1 gives a detailed listing of proper I/O configurations to replace many commonly used 20-pin PALs and GALs. Table 2 compares the Altera EP300-series EPLDs with competitive PAL/GAL devices.

Table 1. EP300-Series Configuration for 20-Pin PAL/GAL Replacement

PAL Part Number	EP300-Series Pin Numbers	EP300-Series Macrocell Numbers	I/O Configuration Mode	Output/Polarity	Feedback
10H8	12 to 19	1 to 8	Combinatorial	Comb./High	None
10L8	12 to 19	1 to 8	Combinatorial	Comb./Low	None
12H6	12	8	Combinatorial	None	Pin
	13 to 18	2 to 7	Combinatorial	Comb./High	None
	19	1	Combinatorial	None	Pin
12L6	12	8	Combinatorial	None	Pin
	13 to 18	2 to 7	Combinatorial	Comb./Low	None
	19	1	Combinatorial	None	Pin
14H4	12 to 13	7 to 8	Combinatorial	None	Pin
	14 to 17	3 to 6	Combinatorial	Comb./High	None
	18 to 19	1 to 2	Combinatorial	None	Pin
14L4	12 to 13	7 to 8	Combinatorial	None	Pin
	14 to 17	3 to 6	Combinatorial	Comb./Low	None
	18 to 19	1 to 2	Combinatorial	None	Pin
16C1	12 to 14	6 to 8	Combinatorial	None	Pin
	15	5	Combinatorial	Comb./Low	None
	16	4	Combinatorial	Comb./High	None
	17 to 19	1 to 3	Combinatorial	None	Pin
16H2	12 to 14	6 to 8	Combinatorial	None	Pin
	15 to 16	4 to 5	Combinatorial	Comb./High	None
	17 to 19	1 to 3	Combinatorial	None	Pin
16L2	12 to 14	6 to 8	Combinatorial	None	Pin
	15 to 16	4 to 5	Combinatorial	Comb./Low	None
	17 to 19	1 to 3	Combinatorial	None	Pin
16H8	12	8	Combinatorial	Comb./High/Z	None
and	13 to 18	2 to 7	Combinatorial	Comb./High/Z	Comb.
16HD8	19	1	Combinatorial	Comb./High/Z	None
16L8	12	8	Combinatorial	Comb./Low/Z	None
and	13 to 18	2 to 7	Combinatorial	Comb./Low/Z	Comb.
16LD8	19	1	Combinatorial	Comb./Low/Z	None
16R4	12 to 13	7 to 8	Combinatorial	Comb./Low/Z	Comb.
	14 to 17	3 to 6	Registered	Reg./Low/Z	Reg.
	18 to 19	1 to 2	Combinatorial	Comb./Low/Z	Comb.
16R6	12	8	Combinatorial	Comb./Low/Z	Comb.
	13 to 18	2 to 7	Registered	Reg./Low/Z	Reg.
	19	1	Combinatorial	Comb./Low/Z	Comb.
16R8	12 to 19	1 to 8	Registered	Reg./Low/Z	Reg.
16P8	12	8	Combinatorial	Comb./Option/Z	None
	13 to 18	2 to 7	Combinatorial	Comb./Option/Z	Comb.
	19	1	Combinatorial	Comb./Option/Z	None
16RP4	12 to 13	7 to 8	Combinatorial	Comb./Option/Z	Comb.
	14 to 17	3 to 6	Registered	Reg./Option/Z	Reg.
	18 to 19	1 to 2	Combinatorial	Comb./Option/Z	Comb.
16RP6	12	8	Combinatorial	Comb./Option/Z	Comb.
	13 to 18	2 to 7	Registered	Reg./Option/Z	Reg.
	19	1	Combinatorial	Comb./Option/Z	Comb.
16RP8	12 to 19	1 to 8	Registered	Reg./Option/Z	Reg.
16V8	12 to 19	1 to 8	Combinatorial Registered	Comb./Reg. Option/Z	Comb./Reg.

10

Table 2. Comparison of EPLD and PAL/GAL Features

Feature	EPLD	PAL/GAL Device		
	EP300 Series	16L8	16R8	16V8
Array logic	AND-OR	AND-OR	AND-OR	AND-OR
Inputs	17	16	10	17
Outputs	8	8	8	8
Array input lines	36	32	32	32
Product terms	72	64	64	64
D-type flip-flops	8	n.a.	8	8
Reprogrammable	Yes	No	No	Yes

Output Enable

Every output of the EP320 and EP330 has a tri-state buffer that is controlled by a dedicated product term. When the product term is high, the output is enabled. Since the Output Enable logic is implemented directly in the AND array, it can be programmed active-high or active-low, or conditionally asserted by any of the selected inputs and feedback paths.

Tri-state combinatorial outputs are implemented similarly in EP300-series EPLDS and PAL/GAL devices. However, PALs and GALs use 1 of the 8 product terms in the macrocell to control the buffer, leaving only 7 product terms for logic. In contrast, EP300-series EPLDs provide an additional product term, allowing the 8-input OR gate to remain intact.

Registered PALs hard-wire the Output Enable function to pin 11, forcing active-low operation. However, EP300-series EPLDs can connect the true complement of pin 11 to the Output Enable product term, giving the designer the ability to preserve exact compatibility or the flexibility to configure the device for a particular system.

Design Tools

Logic is implemented with Altera's A+PLUS Development System, which supports schematic capture, state machine, Boolean equation, and netlist design entry methods. Logic schematics, created with LogiCaps, allow the user to quickly construct a wide range of designs (see the *PLS-SUPREME Data Sheet* in this data book). After the design is entered, A+PLUS automatically translates it into logic equations, performs Boolean minimization, and fits it into an Altera EP300-series EPLD. The device can then be programmed in seconds at the designer's desktop to create customized working silicon. Extensive third-party support also exists for design entry, design processing, and device programming. Altera also provides several free software utilities to quickly convert PAL and GAL designs into EP300-series EPLDs. See *Application Brief 73 (Software Utility Programs)* in the *Development Products* section of this data book, or contact Altera Applications at 1 (800) 800-EPLD for more information.

Memory and Peripheral Interfacing

with EP-Series EPLDs

October 1990, ver. 4

Application Note 3

Introduction

Programmable Logic Devices (PLDs) have long been used for address decoding and other microprocessor support functions. PLDs offer faster decode times and require little PC board space. Erasable Programmable Logic Devices (EPLDs) offer the additional benefits of significant power savings and a more efficient debugging cycle.

This application note discusses the following subjects:

- ❒ Designing address decode and chip-select logic
- ❒ Wait-state generation
- ❒ Dynamic RAM (DRAM) control

This application note shows how to interface the Intel 8086 (synchronous data bus) and the Motorola 68000 (asynchronous data bus) to several memory and peripheral devices with the EP610 and EP330 EPLDs.

Intel 8086 Solutions

This section describes how to use the EP610 EPLD with the Intel 8086 device.

Address Decoding

The circuit in Figure 1 shows an EP610 EPLD that provides chip-select signals in a high-speed serial-data-line multiplexer. This circuit is controlled by an 8086 microprocessor program stored in a 2764 EPROM. A 6164 static RAM stores data packets as they are assembled for transfer. Four serial communication control (SCC) peripheral chips provide an interface between the 8086 CPU and the serial data lines.

The EP610 in this design decodes the address lines of the 8086 into chip-select signals for the computer's peripherals (RAM, ROM, and SCC). The EP610's outputs follow the memory map shown in Table 1. The Boolean equations in this table describe the memory map for a 16-bit address bus; a more complicated memory map can be implemented by changing the equations. An Altera utility program, called DECODER, automatically derives equations for complicated memory maps. See *Application Brief 73 (Software Utility Programs)* in the *Development Products* section of this data book for more information.

Figure 1. 8086-EP610 Interface

The EP610 EPLD controls the ROM, RAM, and SCC chip-select logic, as well as the wait-state generation.

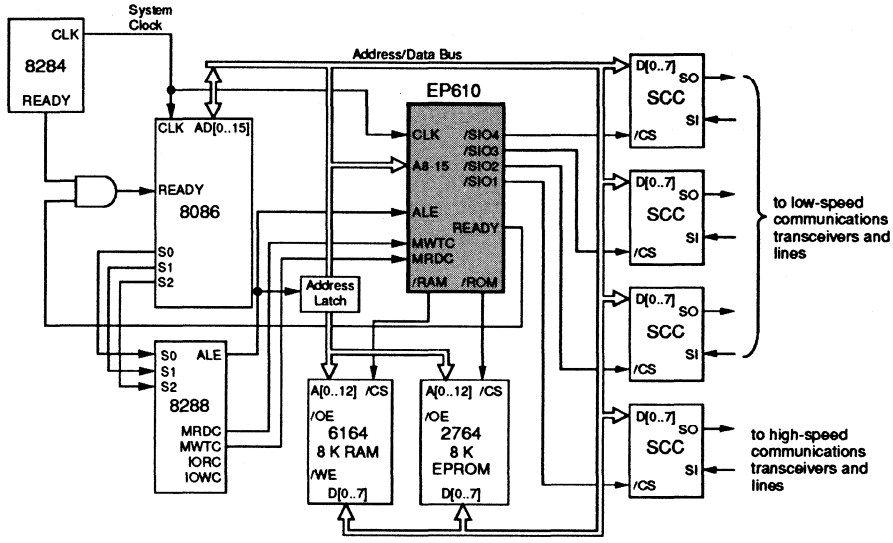


Table 1. Address Decode Memory Map Table

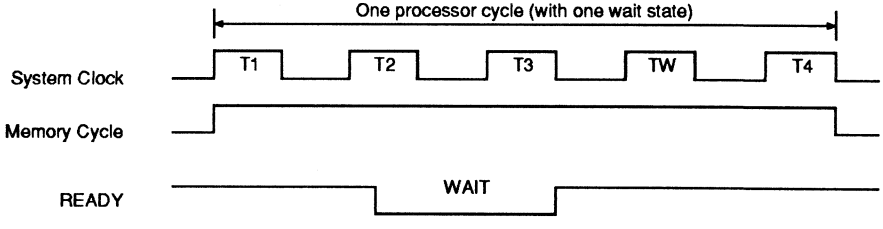
Signal Name	Low Address	High Address	Equation
ROM	0000	1FFF	A15' * A14' * A13'
RAM	2000	3FFF	A15' * A14' * A13
SI00	8000	80FF	A15 * A14' * A13' * A12' * A11' * A10' * A9' * A8'
SI01	8100	81FF	A15 * A14' * A13' * A12' * A11' * A10' * A9' * A8
SI02	8200	82FF	A15 * A14' * A13' * A12' * A11' * A10' * A9 * A8'
SI03	8300	83FF	A15 * A14' * A13' * A12' * A11' * A10' * A9 * A8
SI04	8400	84FF	A15 * A14' * A13' * A12' * A11' * A10 * A9' * A8'

Wait-State Generation

The EP610 also performs wait-state generation. It is inefficient to run a microprocessor at the speed of its slowest peripheral chip. Instead, the microprocessor is usually clocked at its top speed, and a wait-state generator slows the microprocessor's bus cycles when the the slowest peripheral is selected. A wait-state generator is typically implemented with a counter and a handful of decode logic. Figure 2 shows the state table for an 8086 wait-state generator. To implement a wait state, the **READY** signal must be driven low before the falling edge of **T2**. When accessing the ROM, the EP610 asserts the **READY** signal low for one 8086 clock cycle (i.e., one wait state). **READY** is asserted low for two clock cycles (i.e., two wait states) if the RAM is selected. For either wait state, a memory read or write (**MRDC** or **MWTC**) and address latch enable (**ALE**) must be valid. Figure 3 shows the EP610 logic required for both the address decode and wait-state circuits.

Figure 2. EP610 Wait-State Generation

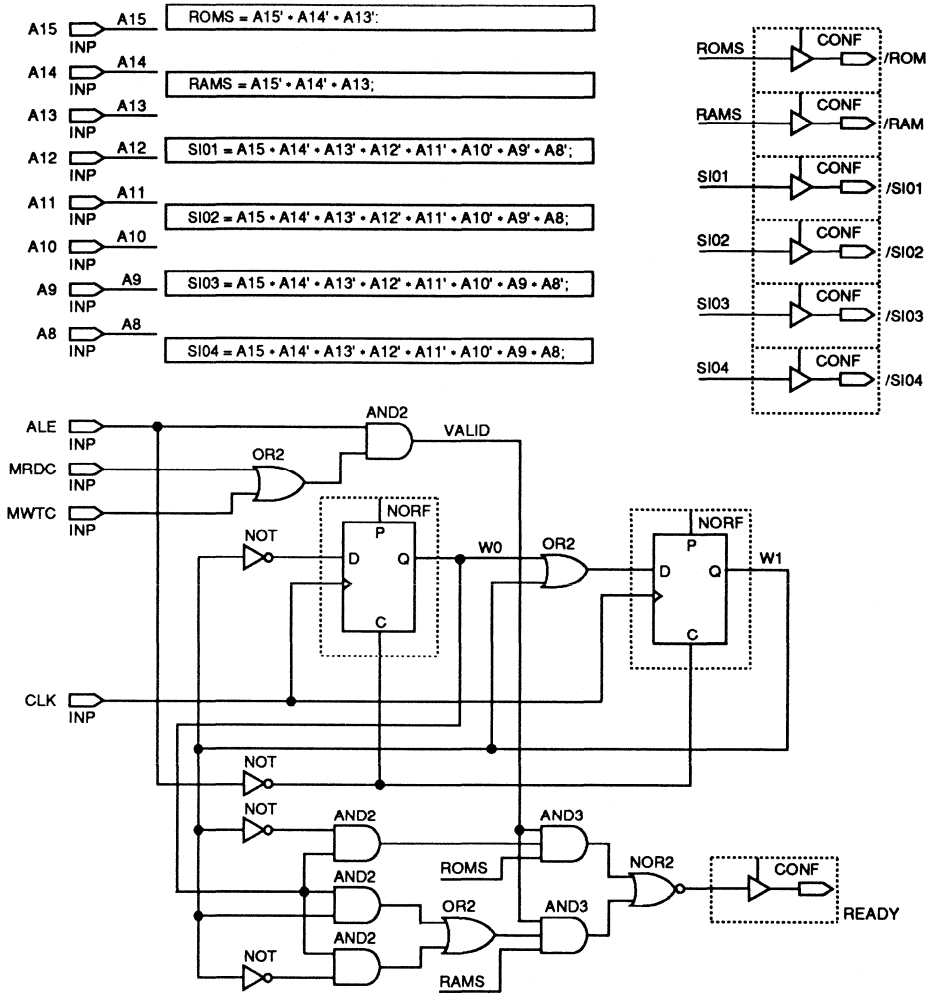
To implement an 8086 wait state, the READY signal must be low before the falling edge of T2.



State Name	W0	W1	Device
STANDBY	L	L	RESET
WAIT1	H	L	ROM
WAIT2	H	H	RAM

10

Figure 3. EP610 Address Decode and Wait-State Generator Logic Schematic



Motorola 68000 Solutions

Figure 4 shows a 68000-based system with an EP610 programmed as a wait-state generator and address decoder. Wait states in a 68000 circuit are easily generated by controlling the **DTACK** (Data Transfer Acknowledge) signal. The clock pulses are counted by two or three flip-flops after the assertion of one of the 68000's data select lines, **LDS** and **UDS**. The flip-flops then assert **DTACK** after a programmed count is reached. The programmed terminal count depends on the speed of the selected peripheral device, allowing different numbers of wait states for different peripherals.

Figure 4. 68000-EP610 Interface

The EP610 implements both address decode and wait-state generation for RAM and ROM memory devices.

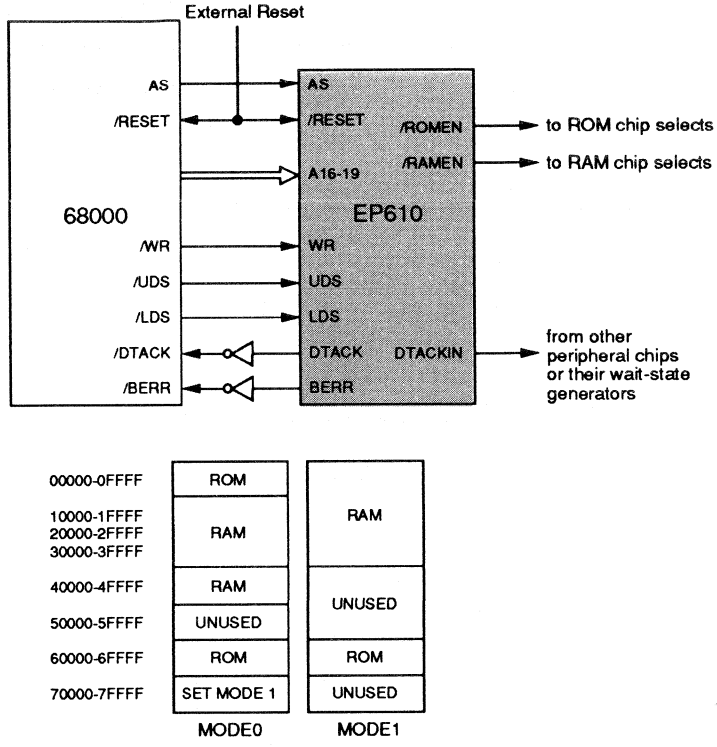
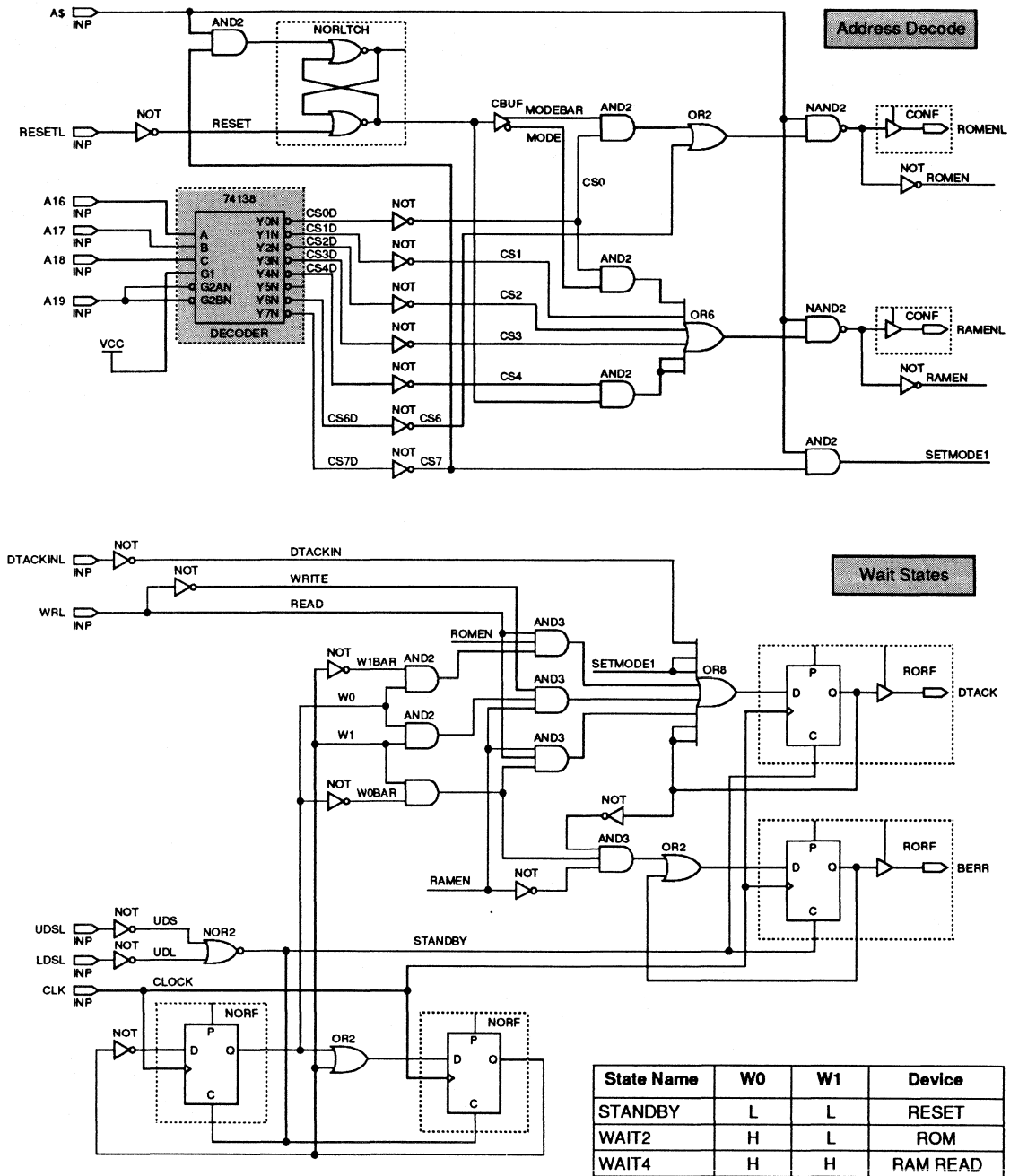


Figure 5 shows the logic schematic for the EP610. The design provides two chip-select signals, one for 256 Kbytes of DRAM (**RAMENL**) and the other for 64 Kbytes of ROM (**ROMENL**). The EP610 asserts **DTACK** one clock cycle after ROM is selected, providing the ROM with two wait states (**WAIT2**), two clock cycles after a RAM read (**WAIT4**), and three clock cycles after a RAM write (**WAIT6**).

As an added element of security, the 68000's bus error line (**BERR**) is asserted if neither RAM nor ROM is selected and if the **DTACKIN** signal is not asserted before the eighth wait state. This feature is useful in industrial control applications, for example, to signal a controller fault or to reset the system after a fault. This design requires only half of an EP610 EPLD for implementation.

Figure 5. EP610 Address Decode and Wait-State Logic



State Name	W0	W1	Device
STANDBY	L	L	RESET
WAIT2	H	L	ROM
WAIT4	H	H	RAM READ
WAIT6	L	H	RAM WRITE

Dynamic RAM Control

DRAM circuits must generate the **RAS**, **CAS**, and **WR** control signals for the DRAMs, **MAS** (memory address select) for the address decoder, and a **DTACK** or **READY** line to acknowledge the data transfer to the CPU. DRAM controllers connected to some 16-bit microprocessors may use either half of the 16-bit bus for 8-bit transfers. In addition, DRAMs must be refreshed by the DRAM controller, a DMA channel, or an interrupt-driven computer software loop during each bus cycle or while in burst mode.

Figure 6 shows a DRAM controller design for the EP330 that handles all of the above actions. The EPLD performs the following steps:

1. Places half of the CPU address on the DRAM's address lines and negates **MAS**.

2. Asserts the **RAS** address strobe.

3. a) Places the second half of the CPU address on the DRAM's address lines (asserts **MAS**).
   b) Asserts the DRAM's **WR** line if the bus transfer is a write operation.

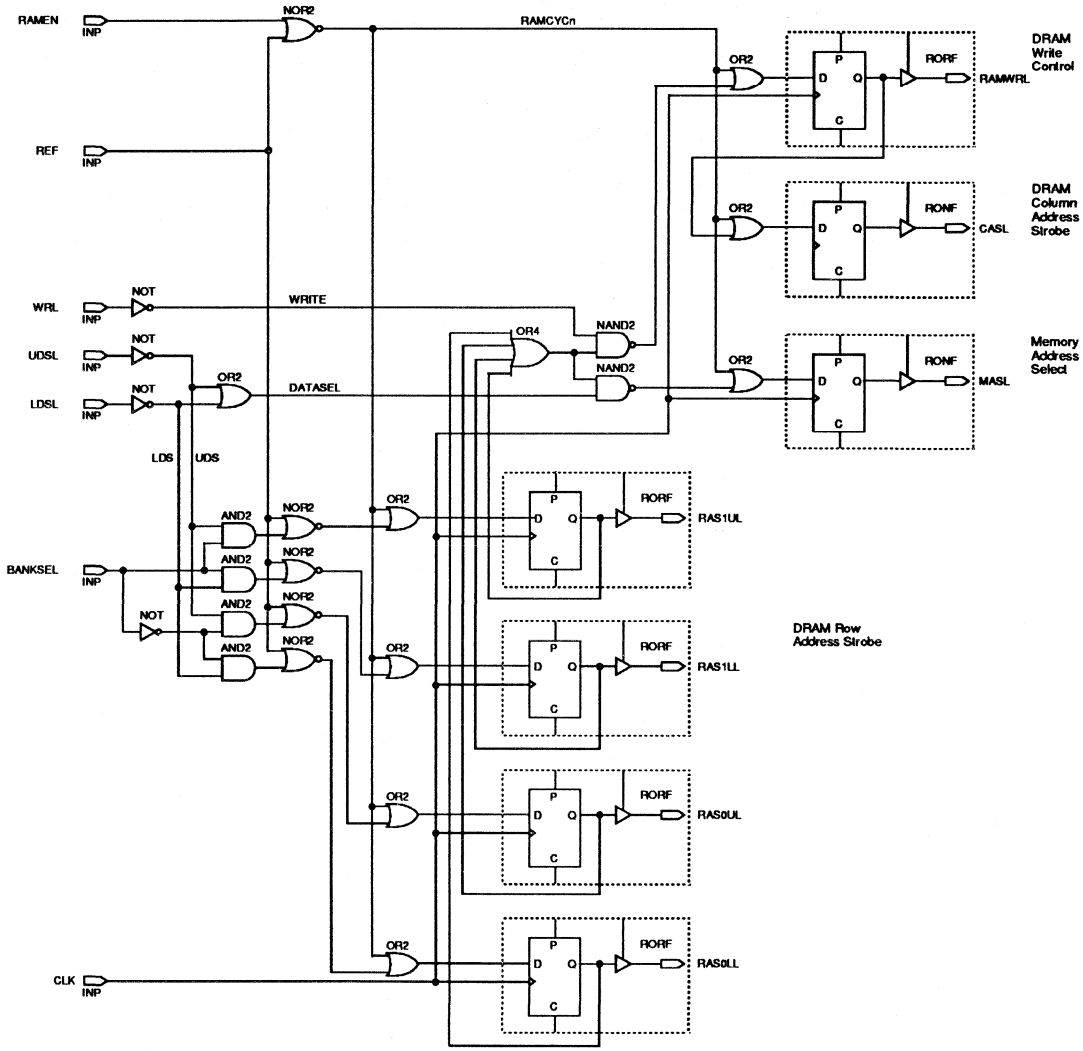
4. Asserts the **CAS** signal.

5. Waits for the bus cycle to finish, then negates all signals.

An EP330 programmed with the design shown in Figure 6 remains in a state with all signals negated (step 1 above) until a bus cycle is started with **RAMEN** and **UDS** or **LDS**. The EP330 then cycles through the steps listed above at a rate set by the clock, stopping at step 5 until the bus cycle ends (i.e., **RAMEN**, **UDS**, and **LDS** negate), and returning to step 1 to wait for the next bus access.

Figure 6. EP330 Dynamic RAM Control

The controller's clock frequency must be exactly the same as the 68000 system clock.



Metastability Characteristics of EPLDs

October 1990, ver. 2

Application Note 9

Introduction

This application note provides the following information:

- ❒ Definition of metastability
- ❒ Description of an experimental setup for metastability measurements
- ❒ Metastability characteristics of Altera EPLDs
- ❒ How to calculate MTBF numbers

The application note describes the problems associated with synchronization of asynchronous signals, in particular, the phenomenon of metastability in clocked flip-flop elements. To help the designer predict and guarantee required mean-time-between-failure (MTBF) rates in circuits targeted for Altera EPLDs, this application note also presents experimental data for Altera EPLDs when the associated flip-flops are used in asynchronous signal synchronizer applications. This information can then be compared to data on standard TTL components, which is also provided.

Altera EPM5000-series MAX EPLD flip-flops show metastability characteristics ranging from those of Advanced Low-Power Schottky (ALS) TTL flip-flops to Fairchild Advanced Schottky (FAST) TTL flip-flops. Altera's EP-series EPLD flip-flops show metastability characteristics that are better than those of Low-Power Schottky (LS) TTL flip-flops.

Synchronization

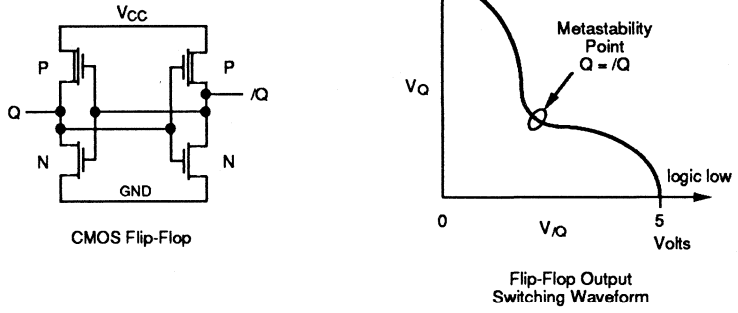
Many designs require communication between asynchronously clocked systems to be synchronized. Most systems are designed synchronously so that all signal transitions within, and generated by, a system use an edge of a master clock as reference. Synchronous systems also require synchronous inputs to avoid race conditions, setup time violations, and other logic problems. The goal, therefore, is to synchronize external inputs with each system's local clock to ensure that operation and MTBF requirements for the composite system(s) are met.

The edge-triggered flip-flop is frequently used to obtain synchronization. Clocked by the system's master clock or a derivative, the flip-flop synchronizes transitions on its data input with the clock, and outputs the result to the system. Its output transitions are synchronous with its clock.

However, when asynchronous signals are synchronized, the minimum timing requirements of the flip-flop cannot be guaranteed. For example, if a signal changes at a flip-flop's data input from low to high at the same time as the clock, the output state is ambiguous. Since the minimum setup

and hold times are not met, electrical parameters and logic functions are no longer guaranteed. The combination of these logical/electrical uncertainties manifests itself in a state known as metastability (see Figure 1).

Figure 1. Metastability Characteristics



Metastability

A flip-flop is typically defined as a bistable element, with the /Q output at logic high and the Q output at logic low, or vice versa. However, the description "bistable" is misleading, since a third stable state is possible where both nodes are at identical voltages. This third state is called "metastable," since the smallest disturbance will push the flip-flop in one direction or the other. Although the flip-flop does eventually stabilize, it hovers in a metastable state while /Q and Q are both at equal (indeterminate) logic levels.

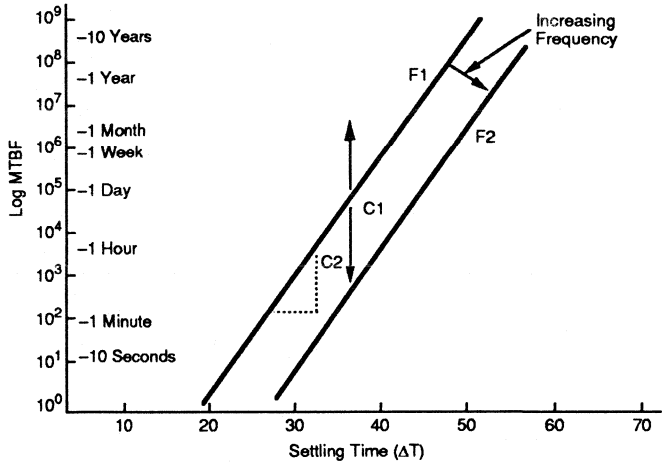
Clocking a flip-flop while its data input is in transition—which may happen during synchronization—can create such a metastable event. The resulting indeterminate output logic levels can, in fact, produce unpredictable results if allowed to propagate throughout the system. The likelihood of encountering this metastable period depends on the width of the window and the signal frequencies being synchronized. The following experiments confirm this conclusion.

The basic equation for the MTBF of a synchronizer due to metastability events is:

$$MTBF = [F_{CLOCK} \times F_{DATA} \times C_{C1} \times e^{(-C2 \times \Delta T)}]^{-1}$$

If this equation is plotted on semi-log graph paper, the MTBF/ΔT relationship appears as a straight line (Figure 2). In the equation shown above, ΔT represents the amount of settling time allowed for the flip-flop to settle to a valid stable state.

Figure 2. Metastability Function & Effects of Parameters



The constants C_1 and C_2 in this equation reflect particular characteristics of the device and, more importantly, the process technology used to manufacture it. Different devices fabricated on the same technology have similar metastability characteristics. Device design tricks and optimizations do not have a marked effect. Fundamental technology parameters such as on-chip capacitances and inverter gain predominate.

The constant C_1 linearly scales the MTBF equation. Therefore, the smaller the value of C_1, the higher the MTBF. C_1 affects the MTBF/ΔT curve in an absolute sense, tending to translate it along the MTBF axis.

The constant C_2 affects the slope of the MTBF/ΔT plot. Therefore, it is a measure of how quickly, in a relative sense, the flip-flop snaps out of metastability. The steeper the relationship on the plot, the better the settling.

The equation indicates that MTBF is a linear function of clock and data frequencies. Results for other operating conditions can be predicted with the data for a set of input frequencies.

Experiment Setup

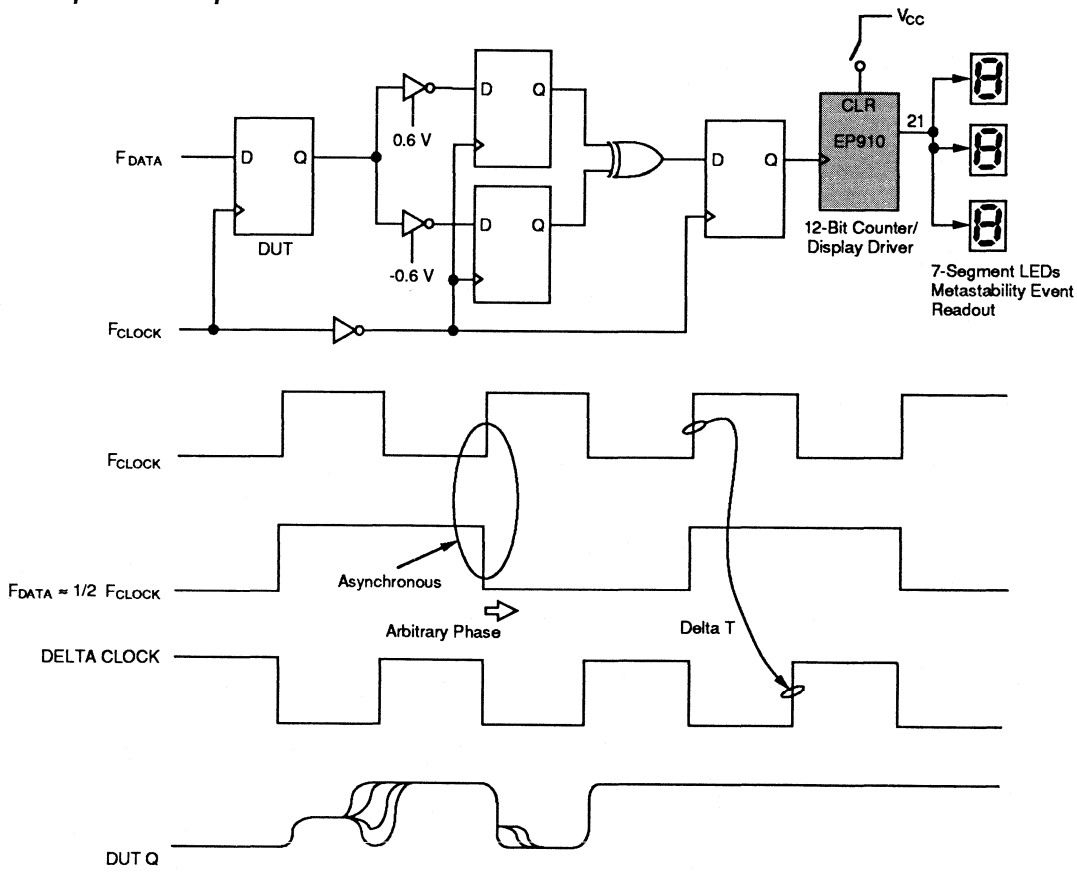
To help evaluate the relationship between metastability-event frequency and input-signal frequencies, the following experiment shows a means for defining and measuring a metastability event. In this experiment, a flip-flop is defined to be in a metastable state whenever its output voltage Q is greater than V_{IL} maximum and less than V_{IH} minimum for longer than normal output transition times. For TTL levels, this state corresponds to 0.8 V < output voltage < 2.0 V.

10

An experimental circuit for measuring a metastable event must include the following items:

- A Device Under Test (DUT), i.e., the synchronizing device to be evaluated
- Two independent signal sources that act as local system clock and data inputs
- A way to compare the DUT's output to V_{IH} and V_{IL} levels. Such a comparison can be performed with dedicated comparator devices, or, as shown in Figure 3, with inverters that have appropriate bias voltages applied to the device grounds. The inverter arrangement is a very-high-speed arrangement—with a few nanoseconds delay, dependent on logic family—and requires fewer power supplies than most dedicated comparators.

Figure 3. Experiment Setup



- A means to strobe the comparators' outputs at variable delay times from the DUT clock to detect a metastable event of a given duration. As shown in Figure 3, strobing can be provided with an inverted version of the DUT clock. By varying the width of the DUT clock pulse, a variable delay between the two rising edges is obtained.

- A counter that counts the metastable events. In this experiment, the data measured is the number of metastability events as a function of ΔT and length of observation. Length of observation divided by the number of events gives an MTBF number. Plotting the data yields the characteristic lines.

Experiment Results

The experiments analyze not only Altera's EPLDs for metastability characteristics, but also several other logic devices, including standard TTL 7474 devices from LS, ALS, and FAST logic families. The plotted data is shown in Figure 4. The results are summarized here:

- Metastability characteristics of devices in a given family are virtually identical since these characteristics are technology-dependent rather than individual device/circuit-dependent.

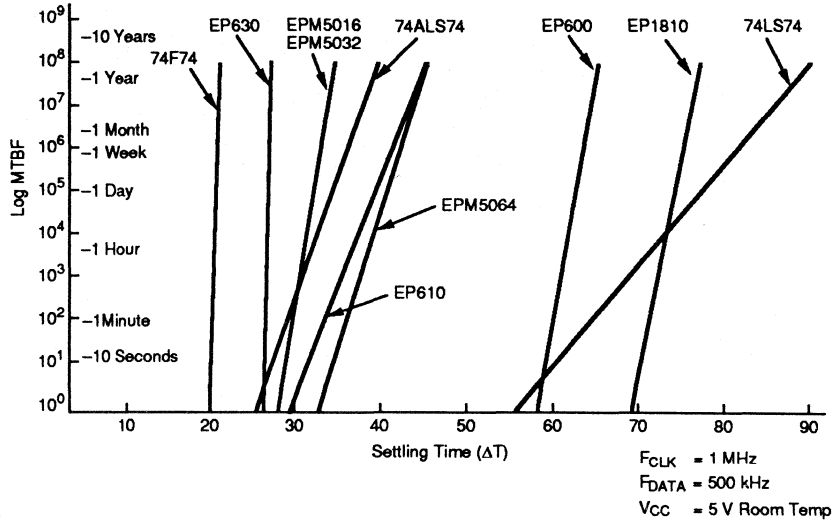
- MAX devices exhibit metastability characteristics that are substantially better than those of EP-series devices because they are fabricated on a faster 0.8-micron double-metal process. The technology dependence of flip-flop settling time is thus reinforced.

- Altera's EPM5000-series MAX EPLDs exhibit metastability characteristics that are equivalent to those of FAST-TTL series devices. When integrating such designs, EPLD flip-flops may be used as synchronizers with equivalent characteristics.

- In cases where very fast settling time is required of a synchronizer, AS-TTL logic still provides characteristics that are superior to high-density solutions such as EPLDs.

- Integrating multiple logic levels into a single EPLD, can often reduce the need for rapid stand-alone settling times. This option is available when isolated performance is not critical.

Figure 4. Results of the Experiment



Given the plotted data, the derivation of the two constants C_1 and C_2 for the MTBF equation is relatively easy. C2 defines the slope of the line. Once C_2 is defined, C_1 can be determined. When $F_{DATA} = 0.5 \times F_{CLOCK}$ (representing a transition on every synchronizing clock edge), the equations reduce to

$$C_2 = \ln (MTBF_2 - MTBF_1) / \Delta T_2 - \Delta T_1$$

$$C_1 = 2 \times e^{(C_2 \times \Delta T)} / MTBF \times f^2$$

Table 1 summarizes the values of C_1 and C_2 for Altera's EPLDs as well as alternative devices. From these values, MTBF calculations can be made for a particular system/clock rate/device combination.

Table 1. Metastability Equation Constants vs. Device

Device	C_1	C_2
EP630	3.081E+52	5.415
EP610	4.567E+12	1.919
EP600	1.916E+22	1.340
EP1810	3.529E+38	2.068
EPM5016/EPM5032	4.981E+20	2.482
EPM5064	4.747E+11	1.641
74F74	6.100E+21	4.000
74LS74	5.270	0.5081

The following example assumes use of an EPM5032 in a synchronizing application and requires MTBF of one year (approximately 3×10^7 seconds). The system clock rate is 10 MHz, while the input to be synchronized has a frequency of 2 MHz. To calculate the minimum settling time allowance required to assure the specified MTBF, the constants shown below are used for the EPM5032 to solve the metastability equation:

$$MTBF = [F_{CLOCK} \times F_{DATA} \times C_1 \times e^{(-C_2 \times \Delta T)}]^{-1}$$

Settling time required is:

$$\Delta T = \ln [MTBF \times F_{CLOCK} \times F_{DATA} \times C_1] / C_2$$

Substitution provides the following result:

$$\Delta T = \ln [3 \times 10^7 \times 10 \times 10^6 \times 2 \times 10^6 \times 4.981E+20] / 2.482$$

$$= \ln [2.989 \times 10^{41}] / 2.482$$

$$= 95.5 / 2.482 = 39 \text{ ns}$$

To ensure an MTBF of one year with an EPM5032, the application should allow approximately 39 ns of settling time before the output of the EPM5032 synchronizing macrocell is evaluated or required to be stable elsewhere in the system. Similar calculations can be made for any combination of MTBFs and clock/data frequencies. Due to the logarithmic relationship between MTBF and settling time, dramatic changes in MTBF can be obtained from small changes in ΔT. For example, if the MTBF requirement is increased from one year to ten years in the calculations shown above, the ΔT allowance need only be increased by 2 ns to 41 ns! Increased design margin is relatively inexpensive and should be examined for each individual application.

Conclusion

Altera's EP-series EPLDs have metastability characteristics that make these devices superior to standard LS-TTL flip-flops when used in synchronizer applications. They also outperform typical low-power PAL devices substantially. Characteristics of EPM5000-series MAX EPLDs offer characteristics that are suitable for today's high-speed applications, and make these devices superior to ALS-TTL. Given the required settling time, any required MTBF may be predicted and obtained for circuits using Altera EPLDs.

When designing synchronizer circuits, it is prudent to provide adequate guardbands. Synchronization is probabilistic at best, and MTBF numbers only show mean or average times taken over a large sample. A circuit may have an MTBF of ten years, yet still has a probability of failure in its first few minutes of use. In all high-reliability applications, the potential of a metastable event can never be totally discounted.

Design Entry
for the EPS448 SAM EPLD

October 1990, ver. 1

Application Note 10A

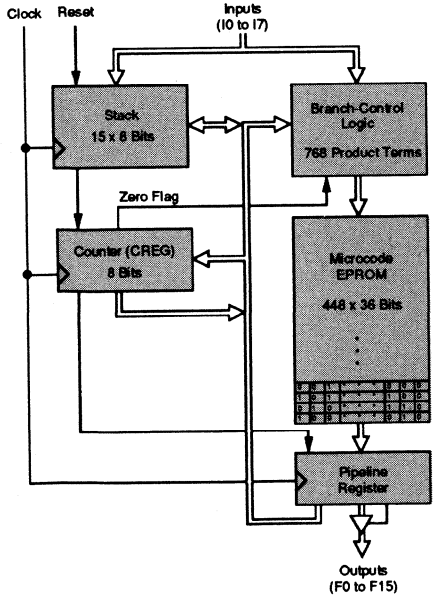
Introduction

This application note describes design entry methods for the Altera EPS448 Stand-Alone Microsequencer (SAM) EPLD. The following subjects are included:

- An overview of the SAM+PLUS Development System used in entering, compiling, simulating, and programming EPS448 designs
- A discussion of applications suited for the EPS448 EPLD
- Descriptions of the two entry languages supported by the EPS448 EPLD—the Altera State Machine Input Language (ASMILE) and the SAM Assembly Language (ASM)
- A 68020 microprocessor bus arbiter application example that demonstrates ASMILE entry
- A graphics controller application example that demonstrates ASM entry and cascading of multiple EPS448 EPLDs in large designs

Figure 1 shows a block diagram of the EPS448 EPLD. (Refer to the *EPS448 SAM EPLD: Stand-Alone Microsequencer Data Sheet* in this data book for more information.) A general knowledge of EPS448 architecture is assumed.

Figure 1. EPS448 Block Diagram

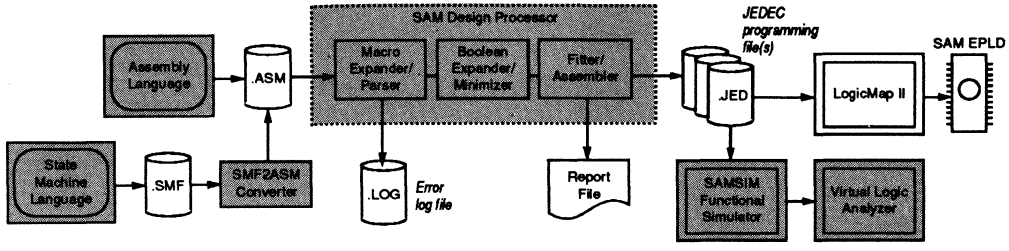


10

SAM+PLUS Development System

The SAM+PLUS Development System (Figure 2) provides an efficient PC-based method for entering and compiling EPS448 designs. SAM+PLUS also allows interactive functional simulation for rapid verification of design operation. PC-compatible programming hardware and LogicMap II software allow EPLD programming right at the designer's desk. The accelerated design process that SAM+PLUS provides is very helpful because control logic is often difficult to design and design changes are common.

Figure 2. SAM+PLUS Development System



SAM+PLUS supports two design entry methods:

- ASMILE — a state machine input language
- ASM — a microassembly language

To create a design with either language, the designer first enters the design file with any standard text editor. The file is then submitted to the SAM Design Processor (SDP), which converts it to ASM format if it is in ASMILE format. The SDP automatically minimizes transition equations in the ASM file and generates a standard JEDEC file for simulating the design and programming the EPS448 EPLD. The SDP also generates a Report File that lists total resources consumed, absolute memory assignments of microassembler instructions, and pin assignments.

The SAM Simulator (SAMSIM) provides functional testing of EPS448 designs, including multi-EPLD applications with horizontal cascading. The Virtual Logic Analyzer (VLA) in SAMSIM provides the designer with a graphical display of the simulation results. In addition, the designer may print out a hard-copy waveform output file.

Choosing EPS448 Applications

The EPS448 architecture supports high-performance synchronous control applications. It has a classic Moore machine architecture, i.e., all outputs are asserted synchronously with respect to the clock. All inputs must also obey a required setup time (t_{SU}) relative to the clock input.

Applications with the following characteristics are most likely to fit into a single EPS448 EPLD:

- ❒ Operating frequency of up to 30 MHz
- ❒ Synchronous operation
- ❒ Up to 8 control inputs exclusive of Clock and nRESET
- ❒ Up to 16 control outputs
- ❒ Up to 256 primary microcode locations
- ❒ Up to 64 multiway-branch microcode locations
- ❒ Transition expressions reducible to four product terms per **IF-THEN** expression

Horizontal cascading is used to obtain more than 16 outputs in an EPS448 design. Similarly, EPS448 EPLDs may be vertically cascaded—sharing a common output bus—if greater microcode depth is required. Horizontal and vertical cascading may be used simultaneously to increase capacity in both dimensions. For example, SAM+PLUS supports horizontal cascading of up to 8 EPS448 EPLDs, for a total output count of 128 lines.

ASMILE Syntax

The ASMILE file consists of the following sections (sections enclosed in brackets are optional):

```
[Header]
PART:
INPUTS:
OUTPUTS:
[EQUATIONS:]
MACHINE:
    [CLOCK:]
    STATES:
    Transition Specifications
END$
```

Note that the ASMILE input format is very similar to the State Machine (SMF) format used with A+PLUS and Altera's EP-series EPLDs.

ASMILE files may be entered with any standard text editor in non-document mode. (Format control characters inserted in document mode are interpreted as syntax errors during compilation.) Filenames have the extension **.SMF**.

The case of characters in the ASMILE file is significant. For example, the names **RWB** and **rwb** are not the same. Comments enclosed in percent symbols (%) may be inserted freely into the source code as shown in the following example:

```
% This is a Comment %
```

Header Section

The Header Section contains design identification information. Typical information includes the designer's name and company, date, design number and revision, and other comments.

Part Section

The Part Section of the ASMILE file (keyword **PART:**) specifies the EPS448 EPLD as the target EPLD for the application.

Inputs Section

The Inputs Section (keyword **INPUTS:**) defines all external inputs to the design and optionally assigns pin numbers to the inputs. SAM+PLUS automatically assigns any remaining pins. Pin assignments are specified in the following format:

```
pin_name @pin_number
```

Only user-defined inputs can appear in the Inputs Section, e.g., the dedicated Clock and nRESET inputs to the EPS448 EPLD are not included.

Outputs Section

The Outputs Section (keyword **OUTPUTS:**) contains a list of all outputs from the design and optional pin assignments. Output pin assignment syntax is the same as for input pins.

Equations Section

The Equations Section of the ASMILE file (keyword **EQUATIONS:**) is used to define intermediate equations to be used later in the design. For example, the following equation can be defined in the Equations Section:

```
EventClk = I1 * /I4 + I3 * I6 * /I7
```

The designer can then use **EventClk** in an **IF-THEN** statement later in the file instead of entering the actual equation.

Machine Section

The Machine Section of the ASMILE (keyword **MACHINE:**) file specifies a state machine's name and the state, output, and transition definitions required for the EPS448 EPLD. It has the following structure:

State machine declaration
- Clock Subsection (optional)
- States Subsection
- Transitions Subsection

State Machine Declaration

This declaration gives the name of the state machine. It has the following format:

```
MACHINE: machine_name
```

Clock Subsection

The optional Clock Subsection (keyword **CLOCK:**) specifies the synchronous clock source for the EPS448 EPLD. It is used primarily for documentation purposes.

States Subsection

The States Subsection (keyword **STATES:**) specifies all states in the machine and the outputs corresponding to the states. The States Subsection has the following format:

```
STATES:        [output_name_1...output_name_n]
state name     [output_value_list]
```

The first line contains a list of output names enclosed in brackets and separated by white space. Each subsequent line contains a state name followed by a binary string enclosed in brackets, which specifies all output values provided when the machine is in that state. For example:

```
STATES:    [ A B C D ]
S0         [ 0 0 0 0 ]
S1         [ 0 1 1 0 ]
S2         [ 1 0 0 0 ]
S3         [ 0 0 0 1 ]
```

This States Subsection specifies a machine with four outputs **A**, **B**, **C**, and **D**. State **S0** has all outputs low; **S1** takes **B** and **C** to logic high; **S2** has only output **A** high; and **S3** has only output **D** high.

Transitions Subsection

The Transitions Subsection in an ASMILE file (no keyword) has the following format:

```
state_name: transition_specification
```

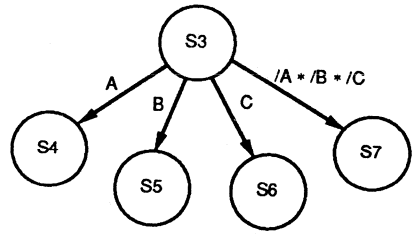
Every state in the machine must have a transition specification that specifies successor states with either unconditional (e.g., **S0: S2**) or conditional (**IF–THEN**) statements.

The first state name in the Transitions Subsection is defined as the initial state of the machine coming out of reset. This state name has special significance as an "inactive" or passive machine state. The position of other transition specifications is not significant.

IF-THEN Statements

The EPS448 architecture implements the user-defined state transition specifications in the branch-control logic block. This block allows up to 64 complex branching expressions to be specified in a single machine. (Up to 192 unconditional state transitions can be specified for a single EPS448 EPLD.) Multiway branching is shown in Figure 3.

Figure 3. EPS448 Multiway Branch



Each **IF-THEN** statement may be a function of any of the eight EPS448 external inputs and may contain up to four product terms after logic minimization. For most designs, these quantities should be sufficient. However, a tradeoff between the number of branch destinations and the number of product terms per destination can be made by pointing multiple **IF-THEN** statements to the same destination. For example, the following expression provides a three-way branch with up to eight product terms available for the specification of transitions to state **S1**:

```
S0  IF (cond1) THEN S1
    IF (cond2) THEN S1
    IF (cond3) THEN S2
    S3
```

Order of IF-THEN Statements

Order is important in **IF-THEN** statements and can determine the machine flow. Transition specifications need not be mutually exclusive in expressions. For example, the following expression at first appears ambiguous:

```
S0: IF I1 * I2 + I5 THEN S1
    IF I5 * I6 + I4 * /I3 THEN S2
    IF I4 THEN S3
    S4
```

If EPS448 inputs **I5** and **I6** both become true during **S0**, either **S1** or **S2** might be the next state. However, the EPS448 priority logic determines the next state on the basis of transition order. Since the **S1** transition is specified before the **S2** transition, it is the next state entered. Similarly, if **I4 * /I3** becomes valid, **S2** is the next state entered before **S3**. This precedence-resolving ability is built into the EPS448 EPLD, which uses a hardware priority-encoder to select the next-state transition. This capability not only resolves conflicts, but can also be exploited in the design to prioritize transitions.

Default Transitions

Another feature of the **IF-THEN** syntax is the implicit default transition. In the previous example, **S4** is the next state entered if **S1**, **S2**, and **S3** are not selected. This feature can reduce design effort and resource requirements substantially, since default transitions do not have to be defined as the negation of non-default transitions. Such inverted expressions tend to consume logic product terms or resources quickly. For example, the following transition specification is valid in ASMILE:

```
S0: IF I1 * I2 + I5 * /I7 + I0 THEN S1
    IF I3 + /I6 * I4 THEN S2
    IF I2 * I3 * I4 * I5 * /I7 THEN S3
    S4
```

If the ASMILE syntax did not support default transitions, the default transition, **S4**, would have to be explicitly defined as shown in the following unminimized example:

```
IF /(I1 * I2 + I5 * /I7 + I0) * /(I3 + /I6 * I4)
   * /(I2 * I3 * I4 * I5 * /I7) THEN S4
```

End Statement

Every ASMILE source file must terminate with the **END$** statement.

ASMILE Design Example

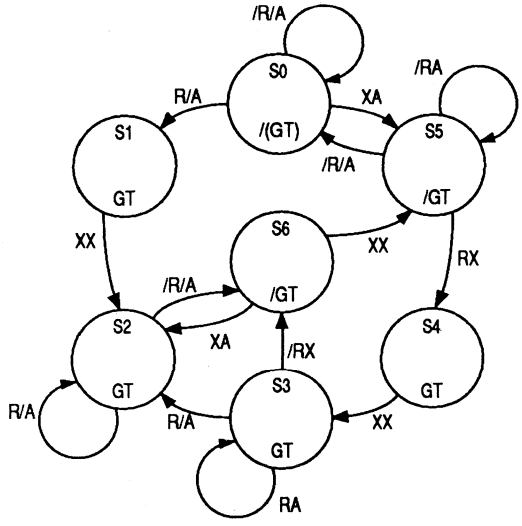
A 68020 microprocessor bus arbiter state machine example is used to illustrate ASMILE design entry. This example assumes knowledge of the 68020 bus exchange protocol. Table 1 shows the flow of the bus arbiter control function. The state machine controls the handshakes between the processor and the bus masters.

Table 1. Bus Arbiter Operation

States	Processor and Requesting Bus Master Actions
S0	Bus master asserts request.
S1 & S2	Processor asserts grant. Bus master arbitrates (if required) among multiple requests. Bus master waits for completion of current cycle.
S6 & S5	Next bus master asserts acknowledge (ACK). Next bus master deasserts request. Processor deasserts grant (waits for ACK to be deasserted). Bus master performs bus operations. Bus master deasserts ACK.
S0	Processor resumes operation.
S4 & S3	Processor rearbitrates.

Figure 4 shows a state machine diagram for the bus arbiter. The 68020-based system runs at 25 MHz, and the bus arbiter machine runs with a 40-ns clock cycle.

Figure 4. Bus Arbiter State Machine Diagram



R — Bus request input
A — Bus grant acknowledge input
G — Bus grant output
T — Tri-state control to bus-control logic
X — Don't care

Three signal lines on a 68020 bus—**REQUEST**, **GRANT**, and **ACKNOWLEDGE**—define the handshake required to arbitrate bus exchanges between multiple bus masters. **S0** represents the "normal," active state of the processor; **S1** and **S2** correspond to the grant phase; **S5** and **S6** to the acknowledge phase; and **S3** and **S4** to the arbitration phase if requests are pending at the end of the current bus exchange.

The file shown in Figure 5, **68020ARB.SMF**, is the actual ASMILE file entered for the state machine in Figure 4. In the Outputs and States sections, output variables **OS0** through **OS6** are defined. Each variable is valid only for one unique state. As the design is simulated, these variables indicate the state of the machine.

Figure 5. 68020 Bus Arbiter State Machine File (68020ARB.SMF)

```
10/1/90 68020 Bus Arbiter for EPS448
% This description uses IF-THEN transition specifications %

PART:   EPS448

% Pin assignments are optional %
INPUTS:  REQUEST  ACK
OUTPUTS: GRANT  TRISTATE  OS0 OS1 OS2 OS3 OS4 OS5 OS6

MACHINE: BUSARBITER
CLOCK:   CLK

% STATES gives the output value mapping %
STATES:  [GRANT  TRISTATE  OS0  OS1  OS2  OS3  OS4  OS5  OS6]
    S0   [0      0         1    0    0    0    0    0     0]
    S1   [1      1         0    1    0    0    0    0     0]
    S2   [1      1         0    0    1    0    0    0     0]
    S3   [1      1         0    0    0    1    0    0     0]
    S4   [1      1         0    0    0    0    1    0     0]
    S5   [0      1         0    0    0    0    0    1     0]
    S6   [0      1         0    0    0    0    0    0     1]

% Transition Specifications follow %
S0: IF REQUEST*/ACK THEN S1
    IF ACK THEN S5
    S0
S1: S2
S2: IF /REQUEST*/ACK + ACK THEN S6
    S2
S3: IF /REQUEST THEN S6
    IF REQUEST*/ACK THEN S2
    S3
S4: S3
S5: IF REQUEST THEN S4
    IF /REQUEST*/ACK THEN S0
    S5
S6: S5

END$
```

ASMILE Design Entry & Compilation

The SAM Design Processor (SDP) is used to compile the design. First, the ASMILE file is automatically translated into an ASM file. Transition equations are then automatically minimized, and "object code" is generated for the EPS448 EPLD. Finally, a JEDEC programming file (**.JED**) is generated. The JEDEC file can also be used as a design template in functional simulation. A report file (**.RPT**) with the results of the compilation process is also generated. Figure 6 shows key portions of this file.

10

Figure 6. Bus Arbiter Report File (68020ARB.RPT) Excerpts

```
SAM Design Processor Utilization Report
Version 2.02 11/13/89
***** Design implemented successfully

10/1/90 68020 Bus Arbiter for SAM
% This description uses IF-THEN Transition Specifications %

                              EPS448
                          .-------_-------.
                RESERVED :  1         28 : RESERVED
                     GND :  2         27 : RESERVED
                     GND :  3         26 : RESERVED
                     GND :  4         25 : GRANT
                     GND :  5         24 : TRISTATE
                   CLOCK :  6         23 : OS0
                     VCC :  7         22 : OS1
                  nRESET :  8         21 : GND
                     GND :  9         20 : OS2
                     GND : 10         19 : OS3
                 REQUEST : 11         18 : OS4
                     ACK : 12         17 : OS5
                RESERVED : 13         16 : OS6
                RESERVED : 14         15 : RESERVED
                          `---------------'

***** DESIGN LISTING                                    c:\samwk\68020.rpt

PART:
        EPS448
INPUTS:
        REQUEST@11, ACK@12
OUTPUTS:
        GRANT@25, TRISTATE@24, OS0@23, OS1@22, OS2@20, OS3@19, OS4@18, OS5@17, OS6@16
PINS:
DEFAULT:
        [000000000]
PROGRAM:
        0D:       [001000000] JUMP S0;
        192D: S0: IF REQUEST * ACK' THEN [110100000] JUMP S1;
                  ELSEIF ACK THEN [010000010] JUMP S5;
                  ELSE [001000000] JUMP S0;
        1D:   S1: [110010000] JUMP S2;
        193D: S2: IF REQUEST' + ACK THEN [010000001] JUMP S6;
                  ELSE [110010000] JUMP S2;
        194D: S3: IF REQUEST' THEN [010000001] JUMP S6;
                  ELSEIF REQUEST * ACK' THEN [110010000] JUMP S2;
                  ELSE [110001000] JUMP S3;
        2D:   S4: [110001000] JUMP S3;
        195D: S5: IF REQUEST THEN [110000100] JUMP S4;
                  ELSEIF REQUEST' * ACK' THEN [001000000] JUMP S0;
                  ELSE [010000010] JUMP S5;
        3D:   S6: [010000010] JUMP S5;
END$

***** PART UTILIZATION                                  c:\samwk\68020.rpt

        4/192 Unconditional Branches ( 2.08%)
        4/ 64 Conditional Branches ( 6.25%)
        0 Warnings 0 Fatal errors
```

ASMILE Design Simulation

After a design has been successfully processed, the user can specify input stimuli in a variety of formats and observe the EPLD response quickly and effectively with the SAMSIM Functional Simulator. SAMSIM supports both hard-copy and on-screen Virtual Logic Analyzer (VLA) output formats. Split windows, multiple zoom levels, and delta time display are a few of the capabilities of this interactive display mode.

SAMSIM supports both interactive and Command File input. Figure 7 shows a simple input Command File for the 68020 bus arbiter design. The first line specifies the source JEDEC file. The next two lines contain logic sequences for the two machine inputs. The **PATTERN CREATE** command specifies a sequence of input logic levels to be applied to selected nodes. The notation **()*n**, where **n** is an integer, is used to hold the indicated logic value on the associated input for **n** clock cycles. The fourth line, **SIMULATE 41**, instructs SAMSIM to run the simulation for 41 clock cycles. The interactive display is invoked in the last line with the **VIEW** command. Typically, Command Files are given the design name with the extension **.CMD** (in this case, **68020ARB.CMD**).

Figure 7. Bus Arbiter Command File for SAMSIM (68020ARB.CMD)

```
JEDEC 68020ARB
PATTERN CREATE REQUEST = (0)*3 1 1 1 1 (0)*12 1 1 1 1 1 0 0 (1)*7 (0)*5
PATTERN CREATE ACK =     (0)*5 (1)*8 (0)*10 (1)*6 (0)*2 (1)*6 (0)*4
SIMULATE 41
VIEW
```

The following list describes other SAMSIM commands that are not used in this example:

TRACE Dumps the entire state of the machine (inputs, outputs, internal register, etc.) for each clock executed.

GROUP Specifies the logical grouping of signals for easy observation or input vector specification.

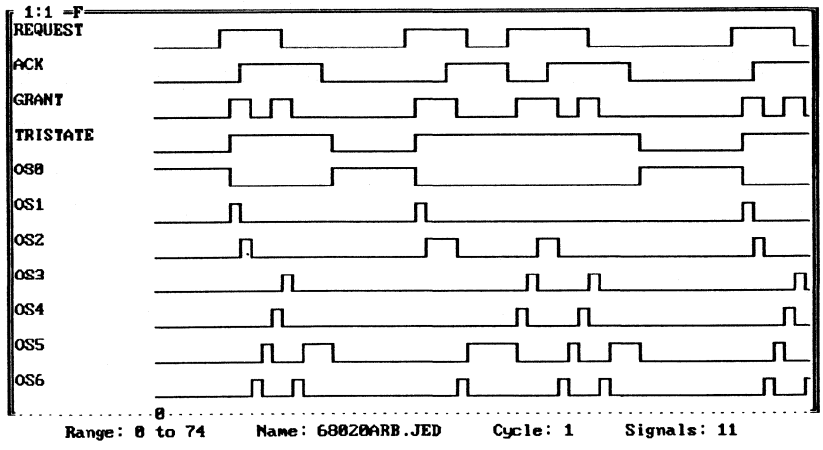
SET Modifies the values of the internal counter, stack, etc.

LINK Logically links EPLD pins for simulation purposes.

RADIX Defines the default radix for all SAMSIM commands. The radix may be binary, hexadecimal, or decimal.

Figure 8 shows the VLA screen after running SAMSIM with the Command File in Figure 7. The screen displays the input stimuli to the bus arbiter design and the resulting state machine operation.

Figure 8. VLA Screen for Bus Arbiter Design



The initial input stimulus applied to the EPS448 design shows a straightforward bus exchange between the 68020 and another bus master. This exchange corresponds to the first **REQUEST/GRANT/ACK** sequence. Upon detecting a **REQUEST**, the 68020 asserts its **TRISTATE** line and issues a **GRANT** pulse, allowing the new bus master to assume control. The alternate bus master asserts **ACK** when it detects that the bus has been granted. When **ACK** finally drops, the 68020 can resume control. The second sequence involves not just a single initial **REQUEST** (bus master 1), but a second **REQUEST** from another bus master (bus master 2) during the time bus master 1 has control. As a result, the 68020 must generate a new **GRANT** pulse during **S4** to **S2**, and hand over bus control to bus master 2 when bus master 1 is finished (i.e., when **ACK** is dropped). When bus master 3 is finished and no requests are pending, the 68020 finally takes control of the bus again and **TRISTATE** goes low.

ASM Syntax

The ASM file consists of the following sections (sections enclosed in brackets are optional):

```
[Header]
PART:
INPUTS:
OUTPUTS:
[PINS:]
[DEFAULT:]
[MACROS:]
[EQUATIONS:]
PROGRAM:
END$
```

ASM files may be entered with any standard text editor in non-document mode. ASM is case-sensitive. It allows comments that are enclosed in percent symbols (%). Filenames are terminated with the extension **.ASM**.

Header

The Header Section contains design identification information. Typical information includes the designer's name and company, date, design number and revision, and other comments.

Part Section

The Part Section of the ASM file (keyword **PART:**) specifies the EPS448 as the target EPLD for the application. Multiple EPS448 EPLDs may be specified for designs requiring more outputs than a single device can supply. The SAM+PLUS software supports horizontal cascading of EPLDs at a source code level. (See the *EPS448 SAM EPLD: Stand-Alone Microsequencer Data Sheet* for more information.)

Inputs Section

The single Inputs Section of the ASM file (keyword **INPUTS:**) defines all external inputs to the design, as well as any required user pin assignments. Pin assignments are specified in the following format:

```
pin_name @pin_number
```

Only user-defined inputs should appear in the Inputs Section. All design inputs must be common in a horizontally cascaded design. A source file can contain only as many inputs as a single EPS448.

Outputs Section

The Outputs Section(s) of the ASM file (keyword **OUTPUTS:**) lists all outputs from the design and pin assignments. Output pin assignment syntax is the same as that of input pin assignments. If multiple EPS448 EPLDs are specified in the Part Section of the design file, multiple Outputs Sections must be inserted in the ASM file, one for each EPS448 component.

Pins Section

The optional Pins Section (keyword **PINS:**) allows external variable names to be mapped onto internal variable names. For example, an active-low system signal called **/WR** might be entered into the transition specifications. To keep the logical sense of such specifications clear, it is helpful to change all active-low external signals to equivalent active-high names internally.

Defaults Section

The optional Default Section (keyword **DEFAULT:**) specifies a default output combination that can be used whenever the output string is not explicitly defined in an instruction. In a single EPS448 specification, the syntax is **DEFAULT: [00...0n]**, where **00...0n** represents a binary string corresponding to the **n** outputs specified for the EPS448 design. Default output values are matched to output pins in the order in which they appear in the Outputs Section. If multiple Outputs Sections appear in a cascaded EPS448 application, the binary string is increased in width to accommodate this change.

Macros Section

The optional Macros Section (keyword **MACROS:**) defines strings that can be substituted universally throughout the ASM source code. Instruction mnemonics may be redefined for efficiency or clarity, or binary output strings may be redefined to have alphanumeric labels. An example of a macro definition is **REG1TOALU = "0101111001100000"**.

Embedded strings are not macro-substituted. To be recognized, macro instances must be delimited by white space. For example, the macro substitution **REG = "0110"** causes the string **0110** to be substituted into **[REG ALU OP] CONTINUE**, but not into **[BREG4 AL OP] CONTINUE**.

Equations Section

The optional Equations Section of the ASM file (keyword **EQUATIONS:**) defines intermediate equations to be used later in the design. For example, an equation such as **EventClk = I1 * /I4 + I3 * I6 * /I7** could be defined in the Equations Section. The designer could then use **EventClk** in an **IF-THEN** statement, such as **IF EventClk THEN JUMP START** instead of entering the actual equation.

Program Section

The Program Section of the ASM file (keyword **PROGRAM:**) specifies the sequence of instructions to be executed and the associated outputs required from the EPS448 EPLD. The format of a basic instruction specification in the Program Section is as follows:

```
label: [output_spec] opcode;
```

The **label** is an optional alphanumeric string that can be used to identify the instruction in branching expressions. **[output_spec]** represents an actual numeric string (in binary, hexadecimal, or decimal format), a macro substitution, or the character **Z** for tri-state (high-impedance) output pins. Hexadecimal and decimal strings are defined by a string of valid digits of correct length, followed by **H** or **D**, respectively. In horizontally cascaded applications, each output is enclosed in brackets. The output specification defined in the Default Section is assumed whenever it has length zero (i.e., empty brackets (**[ ]**) imply the default output specification).

End Statement

Every ASM source file must terminate with **END$**.

Multiway Branch Syntax

To specify multiway branching in the ASM file, a complex expression of the following form is used:

```
IF     (expression1) THEN [output_value] (instruction1)
ELSEIF (expression2) THEN [output_value] (instruction2)
ELSEIF (expression3) THEN [output_value] (instruction3)
ELSE                      [output_value] (instruction4)
```

For example, a complex instruction of this type might appear as follows:

```
IF     I0*I1*I5*/I7 + I3*I4 + I6*/I0 + /I3*/I1 THEN
       [11110011100100000] CALL label1 RETURNTO label2;
ELSEIF I3*/I2 + I5*I6 + /I0*I4*I1 THEN
       [10110000011100011] LOADC 255 GOTO label3;
ELSEIF I4*I6*I0 THEN
       PUSH 15 GOTO label4;
ELSE   [1111111100000001] PUSH1 GOTO label5;
```

Each expression can be a function of any of the eight EPS448 external inputs and can contain up to four product terms.

If more than four product terms are needed to define a transition from one state to another, it is possible to trade off product-term counts for multiway branch destinations. In the following example, **expression1** and **expression2** could consist of four product-term expressions, resulting in eight product terms that could be used to specify the transition to **START**:

```
IF     (expression1) THEN [] JUMP START;
ELSEIF (expression2) THEN [] JUMP START;
ELSEIF (expression3) THEN [] JUMP NEXT1;
ELSE                         JUMP NEXT2;
```

Note the inherent priority scheme in the previous statements. The EPS448 architecture physically implements such a priority scheme in the branch-control logic block.

ASM Opcodes

Thirteen easy-to-use micro-instructions (called opcodes) are built into the ASM syntax. This compact instruction set allows the designer to take full advantage of the advanced features of the EPS448 EPLD, such as the counter and the stack. Three opcode examples are shown here. For a complete list of opcodes, consult the *EPS448 SAM EPLD: Stand-Alone Microsequencer Data Sheet* in this data book.

LOOPNZ (label1) ONZERO (label2) This opcode is useful for one-instruction timing and delay loops. If the count register (CREG) is zero, then the instruction at **label2** is executed. Otherwise CREG is decremented and the instruction at **label1** is executed.

10

POPC GOTO (label1) In this opcode, the top-of-stack is popped into CREG, and the instruction at **label1** is executed.

LOADC (constant1) GOTO (label1) This opcode loads CREG value **constant1**, and executes the instruction at **label1**.

ASM Design Example

A high-performance graphics controller may be used to show the design process with ASM. In this application, two EPS448 EPLDs are horizontally cascaded to generate the control outputs for a graphics subsystem. This subsystem provides primitive graphics-drawing capability for a larger microprocessor-based system.

Figure 9 shows a typical 8086 microprocessor-based system. Beneath the address and data buses is the graphics subsystem to be controlled by the EPS448 EPLDs. The graphics subsystem consists of the following primary elements:

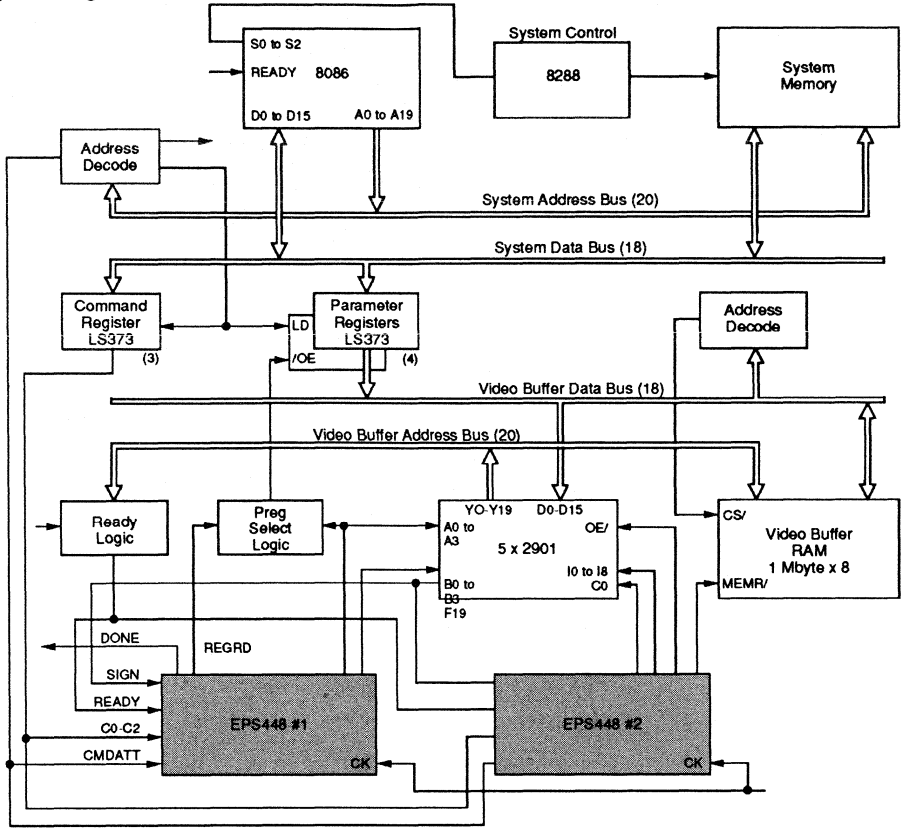
- One 1-Mbyte, high-speed static-RAM video-frame buffer with individual pixel-addressing capability
- Five 2901 bit-slice elements that are used to construct a 20-bit ALU/data path engine
- Two EPS448 EPLDs to provide overall control within the subsystem

This basic graphics engine is a user-microcodable design that can support primitive graphics, such as lines, polygons, and conic sections. This example discusses a single primitive drawing operation that draws circles of arbitrary radii and origins into the frame buffer.

The pair of EPS448 EPLDs must be able to execute the following subfunctions to serve as a controller for this subsystem:

- Read commands issued by the main microprocessor
- Transfer parameters associated with commands to the register file in the 2901 bit slice elements
- Initialize constant registers in 2901 bit slice elements to specified values for the algorithm
- Compute values for pixels on the circle as a function of the specified radius for the first octant (assuming the circle origin is at (0,0))
- Translate (x,y) coordinates into RAM addresses
- Reflect circle pixel coordinates into the remaining seven octants
- Translate pixel coordinates relative to their actual origin
- Perform video-buffer write to all specified pixel addresses
- Issue a done-interrupt to the main processor

Figure 9. EPS448 Graphics Engine



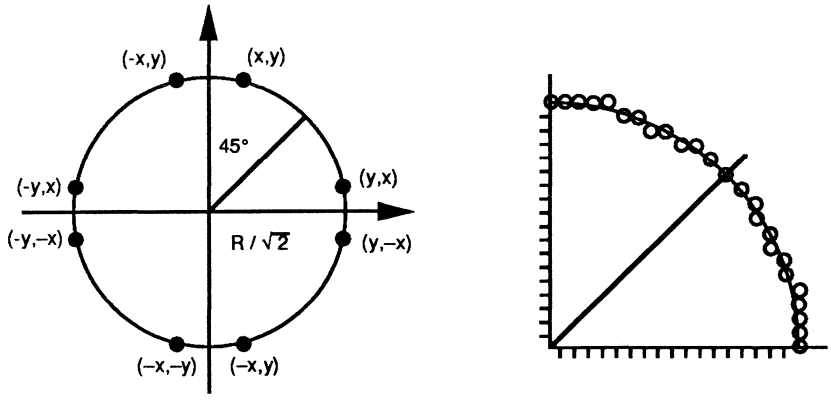
These activities are performed independently of the main microprocessor, freeing it up for other tasks. These other tasks fall into two general categories: controlling bus transfers between elements (registers, ALU, RAM, etc.) and sequencing 2901 ALU computations that generate the pixel addresses for drawing the circle.

Circle Drawing Algorithm

An algorithm based on a methodology developed by Bresenham is used to draw the circle. It uses the symmetry of a circle to calculate the circle points in the first octant and to reflect those coordinates into the other seven octants. (See Figure 10.) In other words, for a given pixel location (x,y), points $(-x,y)$, $(x,-y)$, $(-x,-y)$, (y,x), $(-y,x)$, $(y,-x)$, and $(-x,-y)$ are drawn. After drawing the first octant's points, only two possible choices exist for the next pixel location: a horizontal move (i.e., increment x) or a diagonal move (i.e., increment x and y). The problem is deciding which of the two to pick next, based on the current location.

10

Figure 10. Circle Symmetry



The basic algorithm is shown in Figure 11. The best match between the actual pixel coordinates and the ideal circle points can be obtained by checking an error term equal to the difference in distance from the circle's center to each of the two next-pixel choices. The sign of the term indicates which point will obtain the best fit.

Figure 11. Circle Drawing Algorithm

```
procedure circle (radius, value : integer ;
   var x, y, d  :  integer ;
begin
   x  := 0 ;
   y  := radius ;
   d  := 3 -2 * radius ;
   while x < y do begin
      CircleDraw (x, y, value);
      if d < 0
         then d := d + 4 * x + 6
         else begin
            d := d + 4 * (x - y) + 10 ;
            y := y - 1
         end
      x  := x + 1
   end
   if x = y then CircleDraw (x,y,value) ;
end
```

Timing Considerations

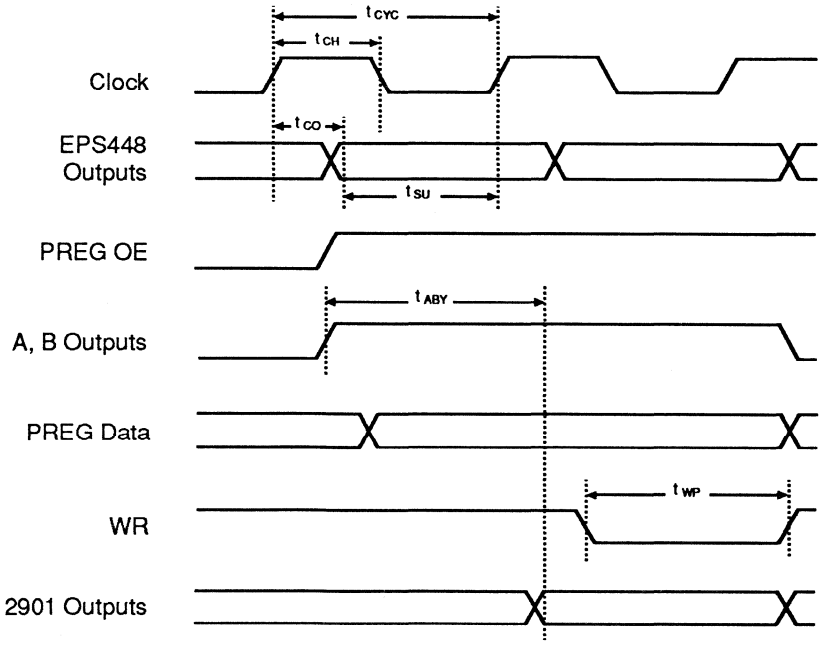
EPS448 timing analysis is straightforward. All times are relative to the synchronous clock input. The t_{SU} parameter specifies the minimum setup time for inputs to gain recognition at the next clock edge, while t_{CO} specifies the clock-to-output delay for user-configured output pins. (Output tri-state and enable times are specified as t_{CZ}, but are not relevant in this application because outputs are always enabled.)

Although the EPS448 EPLD can handle clock rates of 30 MHz, this graphics subsystem design example is driven by a 20-MHz clock. The EPS448 control outputs reflect a t_{CO} of 16 ns, while inputs must obey a 16.5-ns t_{SU} relative to the clock edge.

High-speed static RAM (SRAM) is used for the video-frame buffer for two reasons. First, the memory must be fast enough to keep up with the EPS448 EPLD's high-speed bus cycles. Second, SRAM requires no refresh cycles, unlike dynamic RAM. Thus, more time is available to perform buffer drawing functions.

Memory consists of 8,000 × 8 CMOS SRAM components with an access time of 45 ns, and a minimum write-pulse width of 30 ns. The CMOS 2901 bit-slices require a 30-ns propagation delay from the **A** and **B** register address inputs to valid **Y** output, and a 10-ns setup time prior to the clock's high-to-low transition on **A** and **B** inputs. A timing diagram is shown in Figure 12.

Figure 12. Graphics-Controlled Timing of Primary EPS448



10

The bus cycle uses a two-clock approach. During the first cycle, the 2901 generates a pixel address to be set. During the second cycle, the actual write-pulse is generated by the EPS448 EPLDs to write the frame buffer.

Operations performed entirely within the 2901 bit-slices (e.g., register transfers, ALU operations) are executed in a single clock cycle. Carry-look-ahead circuitry is used with the 2901 bit-slices to improve arithmetic computation times, although it is not explicitly shown in the block diagram.

ASM Example File

Figure 13 shows the source ASM code for the basic circle-drawing process. The following features are included:

- Two EPS448 EPLDs are used in a horizontal cascade configuration.
- Extensive **MACRO** definitions ease design entry and allow the use of user- and 2901-specified mnemonics.
- Two subroutines, **CircPix** and **Trans**, are invoked multiple times to draw the circle pixels. **CircPix** reflects the pixels into all octants of the circle, while **Trans** translates the pixels relative to the actual circle origin and runs the memory-update cycle. These functions use the stack and subrouting resources on the EPS448 EPLD.
- The display is assumed to be 1024 × 1024 pixels, so *x* and *y* pixel coordinates are converted to SRAM address locations by multiplying the *y* coordinate by 1023 and adding the *x* coordinate.
- The signal **CmdAtt** is an input to the EPS448 EPLDs from the main processor. It signals that all parameters are loaded to the parameter registers, and that a circle-drawing operation should be executed. **DoneInt** is a signal from EPS448 to the processor, asserted when the drawing operation is complete.

Figure 13. Circle Drawing Routine (CIRCDRAW.ASM) (Part 1 of 5)

```
% Circle Drawing Routine for EPS448 %
PART: EPS448 EPS448

% SAM Control Output Lines              Inputs                  %
%  A & B Fields (2901)   - 8        C0-2          - 3          %
%  I0-I8 (2901)          - 9        CmdAtt        - 1          %
%  OE (2901)             - 1        Sign          - 1          %
%  Done                  - 1                                   %
%  Cn (2901)             - 1                                   %
%  Wr                    - 1                                   %
%  ALE                   - 1                                   %
%  Rd                    - 1                                   %
%  RegRd                 - 1                                   %
```

Figure 13. Circle Drawing Routine (CIRCDRAW.ASM) (Part 2 of 5)

```
INPUTS: C0,C1,C2,CmdAtt,Sign

OUTPUTS:  A0,A1,A2,A3,B0,B1,B2,B3,I2,I1,I0,I5,I4,I3,I8,I7
OUTPUTS:  I6,Rd,Wr,ALE,RegRd,OE,Cn,Done

DEFAULT: [0000 0000 0000 0000 1110 0100]

MACROS:   CONT = "CONTINUE"

% A & B Fields %
RadiusReg = "0001"
Reg1  = "0001"
Reg2  = "0010"
Reg3  = "0011"
Reg4  = "0100"
Reg5  = "0101"
Reg6  = "0110"
Reg7  = "0111"
Reg8  = "1000"
Reg9  = "1001"
Reg10 = "1010"
Reg11 = "1011"
Reg12 = "1100"

% Source Control %
AQ = "000"
AB = "001"
ZQ = "010"
ZB = "011"
ZA = "100"
DA = "101"
DQ = "110"
DZ =  "111"

% Function %
ADD   = "000"
SUBR  = "001"
SUBS  = "010"
OR    = "011"
AND   = "100"
NOTRS = "101"
EXOR  = "110"
EXNOR = "111"

% Destination  Control %
QREG  = "000"
NOP   = "001"
RAMA  = "010"
RAMF  = "011"
RAMQD = "100"
RAMD  = "101"
RAMQU = "110"
RAMU  = "111"

% Bus Cycle  %
MemWr  = "10001"
RegWr  = "10011"
ALEcyc = "11100"
NoCyc  = "11000"
```

10

Figure 13. Circle Drawing Routine (CIRCDRAW.ASM) (Part 3 of 5)

```
% Misc %
Cn    = "1"
nCn   = "0"
Done  = "1"
nDone = "0"

EQUATIONS:

PROGRAM:

% Processor Initializes: %
% o Load Coloreg, Radius, X0, Y0 %
% o Issues DrawCirc Command %
WAIT:  IF CmdAtt*C0'*C1'*C2' THEN [] JUMP DOIT ;
       ELSE [] JUMP WAIT ;

% Move parameters from buffer to 2901 internal registers %
% Radius -> Reg1 (Y) %
DOIT:  [ Reg1 Reg1 AQ ADD NOP RegWr nCn nDone ] CONT ;
       [ Reg1 Reg1 AQ ADD NOP RegWr nCn nDone ] CONT ;

% X0 -> Reg2 %
[ Reg2 Reg2 AQ ADD NOP NoCyc nCn nDone ] CONT ;
[ Reg2 Reg2 AQ ADD NOP RegWr nCn nDone ] CONT ;
[ Reg2 Reg2 AQ ADD NOP RegWr nCn nDone ] CONT ;

% Y0 -> Reg3 %
[ Reg2 Reg2 AQ ADD NOP NoCyc nCn nDone ] CONT ;
[ Reg3 Reg3 AQ ADD NOP RegWr nCn nDone ] CONT ;
[ Reg3 Reg3 AQ ADD NOP RegWr nCn nDone ] CONT ;

% Load constants to 2901 registers %
% 0 -> Reg4 (X) (AND 0 & anything gives 0) %
[ Reg4 Reg4 ZB AND RAMF NoCyc nCn nDone ] CONT ;

% 3 -> Reg5 (d) % % Put "1" in Reg5 %
[ Reg4 Reg5 ZA ADD RAMF NoCyc Cn nDone ] CONT ;

% Shift Reg5 up 1 to give 2 %
[ Reg5 Reg5 ZB ADD RAMU NoCyc nCn nDone ] CONT ;

% While we have it, preload 2 into Reg9 %
[ Reg5 Reg9 ZA ADD RAMF NoCyc nCn nDone ] CONT ;

% Increment Reg5 to get 3 (whew!!) %
[ Reg5 Reg5 ZA ADD RAMF NoCyc Cn nDone ] CONT ;

% 6 -> Reg8 (const) - just shift 3 up one! %
% Load 1 in CREG to setup for next instruction %
[ Reg5 Reg8 ZA ADD RAMU NoCyc nCn nDone ] LOADC 1D ;

% 10 -> Reg9 (const) %
% Start by shifting Reg9 (now contains 2) up twice to get 8 %
% Reg6 (Temp register) %
SHIFTR9:  [ Reg9 Reg9 ZA ADD RAMU NoCyc nCn nDone ]
          LOOPNZ SHIFTR9 ;
```

Figure 13. Circle Drawing Routine (CIRCDRAW.ASM) (Part 4 of 5)

```
% Increment Reg9 twice to get 10 %
[ Reg9 Reg9 ZA ADD RAMF NoCyc Cn nDone ] CONT ;
[ Reg9 Reg9 ZA ADD RAMF NoCyc Cn nDone ] CONT ;

% Initializing done ! - Begin algorithm %
% d = 3 - 2*radius initially %
[ Reg1 Reg6 ZA ADD RAMU NoCyc nCn nDone ] CONT ;
[ Reg5 Reg6 AB SUBS RAMF NoCyc Cn nDone ] CONT ;

% If x >= y branch to finish up %
OUTERLOOP: [ Reg4 Reg1 AB SUBS RAMF NoCyc Cn nDone ] CONT ;
           IF Sign THEN [] JUMP DrawEnd ;

% Write pixels, translate origin & reflect to all octants %
           ELSE [] CALL CircPix ;

% Test d sign; if >= 0, use POS %
[ Reg5 Reg5 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
IF Sign THEN [] JUMP POS ;

% Compute d = d + 4*x + 6 %
% First 4*x %
ELSE [ Reg4 Reg6 ZA ADD RAMU NoCyc nCn nDone ] CONT ;
     [ Reg6 Reg6 ZA ADD RAMU NoCyc nCn nDone ] CONT ;

% Add 6 %
[ Reg8 Reg6 AB ADD RAMF NoCyc nCn nDone ] CONT ;
[ Reg6 Reg5 AB SUBS RAMF NoCyc Cn nDone ] JUMP IncX ;

% Compute d = d + 4*(x-y) + 10 %
% First x-y %
POS:    [ Reg1 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
        [ Reg4 Reg6 AB SUBS RAMF NoCyc Cn nDone ] LOADC 1D ;

% Then 4*(x-y) %
SHIFTR6:   [ Reg6 Reg6 ZA ADD RAMU NoCyc nCn nDone ]
LOOPNZ SHIFTR6 ;

% Add 10 %
[ Reg9 Reg6 AB ADD RAMF NoCyc nCn nDone ] CONT ;
[ Reg6 Reg5 AB ADD RAMF NoCyc nCn nDone ] CONT ;

% Decrement y %
[ Reg1 Reg1 ZA SUBR RAMF NoCyc nCn nDone ] CONT ;

% Increment x and repeat until x = y %
IncX:      [ Reg4 Reg4 ZA ADD RAMF NoCyc Cn nDone ]
           JUMP OUTERLOOP ;

% Last pixel write / ends octant with x = y (45 degrees) %
DrawEnd:   [] Call CircPix ;
           [] LOADC 16D ;

% Issue Done to processor for 16 clocks %
DoDone:    [ Reg1 Reg1 ZA ADD RAMF NoCyc nCn Done ]
           LOOPNZ DoDone ONZERO WAIT ;

% End Main Routine %
```

10

Figure 13. Circle Drawing Routine (CIRCDRAW.ASM) (Part 5 of 5)

```
% The CircPix routine reflects the pixel into all octants and
calls a routine that translates the pixel relative to x0,y0;
calculates the pixel address as addr = x + y*1023; and runs the
memory cycle. %

CircPix:    [ Reg4 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
            [ Reg1 Reg11 ZA ADD RAMF NoCyc nCn nDone ]
            CALL TRANS ;

% Reflect X to -X %
[ Reg4 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
[ Reg1 Reg11 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Swap X & Y %
[ Reg1 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CONT ;
[ Reg4 Reg11 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Swap -X & Y %
[ Reg4 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
[ Reg1 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Reflect Y %
[ Reg1 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
[ Reg4 Reg6 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Swap -Y & X %
[ Reg1 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
[ Reg4 Reg11 ZA ADD RAMF NoCyc nCn nDone ] CALL TRANS ;

% Reflect -X, -Y %
[ Reg4 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
[ Reg1 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CALL TRANS ;

% Swap -X & -Y %
[ Reg1 Reg6 ZA SUBS RAMF NoCyc Cn nDone ] CONT ;
[ Reg4 Reg11 ZA SUBS RAMF NoCyc Cn nDone ] CALL TRANS ;
[] RETURN ;

% The Trans routine translates relative to x0,y0 and runs the
memory update cycle %

TRANS:  [ Reg3 Reg11 AB ADD RAMF NoCyc nCn nDone ] CONT ;
        [ Reg2 Reg6 AB ADD RAMF NoCyc nCn nDone ] LOADC 10D ;
        [ Reg11 Reg12 ZA ADD RAMF NoCyc nCn nDone ] CONT ;

% Multiply y by 1024 %
MULT1024:   [ Reg11 Reg11 ZA ADD RAMU NoCyc nCn nDone ]
            LOOPNZ  MULT1024 ;

% Subtract y to effectively multiply by 1023 %
DONE1024:  [ Reg12 Reg11 AB SUBR RAMF NoCyc Cn nDone ] CONT ;

% Calculate address %
[ Reg6 Reg11 AB ADD RAMF NoCyc nCn nDone ] CONT ;

% Write pixel in buffer RAM %
RUNBUS: [ Reg11 Reg11 ZA ADD RAMF ALEcyc nCn nDone ] CONT ;
        [ Reg11 Reg11 ZA ADD RAMF MemWr nCn nDone ] RETURN ;
END$
```

The algorithm shown in Figure 13 uses the 2901 operations along with many of the internal addressing modes. Standard mnemonics have been used for the various source, destination, and operation specifiers. These control lines for 2901s are generated by the EPS448 EPLDs. The mnemonics and resulting 2901 functions may be found in any standard 2901 data sheet, available from many vendors.

ASM Design Compilation

The SAM+PLUS Design Processor (SDP) must be invoked to compile the **CIRCDRAW.ASM** file shown in Figure 13. Compilation is an automatic process that generates programming "object code" for the branch-control logic and microcode EPROM blocks on the EPS448 EPLD. In this case, two JEDEC 13
programming files with the extensions **.JD1** and **.JD2** are generated, since two EPLDs are required to implement the design. A Report File (with the extension **.RPT**) is also generated during compilation. This file describes the resources that have been used in the EPS448 EPLDs, pin assignments, and absolute locations within the microcode assigned to the instructions entered. Figure 14 shows the **CIRCDRAW.RPT** file. Note the assigned pin-outs for the two EPLDs as well as the substitution of absolute addresses for logical labels.

10

Figure 14. Report File for Circle Drawing Routine (CIRCDRAW.RPT) (Part 1 of 3)

```
***** Design Implemented Successfully
% Circle Drawing Routine for EPS448 %

                              EPS448
                          .-------_-------.
                      A0 :  1         28 : A1
                      NC :  2         27 : A2
                      NC :  3         26 : A3
                      NC :  4         25 : B0
                      C0 :  5         24 : B1
                   CLOCK :  6         23 : B2
                     VCC :  7         22 : B3
                   RESET :  8         21 : GND
                      C1 :  9         20 : I2
                      C2 : 10         19 : I1
                  CmdAtt : 11         18 : I0
                    Sign : 12         17 : I5
                      I7 : 13         16 : I4
                      I8 : 14         15 : I3
                          .-------_-------.
                              EPS448
                          .-------_-------.
                      NC :  1         28 : NC
                      NC :  2         27 : NC
                      NC :  3         26 : NC
                      NC :  4         25 : I6
                      C0 :  5         24 : Rd
                   CLOCK :  6         23 : Wr
                     VCC :  7         22 : ALE
                   RESET :  8         21 : GND
                      C1 :  9         20 : RegRd
                      C2 : 10         19 : OE
                  CmdAtt : 11         18 : Cn
                    Sign : 12         17 : DONE
                      NC : 13         16 : NC
                      NC : 14         15 : NC
                          .-------_-------.

***** DESIGN LISTING                                          circdraw.rpt

PART:   EPS448, EPS448

INPUTS: C0, C1, C2, CmdAtt, Sign

OUTPUTS:    A0, A1, A2, A3, B0, B1, B2, B3, I2, I1, I0, I5, I4, I3, I8, I7
OUTPUTS:    I6, Rd, Wr, ALE, RegRd, OE, Cn, Done

PINS:

DEFAULT:    [00000000000000000111001100]

PROGRAM:
    0D:         [00000000000000000111001100] JUMP WAIT;
    192D:
    WAIT:       IF CmdAtt * C0' * C1' * C2' THEN
                [00000000000000000111001100] JUMP DOIT;
                ELSE [00000000000000000111001100] JUMP WAIT;
    1D:
    DOIT:       [00010001000000000110011100] JUMP 2D;
```

Figure 14. Report File for Circle Drawing Routine (CIRCDRAW.RPT) (Part 2 of 3)

```
2D:          [00010001000000000110011001] JUMP 3D;
3D:          [00100010000000000111000000] JUMP 4D;
4D:          [00100010000000000110011001] JUMP 5D;
5D:          [00100010000000000110011001] JUMP 6D;
6D:          [00100010000000000111000000] JUMP 7D;
7D:          [00110011000000000110011001] JUMP 8D;
8D:          [00110011000000000110011001] JUMP 9D;
9D:          [01000100011100001111000000] JUMP 10D;
10D:         [01000101100000001111000010] JUMP 11D;
11D:         [01010101011000011111000000] JUMP 12D;
12D:         [01011001100000001111000000] JUMP 13D;
13D:         [01010101100000001111000010] JUMP 14D;
14D:         [01011000100000011111000000] LOADC 1D GOTO SHIFTR9;
15D:
SHIFTR9:     [10011001100000011111000000] LOOPNZ SHIFTR9 ONZERO 16D;
16D:         [10011001100000001111000010] JUMP 17D;
17D:         [10011001100000001111000010] JUMP 18D;
18D:         [00010110101000001111000000] JUMP 19D;
19D:         [01010110001010001111000010] JUMP OUTERLOOP;
20D:
OUTERLOOP:   [01000001001010001111000010] JUMP 193D;
193D:        IF Sign THEN
             [00000000000000000111001001] JUMP DrawEnd;
             ELSE [00000000000000000111001001] CALL CircPix RETURNTO 21D;
21D:         [01010101100000001111000000] JUMP 194D;
194D:        IF Sign THEN
             [00000000000000000111001001] JUMP POS;
             ELSE [01000110100000011111000000] JUMP 22D;
22D:         [01100110100000011111000000] JUMP 23D;
23D:         [10000110001000001111000000] JUMP 24D;
24D:         [01100101001000001111000000] JUMP IncX;
25D:
POS:         [00010110100000001111000000] JUMP 26D;
26D:         [01000110001010001111000010] LOADC 1D GOTO SHIFTR6;
27D:
SHIFTR6:     [01100110100000011111000000] LOOPNZ SHIFTR6 ONZERO 28D;
28D:         [10010110001000001111000000] JUMP 29D;
29D:         [01100101001000001111000000] JUMP 30D;
30D:         [00010001100000101111000000] JUMP IncX;
31D:
IncX:        [01000100100000001111000010] JUMP OUTERLOOP;
32D:
DrawEnd:     [00000000000000000111001001] CALL CircPix RETURNTO 33D;
33D:         [00000000000000000111001001] LOADC 16D GOTO DoDone;
34D:
DoDone:      [00010001100000001111000001] LOOPNZ DoDone ONZERO WAIT;
35D:
CircPix:     [01000110100000001111000000] JUMP 36D;
36D:         [00011011100000001111000000] CALL TRANS RETURNTO 37D;
37D:         [01000110100010001111000010] JUMP 38D;
38D:         [00011011100000001111000000] CALL TRANS RETURNTO 39D;
39D:         [00010110100000001111000000] JUMP 40D;
40D:         [01001011100000001111000000] CALL TRANS RETURNTO 41D;
41D:         [01001011100001001111000010] JUMP 42D;
42D:         [00010110100000001111000000] CALL TRANS RETURNTO 43D;
43D:         [00011011100001001111000010] JUMP 44D;
44D:         [01000110100000001111000000] CALL TRANS RETURNTO 45D;
45D:         [00010110100001001111000010] JUMP 46D;
46D:         [01001011100000001111000000] CALL TRANS RETURNTO 47D;
```

10

Figure 14. Report File for Circle Drawing Routine (CIRCDRAW.RPT) (Part 3 of 3)

```
    47D:        [01000110100010011111000010] JUMP 48D;
    48D:        [00011011100010011111000010] CALL TRANS RETURNTO 49D;
    49D:        [00010110100010011111000010] JUMP 50D;
    50D:        [01001011100010011111000010] CALL TRANS RETURNTO 51D;
    51D:        [00000000000000000111001001] RETURN;
    52D:
    TRANS:      [00111011001000011111000000] JUMP 53D;
    53D:        [00100110001000011111000000] LOADC 10D GOTO 54D;
    54D:        [10111100100000011111000000] JUMP MULT1024;
    55D:
    MULT1024:   [10111011100000011111000000] LOOPNZ MULT1024 ONZERO DONE1024;
    56D:
    DONE1024:   [11001011001001011111000010] JUMP 57D;
    57D:        [01101011001000011111000000] JUMP RUNBUS;
    58D:
    RUNBUS:     [10111011100000011111000000] JUMP 59D;
    59D:        [10111011100000011110001001] RETURN;
END$

***** PART UTILIZATION                                          circdraw.rpt

 59/192 Unconditional Branches ( 30.73%)
  3/ 64 Conditional Branches   (  4.69%)

      x1 Warnings
      0 Fatal errors
```

ASM Design Simulation

ASM designs can be functionally simulated with SAMSIM. Figure 15 shows a sample Command File, called **CIRCDRAW.CMD**, that contains the input stimuli for this circle drawing design. (Command Files are given the design name with the extension **.CMD**.)

The first line specifies the name (without the extension) of the source JEDEC file(s). The **GROUP CREATE** command creates a group called **CF** with three signals (**C0**, **C1**, and **C2**). The group's input patterns can then be specified in the **PATTERN CREATE CF** statement, rather than by entering each signal's stimulus separately. The **PATTERN CREATE** command lists the sequential values that the given input or group of inputs will take during simulation. Hexadecimal format can be used to further streamline group pattern entry. The notation **( )*n** specifies that the enclosed stimulus pattern should be repeated **n** times.

Figure 15. Command File for SAMSIM (CIRCDRAW.CMD)

```
JEDEC CIRCDRAW
GROUP CREATE CF = C0 C1 C2
PATTERN CREATE CF = (0H)*200
PATTERN CREATE CmdAtt = (0)*5 (1)*2 (0)*193
PATTERN CREATE Sign = (0)*200
TRACE CREATE CIRCDRAW.TRC
TRACE ON
SIMULATE 200
VIEW
```

The **TRACE CREATE** command creates a trace buffer file **CIRCDRAW.TRA** (shown in Figure 16). The state of the EPS448 EPLD is dumped into this file after each simulation step. The Trace File includes such information as the value on the top-of-stack, the counter value, and other internal information. **TRACE ON** turns the trace process on and can be discontinued with a **TRACE OFF** command later in the Command File. **SIMULATE 200** specifies a 200-clock simulation to run. Finally, the **VIEW** command enables interactive viewing of the results when simulation is finished.

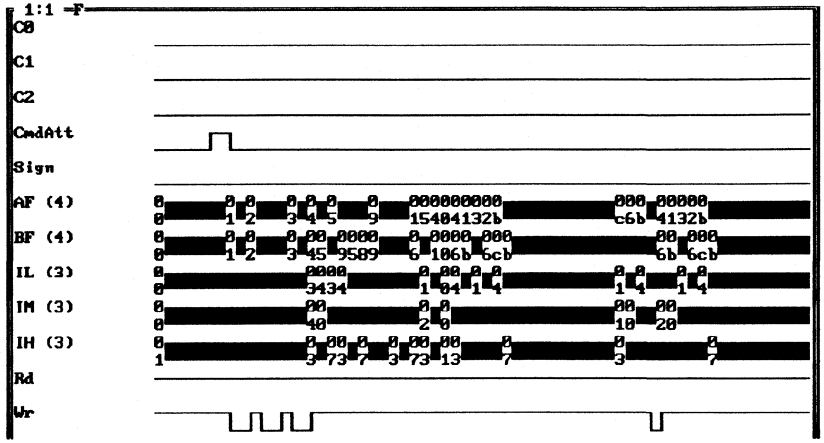
Figure 16. SAMSIM Trace File (CIRCDRAW.TRA)

```
.VC=35D .MA=55D .CO=10D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=36D .MA=55D .CO=9D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=37D .MA=55D .CO=8D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=38D .MA=55D .CO=7D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=39D .MA=55D .CO=6D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=40D .MA=55D .CO=5D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=41D .MA=55D .CO=4D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=42D .MA=55D .CO=3D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=43D .MA=55D .CO=2D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=44D .MA=55D .CO=1D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=45D .MA=55D .CO=0D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 MULT1024: 55D: [12288992D] LOOPNZ MULT1024 ;

.VC=46D .MA=56D .CO=0D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 DONE1024: 56D: [13313506D] CONTINUE ;

.VC=47D .MA=57D .CO=0D .SP=2D .TS=37D C0=0 C1=0 C2=0 CmdAtt=0
Sign=0 57D: [7021024D] CONTINUE ;
```

10

Figure 17 shows the Virtual Logic Analyzer (VLA) interactive output screen that is displayed after SAMSIM has been run with the Command File shown in Figure 15.

Figure 17. VLA Screen for Circle Drawing Routine



Two types of output are displayed in Figure 17: single-signal waveforms and group waveforms. **CmdAtt** is an example of a single-signal waveform that corresponds to an EPLD input. **AF** corresponds to a group of four signals, **A0** to **A3**. For **AF**, the values in the group are displayed in a vertical hexadecimal notation each time any signal in the group changes. (If an explicit value is not displayed, it is the same as the previous time-step's value.) By grouping common signals, much more information can be displayed in a single screen than would otherwise be visible. In this example, **A**, **B**, and **I** outputs are displayed in groups.

The simulation results correspond to approximately the first 40 clocks after the graphics controller receives a **CmdAtt**, which signals the beginning of a circle-drawing operation. The 3 **RegRd** pulses correspond to reading the circle's radius and (x,y) origin from the parameter register. The simple **OE** pulse is the point where the **CircPix** routine is first entered.

The VLA provides many useful features. For example, one command allows the order of waveforms to be changed interactively; another allows arbitrary signal groups to be constructed. An on-line **HELP** command provides instant explanations of all commands. The VLA thus provides an extremely flexible interactive design analysis.

Conclusion

The EPS448 EPLD provides an efficient solution for sophisticated control problems, such as the graphics controller and the 68020 bus arbiter. Other suitable applications for the EPS448 EPLD include industrial-control, graphics, disk controllers, and programmable sequence generators. The PC-based SAM+PLUS software offers two flexible entry languages along with powerful verification and debugging tools. This combination provides a winning approach to control design.

Integrating PAL and PLA Devices with the EPM5032 MAX EPLD

October 1990, ver. 2

Application Note 16

Introduction

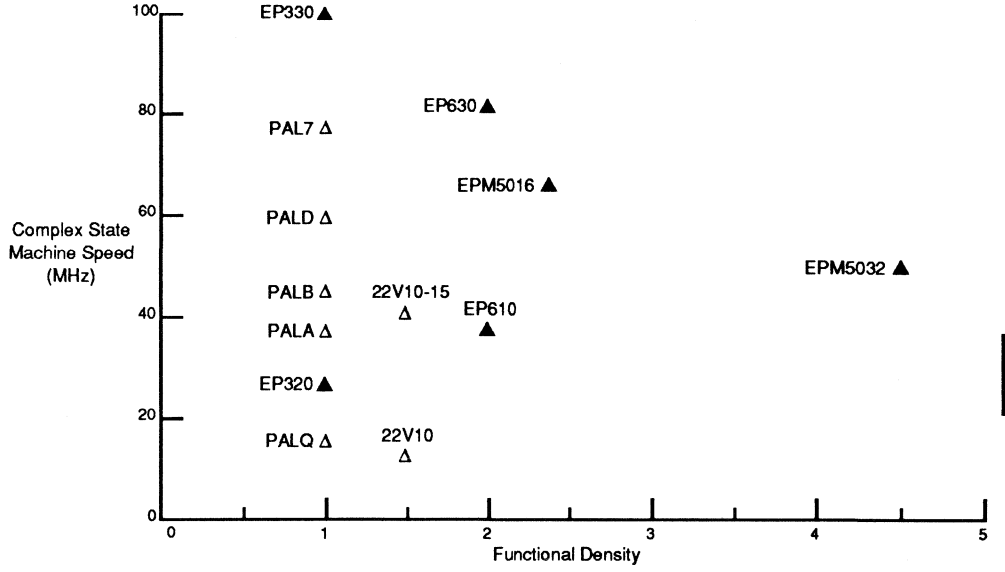
The EPM5032 is a high-speed member of Altera's Multiple Array MatriX (MAX) family of Erasable Programmable Logic Devices (EPLDs). Its highly flexible architecture facilitates general-purpose logic design, and integrates existing functions based on multiple TTL MSI, PLA, and PAL elements. Other benefits include flexible logic utilization, advanced register control, programmable I/O pins, and better system performance. Refer also to the *EPM5016 to EPM5192: High-Speed, High-Density MAX EPLDs Data Sheet* in this data book.

EPM5032 vs. PAL and PLA Architectures

The EPM5032 EPLD provides 28 pins and 32 macrocells, and may be programmed to accommodate 64 latches or 42 flip-flops, in addition to powerful combinatorial and control-logic functions. The device achieves flip-flop toggle rates of over 80 MHz.

EPM5032 architecture allows it to emulate both PAL and PLA structures, and to implement functions that are not possible with these devices. It provides more than four times the integration density of traditional PAL and PLA functions, while supporting complex state machine clock rates of 47 MHz (see Figure 1).

Figure 1. EPM5032 EPLD vs. PAL Devices *The EPM5032 has more than four times the functional density of traditional PALs.*



10

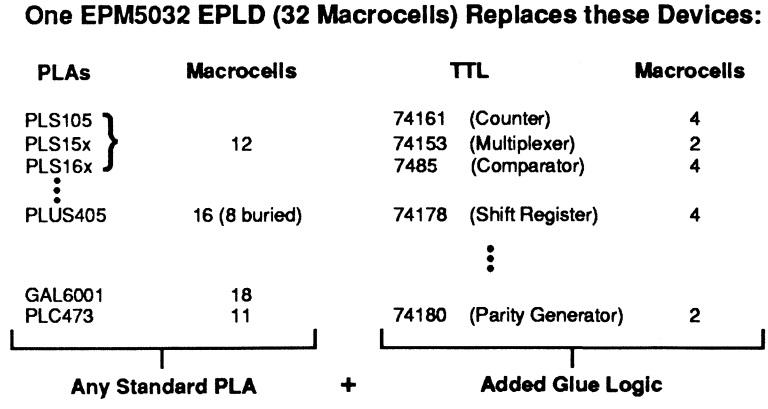
The basic EPM5032 architecture is an evolution of the AND/OR array structure that uses available logic far more efficiently than PAL or PLA devices. Statistical analysis of hundreds of designs indicates that 70% of all applications use no more than 3 product terms. The fixed-product-term allocation of PALs usually wastes 5 of the 8 product terms. In contrast, the MAX approach uses a streamlined, 3-product-term macrocell that may be supplemented—when required—with as many as 64 additional expander product terms. While PLA structures provide more flexible distribution of logic capability, the cost is a considerable loss in performance.

Use of expander product terms increases the effective EPLD density and permits higher integration levels than standard devices. Very complex equations can be implemented in a single macrocell, leaving other macrocells free to perform additional functions. Macrocells can also share expanders to efficiently implement functions with common product terms (e.g., state machines).

Expander product terms can also be used to create additional latches and flip-flops. As many as 32 latches can be created without consuming macrocell registers. Two expanders can be cross-coupled to form an SR latch, and D registers and latches can be implemented with other expander configurations. Although these functions can be created in PAL devices, additional macrocells would be required.

Figure 2 shows that a single EPM5032 can be used to replace multiple PLAs plus a considerable amount of TTL glue logic in a variety of applications.

Figure 2. Using an EPM5032 EPLD to Replace PLAs and TTL Glue Logic



Replacing 7400 TTL Devices

EPM5032 register and I/O features enable the EPLD to emulate 7400-series TTL functions, and to replace programmable logic and associated discrete TTL glue logic. Table 1 shows common TTL functions and their typical EPM5032 device utilization.

Table 1. EPM5032 TTL Function Integration

Function	Part	% Used
4-Bit Latch	7475	8%
4-Bit Mag.Comparator	7485	13%
8-Bit Shift Register	7491	19%
Dual 4:1 Mux	74153	4%
Quad 4:1 Mux	74157	5%
4-Bit Binary Counter	74161	10%
4-Bit Decade Counter	74162	10%
4-Bit Shift Register	74179	10%
9-Bit Parity Generator	74180	16%
Octal Latch	74373	16%

The MAX+PLUS TTL Macro-Function library contains over 340 TTL-equivalent functions to simplify entry of 7400-series functions into a design. Because MAX+PLUS automatically performs all design translation and optimization, a TTL function is implemented simply by entering the symbol. The macrocell architecture provides true TTL emulation. Since MAX+PLUS can also merge schematic designs with Boolean or state machine language descriptions when it compiles a design, the designer can complete each section of a design with the most appropriate entry method.

Replacing PAL Devices

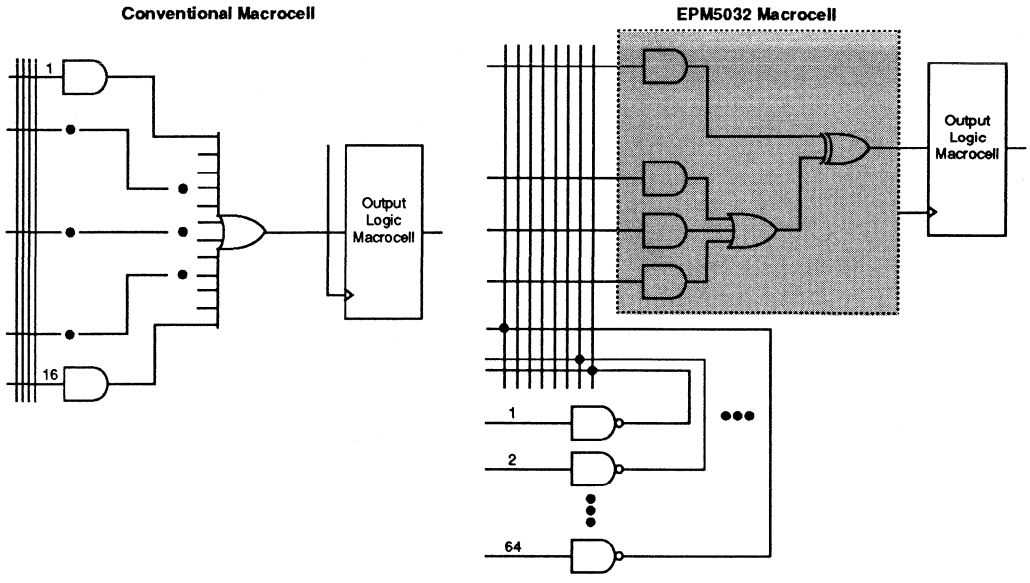
PAL circuits have a programmable-AND/fixed-OR structure. Each macrocell generally includes 7 (e.g., 16L8) or 8 (e.g., 16R8) product terms. Advanced PALs, such as the 22V10, support a variable allocation of product terms for macrocells, with as many as 16 product terms per macrocell. However, sharing product terms between macrocells is not possible, and product terms cannot be reallocated.

Figure 3 shows how the EPM5032 emulates a macrocell with a high product-term count. The AND/OR logic expression of a PAL can be directly implemented with the EPM5032 macrocell. The expander product terms provide product-term expansion if an individual application needs more product terms. An EPM5032 macrocell can have as many as 64 product terms, far more than any existing PAL device. This architecture is ideal for designs with multiple high product-term-count outputs and common input signals. For example, state machines, arithmetic logic circuits, and complex combinatorial functions fit easily on MAX EPLDs such as the EPM5032, but they are often impossible to design efficiently into PALs.

Existing PAL designs can be incorporated into any MAX+PLUS design (Figure 4). Altera's PLD2EQN utility converts a standard PAL JEDEC file into an Altera Hardware Description Language (AHDL) Text Design File (TDF). Altera's ABEL2MAX utility translates an ABEL **.ABL** file into an AHDL TDF. Both utilities are available free from Altera's Electronic Bulletin

10

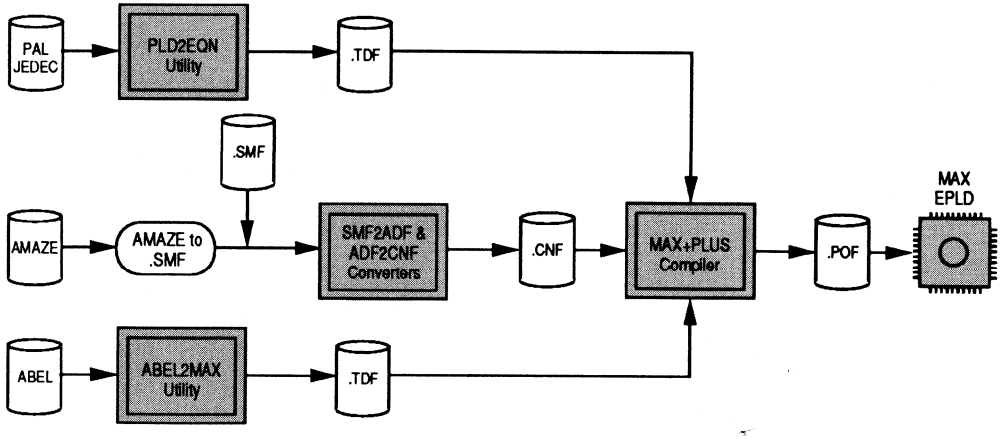
Figure 3. Emulating High Product-Term-Count Macrocells



Board (see the *Electronic Bulletin Board Service Data Sheet* in this data book for more information). A TDF can be processed by the MAX+PLUS Compiler, or the automatically-generated symbol representing the file can be entered into a multiple-PAL design.

Figure 4. PAL and PLA Design File Conversion

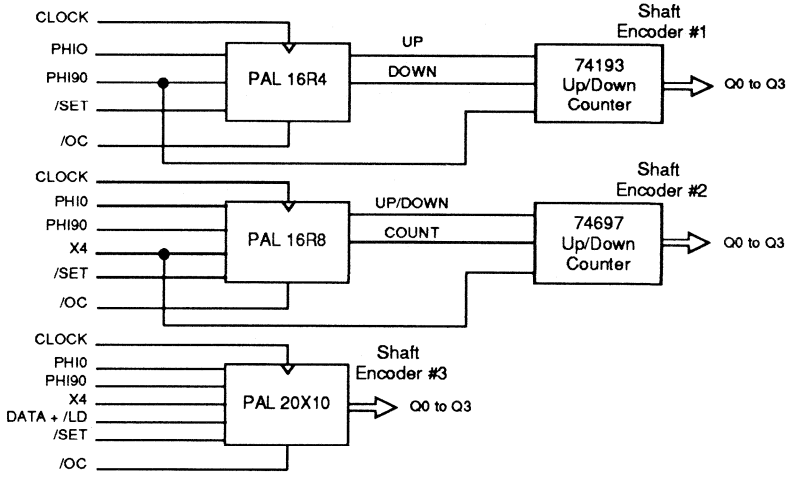
PLD2EQN and ABEL2MAX are free utility programs available from Altera's Electronic Bulletin Board Service. AMAZE-format files can be converted quickly to the Altera State Machine File (SMF) format.



PAL Design Example

The *Shaft Encoders* section of AMD's *PAL Device Handbook* shows a multiple-PAL/TTL implementation of a digital shaft encoder. The circuit measures shaft rotation by counting and comparing pulses produced by a pair of LEDs, a pair of photo sensors, and a rotating disk. The circuits are divided into three major blocks. The first is a set of registers that accepts a pair of 90° out-of-phase digital pulse trains as inputs, one from each photosensor. The registers discriminate the phase difference between the two signals and feed them into the decoder, the next block of the circuit. The decoder converts the register outputs into an up-and-down signal to control the final block, the counter. Outputs from the counter determine the position and direction of shaft rotation. Figure 5 shows the circuitry required: three PALs (16R4, 16R8, and 20X10) and two discrete TTL devices (74193 and 74697). All five parts fit in a portion of a single EPM5032.

Figure 5. Shaft Encoder



The 16R4 (see Figure 5) implements a set of four registers for phase discrimination, as well as decoding circuitry that produces **UP** and **DOWN** control signals for a 74193 counter. Converting these functions from the original PAL JEDEC files to an equivalent AHDL design file (see Design File 1 later in this data sheet) is straightforward. Although the PAL design includes a control signal (**/OC**) for disabling the tri-state buffer outputs, it is removed from the TDF because this signal is not needed in a completely integrated design. The entire translation process takes just a few minutes.

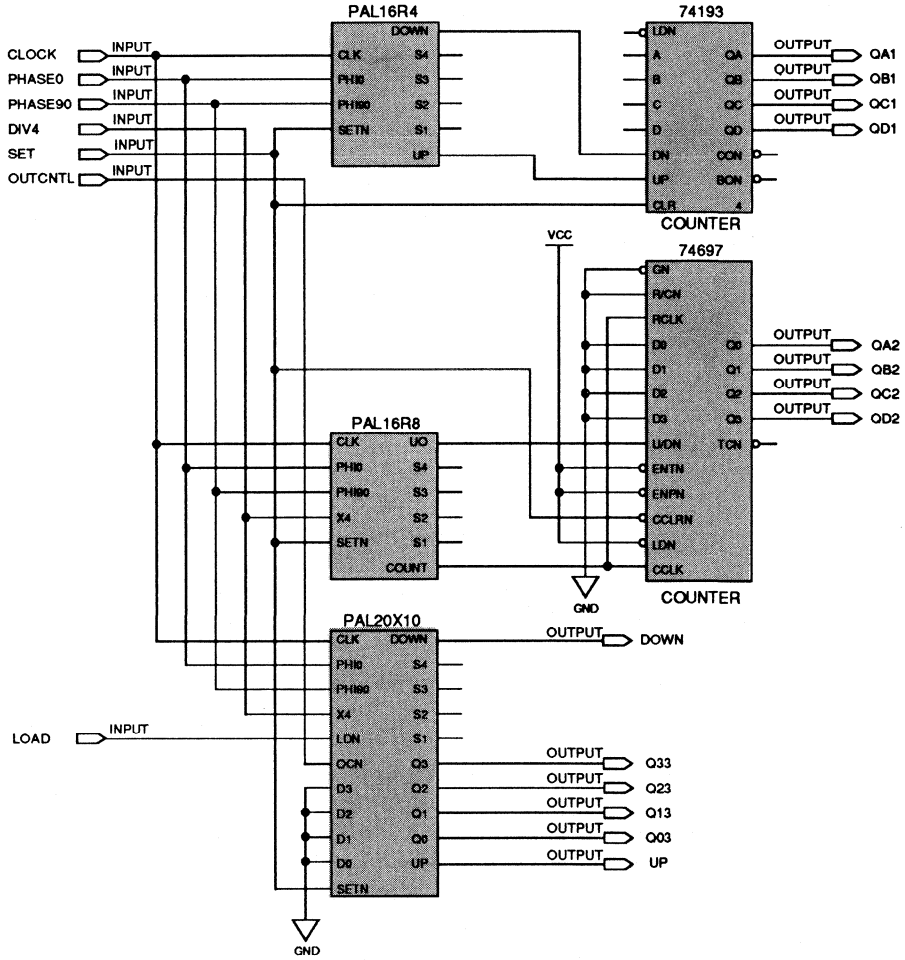
The second PAL in the design (16R8) implements four discrimination registers and decoding circuitry that outputs to a counter. However, the decoded outputs are configured into **UP/DOWN** and **COUNT** control signals, and use a 74697 counter instead of a 74193. The design's PAL JEDEC file translates into an TDF (see Design File 2 later in this data sheet), and the **/OC** control signal is left out. Translation takes only a few minutes.

10

The third PAL (20X10) is more sophisticated. It implements register and decoding functions as well as an up/down counter. Because this PAL includes its own counter and some signals that actually appear on output pins in the integrated design, the Output Enable control is left intact in the TDF shown in Design File 3 later in this data sheet.

After the three PAL designs are converted into AHDL, the resulting TDFs are opened with the MAX+PLUS Text Editor, and symbols representing the TDFs are automatically created by MAX+PLUS. The symbols for the **74193** and **74697** macrofunctions are readily available in the MAX+PLUS TTL MacroFunction Library. As shown in Figure 6, the symbols representing the PAL designs and the macrofunctions are entered with the MAX+PLUS Graphic Editor and then connected to complete the design. This schematic easily fits into a single EPM5032 after compilation.

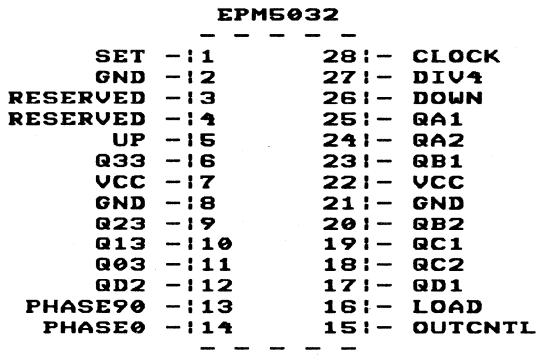
Figure 6. MAX+PLUS Shaft Encoder Schematic



Integration of these 3 PALs plus the 2 discrete counters required 29 macrocells and 29 expanders. Less than 90% of the EPM5032 is used, so there is room left to implement additional logic.

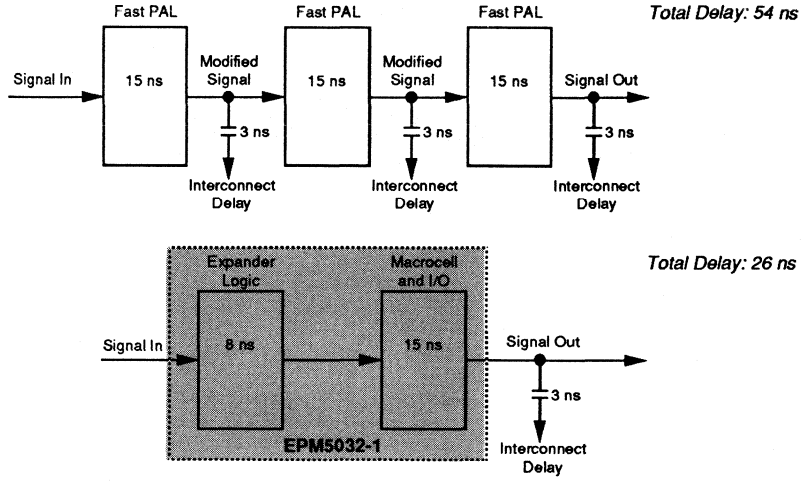
Figure 7 shows the pin-out produced by the MAX+PLUS Compiler as part of the Report File.

Figure 7. Integrated Shaft Encoder



The EPM5032 provides increased performance by replacing multiple PAL and PLA devices. The "single-chip solution" saves more board space than standard devices and eliminates chip-to-chip delays incurred by designs that require multiple PALs or PLAs. (See Figure 8.) For example, a PAL design with 3 logic levels has an overall delay of 54 ns (three 15-ns propagation delays and three 3-ns interconnect delays). The same design can be implemented in the EPM5032 with one level of expander logic and one level of macrocell logic for a total delay of 26 ns (including the 3-ns interconnect delay to the next stage). This single-device implementation provides faster speeds and better overall system performance.

Figure 8. Replacing High-Speed PALs with the EPM5032-1

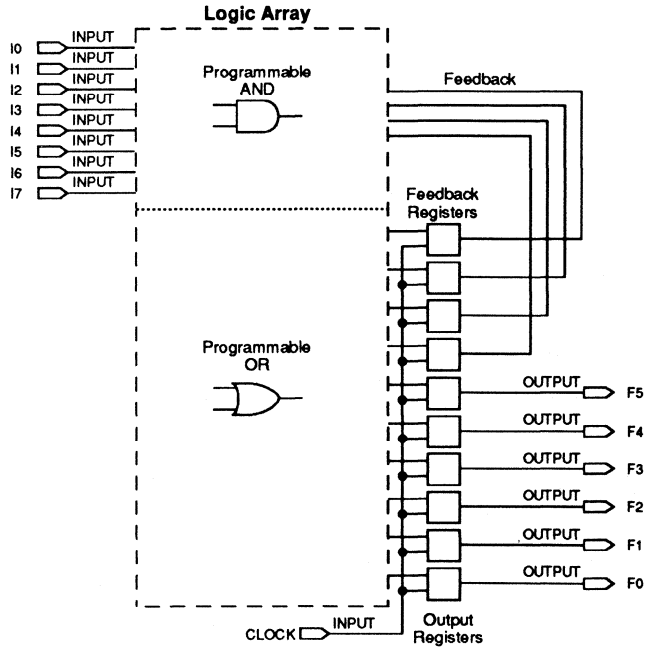


Replacing PLAs

In addition to PAL and random TTL replacement, the EPM5032 can also implement PLA architectures. Each PLA provides an additional programmable array that allows variable product-term distribution.

Figure 9 shows a typical PLA structure that consists of a programmable-AND array (containing 32 to 48 AND gates) feeding a programmable-OR

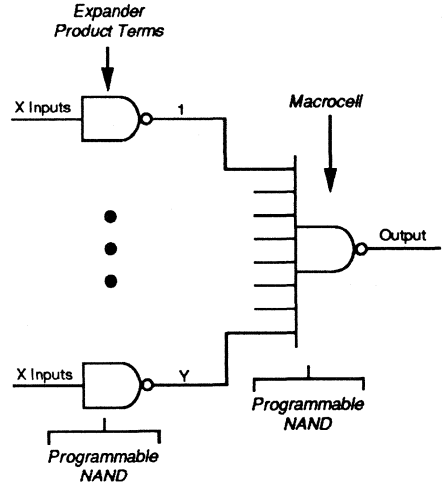
Figure 9. Traditional PLA Architecture



array. Each of the AND gates is fed by 8 to 12 input pins and feedback signals, plus their complements. The OR array allows logical ORing of any of the AND terms in the array. Typically, 8 to 12 programmed OR terms feed either feedback combinatorial buffers, feedback registers, or output pins.

The PLA's programmable-AND/programmable-OR structure is emulated with the expander product-term array, as shown in Figure 10. As in high-product-term PAL applications, the unallocated product terms feed an inverted AND gate in the macrocell to create an AND/OR-equivalent NAND/NAND structure.

Figure 10. Emulating the PLA's AND/OR Structure



As shown previously in Figure 4, existing PLAs can be incorporated into EPM5032 designs. For this purpose, the PLA source files from Signetics, which are usually written in the AMAZE language, must be converted to the Altera State Machine File (SMF) format. AMAZE and SMF formats are similar, so the source files can be quickly converted with a standard text editor. Figure 11 shows AMAZE and SMF files of a state machine beverage dispenser. Note the similarities in state and transition definition. After the AMAZE source file is transformed into an SMF, the SMF2ADF and ADF2CNF Converters provided with MAX+PLUS translate the SMF into a MAX+PLUS-compatible CNF (see Figure 4). The MAX+PLUS Compiler then processes the CNF.

10

Figure 11. SMF vs. AMAZE File Formats

Altera SMF

```
INPUTS: COINDROP, CUPFULL
OUTPUTS: DROPCUP, POURDRNK
STATES: [DROPCUP POURDRNK]
S1          [0          0]
S2          [1          0]
S3          [0          1]
S1: IF COINDROP THEN S2
S2: S3
S3: IF CUPFULL THEN S1
```

Signetics AMAZE

```
STATE VECTORS
S1 = 00b;
S2 = 10b;
S3 = 01b;
INPUT VECTORS
[COINDROP, CUPFULL]
OUTPUT VECTORS
[DROPCUP POURDRNK]
OUT1 = 00b;
OUT2 = 10b;
OUT3 = 01b;
TRANSITIONS
WHILE [S1]
            IF [/COINDROP] THEN [S1] WITH [OUT1]
            IF [COINDROP] THEN [S2] WITH [OUT2]
WHILE [S2] THEN [S3] WITH [OUT3]
WHILE [S3]
            IF [CUPFULL] THEN [S1] WITH [OUT1]
```

PLA Design Example

A serial communications interface with a custom protocol is described in *Custom Communication Protocol—PLS105A, 157, 167, 168A* of the Signetics *Sequencer Solutions* manual. The design shown in Figure 12 consists of the following three blocks, each of which is implemented in a Signetics PLA:

- The Preamble Sequence Detector is a state machine that recognizes a bit pattern that signals the beginning of a valid data block.
- The Cyclic Redundancy Check (CRC) Controller is a state machine that coordinates the loading of parallel-to-serial and serial-to-parallel shift registers and provides status of data and CRC transmissions.
- The 5-bit CRC Generator detects errors.

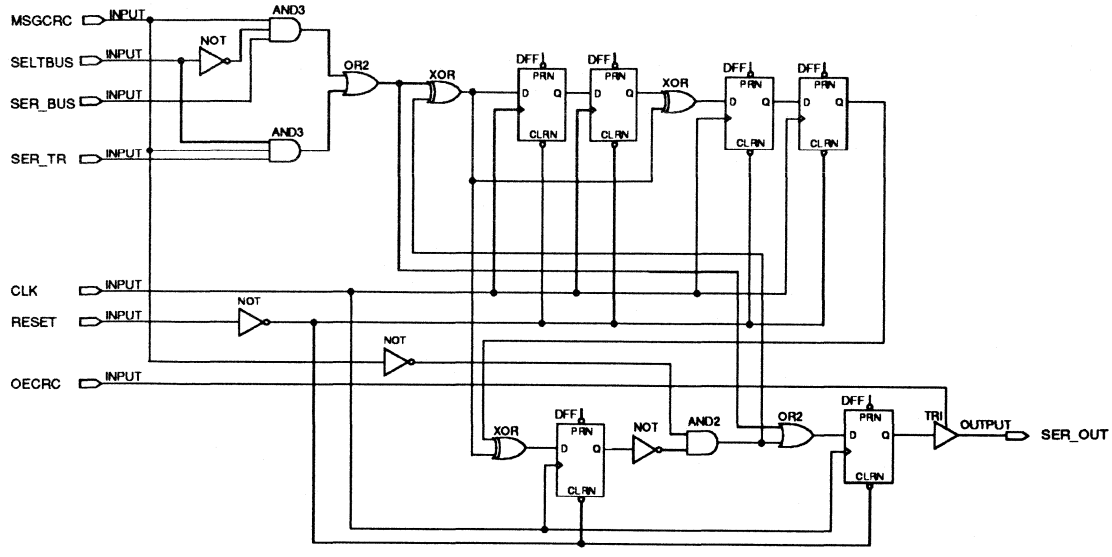
As shown here, these three blocks are integrated with Altera's MAX+PLUS Development System, and the resulting design fits into a single EPM5032.

Preamble Sequence Detector This module is implemented in the Signetics PLS167A. The Signetics AMAZE source file is transcribed into an SMF, and the design fits into less than one third of an EPM5032.

CRC Controller A Signetics source file is converted into an SMF to implement the CRC Controller. The Signetics implementation requires 100% register utilization of a PLS105A, while Altera's CRC Controller implementation leaves plenty of the EPM5032 unused.

CRC Generator The schematic shown in Figure 12 is the CRC Generator. It is submitted directly to MAX+PLUS for processing. This module requires 100% register utilization of a PLS157, but consumes less than 20% of the EPM5032.

Figure 12. CRC Generator



Integration is simplified with MAX+PLUS. Each of the three source files is compiled separately with the MAX+PLUS Compiler, and the automatically generated symbols that represent each module are entered with the MAX+PLUS Graphic Editor. Appropriate connections are made between the symbols to complete the design, and the final schematic is then submitted to the MAX+PLUS Compiler (see Figure 13). The PLS105A, PLS157, and PLS167A designs all fit into one EPM5032 (see Figure 14).

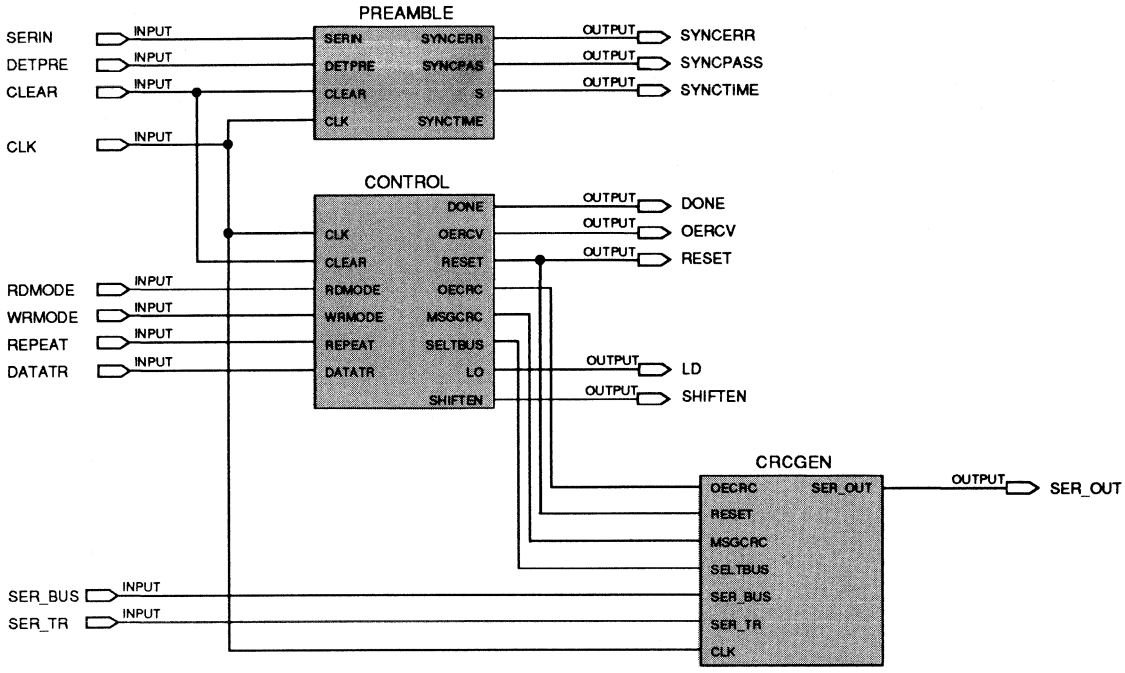
Conclusion

The EPM5032 offers better integration than PALs or PLAs and a flexible architecture that supports a broad range of applications. A single EPM5032 typically replaces three to four conventional PLDs, and can also provide extensive TTL integration. As shown in the design examples, one EPM5032 can be used in place of three PLAs or three PALs and two TTL devices. Designs can be easily converted from PALs or PLAs to the EPM5032. The EPM5032 also offers flexible logic utilization, advanced registers, better control of I/O pins, and system performance benefits.

The design files used to generate the PAL Shaft Encoder and the PLA Serial Communications Interface can be downloaded via modem with the Altera Electronic Bulletin Board Service.

10

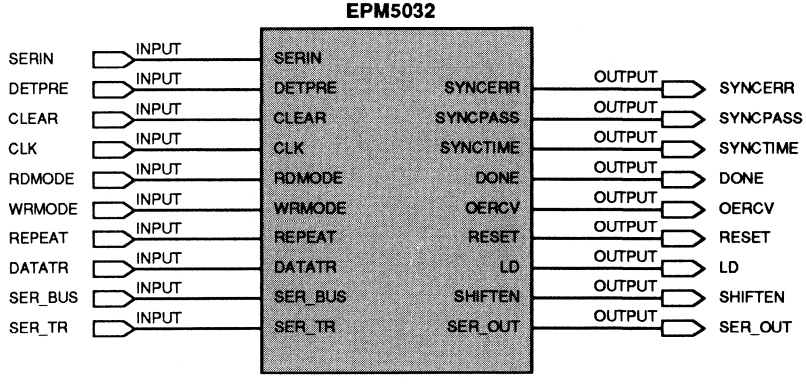
Figure 13. Serial Communications Interface



The following files are available:

PAL Design Files: **SHFTENC1.ADF, SHFTENC2.ADF, SHFTENC3.ADF**
PLA Design Files: **PREAMBLE.SMF, CONTROL.SMF, CRCGEN.GDF**

Figure 14. Integrated Serial Communications Interface



Design File 1. PAL 16R4-Equivalent AHDL File

```
% PLD2EQN version: 1.0 processed: Mon Oct 01 09:04:50 1990 %

TITLE "logic from JEDEC file SHFT_ONE";
DESIGN IS "PAL16R4" DEVICE IS "auto";

%
PAL16R4
SE1
%

SUBDESIGN PAL16R4
(
    CLK, PHI0, PHI90, SETN : INPUT;
    DOWN, S4, S3, S2, S1, UP    : OUTPUT;
)
VARIABLE
    down_din,
    s4_fbk, s4_din,
    s3_fbk, s3_din,
    s2_fbk, s2_din,
    s1_fbk, s1_din,
    up_din  : NODE;
BEGIN

    DOWN        = TRI(down_din, VCC);
    !down_din   =  PHI0 &  PHI90 &  s1_fbk &  s2_fbk &  s3_fbk & !s4_fbk
            # !PHI0 & !PHI90 & !s1_fbk & !s2_fbk & !s3_fbk &  s4_fbk
            #  PHI0 & !PHI90 &  s1_fbk & !s2_fbk & !s3_fbk & !s4_fbk
            # !PHI0 &  PHI90 & !s1_fbk &  s2_fbk &  s3_fbk &  s4_fbk;

    S4      = TRI(s4_fbk, VCC);
    s4_fbk  = DFF(s4_din, CLK, !GND, !GND);
    !s4_din = !s3_fbk
            # !SETN;

    S3      = TRI(s3_fbk, VCC);
    s3_fbk  = DFF(s3_din, CLK, !GND, !GND);
    !s3_din = !PHI90
            # !SETN;

    S2      = TRI(s2_fbk, VCC);
    s2_fbk  = DFF(s2_din, CLK, !GND, !GND);
    !s2_din = !s1_fbk
            # !SETN;

    S1      = TRI(s1_fbk, VCC);
    s1_fbk  = DFF(s1_din, CLK, !GND, !GND);
    !s1_din = !PHI0
            # !SETN;

    UP      = TRI(up_din, VCC);
    !up_din = !PHI0 &  PHI90 & !s1_fbk & !s2_fbk &  s3_fbk & !s4_fbk
            #  PHI0 & !PHI90 &  s1_fbk &  s2_fbk & !s3_fbk &  s4_fbk
            #  PHI0 &  PHI90 &  s1_fbk & !s2_fbk &  s3_fbk &  s4_fbk
            # !PHI0 & !PHI90 & !s1_fbk &  s2_fbk & !s3_fbk & !s4_fbk;
END;
```

Design File 2. PAL 16R8-Equivalent AHDL File (Part 1 of 2)

```
% PLD2EQN version: 1.0 processed: Mon Oct 01 05:17:47 1990 %

TITLE "logic from JEDEC file SHFT_TWO";
DESIGN IS "PAL16R8" DEVICE IS "auto";

%
PAL16R8
SE2
%

SUBDESIGN PAL16R8
(
    CLK, PHI0, PHI90, X4, SETN : INPUT;
    UD, S4, S3, S2, S1, COUNT  : OUTPUT;
)
VARIABLE
    UD_fbk, UD_din,
    S4_fbk, S4_din,
    S3_fbk, S3_din,
    S2_fbk, S2_din,
    S1_fbk, S1_din,
    COUNT_fbk, COUNT_din    : NODE;
BEGIN

    UD      = TRI(UD_fbk, VCC);
    UD_fbk  = DFF(UD_din, CLK, !GND, !GND);
    !UD_din = !S1_fbk &  S2_fbk & !S3_fbk &  S4_fbk
            # !S1_fbk &  S2_fbk &  S3_fbk &  S4_fbk
            # !S1_fbk &  S2_fbk &  S3_fbk & !S4_fbk
            #  S1_fbk &  S2_fbk &  S3_fbk & !S4_fbk
            #  S1_fbk & !S2_fbk &  S3_fbk & !S4_fbk
            #  S1_fbk & !S2_fbk & !S3_fbk & !S4_fbk
            #  S1_fbk & !S2_fbk & !S3_fbk &  S4_fbk
            # !S1_fbk &  S2_fbk & !S3_fbk &  S4_fbk;

    S4      = TRI(S4_fbk, VCC);
    S4_fbk  = DFF(S4_din, CLK, !GND, !GND);
    !S4_din =  S3_fbk
            # !SETN;

    S3      = TRI(S3_fbk, VCC);
    S3_fbk  = DFF(S3_din, CLK, !GND, !GND);
    !S3_din = !PHI90
            # !SETN;

    S2      = TRI(S2_fbk, VCC);
    S2_fbk  = DFF(S2_din, CLK, !GND, !GND);
    !S2_din =  S1_fbk
            # !SETN;

    S1      = TRI(S1_fbk, VCC);
    S1_fbk  = DFF(S1_din, CLK, !GND, !GND);
    !S1_din = !PHI0
            # !SETN;

    COUNT   = TRI(COUNT_fbk, VCC);
    COUNT_fbk   = DFF(COUNT_din, CLK, !GND, !GND);
    !COUNT_din  =  S1_fbk &  S2_fbk & !S3_fbk &  S4_fbk
```

Design File 2. PAL 16R8-Equivalent AHDL File (Part 2 of 2)

```
            # !S1_fbk & !S2_fbk &  S3_fbk & !S4_fbk
            #  X4 & !S1_fbk &  S2_fbk & !S3_fbk & !S4_fbk
            #  X4 &  S1_fbk & !S2_fbk &  S3_fbk &  S4_fbk
            #  S1_fbk &  S2_fbk &  S3_fbk & !S4_fbk
            # !S1_fbk & !S2_fbk & !S3_fbk &  S4_fbk
            #  X4 & !S1_fbk &  S2_fbk &  S3_fbk &  S4_fbk
            #  X4 &  S1_fbk & !S2_fbk & !S3_fbk & !S4_fbk;
END;
```

Design File 3. PAL 20X10-Equivalent AHDL File (Part 1 of 2)

```
%-------------------------------------------------------------
PLD2EQN version: 2.0 processed: Mon Sep 10 17:36:20 1990
    command line options: m p
-------------------------------------------------------------%
TITLE "logic from JEDEC file SHFT_3";
DESIGN IS "SHFT_3" DEVICE IS "auto";
%-------------------------------------------------------------
PAL20X10
SE3
-------------------------------------------------------------%
SUBDESIGN SHFT_3
(
    CLK, PHI0, PHI90, X4, LDN, D3, D2,
    D1, D0, SETN, OCN   : INPUT = GND;
    UP, Q0, Q1, Q2, Q3, S1, S2, S3,
    S4, DOWN    : OUTPUT;
)
VARIABLE
    CLK_clk,
    UP_fbk, UP_din,
    Q0_fbk, Q0_din,
    Q1_fbk, Q1_din,
    Q2_fbk, Q2_din,
    Q3_fbk, Q3_din,
    S1_fbk, S1_din,
    S2_fbk, S2_din,
    S3_fbk, S3_din,
    S4_fbk, S4_din,
    DOWN_fbk, DOWN_din,
    OCN_oen : NODE;
BEGIN
    CLK_clk = CLK;
    UP  = TRI(UP_fbk, OCN_oen);
    UP_fbk  = DFF(UP_din, CLK_clk, !GND, !GND);
    !UP_din = (!PHI0 &  PHI90 & !S1_fbk & !S2_fbk &  S3_fbk & !S4_fbk
            #  PHI0 & !PHI90 &  S1_fbk &  S2_fbk & !S3_fbk &  S4_fbk)
            $ ( PHI0 &  PHI90 &  X4 &  S1_fbk & !S2_fbk &  S3_fbk &  S4_fbk
            # !PHI0 & !PHI90 &  X4 & !S1_fbk &  S2_fbk & !S3_fbk & !S4_fbk);

    Q0  = TRI(Q0_fbk, OCN_oen);
    Q0_fbk  = DFF(Q0_din, CLK_clk, !GND, !GND);
    !Q0_din = (!LDN & !D0 &  SETN
            # !Q0_fbk &  LDN &  SETN)
            $ ( UP_fbk &  LDN &  SETN & !DOWN_fbk
            # !UP_fbk &  LDN &  SETN &  DOWN_fbk);
```

10

Design File 3. PAL 20X10-Equivalent AHDL File (Part 2 of 2)

```
    Q1  = TRI(Q1_fbk, OCN_oen);
    Q1_fbk  = DFF(Q1_din, CLK_clk, !GND, !GND);
    !Q1_din = (!LDN & !D1 &  SETN
            # !Q1_fbk &  LDN &  SETN)
            $ ( UP_fbk & !Q0_fbk &  LDN &  SETN & !DOWN_fbk
            # !UP_fbk &  Q0_fbk &  LDN &  SETN &  DOWN_fbk);

    Q2  = TRI(Q2_fbk, OCN_oen);
    Q2_fbk  = DFF(Q2_din, CLK_clk, !GND, !GND);
    !Q2_din = (!LDN & !D2 &  SETN
            #  LDN & !Q2_fbk &  SETN)
            $ ( UP_fbk & !Q0_fbk & !Q1_fbk &  LDN &  SETN & !DOWN_fbk
            # !UP_fbk &  Q0_fbk &  Q1_fbk &  LDN &  SETN &  DOWN_fbk);

    Q3  = TRI(Q3_fbk, OCN_oen);
    Q3_fbk  = DFF(Q3_din, CLK_clk, !GND, !GND);
    !Q3_din = (!LDN & !D3 &  SETN
            #  LDN & !Q3_fbk &  SETN)
            $ ( UP_fbk & !Q0_fbk & !Q1_fbk &  LDN & !Q2_fbk &  SETN & !DOWN_fbk
            # !UP_fbk &  Q0_fbk &  Q1_fbk &  LDN &  Q2_fbk &  SETN &  DOWN_fbk);

    S1  = TRI(S1_fbk, OCN_oen);
    S1_fbk  = DFF(S1_din, CLK_clk, !GND, !GND);
    !S1_din = (GND)
            $ (!SETN
            # !PHI0);

    S2  = TRI(S2_fbk, OCN_oen);
    S2_fbk  = DFF(S2_din, CLK_clk, !GND, !GND);
    !S2_din = (GND)
            $ (!SETN
            # !S1_fbk);

    S3  = TRI(S3_fbk, OCN_oen);
    S3_fbk  = DFF(S3_din, CLK_clk, !GND, !GND);
    !S3_din = (GND)
            $ (!SETN
            # !PHI90);

    S4  = TRI(S4_fbk, OCN_oen);
    S4_fbk  = DFF(S4_din, CLK_clk, !GND, !GND);
    !S4_din = (GND)
            $ (!SETN
            # !S3_fbk);

    DOWN    = TRI(DOWN_fbk, OCN_oen);
    DOWN_fbk    = DFF(DOWN_din, CLK_clk, !GND, !GND);
    !DOWN_din   = ( PHI0 &  PHI90 &  X4 &  S1_fbk &  S2_fbk &  S3_fbk & !S4_fbk
            # !PHI0 & !PHI90 &  X4 & !S1_fbk & !S2_fbk & !S3_fbk &  S4_fbk)
            $ ( PHI0 & !PHI90 &  S1_fbk & !S2_fbk & !S3_fbk & !S4_fbk
            # !PHI0 &  PHI90 & !S1_fbk &  S2_fbk &  S3_fbk &  S4_fbk);
    OCN_oen = !OCN;
END;
```

References

Advanced Micro Devices, Inc. *PAL Device Handbook* (1988).
Signetics Corporation. *Sequencer Solutions* (1988).

DSP/Imaging Applications with the EPS448 SAM EPLD

October 1990, ver. 2

Application Note 19

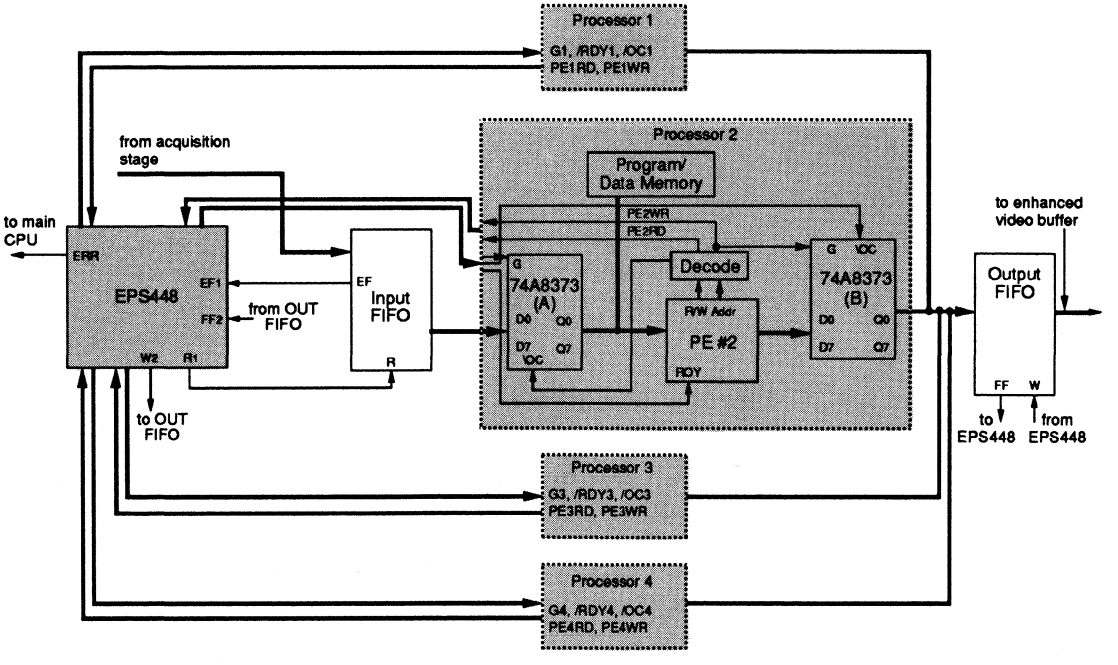
Overview

Digital image processing systems encompass a wide range of applications—from satellite image processing and medical imaging to traditional areas such as radar, sonar, and acoustic image-processing systems. The goal of image processing is to transform the two-dimensional representation of an image into a more desirable form. Use of the technology is often limited by the data throughput of the system. To improve data throughput in these systems, Altera's EPS448 user-configurable Stand-Alone Microsequencer (SAM) EPLD can be used at critical points in the system.

The EPS448 EPLD, an intelligent, programmable microsequencer in a 28-pin DIP package with 30-MHz performance, is ideal for use in DSP imaging systems. It provides a fast, compact, low-power control element in the data path to off-load data housekeeping tasks from the DSP processors. Figure 1 illustrates data flow controlled by the EPS448 EPLD.

Figure 1. Design Sample with the EPS448 EPLD

The EPS448 controls reading of data into the input FIFO, writing of data into the output FIFO, FIFO status checks, and data flow for the four processor rows in this cell.



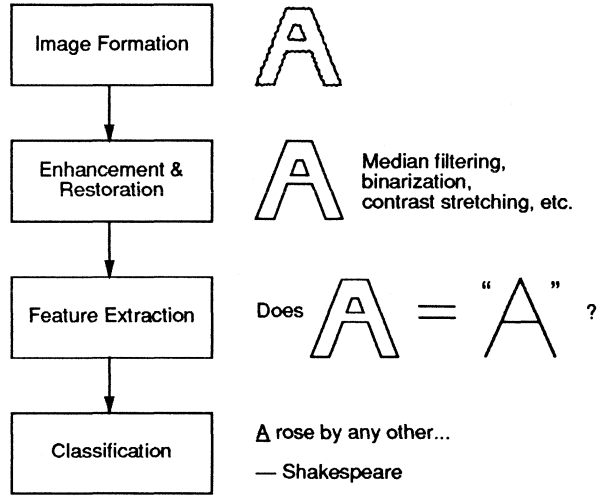
The EPS448 EPLD is also a highly integrated coefficient generator for a variety of DSP algorithms. In addition, the device's exact timing capabilities make it popular in the video output stages of the DSP imaging system.

The user should be familiar with EPS448 architecture and performance, as described in the *EPS448 SAM EPLD: Stand-Alone Microsequencer Data Sheet* in this data book.

Imaging Operations

Four major stages, shown in Figure 2, are common to imaging applications: (1) image formation, (2) enhancement and restoration, (3) feature extraction, and (4) classification. These steps are required for each DSP imaging system and are used here in the design example.

Figure 2. Typical Image Analysis Stages



(1) Image Formation During image formation, an image is represented by a number given to each picture element, or pixel. This image may be represented by the absorption characteristic of body tissue (for x-ray imaging), a temperature profile of a region (for infrared imaging), or the brightness of objects in a particular cross-section (for a picture taken by a camera). The image must be sampled with a sampling rate high enough to maintain the useful information contained in the image. (The sampling rate is determined by the bandwidth of the image.) Assuming the sampling rate is sufficient, the sampled image must also be quantized into some finite number of gray levels, using an analog-to-digital conversion. Quantization is the process in which each pixel is assigned a value based on the average level of gray contained within its area.

(2) Enhancement and Restoration Image enhancement and restoration follow quantization. Image enhancement emphasizes particular features of an image by using a quantitative model built to enhance specific features. For instance, a model may use contrast stretching to stretch gray levels in a small region to occupy a larger region, or median filtering to replace each pixel with the median pixel value in its immediate area. These techniques emphasize particular image characteristics; they don't generate new information about the image. Image restoration approximates an image with known problems (blurring, bad focusing, etc.) and changes it through median filtering, binarization, and contrast stretching. Unlike image enhancement, image restoration is subjective and does not depend on specific quantitative analysis.

(3) Feature Extraction During feature extraction, pixels are grouped with other neighboring pixels to create small subwindows, which vary in size from 3×3 to 64×64 pixels. Real-time calculations are required to measure features such as edge density and variance and to create histograms. ("Real time" refers to the ability to digitize, process, and display signals at standard video rates without any perceptible delay resulting from the DSP processing overhead.)

(4) Classification Feature-based image classification performs the real-time calculations. It needs large amounts of throughput, whose requirements relate to the subwindow size. For example, a typical television quality image of 512×512 pixels and a frame rate of 30 frames per second has a sampling data rate of about 10 MHz. Therefore, the feature extraction stage for a subwindow of 3×3 pixels requires a memory access rate of 10^8 per second. Architectures that permit real-time image analysis and handle these sampling and memory access rates are now available.

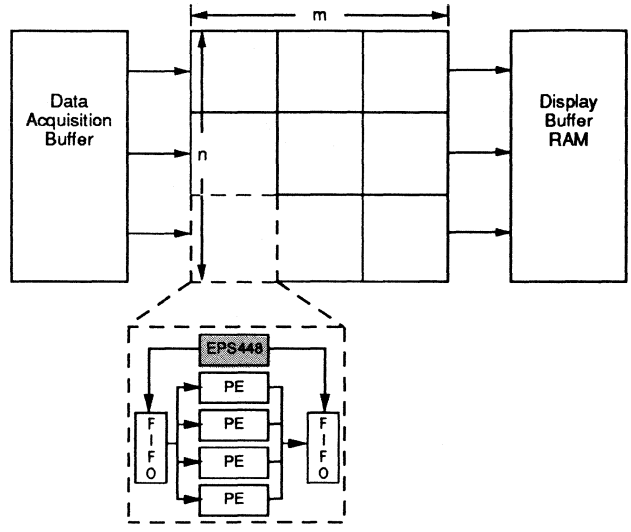
Techniques to Improve Performance

Real-time image analysis requires fast and efficient throughput, which can be achieved through the popular systolic approach. In a systolic system, a set of regularly interconnected cells is formed, with a dedicated processing element (PE) performing each task. Data is pipelined between cells and communicate with the outside world only at the boundary elements. Each stage of the pipeline has its own data storage and does not use global memory. This method of storage allows the cascaded processors to store data between them, permitting new data to enter the pipeline while previous data is processed. The critical performance factors of a systolic system are the speed of the individual processors and the number of stages in the pipeline.

To increase system throughput with better real-time performance, the pipelined processors may also be expanded into a processor matrix by adding parallel processors (see Figure 3). This method allows similar data elements, such as blocks of pixels, to be processed synchronously, multiplying overall system throughput. While this method improves system performance using the speed processors, it complicates data management.

Figure 3. Processor Matrix

A systolic array consists of a number of pipelined and parallel processing cells. The EPS448 can be used to control the system array and elements within a particular cell.



The need for processor element speed has led to advances in processors dedicated to DSP applications, such as the TMS320 family. Careful system data flow design is just as critical to system performance, particularly in a heavily pipelined system. Ideally, data should flow "automatically" through the system, without involving the PEs. By minimizing the housekeeping overhead on the processor elements, this method maximizes useful imaging cycles throughout the system.

Both software and hardware can be used to control data flow between processors. Software may be used to poll the status of the processors to determine when they are ready to accept or generate data, if data rates are slow enough. However, this method severely hampers system performance and is being replaced by interrupt-driven approaches, even in systems less dependent on data throughput. Another approach is the implementation of hardware interrupts, which requires significant software overhead to call and service interrupt routines and to restore the processor to its original task. This overhead is multiplied and becomes unacceptable when a multi-processor DSP system is used.

An ideal system to significantly improve system performance uses the processors exclusively for performing the necessary algorithms, without wasting time for handshakes. Until the advent of high-performance microsequencers, however, this approach was very costly because it required large amounts of hardware to synchronize the processors. Since EPS448 SAM EPLDs are microcoded, this technique can be implemented with a sizable reduction in hardware and cost.

Parallel Data Flow Control

To provide a "transparent" interface between processors, data buffers (e.g., RAMs, FIFOs, and register files) are used to absorb the data output by one processor, when the processors following it in the chain will not yet accept incoming data. These data buffers ensure optimum system performance. The parallel engine element shown in Figure 3 consists of four parallel processors within a cell that share common input and output FIFOs. This structure may be expanded in both the *m* and *n* directions to yield proper performance for each individual application. EPS448 EPLDs can be used not only to control data flow for the pipelined and parallel rows, but also to control it within a particular cell.

Figure 1 shows one cell in this large array. This cell contains an input FIFO, four processor elements (PEs), an output FIFO, and an EPS448 microsequencer that controls the data flow. Each PE shares input and output FIFOs that absorb incoming and outgoing data. Within each FIFO, a 74AS373 is connected to both input and output buses of each PE to latch incoming and outgoing data.

The EPS448 EPLD distributes data to the PEs by controlling the writing and reading functions of the 74AS373s, and performs error checking for the system. It simultaneously supports all data transfers including full error checking between the four processor elements, thus offloading this data housekeeping from the DSP engines.

The first task in handling this data flow is to manage the FIFO read and write cycles. Figure 4 shows the timing diagrams that represent the FIFO processor and the EPS448 signals required for read and write cycles. The **2** in **PE2RD**, **PE2WR**, **G2**, and **/OC2** represents processor 2. The widely used TMS32020 is shown, although any processor is suitable. This processor has two output clocks, **CLKOUT1** and **CLKOUT2**. Both output clocks have 200-ns periods, with **CLKOUT2** lagging by 50 ns. Another output from the processor is **/STRB**, synchronized with **CLKOUT2**. Its rising edge determines when a read or write is complete.

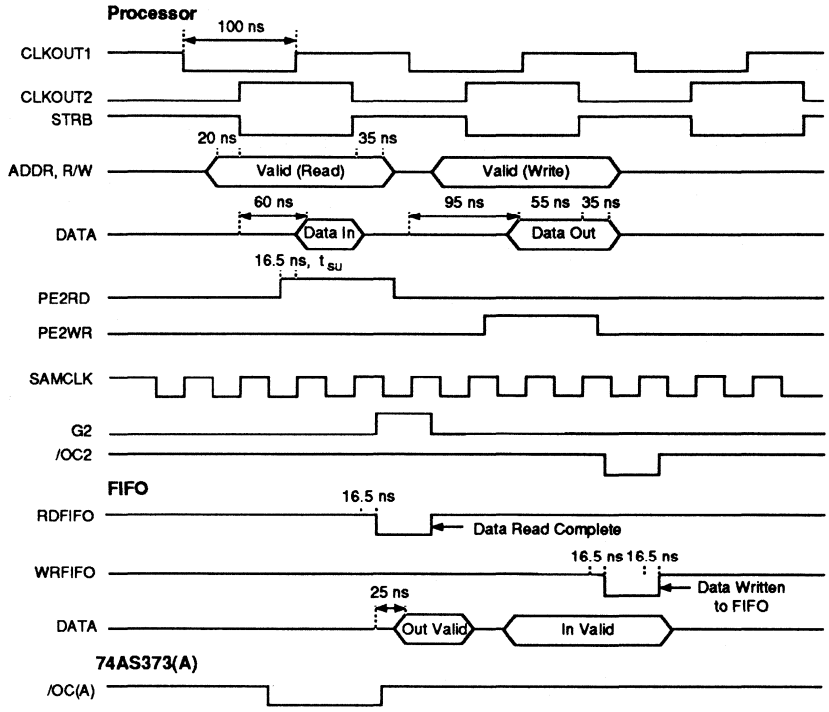
Other relevant processor outputs are the address and **R/W** signals. These signals are issued 20 ns before **/STRB** goes low. They are decoded to generate **PE2RD** and **PE2WR**, which are input signals to the EPS448, and indicate that a read or write has been requested.

Read-Data Timing

Processor 2 reads data in a two-phase cycle (see Figure 1). First, the processor reads the data on its input latch: the output from the 74AS373(A) is enabled, the processor reads this data, and the outputs are disabled. In the second phase, data is read from the FIFO to the 74AS373(A). The entire cycle begins on the falling edge of **/STRB**. The **/OCA** (which is a decode of the processor's **/STRB**), **R/W**, and address signals activate at this time and enable the outputs of 74AS373(A). The 74AS373(A) remains enabled until the rising edge of **/STRB**. On this edge, the processor reads the data of 74AS373(A) and **/OCA** is deasserted, i.e., 74AS373(A) is disabled.

Figure 4. Data Flow Timing

These signals are required for reading data into the input FIFO and writing data into the output FIFO.



Once the processor has read the data, the EPS448 EPLD must refill the 74AS373(A). The EPLD inititates the read from the FIFO into 74AS373(A) when it detects **PE2RD** on the rising edge of **SAMCLK**, 50 ns after **/STRB** goes low. The EPS448 EPLD waits for one clock cycle until it drives **/RFIFO** low. **/RFIFO** requests the FIFO to output a word. The input enable to the 74AS373(A), **G**, is also activated by the EPS448 EPLD at this time. **/RFIFO** and **G** are actually asserted on the EPS448 pins, 16.5 ns after **SAMCLK** (clock-to-output delay). If the FIFO has access time of 25 ns, the output data of the FIFO appears at the inputs of the 74AS373(A), 41.5 ns after **PE2RD** is detected (16.5 + 25 ns). For 50 ns, the **/RFIFO** pulse is low. Data is valid at 74AS373(A) for 25 ns (50 ns **RFIFO** low, 25 ns FIFO access time), meeting the setup time of 74AS373(A). On the rising edge of **/RFIFO**, the data is latched into 74AS373(A).

This process requires two **SAMCLK** signals (one detects **PE2RD** and one asserts **/RFIFO** to the FIFO), and half the 200-ns processor cycle time. If the FIFO is empty when the read is requested (i.e., **EF1** high), the EPS448 deactivates the processor's **RDY** signal and informs the system processor of the error status. In a real system, the EPS448 EPLD may have a set of routines to allow the processing element to recover dynamically from the error. The EPS448 EPLD provides up to 448 microcoded states to support this complex error recovery.

Write-Data Timing

PE2WR becomes active when the processor issues a write cycle. Figure 4 shows that after **PE2WR** goes high, the inputs of 74AS373(B) are enabled (since **G** is tied to **PE2WR**), and the latch is ready to accept a new data word. To write data into the output FIFO, **/OCB** and **/WFIFO** appear 16.5 ns (one clock-output delay) after the clock following the detection of **PE2WR**. Processor 2 then takes control of the FIFO data bus and begins the FIFO read. After 50 ns, the **/OCB** and **/WFIFO** are deasserted (i.e., go high), and data from the 74AS373(B) is written into the FIFO.

The write process also requires two **SAMCLK**s: one detects **PE2WR** and one asserts **/W** to the FIFO. If the FIFO is full (**FF2** high) at the time of the write request, the EPS448 EPLD deactivates **RDY** to the processor and informs the system processor of the error status.

Data Flow Microcode

EPS448 subsystem control tasks include checking the processors for read and write requests, servicing these requests, and checking for FIFO error conditions. The SAM Assembly Language (ASM) is used to enter EPS448 designs (see Figure 5). This language consists of simple **IF-THEN** constructs. Output specifications for any particular state are enclosed in brackets, and appear in the same order in which they are placed in the Outputs section (i.e., the first output in brackets represents **/RFIFO**). Macros are used to substitute text for commonly used output strings. For instance, **IDLE** substitutes for **G = 0** (inactive), **RDY = 1** (active), and **/OC = 1** (inactive). Data flow control starts at the label **PE1SRVC**.

Figure 5. Data Flow Microcode for EPS448

```
OUTPUTS:  /RFIFO, /WFIFO, G1, RDY1, /OC1, G2, RDY2,
          /OC2, ..., ERR
MACROS:   IDLE = 011
PE1SRVC:  % Similar to PE2SRVC %

PE2SRVC:  IF EF1 + FF2 THEN [11 001 001 ... 1]
          GOTO SYSERR;
          % Processor not ready %
          ELSEIF PE2RD THEN [11 IDLE IDLE ... 0]
          GOTO RDPE2;
          % Ready to read %
          ELSEIF PE2WR THEN [11 IDLE IDLE ... 0]
          GOTO WRPE2;
          % Ready to write %
          ELSE [11 001 001 ... 0] GOTO SYNCH;
          % Error, need to synchronize %
RDPE2:    [01 IDLE 111 ... 0] GOTO PE3SRVC;
          % Read from input FIFO %
WRPE2:    [10 IDLE 010 ... 0] GOTO PE3SRVC;
          % Write to output FIFO %

PE3SRVC:  % Similar to PE2SRVC %
PE4SRVC:  % Similar to PE2SRVC %

SYSERR:   % Recovery, wait for error conditions to clear %
SYNCH:    % Stop processors, synch subsystem %
```

10

The starting point for data flow control in **PE1SRVC** is the same as that of the **PE2SRVC** subroutine used to detail processor read and write cycles. One of four different states becomes active when **PE2SRVC** is called (this label is reached when the first processor has been serviced). The first state becomes active if the input FIFO is empty or the output FIFO is full (**EF1** + **FF2**) at the label **PE2SRVC**. When this error condition takes place, the processors' **RDY** signals are forced low, and a transition to **SYSERR** occurs. This system error should not happen, since the FIFO depths should be chosen according to worst-case data build-up. However, if either error condition does occur, the user can customize the **SYSERR** subroutine to a particular system. **SYSERR** can inform the main CPU of the subsystem error and make the EPS448 EPLD clear the error status.

In the second state, if **PE2RD** is true, all outputs are static to prepare for the performance of the required read cycle, assuming no error occurs at the label **PE2SRVC**. The label **RDPE2** becomes active on the next **SAMCLK**, when **/RFIFO** and **G2** become active. This state causes a read from the FIFO to 74AS373(A). On the next **SAMCLK**, a transition to **PE3SRVC** occurs when processor 3 is serviced and all outputs relating to processor 2 are deactivated.

The third state occurs if **PE2WR** is true at the **PE2SRVC** label, which prompts a write cycle to begin. All outputs remain static until **SAMCLK** causes a transition to the only state at the **WRPE2** label. This state's outputs force **/WFIFO** and **/OC** low, allowing the processor to write data to the output FIFO. On the following **SAMCLK**, a transistion to **PE3SRVC** takes place when processor 3 is serviced and all outputs relating to processor 2 are deactivated.

In the fourth state, neither error condition is true and both **PE2RD** and **PE2WR** are inactive. This error condition informs the user that the processors are not synchronized. The subsystem is designed so that the processor 2 lags behind processor 1 by 100 ns, which is the same time it takes to read or write from processor 1. In addition, it is assumed that all processors in the image-processing system execute instructions requiring the same amount of time. If processor 1 has just read data, processor 2 should also be ready to read data when **PE2SRVC** is reached. If processor 2 is not ready when processor 1 finishes, a synchronization error has occurred, and all processors' **RDY** signals are pulled low. A transition to **SYNCH** then takes place, which resynchronizes the subsystem processors through a subroutine created by the user.

The benefits of the EPS448 subsystem include the increased data throughput and the consistent data flow from the input stage, through the processors, to the output FIFO. The microsequencer provides data flow control, while relieving the processors of synchronization and FIFO error condition checking. The EPS448 EPLD runs at 20 MHz to allow four processor reads and four processor writes to occur continually every 800 ns, generating an output data word every 200 ns. (This is the same amount of time a processor requires to perform one instruction.) Without the microsequencer,

each PE would require two additional cycles of overhead between the generation of each data word to check for data at the input and output FIFOs. A 30-MHz EPS448 EPLD is also available to support faster processors and subsystems that have more than four processors for each microsequencer.

Other Parallel Processing Control

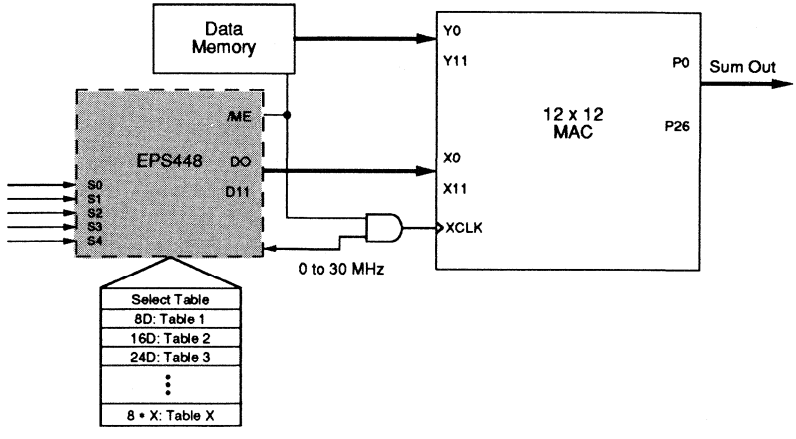
Parallel approaches that require local neighborhood operations are also in spatial operation techniques used for image enhancement. Calculations are performed on a pixel and its nearest neighbors. If a 5 × 5 pixel area of an image is to be analyzed, a 5 × 5 array of processors ideally would perform the algorithm. The EPS448 EPLD controls the loading of the pixel words into the processors. Five bits from the image's first line are loaded into the top row of processors. The EPS448 EPLD counts through the unnecessary pixels in the line following the neighborhood's pixels, the blanking pixels beyond the edge of the screen, and the unnecessary pixels preceding the active pixels in the second line. The EPS448 then enables the five pixels on the second line into the second row of processors, and the process continues. The EPS448 EPLD is well-suited to this application because it has an on-chip counter and performs subroutine calls and looping.

FIR Filter Implementation with SAM

The finite duration impulse response (FIR) filter is a basic building block in DSP applications. The EPS448 EPLD is ideal for creating an N-tap FIR filter, and for using a Multiplier-Accumulator (MAC) for coefficient generation (see Figure 6). An 8-tap FIR filter requires 8 data words and 8 coefficients as inputs to a MAC. The EPS448 EPLD can supply coefficients to a MAC as fast as 30 MHz, enabling the device to act as a fast PROM in delivering coefficients to the input registers of the MAC. The EPS448 EPLD also replaces the address counter and provides necessary control lines.

Figure 6. Selectable Coefficient Generation

The EPS448 acts as a large EPROM plus address logic for storing coefficient values. Here a selectable 8-tap, 12-bit filter is implemented with a 12 x 12 MAC.

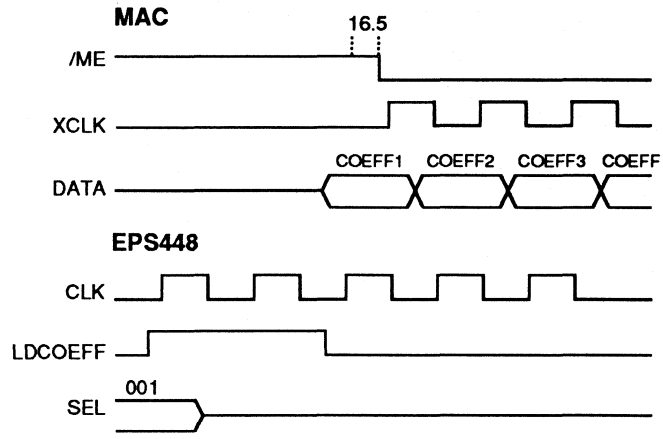


10

Figure 7 shows the signals necessary to interface the EPS448 to a 12×12 bit MAC that has 35-ns multiply-accumulate times.

Figure 7. Coefficient Generator Timing

The timing and signals required for the 8-bit tap filter, SEL, and LDCOEFF are the necessary control signals into EPS448 coefficients that are available 165 ns after the rising edge of each EPS448 clock.



The EPS448 EPLD has 448 microcode addresses, each containing a 36-bit word that consists of 20 internal control bits and 16 output bits. As a result, the EPLD contains a 448 × 16 PROM and a stack that can implement a 448-way branch in only 2 clock cycles. The EPS448 EPLD can thus store and quickly obtain access to coefficient tables of varying sizes to implement multiple algorithms.

Figure 8 shows ASM code that allows the EPS448 EPLD to provide selectable coefficient strings to implement eight-tap filters. If **LDCOEFF** from the first instruction (**If LDCOEFF ANDPUSHI 11111000**) is true, the EPS448 inputs are ANDed with the constant **11111000**, and the result is pushed onto the EPS448 internal stack. This process guarantees that the three least significant bits will be zeros. The remaining five input bits are the select inputs that determine which coefficient table to access.

The clock after the **ANDPUSHI** is instructed to **RETURN**. This instruction pops the top-of-stack, making it the next label (or address location) in the EPS448. For instance, if the select inputs are **00001**, address **00001000 * 11111000 = 8D** (decimal) is off the stack and becomes the next label; if the select inputs are **00010**, address **16D** is the next label. In this example, the **11111000** mask forces a branch to an address that is a multiple of 8 (the number of taps). If the number of taps is changed, the mask is also modified. For example, if a 12-tap filter is required, the mask is changed to **11110100**. For this 8-tap filter and select inputs **00001**, a transition occurs after the **RETURN** command to address **8D** on the clock. (Address **8D** is the address of the first coefficient table.)

Figure 8. EPS448 Microcode for Coefficient Generator

```
INPUTS:     S4, S3, S2, S1, S0, LDCOEFF          % Selects are 3rd, 4th, & 5th for mask %

OUTPUTS:    COEFF11 ... COEFF0, /ME              % 12 coefficients and memory enable    %

SELECT:     IF LDCOEFF [0 ... 0 1]               % Select table                         %
              ANDPUSHI 11111000 GOTO TABLESEL;
            ELSE [0 ... 0 1]  GOTO SELECT;       % Wait for LDCOEFF                     %

TABLESEL:   [0 ... 0 1] RETURN;                  % Transition to selected table         %

8D:         [H4C 0] CONTINUE;                    % HEX COEFFICIENTS, /ME                 %
            [0B5 0] CONTINUE;
            [679 0] CONTINUE;
            [D40 0] CONTINUE;
            [51A 0] CONTINUE;
            [D41 0] CONTINUE;
            [A09 0] CONTINUE;
            [624 0] CONTINUE;
16D:        [904 0] CONTINUE;
            [4B8 0] CONTINUE;
```

After the transition to address **8D**, the first coefficient and memory enable **/ME** are output. **/ME** enables the data memory and is ANDed with the system clock to generate **XCLK**, which clocks in the *x* data bits. **XCLK** requires a 15-ns data setup time and a 3-ns hold time. Since the coefficient outputs arrive 15 ns before **XCLK**, this condition is met. Hereafter, a new 12-bit coefficient is loaded into the *x* data input registers of the MAC on every system clock.

When the last coefficient is supplied to the MAC, the FIR filter is implemented, and the EPS448 issues a flag to the data source, informing it that data should no longer be enabled. At this point, a transition to **SELECT** occurs that provides access to the appropriate location in the coefficient table, depending on the inputs. **SELECT** may access the same set of coefficients or any other set in the table. Because the EPS448 EPLD has 448 memory locations, 56 possible table locations can be accessed for this 8-tap filter.

To choose the appropriate set of coefficients and to meet the setup time for **XCLK**, a 3-clock cycle latency occurs with this approach. (It occurs whether there are 56 eight-tap filters, equal to 448 states, or 224 two-tap filters.) This latency is possible because the EPS448 EPLD performs a 448-way branch in only 2 clock cycles. After the third clock cycle, coefficient data becomes available every clock cycle, or as fast as 30 MHz. This approach has advantages over both FIFO and PROM approaches. Although data can be written directly if FIFOs are used, very fast FIFOs (with access time of 35 ns) are very expensive.

Also, coefficients must be reloaded before being reused (which requires *n* clock cycles, where *n* is the number of taps), or a retransmit feature must be used. Retransmit slows the cycle time of a 35-ns FIFO to 120 ns.

The EPS448 EPLD provides advantages over PROMs, including power savings, cost reduction, and integration. In addition, address counters that are required with the PROM approach are unnecessary with the EPS448 EPLD. Also, if multiple PROMs are used because of filter length or width, the PROM address counter must be buffered. Using multiple PROMs consumes more board area because of the additional devices and the routing of address lines. However, the EPS448 provides these benefits while typically consuming 90 mA at 30 MHz.

Conclusion

The programmable EPS448 architecture is ideal for implementing complex designs typical of DSP systems. Because of its flexibility and high performance, the EPS448 EPLD is well-suited for controlling data flow for both parallel and pipelined systems, and for coefficient generation. Finally, the EPS448 provides both low-power operation and high performance (30 MHz).

Fast Bus Controllers with the EPM5016 MAX EPLD

October 1990, ver. 2

Application Note 20

Introduction

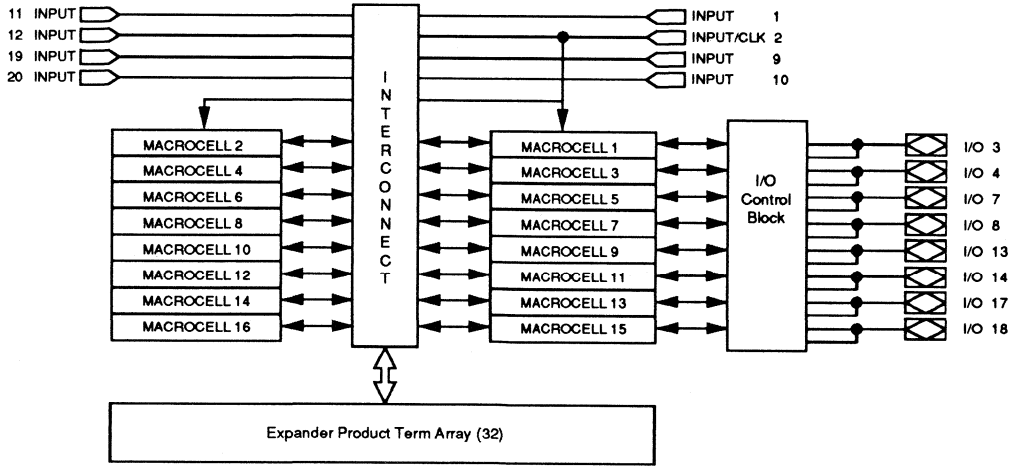
Today's advanced microprocessors are capable of running in systems with clock speeds up to 33 MHz. To realize the performance potential of these microprocessors, their memory interfaces must be equally fast. High-performance memory devices, however, are expensive. This cost/performance tradeoff can be improved by using a combination of fast and slow memories. To accommodate the slower memories, wait states are added into the microprocessor bus cycle.

This application note describes how to integrate the wait-state and bus-control logic into an Altera EPM5016 MAX EPLD, which is then integrated into an 80386 microsystem design. The application note also describes how to create and process the design with the MAX+PLUS Development System.

EPM5016 Overview

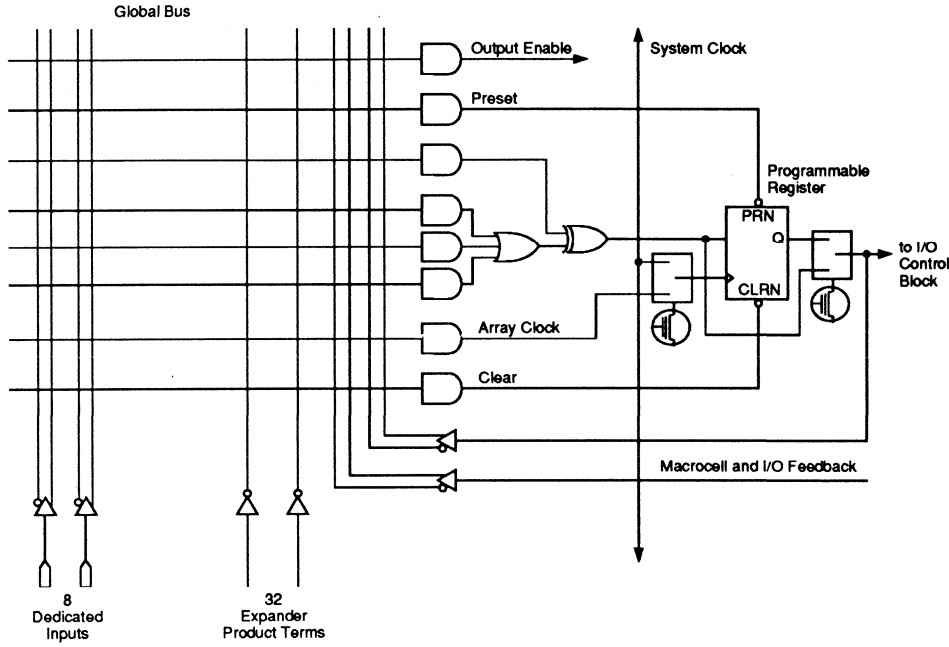
The EPM5016 is the fastest member of the MAX EPM5000-series EPLDs, with propagation delays of 15 ns, system clock rates of 66 MHz, and counter frequencies of 100 MHz. The EPM5016 can also drive 24 mA, which allows it to directly connect to buses. It is available in a windowed ceramic or plastic 20-pin DIP, 20-pin PLCC, or 20-pin SOIC package, accommodating designs with up to 15 inputs and 8 outputs. The EPM5016 architecture is based on a single flexible Logic Array Block (LAB) that encompasses 3 components: the macrocell array, the expander product-term array, and the I/O control block. See Figure 1.

Figure 1. EPM5016 Block Diagram



The macrocell array contains 16 macrocells (see Figure 2). Each macrocell has a programmable-AND/fixed-OR array and a configurable register that provides D, T, JK, SR, or flow-through latch operation with independent programmable clock options. Each macrocell also contains 3 product terms for logic implementation. If necessary, the expander product-term array can supply up to 32 additional product terms. Each expander product term can be used and shared by any of the macrocells.

Figure 2. MAX Macrocell



The EPM5016 has 8 bidirectional I/O pins with 24-mA output drivers to allow direct interfacing to a variety of system buses. All of the I/O pins are individually configurable for dedicated input, dedicated output, or bidirectional operation. Each I/O pin has a dedicated feedback and a tri-state buffer. Macrocells and I/O pins have separate feedbacks—a feature called "dual feedback"—that enable the user to bury macrocell logic and retain the pins for inputs. For complete details about EPM5016 architecture and timing, see the *EPM5016 to EPM5192: High-Speed, High-Density MAX EPLDs Data Sheet* in this data book.

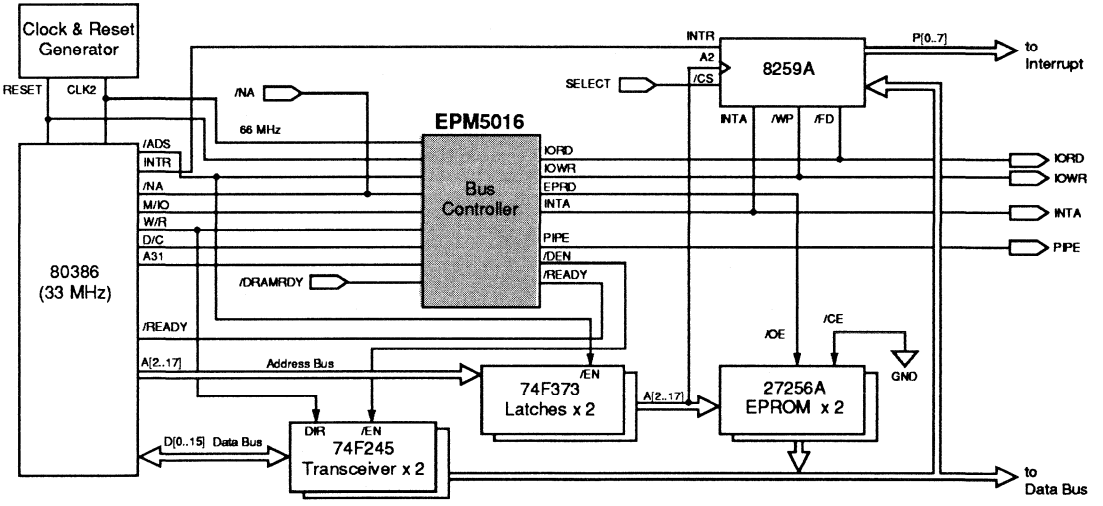
EPM5016 designs are created and programmed with Altera's MAX+PLUS Development System. MAX+PLUS is a complete CAE system that offers hierarchical design entry tools, automatic design compilation and fitting, timing simulation, and device programming. The MAX+PLUS Compiler

features advanced logic synthesis algorithms, allowing designs to be entered in a variety of high-level formats while ensuring the most efficient use of EPLD resources. The combination of flexible device architecture and advanced CAE tools ensures rapid design cycles. A design can go from conception to completion in a single day.

Bus Controller Functional Description

Figure 3 shows the block diagram of an 80386 microsystem that incorporates peripheral logic, memory, and an 8259A interrupt controller. The EPM5016, shown in the center of the diagram, serves as the system bus controller. The EPM5016 decodes the 80386 status signals to control the peripheral logic, i.e., the data transceiver, interrupt controller, and other external logic. It also extends bus cycles by adding wait states to interface to slower peripherals and memory devices.

Figure 3. 80386 Subsystem Block Diagram



The 80386 halts processing to allow wait states to be added into the bus cycle when the signal **/READY** is high. (The slash (/) indicates an active-low signal.) The EPM5016 bus controller tracks each bus cycle operation and causes **/READY** to go high when wait states are needed. For example, read operations from 200-ns EPROM memory in 33-MHz systems require 14 wait states.

The EPM5016 also decodes the bus control signals **IORD** (I/O read), **IOWR** (I/O write), and **INTA** (interrupt acknowledge). The 24-mA output drivers on the EPM5016 eliminate the need to buffer these bus signals externally.

The 16 data signals originating from the 80386 are isolated from the system data bus with two 74245 8-bit transceivers. The tri-state control on the transceivers is provided by the signal /DEN from the EPM5016. The direction signal is controlled directly by the Read/Write signal (W/R).

Two 74373s (8-bit latches) latch the 80386 address signals at the beginning of the bus cycle to maintain a valid address throughout the cycle. The latches are controlled by the 80386 signal /ADS. The high performance of the EPM5016 easily supports the 33-MHz bus cycles. In fact, a 66-MHz (2× 33-MHz) clock is used to clock the design for two reasons. First, /ADS can be connected directly to the address latches since the EPM5016 control signals for the peripheral logic are active before the end of the first bus cycle. Second, the wait-state generator offers greater granularity with 15-ns cycles than with 30-ns cycles.

Bus Controller Interface Signals

The design for the bus controller circuit requires 9 inputs: all 8 dedicated inputs of the EPM5016 and 1 I/O pin. Five of the 9 inputs to the EPM5016 are signals from the 80386 microprocessor. The functions of these 5 signals are given in Table 1. The other inputs are described subsequently.

Table 1. EPM5016 Input Functions

Input	Function
M/IO	Memory or I/O
W/R	Read or Write status
D/C	Data or Control status
A31	Address bit 31 for memory mapping of the EPROM
/ADS	Address data strobe indicating the beginning of the bus cycle

Systems that have functions set up for pipelining require external logic to generate the signal /NA (next address). /NA feeds both the 80386 and the EPM5016 bus controller. When /NA is activated, the 80386 places the next address on the address signals so that they may be latched. Applications that require pipelining receive minimal benefit from wait states. In such cases, the /NA signal is used simply to disable the EPM5016.

/DRAMRDY is an externally generated signal that, when high, halts the 80386 by causing the /READY signal to be high. When /DRAMRDY goes low, the 80386 continues processing.

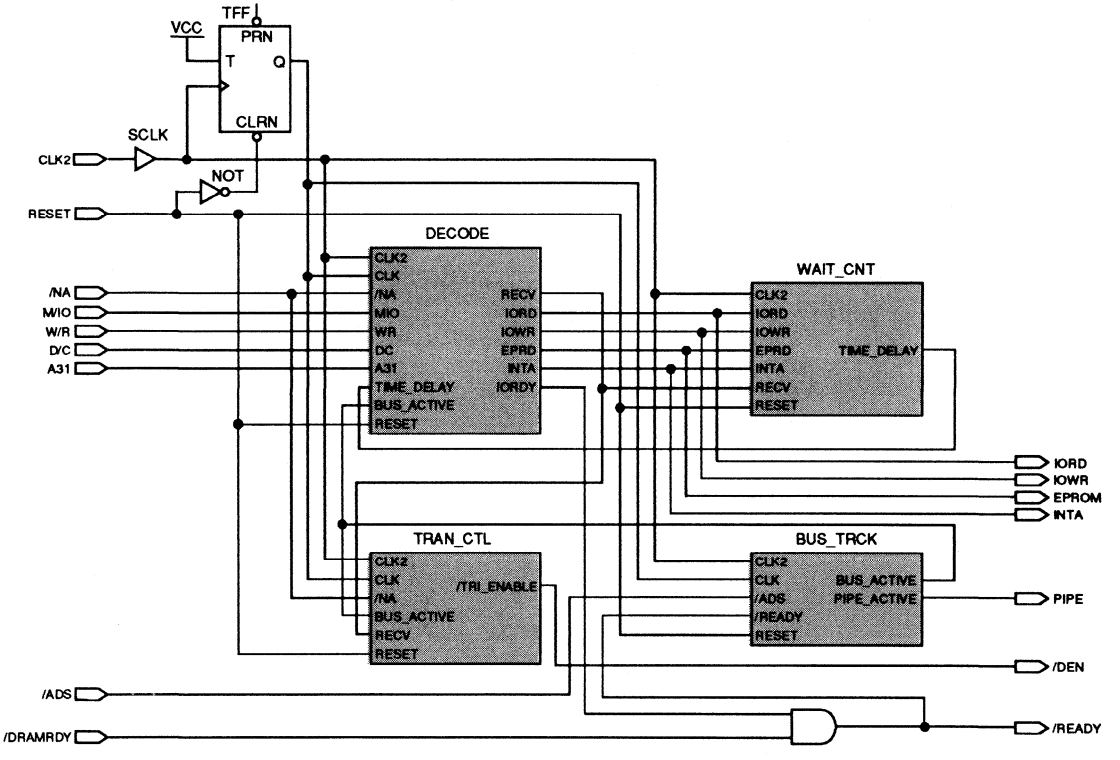
The 66-MHz clock (CLK2), which is internal to the EPM5016, feeds a toggle flip-flop to generate a divide-by-two signal (CLK) that matches the 33-MHz system clock signal. CLK tracks the microprocessor clock phase.

The last input signal, RESET, is connected directly to the Reset signal of the microprocessor. RESET feeds the Preset or Reset of each register to set the EPM5016 to the correct start-up state.

Designing with MAX+PLUS

Figure 4 shows the MAX+PLUS-generated schematic for the bus controller placed into the EPM5016. MAX+PLUS enables designs to be created with up to eight levels of hierarchy. Any part of a design may be either a text or a graphic file. In this design, the four symbols with names **BUS_TRCK**, **DECODE**, **WAIT_CNT**, and **TRAN_CTL** are text files designed with the Altera Hardware Description Language (AHDL). AHDL allows designs to be represented with behavioral description such as state machines, arithmetic functions, comparator functions, or Boolean equations.

Figure 4. Bus Controller Schematic



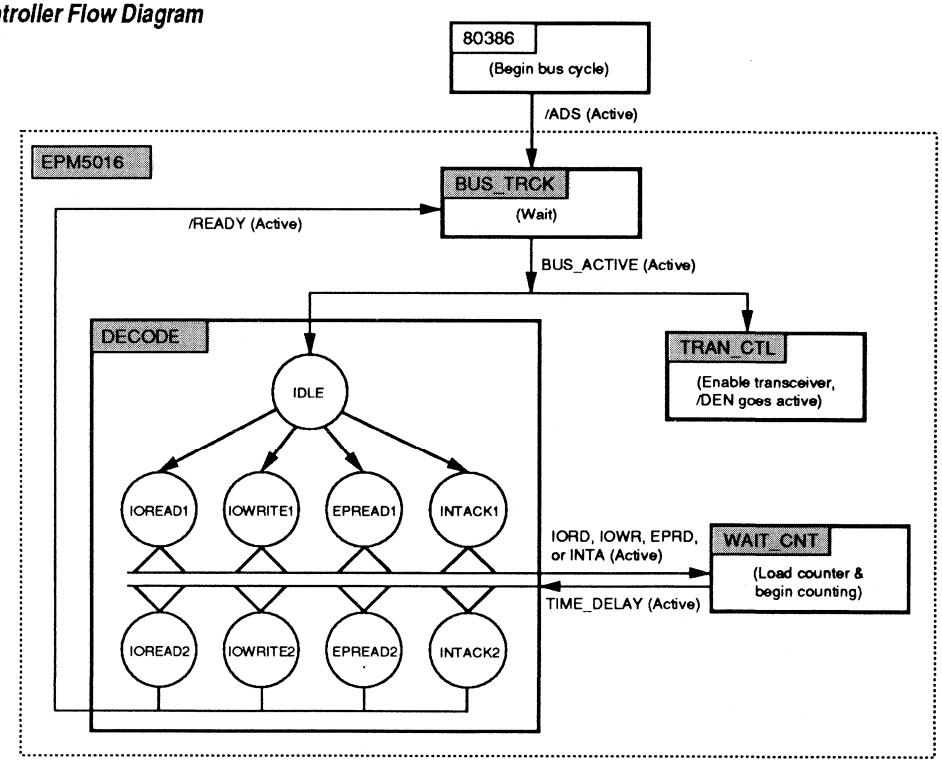
The top level of the design, shown in Figure 4, was created with the MAX+PLUS Graphic Editor. The four symbols (whose underlying logic is expressed in AHDL behavioral descriptions) were automatically created by MAX+PLUS. When the symbols were created, inputs to the text files became pinstubs on the left side of the symbol, and outputs became pinstubs on the right. In this design, the **BUS_TRCK**, **TRAN_CTL**, and **DECODE** functional blocks are state machines. **WAIT_CNT** is a counter with synchronous Preload and Enable.

10

Design Description

Figure 5 shows the flow diagram for the bus controller. First, the 80386 causes **/ADS** to go low, indicating that a bus cycle has begun. **/ADS** then causes **BUS_TRCK** to activate the signal **BUS_ACTIVE**, which indicates that the processor is in an active bus cycle. **BUS_ACTIVE** feeds the functions **DECODE** and **TRAN_CTL**. The **TRAN_CTL** function then scans the 80386 control signals and enables the data transceiver buffers when they are required. **DECODE** also scans the 80386 control signals and decodes them into specific control signals (**IORD**, **IOWR**, **EPRD**, and **INTA**) that drive the peripheral logic and the block function, **WAIT_CNT**. **WAIT_CNT** is a loadable 4-bit counter that counts the required number of wait states for bus cycles. When **WAIT_CNT** finishes the wait state count, it causes **time_delay** to go low, enabling **DECODE** to release the **/READY** signal and finish the bus cycle.

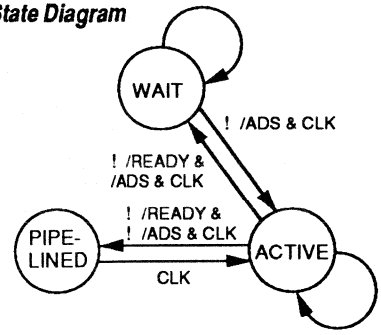
Figure 5. Bus Controller Flow Diagram



BUS_TRCK

The **BUS_TRCK** function is a state machine that determines when the 80386 is in a bus cycle. Figure 6 shows the state diagram for **BUS_TRCK**.

Figure 6. BUS_TRCK State Diagram



The AHDL description of **BUS_TRCK** is shown in Figure 7. **BUS_TRCK** has two output signals: **bus_active** and **pipe_active**, of which only **bus_active** is used in the top level of the design. The **bus_active** signal

Figure 7. BUS_TRCK AHDL File

```
SUBDESIGN bus_trck
(
     clk2,          % 80386 2x clock (66MHz)  %
     clk,           % 80386 33MHz clock       %
     /ads,          % low to begin bus cycles %
     /ready,        % low to end bus cycles   %
     reset:         % high to reset           %
          INPUT;
     bus_active,    % low during active bus cycles   %
     pipe_active:   % low after pipelined bus cycles %
          OUTPUT;
)

VARIABLE
     bus_cycle : MACHINE OF BITS (bus[1..0])  % define state %
                 WITH STATES ( wait,          % bits and     %
                               active,        % state names  %
                               pipelined );
BEGIN
     DEFAULTS
          bus_active = VCC;      % define signals to be  %
pipe_active = VCC;               % normally high         %
     END DEFAULTS;
     bus_cycle.clk = clk2;            % state machine runs at %
     bus_cycle.reset = reset;         % 66 MHz                %
     CASE ( bus_cycle ) IS
          WHEN wait =>                % check for clk to      %
               IF (!/ads & clk) THEN  % determine phase      %
                    bus_cycle = active;
               END IF;
          WHEN active =>
               IF (!/ready & /ads & clk) THEN
                    bus_cycle = wait;
                    ELSIF (!/ready & !/ads & clk) THEN
                         bus_cycle = pipelined;
               END IF;
               bus_active = GND;      % activate bus_active  %
          WHEN pipelined =>
               IF (clk) THEN bus_cycle = active;
               END IF;
               pipe_active = GND;     % activate pipe_active %
     END CASE;
END;
```

is active-low, indicating that the 80386 is executing a non-pipelined bus cycle.

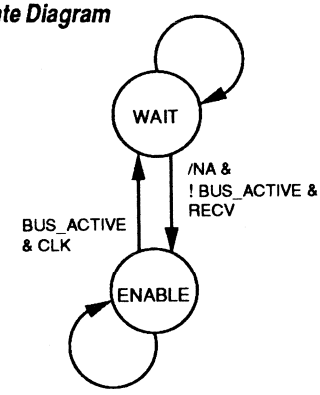
The state machine **BUS_CYCLE** is defined in the Internal Declaration Section of the AHDL file. The register bits for the state machine are defined as **bus1** and **bus0** with the bus format **bus[1..0]**, and the states are defined as **wait**, **active**, and **pipelined**. MAX+PLUS automatically generates the state assignments for each of these states during compilation.

After the **BEGIN** statement, the Clock and the Reset for the state machine are defined with the statements **bus_cycle.clk = clk2** and **bus_cycle.reset = reset**. Next, the transition equations for the state machine are defined with **CASE** and **IF-THEN** statements for the conditional state transfers. When **/ads** goes low and **CLK** is high in the state **wait**, then the state machine changes to the state **active**, and the signal **bus_active** goes low.

TRAN_CTL

The **TRAN_CTL** state machine controls the Output Enable of the data bus's two 74245 data transceivers. See Figure 8.

Figure 8. TRAN_CTL State Diagram



The AHDL file describing **TRAN_CTL**, which is a single-bit state machine called **tran_en**, is shown in Figure 9. When a non-pipelined bus cycle is executed with the statement **na & !bus_active**, the 80386 control signals are read by **tran_en** to determine whether the transceiver needs to be enabled. The enable is released when the bus cycle is completed with the statement **bus_active & clk**.

Figure 9. TRAN_CTL AHDL File

```
SUBDESIGN tran_ctl
(
     clk,          % 80386 clock                    %
     clk2,         % 80386 2x clock (66MHz)         %
     /na,          % low to begin bus cycles        %
     bus_active,   % low during active bus cycles  %
     recv,         % low for recover                %
     reset:        % high for reset of device       %
     INPUT;

     /tri_enable:  % low to enable io transceiver  %
     OUTPUT;
)

VARIABLE

     tran_en : MACHINE OF BITS ( tran )
               WITH STATES (wait, enable);

BEGIN
     tran_en.clk = clk2;
     tran_en.reset = reset;

     CASE ( tran_en ) IS
          WHEN wait =>
             IF (/na & !bus_active & recv)
                  THEN tran_en = enable;
             END IF;
             % set /tri_enable high in the state wait %
             /tri_enable = VCC;
          WHEN enable =>
             IF (bus_active & clk)
                  THEN tran_en = wait;
```

DECODE

DECODE is a state machine that tracks the 80386 control signals to decode the bus cycle. (See Figure 5 for the state machine diagram of **DECODE**.) The AHDL file for **DECODE** is shown in Figure 10. The state machine **io_state** has 10 states and uses 6 state registers (2 more than the required 4) because all of the registers are used as outputs. The state register outputs are defined in the Internal Declarations Section as binary numbers. (AHDL allows all values to be represented in binary, octal, decimal, or hexadecimal formats.)

The state machine **io_state** waits in its initial state called **idle**. When a bus cycle occurs, i.e., **bus_active** goes low, the control signals from the 80386 are tested to see if wait states are required. If so, the state machine moves to a state that deactivates **/READY** to the 80386 and waits for the wait-state counter to time out. The signal **time_delay**, which is an output from **WAIT_CNT**, indicates that the wait-state counter has completed counting the required number of wait states. For example, a read from EPROM would take the state machine to **epread1**, where it would wait for **time_delay** to go high. When **time_delay** goes high, **io_state** moves to **epread2** to enable the processor, and then returns to the **idle** state on the next clock cycle.

Figure 10. DECODE AHDL File

```
SUBDESIGN decode
(    clk2,          % 80386 2x clock (66MHz)                  %
     clk,           % 80386 clock                             %
     /na,           % low to begin bus cycles                 %
     mio,           % high for memory, low for i/o cycles     %
     wr,            % high for write, low for read cycles     %
     dc,            % high for data, low for ctrl cycles      %
     a31,           % processor address line A31              %
     time_delay,    % time delay input                        %
     bus_active,    % low during active bus cycles            %
     reset:         % high for reset of device                %
     INPUT;
     recv,          % low during float and recovery           %
     iord,          % low to read i/o                         %
     iowr,          % low to write i/o                        %
     eprd,          % low to read eproms                      %
     inta,          % low for interrupt acknowledge           %
     iordy:         % low to indicate ready                   %
     OUTPUT;                                                    )
VARIABLE io_state : MACHINE OF BITS ( io[5..0]
     WITH STATES       ( idle    = b"111111",  % state         %
                         ioread1  = b"011111",  % assignements  %
                         ioread2  = b"011101",
                         iowrite1 = b"01111",
                         iowrite2 = b"111101",
                         epread1  = b"110111",
                         epread2  = b"110101",
                         intack1  = b"111011",
                         intack2  = b"111001",
                         recover  = b"111110" );
BEGIN
     (iord,iowr,eprd,inta,iordy,recv) = io[]; % assign outputs %
     io_state.clk = clk2;                     % to state       %
     io_state.reset = reset;                  % register bits  %
     CASE ( io_state ) IS
          WHEN idle =>
               IF (/na & !bus_active & clk) THEN
                    IF (a31 & mio & dc & !wr) THEN
                         io_state = epread1;
                    ELSIF (!a31 & !mio & dc & wr) THEN
                         io_state = iowrite1;
                    ELSIF (!a31 & !mio & dc & !wr) THEN
                         io_state = ioread1;
                    ELSIF (!a31 & !mio & !dc & !wr) THEN
                         io_state = intack1;
                    ELSIF (mio & !dc & wr) THEN
                         io_state = iowrite2;
                    ELSE  io_state = recover;
                    END IF;
               END IF;
          WHEN epread1 =>
               IF (time_delay) THEN io_state = epread2; END IF;
          WHEN epread2 =>
               IF (clk) THEN io_state = idle; END IF;
          WHEN iowrite1 =>
               IF (time_delay) THEN io_state = iowrite2; END IF;
          WHEN iowrite2 =>
               IF (!mio & clk) THEN io_state = recover;
                    ELSIF (mio & clk) THEN io_state = idle; END IF;
          WHEN ioread1 =>
               IF (time_delay) THEN io_state = ioread2; END IF;
          WHEN ioread2 =>
               IF (clk) THEN io_state = recover; END IF;
          WHEN intack1 =>
               IF (time_delay) THEN io_state = intack2; END IF;
          WHEN intack2 =>
               IF (clk) THEN io_state = recover; END IF;
          WHEN recover =>
               IF (time_delay & clk) THEN io_state = idle; END IF;
     END CASE;
END;
```

WAIT_CNT

WAIT_CNT, shown in Figure 11, is a counter that generates the wait states required to accommodate the slow hardware modules in the system. The counter is idle when it is not counting wait states. The counter is activated by **iord**, **iowr**, **inta**, or **eprd**, all of which originate from **DECODE**. Once activated, the counter loads a preassigned value and begins to count until the time-out count reaches zero. At that time, the signal **time_delay** goes low, releasing the bus for the next cycle.

Figure 11. WAIT_CNT AHDL File

```
CONSTANT I/O        = 5;  % define constants %
CONSTANT EPROM      = 1;
CONSTANT RECOVER    = 10;
CONSTANT TIME_UP    = 15;
CONSTANT IDLE       = 0;

SUBDESIGN wait_cnt
(
     clk2,          % 80386 CLK2                      %
     iord,          % low to read io                  %
     iowr,          % low to write to                 %
     eprd,          % low to read EPROMs              %
     inta,          % low for interrupt acknowledge   %
     recv,          % low during float and recovery   %
     reset:         % high to reset                   %
     INPUT;
     time_delay:    % time delay output %
     OUTPUT;
)

VARIABLE
     timer[3..0] : DFF;

BEGIN
     timer[].clk = clk2;
     timer[].clrn = !reset;
     % load counter %
     IF (timer[] == IDLE) THEN
          IF (!iord # !iowr # !inta) THEN timer[] = I/O;
               ELSIF (!eprd) THEN timer[] = EPROM;
          ELSIF (!recv) THEN timer[] = RECOVER;
               ELSE timer[] = IDLE;
          END IF;
     % check to see if count is done %
     ELSIF (timer[] == TIME_UP) THEN time_delay = VCC;
          IF (iord & iowr & eprd & inta) THEN timer[] = IDLE;
               ELSE timer[] = TIME_UP;
          END IF;
     % count %
     ELSE timer[] = timer[] + 1;
     END IF;
END;
```

The constants at the beginning of the file shown in Figure 11 define the different number of wait states required for each type of read/write function. The constants are then included in the equations that are used to load the timer. Constants allow descriptive names, rather than numbers, to be placed into equations. For example, in Figure 11, **EPROM** is assigned the value **1** in the Constant Section. When data is being read from **EPROM**, the timer is loaded with this value. The cycle is completed once the counter has reached 15 (14 clock wait states).

The conditional equations for the counter in **WAIT_CNT** are created with **IF-THEN** statements. The counter registers are defined in the Internal Declarations Section with the statement **timer[3..0] : DFF**, which assigns the names **timer3**, **timer2**, **timer1**, and **timer0** to four D-type flip-flops. The statement **timer = timer + 1** at the end of the AHDL file causes the registers to count.

Compilation

Once the design for the EPM5016 (Figure 4) has been entered, it is submitted to the MAX+PLUS Compiler for processing. The Compiler extracts a netlist from the graphic and text files defining the hierarchical design and checks it against design rules. It then performs logic synthesis, minimizing and optimizing it for MAX architecture. If any errors are detected during compilation, MAX+PLUS highlights the exact location in either the schematic or the AHDL file where the error occurred.

The Compiler's Fitter module then maps the design into the EPM5016. It uses a set of heuristic algorithms that automatically route design signals. A Report File that documents the resource utilization for the design is also generated at this time. Figure 12 shows a portion of the Report File for the bus controller design. Sixteen macrocells are required for this design, 8 of which are buried. In addition, 19 expanders are used, leaving 13 expanders for additional logic.

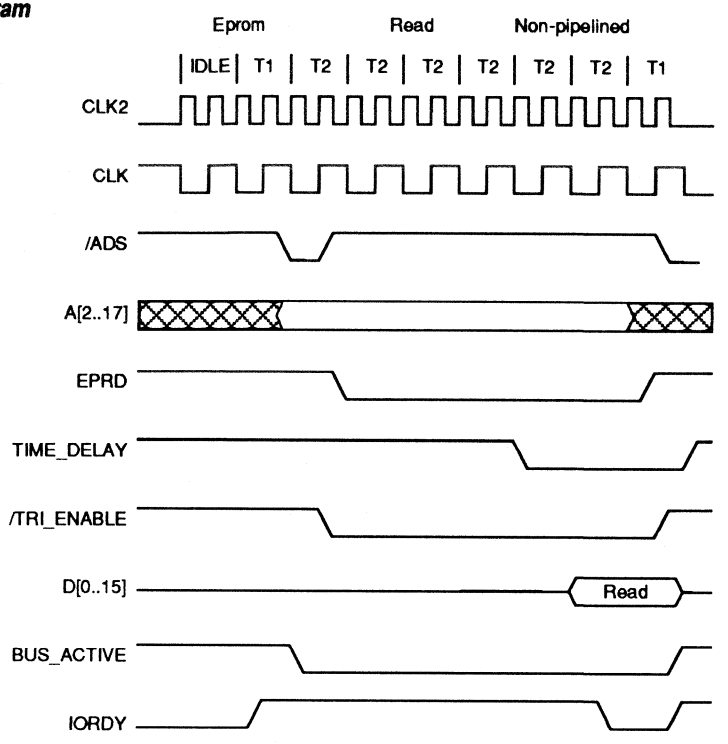
Figure 12. Report File Excerpt

```
** RESOURCE USAGE **

Logic Array Block      Macrocells    I/O Pins     Expanders

A:      MC1 - MC16    16/16(100%)   8/ 8(100%)  19/32( 59%)

Total dedicated input pins used:      8/  8 (100%)
Total I/O pins used:                  8/  8 (100%)
Total macrocells used:               16/ 16 (100%)
Total expanders used:                19/ 32 ( 59%)

Total input pins required:            9
Total output pins required:           7
Total bidirectional pins required:    0
Total macrocells required:           16
Total expanders in database:         17
```

A Read from EPROM

Figure 13 shows the bus cycle timing for the 80386. In the EPROM interface, the **OE** input of each EPROM device is connected directly to the **EPRD** signal from the EPM5016 bus controller, and **/CE** is always grounded. The wait-state requirement is calculated by adding worst-case delays and comparing the total delay time to the 80386 bus cycle time.

Figure 13. 80386 Bus Cycle Timing Diagram



The EPROM output signals become valid 200 ns after an address is placed on the address signals. To activate the **/READY** signal, the number of wait states required must include the EPROM address-to-valid delay *plus* the latch clock-to-output delay *minus* one state machine cycle. Thus, the total delay required, in nanoseconds, is:

$$t_{WAIT} = t_{EPROM} + t_{LATCH} - t_{CYCLE}$$
$$198 = 200 + 13 - 15$$

Each clocked wait state is 15 ns. Therefore, 14 wait-state cycles are required to meet the specification of the EPROM hardware (15 * 14 = 210; 210 > 198).

Simulation

Input vectors for simulation are created either in text format via a Vector File, or in graphic format by drawing the waveforms in the MAX+PLUS Waveform Editor. In the Vector File example shown in Figure 14, the input vectors are defined to simulate a read from the EPROM.

10

Figure 14. Vector File

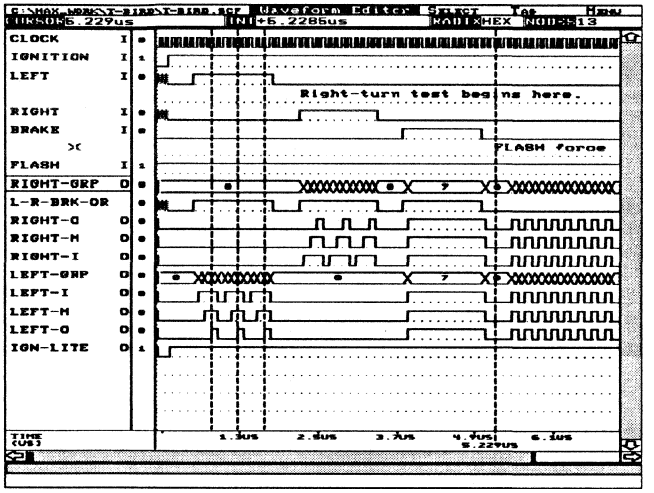
```
GROUP CREATE timer = !WAIT_CNT:66!timer3.Q
                     !WAIT_CNT:66!timer2.Q
                     !WAIT_CNT:66!timer1.Q
                     !WAIT_CNT:66!timer0.Q ;


OUTPUTS CLK READY EPRD DEN
                  !TRAN_CTL:59!tran_en
                  !BUS_TRCK:36!bus_cycle
                  !DECODE:65!io_state
                  timer
                  CLK ;

INPUTS  RESET;
PATTERN
0>    1
100>  0;

INPUTS  CLK2;           % define CLK2 to be 66 MHZ %
INTERVAL  7.5;
START  200; STOP  5us;
PATTERN  1 0;

% define static inputs %
INPUTS  W/R PA31 MA M/IO D/C DRAMRDY;
PATTERN
0>      0   1   1   1    1   1;

INPUTS ADS;
PATTERN
0> 1
430> 0
490> 1;
```

The first statement in the Vector File, which begins with the word **GROUP**, groups the timer bits together into a bus. The actual input vectors are easily entered. The input **RESET**, for example, is defined to be a 1 for the first 100 ns, then 0 until the end of simulation. For the **CLK2** input, the statement **INTERVAL 7.5** defines the cycle transition time to be 7.5 ns. The **START** and **STOP** commands define the period of time for the vector pattern to repeat. The command **PATTERN 1 0;** causes **CLK2** to toggle. The other input signals are similarly specified.

The outputs from simulation can also be viewed in either graphic or text format. The MAX+PLUS Simulator automatically generates a graphic Channel File that contains waveforms of simulation inputs and outputs, which can be viewed and edited in the Waveform Editor. For example, if some inputs must be changed for further simulation, the Channel File can be edited graphically in the Waveform Editor and then submitted again to the Simulator for further simulation.

Figure 15 shows the output waveforms generated by simulating with the Vector File shown in Figure 14. The state machines are displayed with the state names rather than the individual state bits in the waveforms.

Figure 15. Simulator Channel File



Conclusion

The MAX architecture and the advanced MAX+PLUS software tools allow designers to enter and fit a broad range of complex designs into the EPM5016. In addition, the high-drive, 24-mA outputs of the EPM5016 allow direct connection to most system buses. As systems are being challenged to run faster and faster, the EPM5016 rises to the occasion by offering counter speeds up to 100 MHz. Complex state machines, such as the bus controller shown in this application note, may be clocked up to 66 MHz, which is twice the speed of today's fastest processors.

References

Intel Corporation. *386 Microprocessor Hardware Reference Manual* (1988).

Intel Corporation. *386DX Microprocessor Data Sheet* (April 1989).

10

Notes:

Designing with AHDL

October 1990, ver. 1

Application Note 22

Introduction

Text-based logic design has been greatly simplified by the introduction of advanced behavioral languages that integrate very complex logic with easy-to-understand text descriptions. The Altera Hardware Description Language (AHDL) provides such powerful text-design support for Altera's Multiple Array MatriX (MAX) family of EPLDs. AHDL incorporates the benefits of earlier-generation text languages—such as ABEL, CUPL, PALASM—and the Altera Design File (ADF) and State Machine File (SMF) formats. With AHDL, Altera has gone beyond these languages by adding features for specific applications, such as state machines, truth tables, and **IF-THEN-ELSE** decision logic. AHDL allows any logic function to be easily described, providing a comprehensive tool for text-based design.

AHDL is completely integrated into Altera's MAX+PLUS design environment. AHDL designs can be created with the MAX+PLUS Text Editor or any standard text editor. These text designs can be freely combined with schematics in MAX designs. The MAX+PLUS Compiler then performs logic synthesis and automatic fitting to use EPLD resources most efficiently. A full timing simulator is also provided for rapid design debugging. For more information on the MAX+PLUS development software, refer to the *PLS-MAX: MAX+PLUS Programmable Logic Software Data Sheet* in this data book.

AHDL Constructs

AHDL provides several behavioral constructs that define the syntax for different types of logic. This application note describes the following AHDL constructs:

- Groups
- Logic, arithmetic, and comparison operators
- Function prototypes
- **IF-THEN** statements
- **CASE** statements
- Truth tables
- State machines

Complex logic functions are easily implemented with AHDL constructs, which allow any logic function to be described in a compact, readable format. The features of these powerful constructs are discussed here. Examples are given in "AHDL Design Examples" later in this application note.

10

Groups for Bus Designs

Groups in AHDL are analogous to buses in schematic capture designs. The group construct allows a collection of bits to be treated as a single element. AHDL groups may be defined with the following two notations:

- A symbolic name followed by a range of decimal numbers enclosed in brackets, e.g., **data[15..0]**
- A list of symbolic names separated by commas enclosed in parentheses, e.g., **(a,b,c,d)**

Groups may be combined with other AHDL constructs to simplify implementation of bus-oriented designs.

Logical, Arithmetic, and Comparison Operations

AHDL provides a complete set of logical, arithmetic, and comparison operators. Any number of operators can be applied to define a node or group of nodes. Operations may be performed on groups, single nodes, numbers, and constants. The arithmetic operators can be used for functions such as adders and counters; comparison operators allow decoding logic to be implemented quickly. A complete list of AHDL operators is shown in Table 1.

Table 1. AHDL Operators

Operator	Function	Example
!, NOT	One's complement	!eof
&, AND	Logical AND	bread & butter
!&, NAND	Logical NAND	a[3..1] !& b[6..4]
#, OR	Logical OR	trick # treat
!#, NOR	Logical NOR	here !# there
$, XOR	Exclusive OR	(a,b,c) $ data[2..0]
!$, XNOR	Exclusive NOR	x2 !$ x4
+	Addition	count[4..1] + 1
-	Two's complement	-data[7..0]
-	Subtraction	a[15..0] - b[31..16]
==	Equal to	count[7..0] == 200
!=	Not equal to	(a[] & MASK) != 0
<	Less than	(a[] + b[]) < c[]
<=	Less than or equal to	comp[23..0]<= H"ABCDEF"
>	Greater than	count[] > MAXIMUM
>=	Greater than or equal to	address[15..14] >= 2

Function Prototypes for Hierarchical Text Design

A designs can incorporate ("call") another lower-level design with the **FUNCTION** construct, which specifies the inputs and outputs of a lower-level function. The logic of a lower-level design may then be called any number of times within an AHDL design. Any user-created schematic or text design can be called with a function prototype. In addition, MAX+PLUS provides a library of over 300 TTL and custom macrofunctions that also can be integrated into AHDL.

IF-THEN and Case Statements for Conditional Logic

For conditional logic, AHDL provides both **IF-THEN** and **CASE** statements. The **IF-THEN** statement describes conditional logic based on the evaluation of one or more Boolean expressions. The keywords **ELSE** and **ELSIF** prioritize the order in which conditions are evaluated.

The **CASE** statement is an alternative method for describing conditional logic. It is more compact than the **IF-THEN** statement when multiple conditions of a single Boolean expression are evaluated. For example, a **CASE** statement can easily define logic based on the different possible values of an address bus. The **CASE** statement is also especially helpful for describing state machine transitions. The keyword **OTHERS** in a **CASE** statement allows default operation when the group value is not specified in one of the **WHEN** clauses. Both the **IF-THEN** and **CASE** statements can be nested for very complex conditional logic.

Truth Table for Decode Logic

The **TABLE** construct in AHDL decodes output values for a set of inputs. The **TABLE** construct eliminates the need to extract Boolean expressions from a function table; the table itself is directly implemented with the construct. Functions that are intuitively represented with a truth table—such as BCD decoders, address decoders, and state machine transition logic—can be entered with this construct.

State Machines for Control Logic

Control logic based on state machines is implemented with the **MACHINE** construct. A designer simply defines state names and describes the state transitions to implement a state machine. The MAX+PLUS Compiler automatically selects the most efficient register type (D or T) and calculates the next-state equations. The determination of the required number of state bits and the assignment of these bits can also be automatic, or they can be completely controlled by the designer. State transitions and state machine outputs are easily described with **CASE**, **IF-THEN**, or **TABLE** constructs.

A single design can contain any number of these AHDL constructs. This flexibility enables a system designer to define a complete behavioral description for any MAX EPLD design quickly and efficiently.

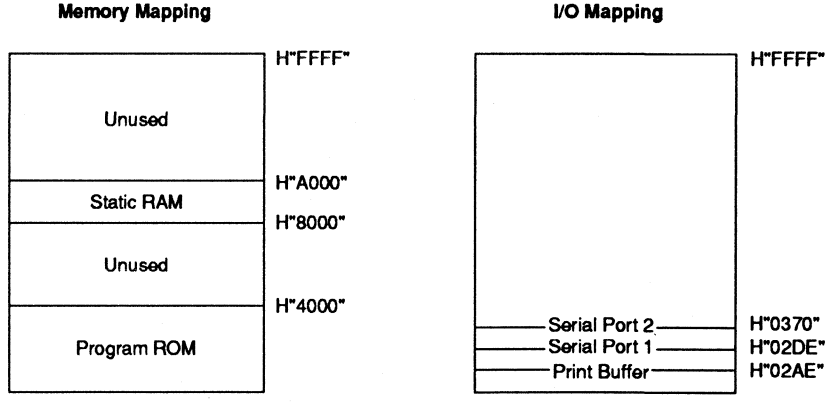
AHDL Design Examples

The sample designs provided in this section show all AHDL keywords in upper case and user-defined symbolic names in lower case. Consistent formatting (e.g., indentation of **IF-THEN** and **CASE** statements) and comments enclosed in percent symbols (%) are used to increase readability. However, designers are free to use whatever formatting method suits them.

Address Decoders

EPLDs are commonly used to decode microprocessor address lines to provide all system memory and I/O chip selects. I/O and memory mapping for a generalized 16-bit microprocessor is shown in Figure 1. This system contains a bank of ROM and a bank of RAM in memory space. I/O space includes a print buffer and two serial ports.

Figure 1. 16-Bit Microprocessor Memory and I/O Space



AHDL groups and comparison operators implement the address decoding for this system. Grouped address lines can be treated as a single element, which is then compared to the boundaries defined by the memory and I/O mapping in Figure 1. Figure 2 shows an AHDL design that implements the address decoder.

Figure 2. Address Decoder with Group Operations

```
SUBDESIGN decode (

    % The SUBDESIGN declaration of AHDL defines the interface %
    % to the logic function. Entering this section is         %
    % equivalent to entering input, output, and bidirectional %
    % pins in a schematic.                                    %

    addr[15..0],m/io           :INPUT;
    rom, ram, print, sp[2..1]  :OUTPUT;
)

BEGIN

    % The BEGIN keyword marks the beginning of the SUBDESIGN  %
    % body. The designer describes the behavior of the        %
    % function in this section. The memory map in Figure 1    %
    % is described in the SUBDESIGN body with AHDL comparison %
    % operators. This example is for a generalized 16-bit     %
    % microprocessor system.  This function can be easily     %
    % modified to implement the address decoding for any      %
    % microprocessor-based system.                            %

     rom   =  m/io & (addr[] <  H"4000");
     ram   =  m/io & (addr[] <  H"A000")
                   & (addr[] >= H"8000");
     print = !m/io & (addr[] == H"02AE");
     sp1   = !m/io & (addr[] == H"02DE");
     sp2   = !m/io & (addr[] == H"0370");

END;
```

Address decoders can also be described with the **TABLE** construct. Figure 3 shows the address decoder described in Figure 2 implemented with a truth table.

Figure 3. Address Decoder with Truth Table

```
SUBDESIGN decode (

     addr[15..0], m/io          :INPUT;
     rom, ram, print, sp[2..1]  :OUTPUT;
)

BEGIN

    TABLE
    m/io,   addr[15..0]       => rom, ram, print, sp[];
     1   , B"00XXXXXXXXXXXXXX" =>  1 ,  0 ,    0  , B"00";
     1   , B"10XXXXXXXXXXXXXX" =>  0 ,  1 ,    0  , B"00";
     0   , B"0000001010101110" =>  0 ,  0 ,    1  , B"00";
     0   , B"0000001011011110" =>  0 ,  0 ,    0  , B"01";
     0   , B"0000001101110000" =>  0 ,  0 ,    0  , B"10";
    END TABLE;

END;
```

The choice of which method to use is up to the designer. The truth table method provides a more compact format when the address range can be decoded with one line in a table. The comparison operators provide quicker and more intuitive integration when multiple lines of a table are required. However, both methods produce the same results.

Counters

Almost every design requires at least one counter for timing or control logic. EPLDs contain a superior mix of combinatorial and registered logic, making them ideal for integrating counters.

The MAX+PLUS TTL MacroFunction Library contains all of the common TTL building blocks used to integrate counter logic. The hierarchical features of AHDL allow easy integration of these TTL-equivalent counters.

Figure 4 shows an AHDL design that integrates two **74161** counters to create an eight-bit counter. This design is easily modified to link several **74161**s or other TTL counters. A counter of any width can be created by linking the appropriate number of functions.

Figure 4. Eight-Bit Counter with Two 74161 Macrofunctions

```
FUNCTION 74161    (d,c,b,a,ldn,enp,ent,clrn,clk)
         RETURNS (rco,qd,qc,qb,qa);

% The Function Prototype defines the inputs and outputs of     %
% the 74161 in the MAX+PLUS TTL MacroFunction Library.         %

SUBDESIGN ttlcount (
     clk, ld, en, clr, d[7..0]   :INPUT;
     c8, q[7..0]                 :OUTPUT;
)

VARIABLE
     c4 :NODE;

BEGIN
     (c4,q[3..0]) = 74161(d[3..0], !ld, en, en, !clr, clk);
     (c8,q[7..0]) = 74161(d[7..4], !ld, en, c4, !clr, clk);
END;
```

Counters can also be described with the powerful operators and constructs built into AHDL. Figure 5 shows a nine-bit counter that provides the same load, hold, and count functions as the **74161**.

Figure 5. Nine-Bit Counter with Load and Enable

```
SUBDESIGN ahdlcnt (
   clk, ld, en, clr, d[8..0]  :INPUT;
   q[8..0]                    :OUTPUT;
)

VARIABLE
     count[8..0] :DFF;

BEGIN
     count[].clk = clk;
     count[].clrn = !clr;
     IF ld THEN
          count[] = d[];
     ELSIF en THEN
          count[] = count[] + 1;
     ELSE
          count[] = count[];
     END IF;
     q[] = count[];
END;
```

This example can be modified to integrate a counter of any width by changing the width of the **d[ ]**, **q[ ]**, and **count[ ]** groups. To incorporate more counter functions, additional conditional logic may be specified. For example, adding count-down capability to the previous design requires an additional input signal and a slight modification to the **IF–THEN** statement. The modified design is shown in Figure 6.

Figure 6. Nine-Bit Up/Down Counter with Load and Enable

```
SUBDESIGN ahdlcnt (
     clk, ld, en, clr, up/down, d[8..0]   :INPUT;
     q[8..0]                              :OUTPUT;
)

VARIABLE
     count[8..0] :DFF;

BEGIN
     count[].clk = clk;
     count[].clrn = !clr;
     IF ld THEN
          count[] = d[];
     ELSIF en THEN
          IF up/down THEN
               count[] = count[] + 1;
          ELSE
               count[] = count[] - 1;
          ENDIF;
     ELSE
          count[] = count[];
     END IF;
    q[] = count[];
END;
```

Arithmetic Logic

AHDL provides arithmetic and comparison operators to create compact arithmetic designs. A function frequently described in MAX EPLDs is an integrator that stores the result of an adder and feeds the result back to the adder. With this function, a series of numbers can be integrated into a sum.

Figure 7 shows an example of a 16-bit integrator. The 16-bit result is cleared to zero when the **clear** input signal is high. Each clock adds the value represented by the 8-bit bus (**add[ ]**) to the previously calculated sum in a bank of registers (**result[ ]**). The result is connected to the output signals (**sum[ ]**).

Figure 7. 16-Bit Integrator

```
SUBDESIGN 16int (
     clock, add[7..0], clear    : INPUT;
     sum[15..0]                 : OUTPUT;
)

VARIABLE
     result[15..0]              : DFF;

BEGIN
     result[].clk = clock;
     result[].clrn = !clear;
     result[] = result[] + add[];
     sum[] = result[];
END;
```

DRAM Controller State Machine

Dynamic RAM devices (DRAMs) are useful for inexpensive, efficient, high-density storage. However, DRAMs require controllers that ensure proper operation. Part of the controller is typically based on a state machine that controls the active-low row-address and column-address strobes and select lines that select the row, column, or refresh addresses. Figure 8 shows an AHDL design that controls these signals (**/ras**, **/cas**, and **s[ ]**) for a simple DRAM system.

This simple example of a DRAM controller—which uses state transitions defined with **CASE** and **IF-THEN** statements—can be modified to create a more complex controller. The implementation of an 8-Mbyte DRAM system is described in "AHDL Top-Down Design Methodology" in this application note.

Figure 8. DRAM Controller State Machine

```
SUBDESIGN dram_sm (
    clk, /reset, /mreq, refresh     :INPUT;
    /ras, /cas, s[1..0]             :OUTPUT;
)

VARIABLE
    control : MACHINE OF BITS (ras, cas, sel[1..0])
              WITH STATES (
                          idle = B"0000",
                    strobe_row = B"1000",
                    strobe_col = B"1110",
                  refresh_dram = B"1011"
                        );

BEGIN
    % When a memory request (/mreq) is received, the          %
    % controller generates a /ras, then switches multiplexer  %
    % select and generates a /cas. When a refresh is          %
    % requested, the controller activates /ras and waits for  %
    % refresh request to be deasserted before going back to   %
    % the idle state.                                         %

    control.clk = clk;
    control.reset = !/reset;
    CASE control IS
        WHEN idle =>
            IF refresh THEN
                control = refresh_dram;
            ELSIF !/mreq THEN
                control = strobe_row;
            END IF;
        WHEN strobe_row =>
            control = strobe_col;
        WHEN strobe_col =>
            control = idle;
        WHEN refresh_dram
            IF !refresh THEN
                control = idle;
            END IF;
    END CASE;
    /ras = !control.ras;
    /cas = !control.cas;
     s[] =  control.sel[];
END;
```

Sync Detector State Machine

A truth table is also very efficient for representing state machine transitions. Figure 9 shows a function that synchronizes a serial receiver to the incoming data stream. This synchronization-detector state machine searches an incoming serial data stream for a pattern of six successive 1s.

10

Figure 9. Synchronous Detector State Machine

```
SUBDESIGN sync_det (
    clock, data_in  : INPUT;
    sync            : OUTPUT;
)

VARIABLE
    detect : MACHINE OF BITS (q[2..0])
             WITH STATES (
                 zero,
                 one,
                 two,
                 three,
                 four,
                 five );

BEGIN
    detect.clk = clock;
    TABLE
           detect, data_in => detect,  sync;
           zero   ,    1   => one    ,    0;
           one    ,    1   => two    ,    0;
           one    ,    0   => zero   ,    0;
           two    ,    1   => three  ,    0;
           two    ,    0   => zero   ,    0;
           three  ,    1   => four   ,    0;
           three  ,    0   => zero   ,    0;
           four   ,    1   => five   ,    0;
           four   ,    0   => zero   ,    0;
           five   ,    1   => five   ,    1;
           five   ,    0   => zero   ,    0;
    END_TABLE;
END;
```

This design samples the incoming data stream and moves to the subsequent state whenever the serial input is **1**. Any **0** sampled in the stream causes the state machine to make a transition back to state **zero**. The synchronous signal is true whenever the machine is in state **five** (i.e., five successive **1**s have been sampled) and the **data_in** is at **1**.

State names are not assigned state values in this example because the MAX+PLUS Compiler automatically determines an efficient set of state-bit assignments. However, some state machines have state bits that are used as outputs, in which case the state bit assignments must be explicitly specified as shown in Figure 8.

MAX+PLUS Text Editor

Any ASCII text editor may be used to create AHDL designs. However, the MAX+PLUS Text Editor provides commands that specifically support EPLD design. Like the MAX+PLUS Graphic Editor, the Text Editor provides automatic error location. Whenever an error is detected during compilation, an error message is generated and the line containing the error is highlighted. Automatic error location allows the designer to quickly debug a text file for syntax and electrical rule errors (e.g., outputs tied together).

The MAX+PLUS Text Editor also offers delay prediction, which allows a logic designer to specify a signal path in an AHDL design for analysis. The delay prediction command then reports the worst-case timing for the path. For example, the worst-case frequency of a state machine may be determined by performing delay prediction on paths from state bit to state bit. These error location and delay prediction features enable designers to quickly create AHDL designs.

AHDL Top-Down Design

The following four steps illustrate the top-down approach to AHDL design:

1. Create a block diagram of the design. In this step, the major functions and the communication between them are defined.

2. Declare the inputs and outputs of the design. The required input, output, and bidirectional signals can be determined from the block diagram, and then declared in the AHDL Subdesign Section.

3. Declare the major functions of the design. Each block of the diagram is a major function that can be easily translated into AHDL variables. These variables are declared in the Internal Declarations Subsection.

4. Describe the major functions. The internal logic for each of the major functions is implemented with AHDL constructs in the body of the Subdesign Section.

Entering comments in the AHDL design file also enhances the readability of an AHDL design.

The following example follows this top-down design method to create a DRAM controller. This controller controls an 8-Mbyte DRAM system organized in 32-bit words. Refer to *Application Brief 85 (DRAM Controller Using an EP1830 EPLD)* for a complete operational description of this design.

Step 1: Block Diagram

Figure 10 shows the block diagram of an 8-Mbyte DRAM controller. The design controls all access to dynamic RAM organized on a 32-bit bus. The controller interfaces directly to a microprocessor address and control bus and a 20-MHz clock signal. Reset capability is also provided. This system produces row address strobe (**RAS**), column address strobe (**CAS**), and the dynamic RAM address. In addition, the controller provides the data strobe-acknowledge (**DSACK**) to the processor. The controller consists of a refresh timer, a refresh counter, an address multiplexer and a **RAS/CAS** generator. The state transition diagram for the **RAS/CAS** generation is shown in Figure 11.

Figure 10. Eight-Mbyte DRAM Controller

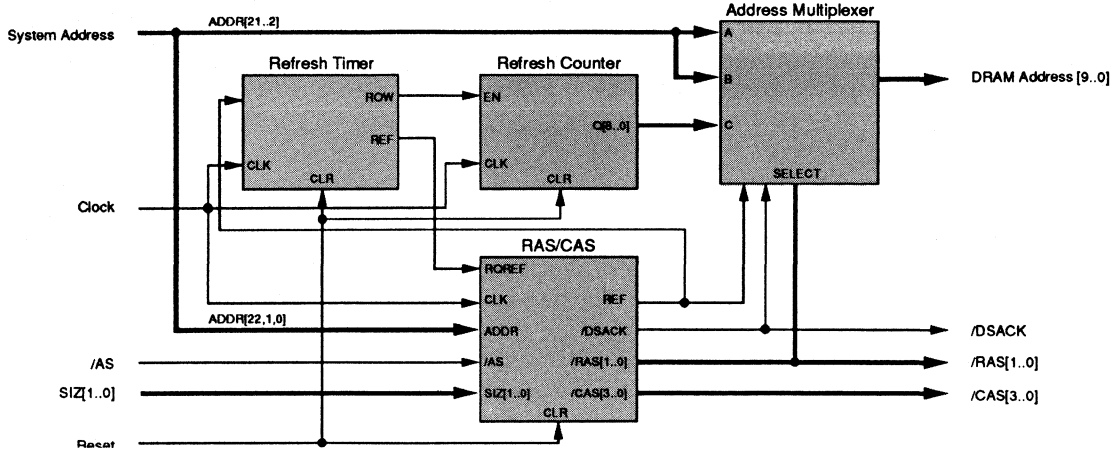
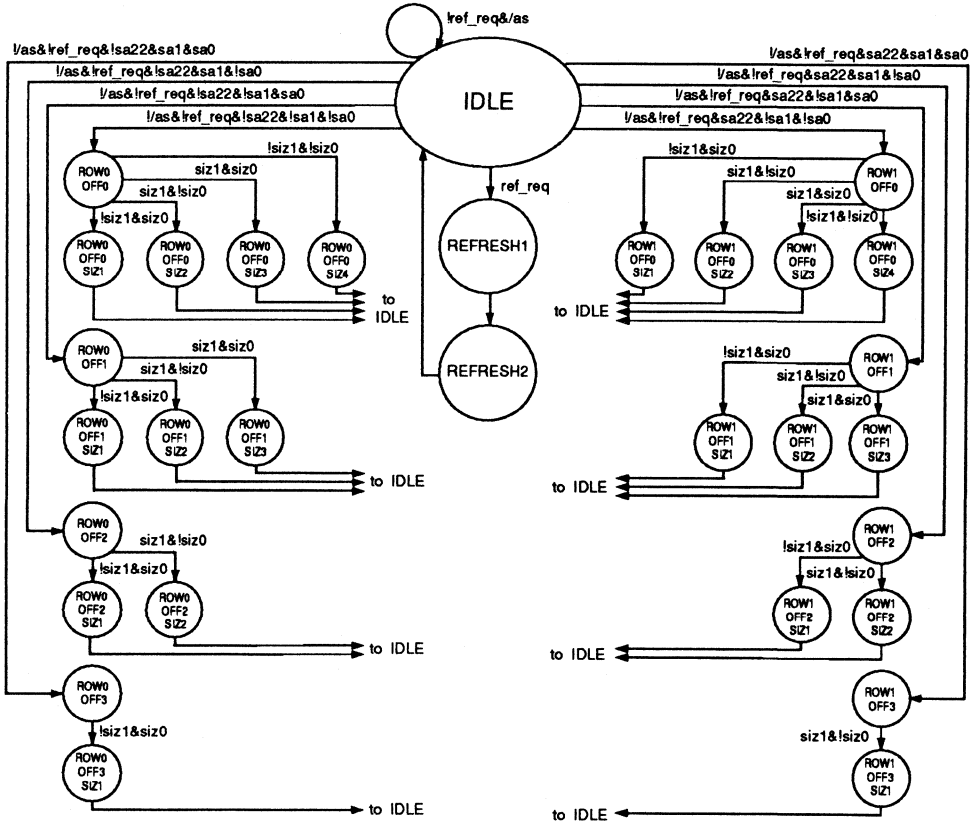


Figure 11. State Transition Diagram for RAS/CAS Generator



Step 2: Declare the Inputs & Outputs

The interface to the DRAM controller can be determined from the block diagram. The input and output declarations for this design are shown below.

```
TITLE "8-Megabyte DRAM Controller";

CONSTANT MAX_REF_TIMER = 311;

SUBDESIGN dram_ctl (

    sa[22..0],              % system address                %
    clock,                  % 20-MHz clock                  %
    size[1..0],             % size of memory operation      %
    /as,                    % address strobe               %
    reset         :INPUT;   % master reset                  %

    da[9..0],               % DRAM address                  %
    /dsack,                 % data strobe acknowledge       %
    /ras[1..0],             % row address strobe            %
    /cas[3..0]    :OUTPUT;  % column address strobe         %
)
```

Step 3: Declare the Major Functions

Each block in the block diagram is a major function of the DRAM controller. These major functions are declared as variables in the Internal Declarations Section of an AHDL design:

```
VARIABLE

    address_mux[9..0]   :NODE;

    % The address multiplexer is a purely combinatorial     %
    % function. It selects the row, column, or refresh      %
    % address based on the current cycle. The variable      %
    % type NODE defines a general-purpose signal that       %
    % allows intermediate logic to be defined.              %

    ref_address[8..0]   :DFF;

    % The refresh counter contains a 9-bit counter that     %
    % requires 9 registers to be declared. The variable     %
    % type DFF defines a D-type register.                   %

    ref_timer[8..0]     :DFF;
    timeout             :NODE;

    % The refresh timer also requires a 9-bit               %
    % counter. This function also provides a timeout        %
    % signal that is assigned to type NODE.                 %
```

```
ref_req :DFF;
control :MACHINE OF BITS(cas[3..0],ras[1..0],dsack,ref)
    WITH STATES (
                                    idle = B"00000011",
                                 row0off0 = B"00000111",
                                 row0off1 = B"00000111",
                                 row0off2 = B"00000111",
                                 row0off3 = B"00000111",
                              row0off0siz1 = B"00010101",
                              row0off0siz2 = B"00110101",
                              row0off0siz3 = B"01110101",
                              row0off0siz4 = B"11110101",
                              row0off1siz1 = B"00100101",
                              row0off1siz2 = B"01100101",
                              row0off1siz3 = B"11100101",
                              row0off2siz1 = B"01000101",
                              row0off2siz2 = B"11000101",
                              row0off3siz1 = B"10000101",
                                 row1off0 = B"00001011",
                                 row1off1 = B"00001011",
                                 row1off2 = B"00001011",
                                 row1off3 = B"00001011",
                              row1off0siz1 = B"00011001",
                              row1off0siz2 = B"00111001",
                              row1off0siz3 = B"01111001",
                              row1off0siz4 = B"11111001",
                              row1off1siz1 = B"00101001",
                              row1off1siz2 = B"01101001",
                              row1off1siz3 = B"11101001",
                              row1off2siz1 = B"01001001",
                              row1off2siz2 = B"11001001",
                              row1off3siz1 = B"10001001",
                                 refresh0 = B"00001110",
                                 refresh1 = B"00001110");

% The RAS/CAS generator contains a latch for refresh   %
% request and a state machine named control. The       %
% state names are extracted from the state machine     %
% transition diagram.                                  %
```

Step 4: Describe the Major Functions

Each of the major functions is described with AHDL constructs. The subdesign body of the AHDL design is shown below. Comments in this section describe the operation of each function.

```
BEGIN

%-------------------------------------------------------%
%                   Address Multiplexer                 %
%-------------------------------------------------------%

% The address multiplexer controls the address that is   %
% connected to the DRAM devices. During the idle state   %
% and the RAS-generation states, this multiplexer selects %
% the row address (sa[21..12]). During the CAS-generation %
% states, the column address (sa[11..2]) is selected.    %
% During refresh cycles, the refresh address is selected. %

    IF ref THEN
        da[8..0] = ref_address[];
    ELSIF dsack THEN
        da[] = sa[11..2];
    ELSE
        da[] = sa[21..12];
    END IF;

    da[] = address_mux[];

%-------------------------------------------------------%
%                     Refresh Counter                   %
%-------------------------------------------------------%

% The refresh counter increments the refresh address every %
% refresh cycle. The counter is initialized to zero by a  %
% reset signal.                                           %

    ref_address[].clk = clock;
    ref_address[].clrn = !reset;
    IF timeout THEN
        ref_address[] = ref_address[] + 1;
    ELSE
        ref_address[] = ref_address[];
    END IF;

%-------------------------------------------------------%
%                     Refresh Timer                     %
%-------------------------------------------------------%

% A refresh cycle is required on all 512 rows every 8 ms. %
% With a clock frequency of 20 MHz, the number of cycles  %
% between refresh requests is 312. The refresh timer is a %
% free-running counter that counts from 0 to 311. Refresh %
% request is generated when the counter is at 311.        %

    ref_timer[].clk = clock;
    ref_timer[].clrn = !reset;
    IF (ref_timer[] == MAX_REF_TIMER) THEN
        ref_timer[] = 0;
        timeout = VCC;
    ELSE
        ref_timer[] = ref_timer[] + 1;
    END IF;
```

```
%-------------------------------------------------------%
%                    RAS/CAS Generator                  %
%-------------------------------------------------------%

% The RAS/CAS generator controls cycling through the RAS, %
% CAS, and refresh cycles. The RAS/CAS generator waits in %
% the idle state until either a refresh request (refreq)  %
% or an address strobe (/as) is received. When a refresh  %
% is received, /ras1 and /ras0 are asserted for two clock %
% cycles. The first cycle after /as is received, the      %
% appropriate /ras is asserted. The next clock cycle      %
% asserts the appropriate /cas signals based on the offset %
% and size of the request. The /dsack signal is also      %
% asserted during this cycle. The third cycle deasserts   %
% /ras, /cas, and /dsack, and returns to the idle state.  %

    ref_req.clk = clock;
    ref_req = timeout
            # ref_req & control.ref;
    control.clk = clock;
    control.reset = reset;
    CASE control IS
        WHEN idle =>
            IF ref_req THEN
                control = refresh0;
            ELSIF !/as & (sa[1..0] == 0) THEN
                IF sa22 THEN
                    control = row1off0;
                ELSE
                    control = row0off0;
                END IF;
            ELSIF !/as & (sa[1..0] == 1) THEN
                IF sa22 THEN
                    control = row1off1;
                ELSE
                    control = row0off1;
                END IF;
            ELSIF !/as & (sa[1..0] == 2) THEN
                IF sa22 THEN
                    control = row1off2;
                ELSE
                    control = row0off2;
                END IF;
            ELSIF !/as & (sa[1..0] == 3) THEN
                IF sa22 THEN
                    control = row1off3;
                ELSE
                    control = row0off3;
                END IF;
            END IF;
        WHEN row0off0 =>
            IF (size[] == 1) THEN
                control = row0off0siz1;
            END IF;
            IF (size[] == 2) THEN
                control = row0off0siz2;
            END IF;
            IF (size[] == 3) THEN
                control = row0off0siz3;
            END IF;
```

```
    IF (size[] == 0) THEN
        control = row0off0siz4;
    END IF;
WHEN row0off1 =>
    IF (size[] == 1) THEN
        control = row0off1siz1;
    END IF;
    IF (size[] == 2) THEN
        control = row0off1siz2;
    END IF;
    IF (size[] == 3) THEN
        control = row0off1siz3;
    END IF;
WHEN row0off2 =>
    IF (size[] == 1) THEN
        control = row0off2siz1;
    END IF;
    IF (size[] == 2) THEN
        control = row0off2siz2;
    END IF;
WHEN row0off3 =>
    IF (size[] == 1) THEN
        control = row0off3siz1;
    END IF;
WHEN row1off0 =>
    IF (size[] == 1) THEN
        control = row1off0siz1;
    END IF;
    IF (size[] == 2) THEN
        control = row1off0siz2;
    END IF;
    IF (size[] == 3) THEN
        control = row1off0siz3;
    END IF;
    IF (size[] == 0) THEN
        control = row1off0siz4;
    END IF;
WHEN row1off1 =>
    IF (size[] == 1) THEN
        control = row1off1siz1;
    END IF;
    IF (size[] == 2) THEN
        control = row1off1siz2;
    END IF;
    IF (size[] == 3) THEN
        control = row1off1siz3;
    END IF;
WHEN row1off2 =>
    IF (size[] == 1) THEN
        control = row1off2siz1;
    END IF;
    IF (size[] == 2) THEN
        control = row1off2siz2;
    END IF;
WHEN row1off3 =>
    IF (size[] == 1) THEN
        control = row1off3siz1;
    END IF;
WHEN refresh0 =>
    control = refresh1;
```

```
        WHEN OTHERS =>
            control = idle;
    END CASE;

    /dsack = !dsack;
    /ras[] = !ras[];
    /cas[] = !cas[];

END;
```

Conclusion

AHDL allows the designer to describe any MAX EPLD design with a variety of AHDL constructs and the MAX+PLUS TTL MacroFunction Library. The MAX+PLUS Text Editor provides error location and delay prediction to ensure error-free AHDL designs. MAX+PLUS provides a hierarchical design environment to integrate multiple text and graphic functions into a single design, so that each portion of a design can be represented in the most intuitive form. AHDL designs are compiled with the MAX+PLUS Compiler, which provides automatic logic minimization, logic synthesis, and device fitting to guarantee efficient use of MAX EPLD resources. In addition, MAX+PLUS provides a timing simulator and advanced timing analyzer for design verification. AHDL, together with Altera's MAX+PLUS Development System, provides unsurpassed entry, processing, and verification for text-based designs.

For a copy of any of the sample designs shown in this application note, contact Altera Applications at 1 (800) 800-EPLD.

Counter Design for EP-Series EPLDs

October 1990, ver. 3

Application Brief 8

Introduction

Counters—often the most useful building blocks in digital design—are easily constructed in EPLDs. This application brief discusses efficient counter design for EP-series devices, using both schematic capture and Boolean equation design entry methods. EP-series EPLDs can implement binary, decade, and Gray-code counters that include load, enable, clear, and cascade options.

Choosing a Flip-Flop

Toggle (T) flip-flops are used to build the simplest and most efficient counters. The advantages offered by T flip-flops become apparent when the two 8-bit binary counter designs shown in Figures 1 and 2 are compared. The Altera Design File (ADF) counter in Figure 1 uses D flip-flops, which

Figure 1. Counter with D Flip-Flops

Counters that use D flip-flops require an additional product term for each successive significant bit.

```
Altera Corporation
EP610
8-BIT BINARY COUNTER WITH D FLIP-FLOPS

PART:  EP610
INPUTS:  ENABLE, RESET, CLOCK
OUTPUTS:  Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7
NETWORK:
   ENABLE = INP(ENABLE)     CLR = INP(RESET)
   CK     = INP(CLOCK)
   Q0,Q0  = RORF(Q0d,CK,CLR,,)
   Q1,Q1  = RORF(Q1d,CK,CLR,,)
   Q2,Q2  = RORF(Q2d,CK,CLR,,)
   Q3,Q3  = RORF(Q3d,CK,CLR,,)
   Q4,Q4  = RORF(Q4d,CK,CLR,,)
   Q5,Q5  = RORF(Q5d,CK,CLR,,)
   Q6,Q6  = RORF(Q6d,CK,CLR,,)
   Q7,Q7  = RORF(Q7d,CK,CLR,,)
EQUATIONS:
   Q7d = /Q0 * Q7 + /Q1 * Q7 + /Q2 * Q7 + /Q3 * Q7
       + /Q4 * Q7 + /Q5 * Q7 + /Q6 * Q7 + /ENABLE * Q7
       + ENABLE * Q0 * Q1 * Q2 * Q3 * Q4 * Q5 * Q6 * /Q7;

   Q6d = /Q0 * Q6 + /Q1 * Q6 + /Q2 * Q6 + /Q3 * Q6
       + /Q4 * Q6 + /Q5 * Q6 + /ENABLE * Q6
       + ENABLE * Q0 * Q1 * Q2 * Q3 * Q4 * Q5 * /Q6;

   Q5d = /Q0 * Q5 + /Q1 * Q5 + /Q2 * Q5 + /Q3 * Q5
       + /Q4 * Q5 + /ENABLE * Q5
       + ENABLE * Q0 * Q1 * Q2 * Q3 * Q4 * /Q5;

   Q4d = /Q0 * Q4 + /Q1 * Q4 + /Q2 * Q4 + /Q3 * Q4
       + /ENABLE * Q4
       + ENABLE * Q0 * Q1 * Q2 * Q3* /Q4;

   Q3d = /Q0 * Q3 + /Q1 * Q3 + /Q2 * Q3 + /ENABLE * Q3
       + ENABLE * Q0 * Q1 * Q2 * /Q3;

   Q2d = /Q0 * Q2 + /Q1 * Q2 + /ENABLE * Q2
       + ENABLE * Q0 * Q1 * /Q2;

   Q1d = /Q0 * Q1 + /ENABLE * Q1 + ENABLE * Q0 * /Q1;

   Q0d = /ENABLE * Q0 + ENABLE * /Q0;
END$
```

Figure 2. Counter with T Flip-Flops

Counters that use T flip-flops require only one product term for each significant bit.

```
Altera Corporation
EP610
8-BIT BINARY COUNTER WITH T FLIP-FLOPS

PART:  EP610
INPUTS:  ENABLE, RESET, CLOCK
OUTPUTS:  Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7
NETWORK:
   ENABLE = INP(ENABLE)     CLR = INP(RESET)
   CK     = INP(CLOCK)
   Q0,Q0  = RORF(Q0t,CK,CLR,,)
   Q1,Q1  = RORF(Q1t,CK,CLR,,)
   Q2,Q2  = RORF(Q2t,CK,CLR,,)
   Q3,Q3  = RORF(Q3t,CK,CLR,,)
   Q4,Q4  = RORF(Q4t,CK,CLR,,)
   Q5,Q5  = RORF(Q5t,CK,CLR,,)
   Q6,Q6  = RORF(Q6t,CK,CLR,,)
   Q7,Q7  = RORF(Q7t,CK,CLR,,)
EQUATIONS:
   Q7t = ENABLE * Q0 * Q1 * Q2 * Q3 * Q4 * Q5 * Q6;

   Q6t = ENABLE * Q0 * Q1 * Q2 * Q3 * Q4 * Q5;

   Q5t = ENABLE * Q0 * Q1 * Q2 * Q3 * Q4;

   Q4t = ENABLE * Q0 * Q1 * Q2 * Q3;

   Q3t = ENABLE * Q0 * Q1 * Q2;

   Q2t = ENABLE * Q0 * Q1;

   Q1t = ENABLE * Q0;

   Q0t = ENABLE;

END$
```

10

require an additional product term for each successive significant bit (in the Equations Section). The most significant bit, **Q7**, requires 9 product terms. The counter in Figure 2 uses T flip-flops. Each counter bit requires only one product term. Thus, D flip-flops require significantly more gated logic to construct the equivalent counter function.

Altera EP-series EPLDs provide eight product terms per macrocell and offer programmable D, T, JK, and SR flip-flops. Using D flip-flops, only a seven-bit counter fits within each EPLD macrocell (the nth bit requires $n+1$ product terms). Counters built with T flip-flops can be any size, and still only consume one product term per bit. Therefore, T flip-flops are generally the best choice for counter design.

Designing Counters with LogiCaps & Macrofunctions

Altera's A+PLUS TTL MacroFunction Library provides 21 predefined MSI counters, shown in Table 1. These counters range in function from decade to binary, and include a variety of features such as load, clear, enable, and cascade options. Some entries are TTL part numbers followed by a "T" (e.g., **74163T**), indicating that T flip-flops were used to implement the counter. The library also includes some specially-designed A+PLUS counters, such as **8COUNT** (cascadable 8-bit up/down counter), **FREQDIV** (4-bit frequency divider), and **GRAY4** (4-bit Gray-code counter).

Table 1. Counter Macrofunctions

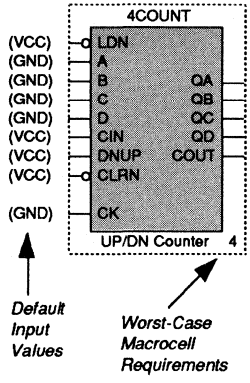
Name		Type			Options				
	Bits	Decade	Binary	Gray	Up	Down	Load	Clear	Enable
7493	4		✔		✔			✔	
74160	4	✔			✔		✔	✔	✔
74160T	4	✔			✔		✔	✔	✔
74161	4		✔		✔		✔	✔	✔
74161T	4		✔		✔		✔	✔	✔
74162	4	✔			✔		✔	✔	✔
74162T	4	✔			✔		✔	✔	✔
74163	4		✔		✔		✔	✔	✔
74163T	4		✔		✔		✔	✔	✔
74190	4	✔			✔	✔	✔		✔
74190T	4	✔			✔	✔	✔		✔
74191	4		✔		✔	✔	✔		✔
74191T	4		✔		✔	✔	✔	✔	✔
74192T	4	✔			✔	✔	✔	✔	✔
74193T	4		✔		✔	✔	✔	✔	✔
74393	4		✔		✔				
GRAY4	4			✔	✔				
4COUNT	4		✔		✔	✔	✔	✔	✔
8COUNT	8		✔		✔	✔	✔	✔	✔
FREQDIV	4		✔		✔				
16CUDSLR	16		✔		✔	✔			

Macrofunction documentation, shown in Figure 3, provides a symbol name, a function table, and an ADF declaration statement. If macrofunction inputs are left unconnected, a default value (shown in Figure 3) is used. For example, **VCC** is the default value for **LDN**. Worst-case macrocell requirements are shown in the lower right-hand corner of the symbol (e.g., four macrocells for **4COUNT**). The "Declaration" line shows the ADF syntax for the function. The function table shows the macrofunction operation under all input conditions. The logic schematic in Figure 4 shows the gate-level logic for **4COUNT**.

10

Figure 3. 4COUNT Macrofunction Documentation

4COUNT Symbol



Altera Design File Declaration and Default Input Values

Name: 4COUNT (4-Bit Up/Down Counter with Synchronous Load and Asynchronous Clear)

Declaration: 4COUNT(CLRN,LDN,DNUP,CIN,A, B,C,D,CK,QD,QC,QB,QA,COUT)

(LDN = Load, Active Low;
CIN = Carry In; DNUP = Down/Up;
CLRN = Clear, Active Low;
CK = Clock; COUT = Carry Out)

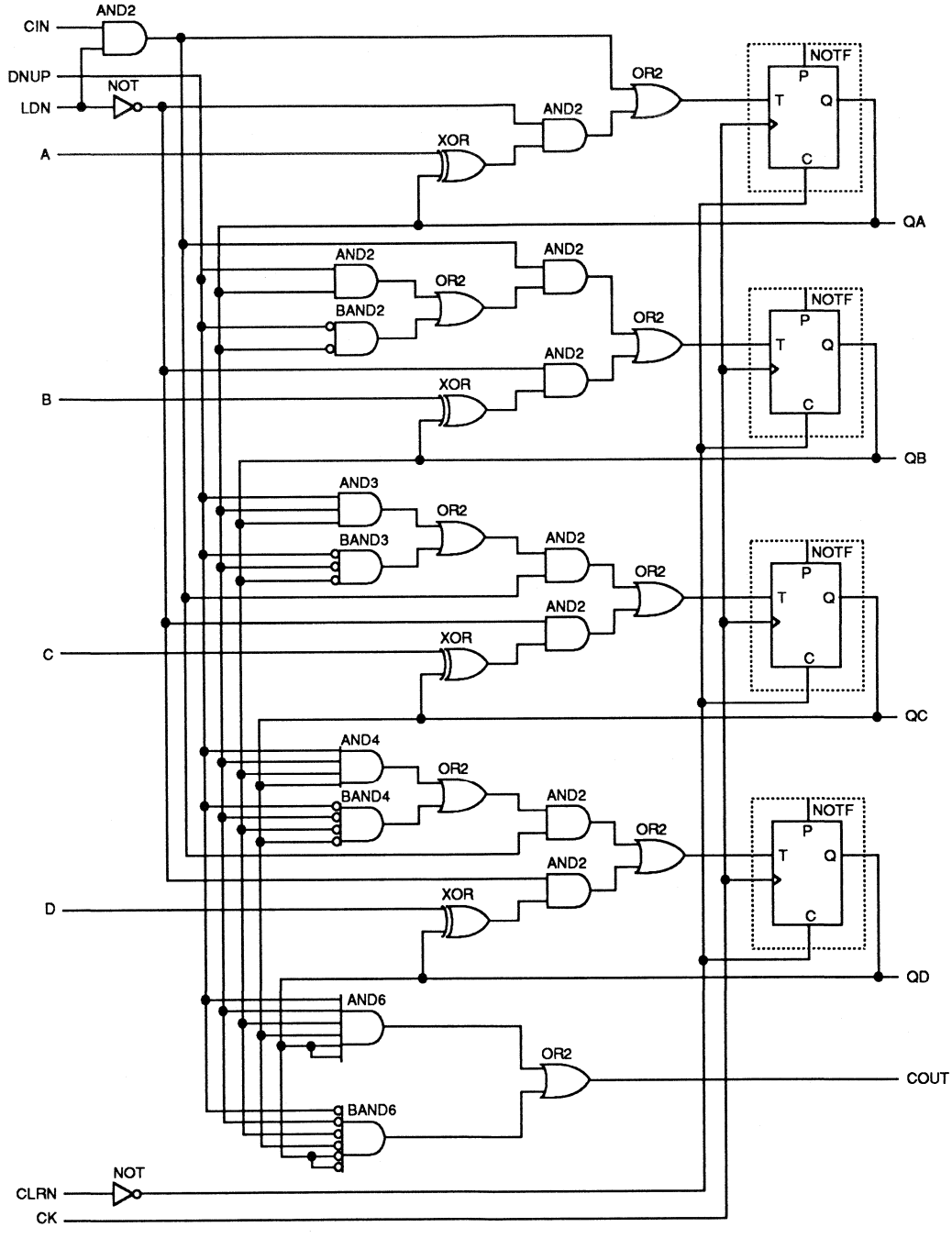
EPLDs: EP610, EP630, EP640, EP910, EP1810, EP1830

Default Signal Levels: GND — CK, A, B, C, D
VCC — CLRN, LDN, DNUP, CIN

4COUNT Function Table

Inputs									Outputs				
CK	LDN	CLRN	DNUP	CIN	D	C	B	A	QD	QC	QB	QA	COUT
X	X	L	X	X					L	L	L	L	X
↓	L	H	X	X	d	c	b	a	d	c	b	a	X
↓	H	H	X	L					Hold				X
↓	H	H	L	H					Count Down				L
↓	H	H	H	H					Count Up				L
↓	H	H	H	H					H	H	H	H	H
↓	H	H	L	H					L	L	L	L	H

Figure 4. 4COUNT Schematic File



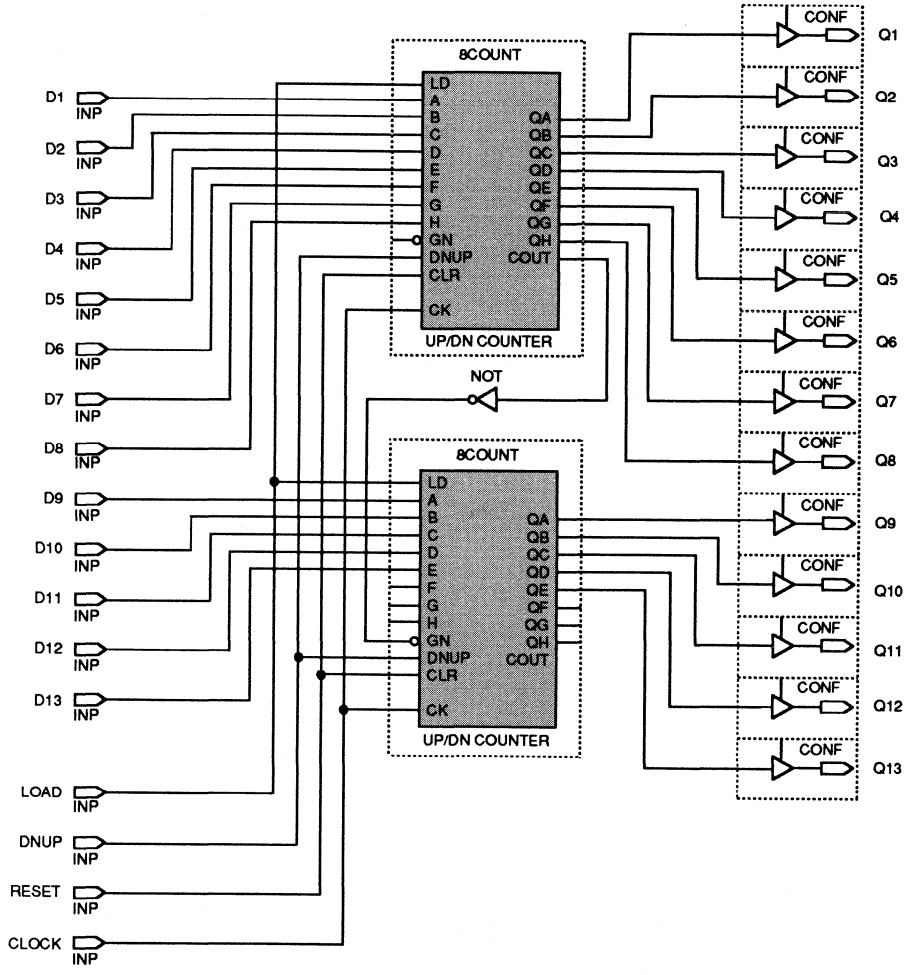
10

MacroMunching

MacroMunching, a logic synthesis feature of the A+PLUS software, automatically removes unused portions of a macrofunction. (See *PLS-SUPREME: Enhanced A+PLUS Programmable Logic Software Data Sheet* in this data book for a description of MacroMunching.) For example, if a design requires only a 3-bit counter, **4COUNT** may still be used. By leaving the extra counter bit **QD** unconnected, its flip-flop and related gated logic are removed automatically from the design and do not consume any of the EPLD's resources. A+PLUS macrofunctions and the MacroMunching capability make it easy to build counters of any size. Figure 5 shows a 13-bit counter that uses A+PLUS macrofunctions.

Figure 5. 13-Bit Up/Down Counter

This counter is constructed by cascading two 8-bit counters. MacroMunching automatically removes the unused bits.



Designing Counters with Boolean Equations

Figures 6, 7, and 8 show ADFs for a 4-bit binary up counter (**74161**), a 4-bit decade up/down counter (**74190T**), and an 8-bit binary up/down counter (**8COUNT**) with load, enable, and clear options. Other counters may be designed with the Boolean equations given in these ADFs.

Figure 6. 4-Bit Binary Counter with Load and Clear

```
Altera Corporation
10/1/90
1.00
B
EP610
4-BIT BINARY UP COUNTER WITH SYNCHRONOUS LOAD,
ASYNCHRONOUS CLEAR

PART:  EP610

INPUTS:  /CLEAR, /LOAD, ENP, ENT, A, B, C, D,
CLOCK

OUTPUTS:  QD, QC, QB, QA, RCO

NETWORK:
   CLRn = INP(/CLEAR)
   LDn  = INP(/LOAD)
   ENP  = INP(ENP)
   ENT  = INP(ENT)
   A    = INP(A)
   B    = INP(B)
   C    = INP(C)
   D    = INP(D)
   CK   = INP(CLOCK)
   QA,QA = RORF(QAd,CK,CLR,GND,VCC)
   QB,QB = RORF(QBd,CK,CLR,GND,VCC)
   QC,QC = RORF(QCd,CK,CLR,GND,VCC)
   QD,QD = RORF(QDd,CK,CLR,GND,VCC)
   RCO   = CONF(RCOc,VCC)
   CLR = NOT(CLRn)              % ACTIVE LOW CLEAR
%

EQUATIONS:
   RCOc = ENT * QD * QC * QB * QA;

   QDd = LDn' * D               % LOAD  %

       + ENT' * LDn * Q         % HOLD  %

       + ENP' * LDn * QD
       + LDn * QD * QA'         % COUNT %
       + LDn * QD * QB'
       + LDn * QD * QC'
       + ENT * ENP * LDn * QD' * QA * QB * QC;

   QCd = LDn' * C
       + ENT' * LDn * QC
       + ENP' * LDn * QC
       + LDn * QC * QA'
       + LDn * QC * QB'
       + ENT * ENP * LDn * QC' * QA * QB;

   QBd = LDn' * B
       + ENT' * LDn * QB
       + ENP' * LDn * QB
       + LDn * QB * QA'
       + ENT * ENP * LDn * QB' * QA;

   QAd = LDn' * A
       + ENT' * LDn * QA
       + ENP' * LDn * QA
       + ENT * ENP * LDn * QA';

END$
```

Figure 7. 4-Bit Up/Down Decade Counter with Load

```
Altera Corporation
10/1/90
1.00
A
EP610
4-BIT UP/DOWN DECADE COUNTER (74190T)


PART:  EP610

INPUTS:  /LOAD, DNUP, /ENABLE, CLOCK, A, B, C, D

OUTPUTS:  QA, QB, QC, QD, /RCO, MNMX

NETWORK:
   Gn   = INP(/ENABLE)       A = INP(A)
   LDn = INP(/LOAD)          B = INP(B)
   DNUP = INP(DNUP)          C = INP(C)
   CK = INP(CLOCK)           D = INP(D)
   QA,QA = TOTF(QAt, CK, GND, GND, VCC)
   QB,QB = TOTF(QBt, CK, GND, GND, VCC)
   QC,QC = TOTF(QCt, CK, GND, GND, VCC)
   QD,QD = TOTF(QDt, CK, GND, GND, VCC)
   /RCO  = CONF(RCOc, VCC)
   MNMX  = CONF(MNMXc,VCC)

EQUATIONS:
   TEN = (LDn * QA * QD * /DNUP
       +  LDn * /QA * /QD * DNUP * /QB * /QC)';

   MNMXc = QD * QA * /DNUP * /QC */QB * /Gn * LDn
         + /QD * /QA * DNUP * /QC * /QB * /Gn
           * LDn;

   RCOc = (QD * QA * /DNUP * /QC * /QB * /Gn * LDn
         + /QD * /QA * DNUP * /QC * /QB * /Gn
           * LDn)';

   QDt = /Gn * /QD * D * /LDn
       + /Gn * QD * /D * /LDn
       + /Gn * QD * LDn * QA * /DNUP
       + /Gn * LDn * /QA * DNUP * /QB * /QC
       + /Gn * LDn * QA * /DNUP * QB * QC;

   QBt = TEN * /Gn * /QB * B * /LDn
       + TEN * /Gn * QB * /B * /LDn
       + TEN * /Gn * LDn * /QA * DNUP
       + TEN * /Gn * LDn * QA * /DNUP;

   QCt = TEN * /Gn * /QC * C * /LDn
       + TEN * /Gn * QC * /C * /LDn
       + TEN * /Gn * LDn * /QA * /QB * DNUP
       + TEN * /Gn * LDn * QA * QB * /DNUP;

   QAt = /Gn * LDn
       + /Gn * QA * /A
       + /Gn * /QA * A;

END$
```

10

Figure 8. 8-Bit Binary Up/Down Counter with Load, Enable, and Clear

```
Altera Corporation
10/1/90
1.00
A
EP910
8 BIT BINARY UP/DOWN COUNTER
WITH LOAD, ENABLE, AND CLEAR

PART:  EP910
INPUTS:  LOAD, ENABLE, DNUP, CLEAR, CLOCK,
         A, B, C, D, E, F, G, H
OUTPUTS:  QA, QB, QC, QD, QE, QF, QG, QH, COUT
NETWORK:
    LD = INP(LOAD)        A = INP(A)
    CK = INP(CLOCK)       B = INP(B)
   CLR = INP(CLEAR)       C = INP(C)
  DNUP = INP(DNUP)        D = INP(D)
ENABLE = INP(ENABLE)      E = INP(E)
                          F = INP(F)
                          G = INP(G)
                          H = INP(H)
QA,QA = TOTF(QAt, CK, CLR, GND, VCC)
QB,QB = TOTF(QBt, CK, CLR, GND, VCC)
QC,QC = TOTF(QCt, CK, CLR, GND, VCC)
QD,QD = TOTF(QDt, CK, CLR, GND, VCC)
QE,QE = TOTF(QEt, CK, CLR, GND, VCC)
QF,QF = TOTF(QFt, CK, CLR, GND, VCC)
QG,QG = TOTF(QGt, CK, CLR, GND, VCC)
QH,QH = TOTF(QHt, CK, CLR, GND, VCC)
COUT  = CONF(COUTc,VCC)

EQUATIONS:
   COUTc = /QH * /DNUP * /QA * /QB * /QC *
           /QD * /QE * /QF * /QG * /LD * /ENABLE
         + QH * DNUP * QA * QB * QC *
           QD * QE * QF * QG * /LD * /ENABLE;

   QHt = LD * /QH * H                    % LOAD %
       + LD * QH * /H

       + /QA * /QB * /QC * /QD *   % COUNT DOWN %
         /QE * /QF * /QG * /DNUP * /LD * /ENABLE

       + QA * QB * QC * QD * QE *    % COUNT UP %
         QF * QG * DNUP * /LD * /ENABLE;

   QGt = LD * /QG * G
       + LD * QG * /G
       + /QA * /QB * /QC * /QD * /QE *
         /QF * /DNUP * /LD */ENABLE
       + QA * QB * QC * QD * QE *
         QF * DNUP * /LD * /ENABLE;

   QFt = LD * /QF * F
       + LD * QF * /F
       + /QA * /QB * /QC * /QD * /QE *
         /DNUP * /LD * /ENABLE
       + QA * QB * QC * QD * QE *
         DNUP * /LD * /ENABLE;

   QEt = LD * /QE * E
       + LD * QE * /E
       + /QA * /QB * /QC * /QD * /DNUP *
         /LD * /ENABLE
       + QA * QB * QC * QD * DNUP *
         /LD * /ENABLE;

   QDt = LD * /QD * D
       + LD * QD * /D
       + /QA * /QB * /QC * /DNUP * /LD * /ENABLE
       + QA * QB * QC * DNUP * /LD * /ENABLE;

   QCt = LD * /QC * C
       + LD * QC * /C
       + /QA * /QB * /DNUP * /LD * /ENABLE
       + QA * QB * DNUP * /LD * /ENABLE;

   QBt = /QA * /DNUP * /LD * /ENABLE
       + LD * /QB * B
       + LD * QB * /B
       + QA * DNUP * /LD * /ENABLE;

   QAt = /ENABLE * /LD
       + LD * /A * QA
       + LD * A * /QA;

END$
```

Designing Asynchronous Latches for EP-Series EPLDs

October 1990, ver. 3

Application Brief 9

Introduction

This application brief discusses how to implement asynchronous latches in EP-series EPLDs with both LogiCaps schematic capture and Boolean equation design entry. The following topics are covered:

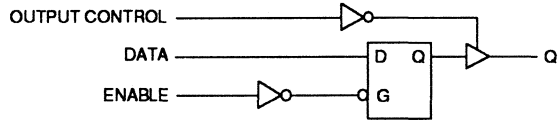
- Transparent D latch (equivalent to 74LS373)
- SR latch (equivalent to 74LS279)
- Programmable I/O polarity options

Transparent D Latch

Transparent, asynchronous D latches commonly implemented with 74LS373 TTL devices hold data at the latch input until a control signal is applied. Once enabled, the output follows the data input. Figure 1 shows the symbolic representation, function table, A+PLUS macrofunction symbol, and gate-level logic for a standard D latch.

Figure 1. Transparent D Latch

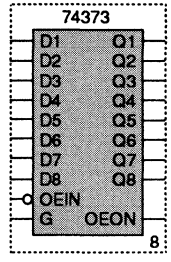
Symbolic Representation



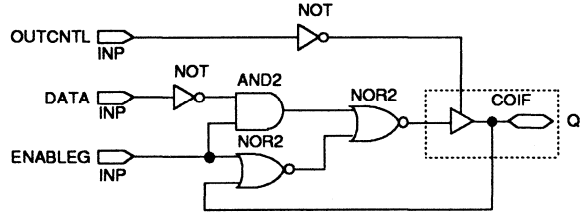
Function Table

OUTPUT CONTROL	ENABLE (G)	DATA	OUTPUT (Q)
L	H	H	H
L	H	L	L
L	L	X	Q
H	X	X	Z

Macrofunction Symbol



LogiCaps Schematic



Only one macrocell that uses combinatorial feedback is required to implement the D latch in an EP-series EPLD. (MAX EPLD registers can be configured as D latches.) This latch can be entered with the **74373** macrofunction or with gate-level logic in LogiCaps, as shown in Figure 1, or designed with Boolean logic in an Altera Design File (ADF). An ADF for the transparent D latch is given in Figure 2.

Figure 2. Transparent D Latch Implemented in an Altera Design File

```
Altera Corporation
10/1/90
1.0
EPLD
TRANSPARENT D-TYPE LATCH

OPTIONS:  TURBO=ON

PART:  AUTO                          % Automatic part selection %

INPUTS:  DATA,ENABLEG,OUTCNTL

OUTPUTS:  Q

NETWORK:

DATA    = INP(DATA)
ENABLEG = INP(ENABLEG)
OUTCNTL = INP(OUTCNTL)
Q,Q = COIF(Qc,OE)

EQUATIONS:

Qc = /((/DATA*ENABLEG) + /(ENABLEG + Q));
                                          % Active high output %

OE = /OUTCNTL;

END$
```

Asynchronous SR Latch

Asynchronous SR latches are constructed with either cross-coupled NOR or NAND gates. For the NOR-NOR implementation, the output goes to a logical one (high) if the Set (**S**) input is high. The output produces a logical zero (low) if the Reset (**R**) input is high. Figure 3 gives symbolic representations of an SR latch, the A+PLUS macrofunction symbol, function table, and gate-level LogiCaps schematics.

Only one macrocell that uses combinatorial feedback is required to implement this latch in an EP-series EPLD. The SR latch can be entered with the **74279** macrofunction or with gate-level logic in LogiCaps, as shown in Figure 3, or designed with Boolean logic in an ADF. See Figure 4 for an ADF that implements an SR latch.

Figure 3. SR Latch

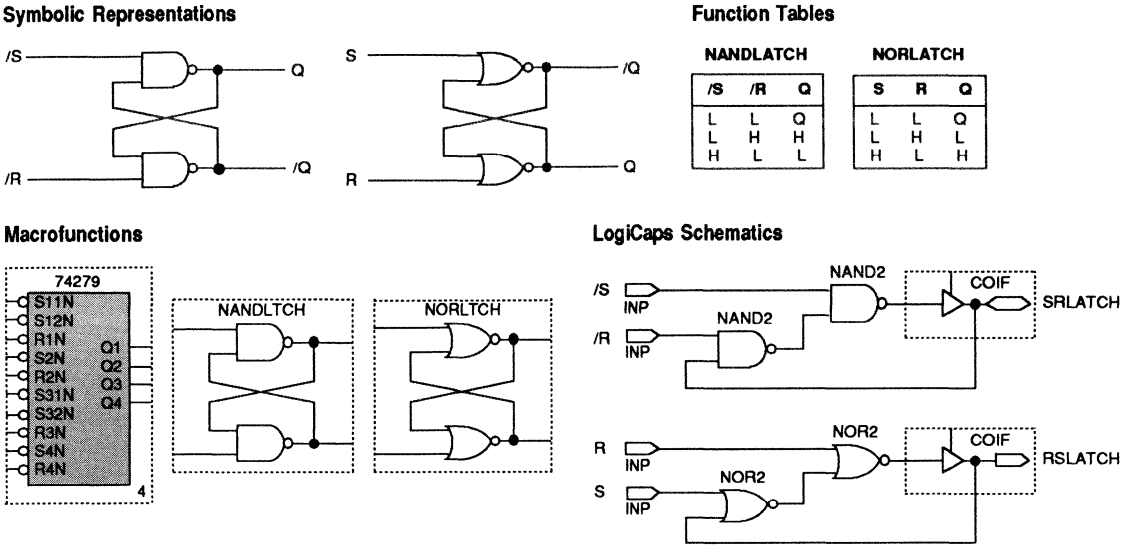


Figure 4. SR Latch Implemented in an Altera Design File

```
Altera Corporation
10/1/90
1.0
EPLD
SR LATCH

OPTIONS:  TURBO=ON

PART:  AUTO

INPUTS:  S,R

OUTPUTS:  RSLATCH

NETWORK:

S = INP(S)
R = INP(R)
RSLATCH,Q = COIF(RSLATCHc,VCC)

EQUATIONS:

RSLATCHc = /(/(S+Q)+R);              % NOR-NOR implementation %

END$
```

10

Programmable Polarity

Altera EP-series EPLDs offer superior flexibility to TTL devices for specifying latch operation. For the transparent latch, the output polarity is active high (**Q** = **DATA** when **ENABLEG** is high) when the **DATA** input pin drives the **NOT** gate, as shown in the LogiCaps schematic in Figure 1. For an active-low output (**Q** = **/DATA** when **ENABLEG** is high), it is necessary to remove the inverter and connect the **DATA** input pin directly to the **AND2** gate. In addition, the **OUTCNTL** and **ENABLEG** inputs can also be defined as active high or active low. For example, connecting an inverter after the **ENABLEG** input pin causes the circuit to pass **DATA** when **ENABLEG** is low and latch **DATA** when **ENABLEG** is high.

For Altera EPLDs whose I/O pins do not support direct combinatorial feedback (EP300-, EP600-, and EP900-series), input feedback can be substituted to route the signal back into the AND array. For example, the SR latch implemented with **NAND2** gates uses **COIF** I/O architecture (see the LogiCaps schematic in Figure 3). When using the **COIF** primitive, the output must always be enabled to ensure that the feedback path is connected to the logic.

A+PLUS State Machine Design Entry

October 1990, ver. 2

Application Brief 14A

Introduction

A+PLUS state machine design entry—supported by the PLCAD-SUPREME and PLS-SUPREME development products—provides a high-level approach for entering state machine designs. State Machine Files (SMFs) can be created with **IF-THEN** or **CASE** statements and optional truth tables. State machine entry can be used separately or together with LogiCaps schematic capture, TTL macrofunctions, and Boolean equations for added power.

The State Machine File (SMF) format is a superset of the Altera Design File (ADF) format. (It is also similar to the ASMILE state machine entry language available with the SAM+PLUS Development Software.) Users who are unfamiliar with the ADF format should first read *Application Brief 71 (A+PLUS Boolean Equation Design Entry)* in this data book.

SMF Syntax

State machine design files are created in the Altera SMF syntax with an ASCII text editor (in non-document mode). The SMF is divided into several sections, many of which are defined by a keyword. Each section is described here. A sample SMF is shown in Figure 1.

During design processing, the A+PLUS State Machine Converter (SMV) translates the SMF design file into a Boolean-equation-based ADF. The Altera Design Processor (ADP) then processes the ADF, choosing the best register (T or D) for each state variable. Finally, the ADP minimizes the resulting logic, fits the design into an appropriate EPLD, and creates a JEDEC file for device programming.

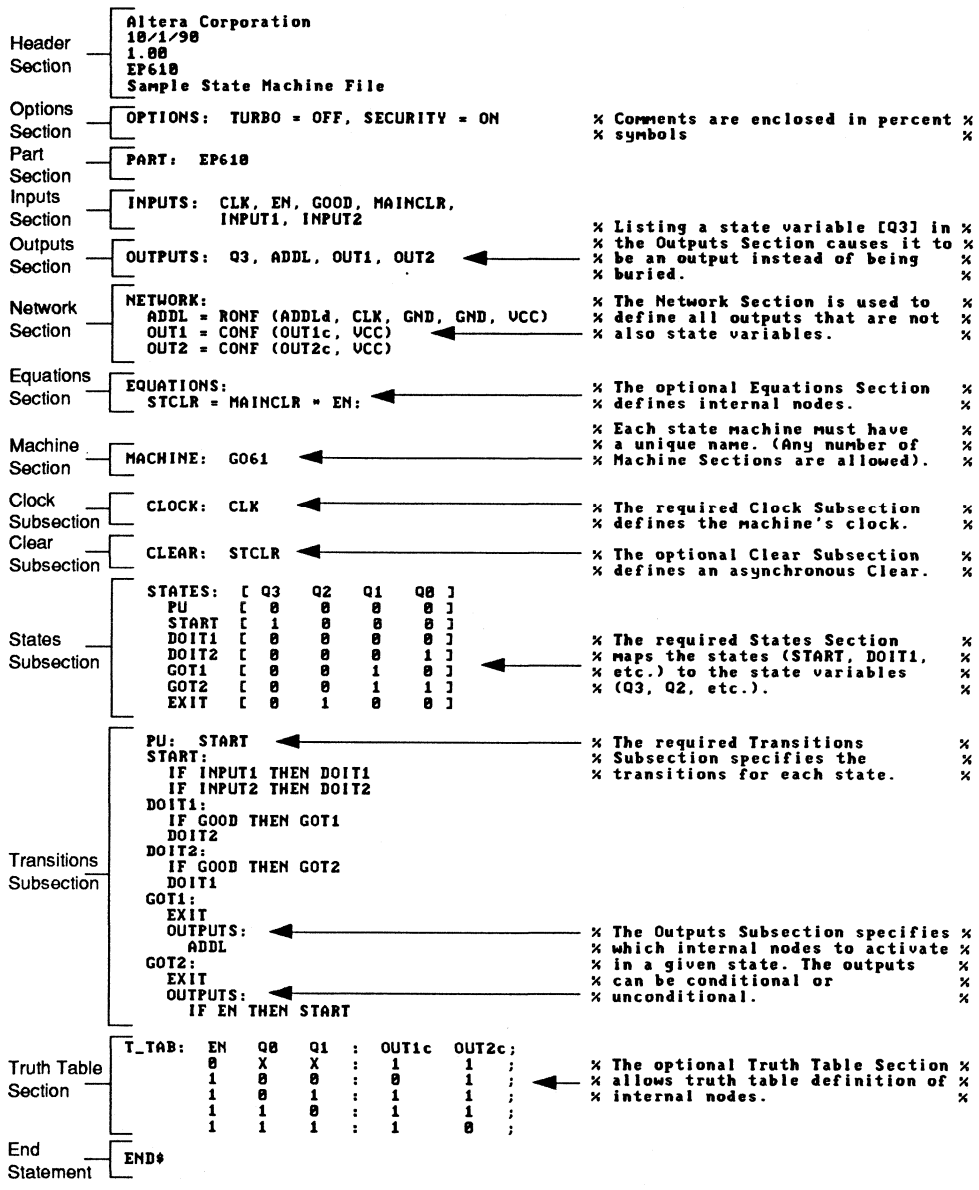
Header Section

The optional Header Section includes all "bookkeeping" information, such as design title, revision number, designer, and any other desired information. It has no effect on design processing. This section has no keyword.

Options Section

The Options Section (keyword **OPTIONS:**) controls the Turbo Bit and Security Bit.

Figure 1. Sample State Machine File (SMF)



Part Section

The Part Section (keyword **PART:**) specifies the target EPLD. If the **AUTO** option is used, A+PLUS automatically selects the best EPLD for the design. If pin assignments are specified in the SMF, the **AUTO** option is not

available, since the assigned pins must necessarily correspond to a particular EPLD and package.

Inputs Section

The Inputs Section (keyword **INPUTS:**) specifies all EPLD inputs for the design. Pin numbers can be assigned by appending an "at" symbol (**@**) plus the pin number to the end of the pin name.

Outputs Section

The Outputs Section (keyword **OUTPUTS:**) declares all outputs used in the design. Pin assignments are made in the same way as input pins.

Network Section

The optional Network Section (keyword **NETWORK:**) is used only to define state machine outputs that are not also state variables. (SMV automatically creates the network statements required for each input and each state variable during file conversion.)

Equations Section

The optional Equations Section (keyword **EQUATIONS:**) uses the same standard Boolean operators available for Boolean equation design entry (AND = * or &; OR = + or #; NOT = /, ', or !). Equations can be used to define outputs that are not state variables. Intermediate equations are also supported and may be used to define a transition condition or a clock for the state machine.

Machine Section

The Machine Section (keyword **MACHINE:**) specifies the state machine name. Several state machines can be included in the same file.

Clock Subsection

The Clock Subsection (keyword **CLOCK:**) defines the clock for the state machine. The clock may come directly from an input pin (listed in the Inputs Section), or it can be defined as an internal node within the design. Internal nodes are defined by the Equations Section, the Truth Table Section, or other state machines.

Clear Subsection

The optional Clear Subsection (keyword **CLEAR:**) defines a master asynchronous clear for the state machine. The Clear signal may come directly from an input pin or from any internal node within the design.

States Subsection

The States Subsection (keyword **STATES:**) defines all machine states. Each state name is listed in a column. The state variables of the machine are enclosed in square brackets following the state names. Each state must be assigned a unique combination of state variables.

Transitions Subsection

The Transitions Subsection (no keyword) defines the transition for each state. Each state name, followed by a colon (:), is listed in this section. The state name is followed by a conditional **IF-THEN** or **CASE** statement, or an unconditional transition. To ensure that no ambiguity exists if more than one condition is true, the first **IF-THEN** or **CASE** statement has precedence over all others. For unconditional transitions, only the name of the next state is listed. If none of the conditions is true, and no unconditional transition is specified, the machine state remains unchanged.

For example, when the machine shown in Figure 1 is in state **START**, and **INPUT1** and **INPUT2** are both **0**, the next state is **START**.

Outputs Subsection

The optional Outputs Subsection of the Transitions Subsection (keyword **OUTPUTS:**) follows the transition(s) for a state. When this section is used, all node names that follow the **OUTPUTS:** keyword are set high during the current state. An **IF-THEN** statement may be included to make the node a function of both the current state and current input conditions.

Truth Table Section

The optional Truth Table Section (keyword **T_TAB:**) provides another method for defining internal nodes. Both sides of the table consist of internal nodes. See Figure 1.

End Statement

All State Machine Files must end with the **END$** statement.

Defining Outputs from the Machine

Outputs from the state machine can consist of any internal nodes within the design. Internal nodes are inputs, state variables, or nodes defined in the Equations Section, Outputs Subsection, and Truth Table Section. By listing a state variable name in the Outputs Section (i.e., the Outputs Section that follows the Inputs Section, *not* the Outputs Subsection of the Transitions Subsection), the state variable becomes an output. All other internal nodes must be routed through the Network Section, where the type of output architecture must be specified (i.e., registered or combinatorial).

State Machine Partitioning
for EP-Series EPLDs

October 1990, ver. 3 **Application Brief 18**

Introduction

State machine partitioning offers the following advantages:

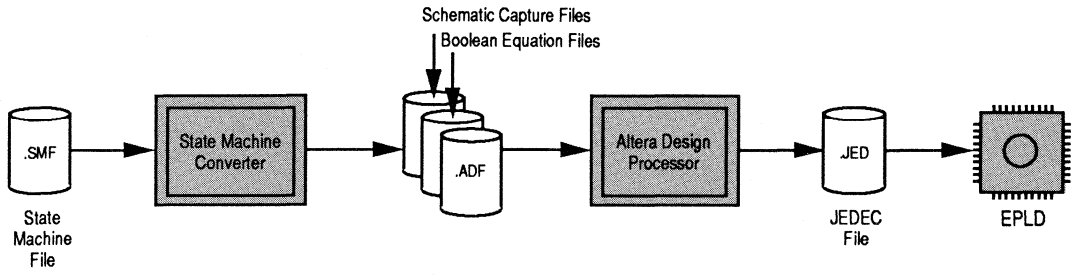
- ❒ Complex state machines are reduced to two or more simple state machines.
- ❒ Complex designs can be fit into EPLDs.
- ❒ The designer is able to use a systematic approach when working with state machine designs.

This application brief describes a systematic method for partitioning large, complex state machines into two or more smaller and simpler linked state machines. Partitioning is useful for implementing a state machine in an Erasable Programmable Logic Device (EPLD) when the size of logic equations (number of product terms) for the next-state decoder exceeds the EPLD macrocell resources.

When a state machine is partitioned, the resulting design often requires a larger number of state registers, which may reduce efficiency when the design is implemented in silicon. However, although the number of state registers increases, the equations for each register become smaller. Consequently, the partitioned design may be placed into fixed-width-array logic devices such as EPLDs.

Partitioning does not always reduce the size of the equations, especially for smaller, "tightly connected" machines. The results depend on how the machine is divided, the complexity of dependent transition equations, and the state assignments. Sometimes the designer must try different divisions and state assignments to find the best implementation. The best solution, however, is to use the A+PLUS State Machine Converter, with its high-level language syntax for easy design input and modification. See Figure 1.

Figure 1. A+PLUS State Machine Converter

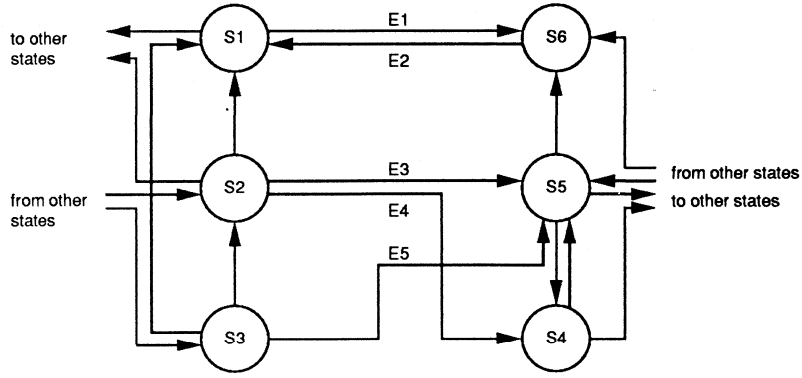


10

Selecting the Partition

Figure 2 shows part of a state diagram that describes a complex state machine with states **S1** to **S6**. Each state represents a unique combination of the state variables **U1** to **UN**. For example, **S3** could represent the combination **U1*/U2*/U3*U4**.

Figure 2. State Machine before Partitioning



The states are connected by directional arrows, marked with the symbolic labels **E1** to **E5**, that show possible transitions to and from states. These labels represent arbitrarily complex or simple equations, which are normally functions of state machine inputs that allow the state machine to respond to external events.

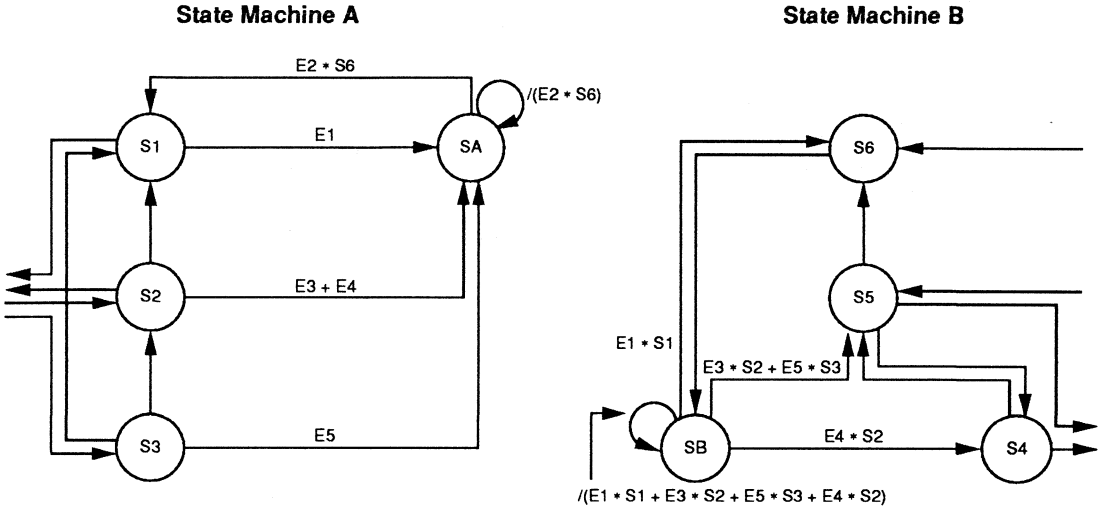
Partitioning the State Machine

The state machine can be split so that states **S1, S2, S3** (and all states on the left that are not shown in the diagram) become a separate machine. **E1** to **E5** represent all transitions between the two halves of the diagram.

A line should be drawn so that it crosses all transitions between the two halves of the machine. For each half of the machine, the other half should be replaced with a new state, called an **IDLE** state. This state allows one half of the state machine to be idle while the other half is busy. When a transition between the two halves occurs, the half of the machine that is currently active goes into the idle state, while the currently idle half becomes active.

Figure 3 shows the sample machine after it has been partitioned. States **SA** and **SB** are the idle states for each half. All transitions between the two halves are routed to the idle states. Transitions leaving the idle states depend on the original transition equation and the current state of the opposite half. Equations for the idle states are simply the logical inverse of the transitions leaving each idle state. Rules for partitioning a state machine are shown in Figure 4.

Figure 3. State Machine after Partitioning



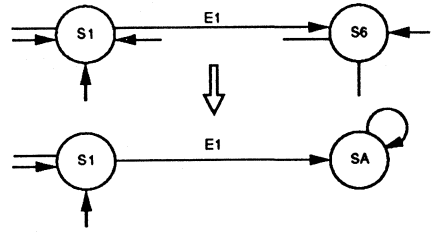
Sample Design

Figure 5 shows a state diagram for part of a design that converts a manchester-encoded serial data stream into a standard UART-compatible serial data stream, and extracts information from the data. The state diagram clearly shows the method and usefulness of state machine partitioning.

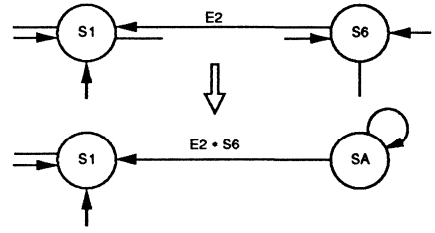
10

Figure 4. Rules for State Machine Partitioning

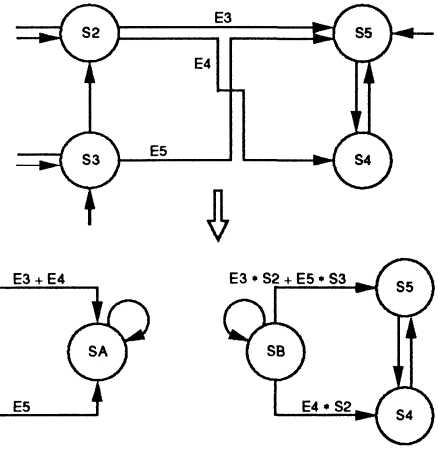
1. *A transition leading into an idle state has the same equation as the corresponding transition from the original machine.*



2. *A transition leading out of an idle state has the same equation as the corresponding transition from the original machine, logically ANDed with the state of the other half of the machine.*



3. *Multiple transitions with the same source and destination states can be replaced by a single transition with an equation that is the logical OR of the two original equations. This equation corresponds to two IF-THEN statements that have the same destination in the State Machine File (SMF), making SMF modification easier when a design is partitioned.*



4. *Idle states have a loopback or hold transition, with an equation that is the logical inverse of all equations for transitions leaving that state, logically ORed together. The hold transition automatically occurs when an ELSE transition for a given state does not exist.*

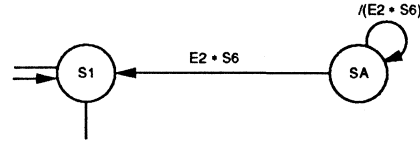
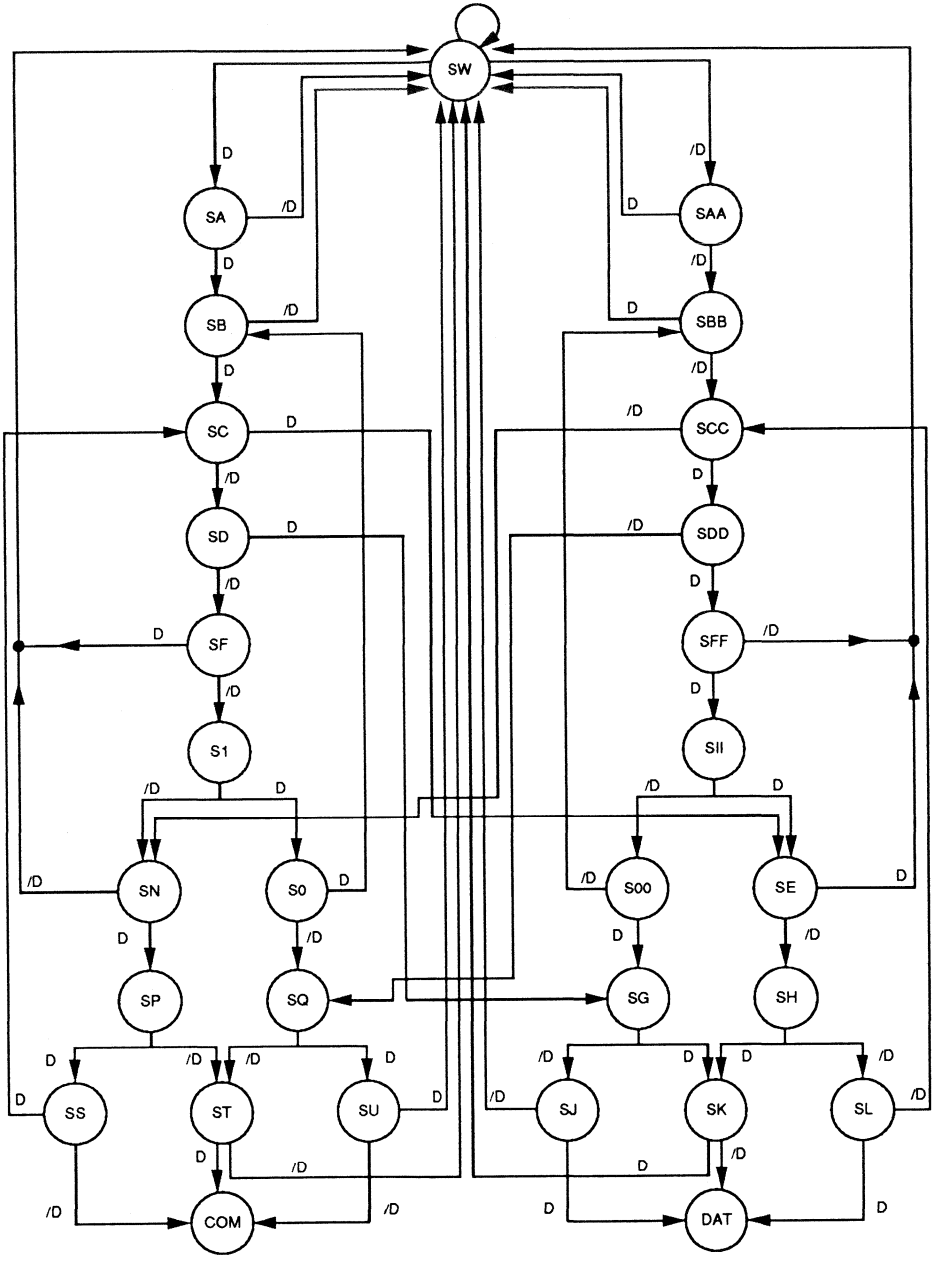


Figure 5. Manchester Synchronization Detection State Diagram



10

The Manchester synchronization detection design was minimized to 5 equations that have from 8 to 14 product terms. Figures 6 and 7 show the non-partitioned design before and after design processing. In its current form, this design will not fit into an EP-series EPLD.

Figure 6. Manchester Synchronization Detection State Machine File before Processing (Part 1 of 2)

```
PART:     EP910J
INPUTS:   D, CLK
OUTPUTS:  % no outputs %
MACHINE:  hd15530_state_machine

    CLOCK: clk
                % state assignments %
     STATES:   [ SV4  SV3  SV2  SV1  SV0   ]
        SW     [  0    0    0    0    0    ]
        SA     [  0    1    0    0    0    ]
        SB     [  0    1    1    0    0    ]
        SC     [  1    1    1    0    0    ]
        SD     [  1    0    1    0    0    ]
        SF     [  0    0    1    0    0    ]
        SI     [  0    0    1    1    0    ]
        SN     [  1    0    1    1    0    ]
        SO     [  0    1    1    1    0    ]
        SP     [  1    0    0    1    0    ]
        SQ     [  1    1    1    1    0    ]
        SS     [  0    0    0    1    0    ]
        ST     [  1    1    0    1    0    ]
        SU     [  0    1    0    1    0    ]
        COM    [  0    0    0    1    1    ]
        SAA    [  1    0    0    0    0    ]
        SBB    [  1    0    0    0    1    ]
        SCC    [  0    0    0    0    1    ]
        SDD    [  0    1    0    0    1    ]
        SFF    [  1    1    0    0    1    ]
        SII    [  1    1    1    0    1    ]
        SOO    [  1    0    1    0    1    ]
        SE     [  0    1    1    0    1    ]
        SG     [  0    0    1    0    1    ]
        SH     [  0    1    1    1    1    ]
        SJ     [  1    0    1    1    1    ]
        SK     [  0    0    1    1    1    ]
        SL     [  1    1    1    1    1    ]
        DAT    [  1    0    0    1    1    ]

                % transitions %
        SW:   IF  D   THEN    SA   % ELSE %    SAA
        SCC:  IF  D   THEN    SDD  % ELSE %    SN
        SS:   IF  D   THEN    SC   % ELSE %    COM
        COM:  IF  D   THEN    COM  % ELSE %    COM
        SF:   IF  D   THEN    SW   % ELSE %    SI
        SG:   IF  D   THEN    SK   % ELSE %    SJ
        SI:   IF  D   THEN    SO   % ELSE %    SN
        SK:   IF  D   THEN    SW   % ELSE %    DAT
        SA:   IF  D   THEN    SB   % ELSE %    SW
        SDD:  IF  D   THEN    SFF  % ELSE %    SQ
        SU:   IF  D   THEN    SW   % ELSE %    COM
        SB:   IF  D   THEN    SC   % ELSE %    SW
```

Figure 6. Manchester Synchronization Detection State Machine File before Processing (Part 2 of 2)

```
        SE:  IF  D   THEN    SW  % ELSE %    SH
        SO:  IF  D   THEN    SB  % ELSE %    SQ
        SH:  IF  D   THEN    SK  % ELSE %    SL
        SAA: IF  D   THEN    SW  % ELSE %    SBB
        SBB: IF  D   THEN    SW  % ELSE %    SCC
        SP:  IF  D   THEN    SS  % ELSE %    ST
        DAT: IF  D   THEN    DAT % ELSE %    DAT
        SD:  IF  D   THEN    SG  % ELSE %    SF
        SOO: IF  D   THEN    SG  % ELSE %    SBB
        SN:  IF  D   THEN    SP  % ELSE %    SW
        SJ:  IF  D   THEN    DAT % ELSE %    SW
        SFF: IF  D   THEN    SII % ELSE %    SW
        ST:  IF  D   THEN    COM % ELSE %    SW
        SC:  IF  D   THEN    SE  % ELSE %    SD
        SII: IF  D   THEN    SE  % ELSE %    SOO
        SQ:  IF  D   THEN    SU  % ELSE %    ST
        SL:  IF  D   THEN    DAT % ELSE %    SCC
END$
```

Figure 7. Minimized Logic Equation File (Part 1 of 2)

```
OPTIONS: TURBO = ON, SECURITY = OFF
PART:    EP910J
INPUTS:  D, CLK
OUTPUTS: SV4, SV3, SV2, SV1, SV0

NETWORK:
    clk = INP(clk)
    D = INP(D)
    SV4 = NORF(SV4.d, clk, GND, GND)
    SV3 = NOTF(SV3.t, clk, GND, GND)
    SV2 = NOTF(SV2.t, clk, GND, GND)
    SV1 = NOTF(SV1.t, clk, GND, GND)
    SV0 = NOTF(SV0.t, clk, GND, GND)

EQUATIONS:
    SV0.t = SV4' * SV2' * SV1 * SV0' * D'
          + SV4' * SV2' * SV1' * SV0 * D'
          + SV3 * SV2' * SV1' * SV0 * D'
          + SV4 * SV2 * SV1' * SV0' * D
          + SV4 * SV3' * SV2' * SV1' * SV0' * D'
          + SV4 * SV3' * SV2' * SV1' * SV0 * D
          + SV4' * SV3' * SV2 * SV1 * SV0 * D
          + SV4' * SV3 * SV2 * SV1' * SV0 * D
          + SV4 * SV3' * SV2 * SV1 * SV0 * D'
          + SV4 * SV3 * SV2' * SV1 * SV0' * D;

    SV1.t = SV4' * SV1' * SV0 * D'
          + SV4' * SV3' * SV2 * SV1' * D'
          + SV4' * SV2' * SV1 * SV0' * D
          + SV4 * SV3' * SV2 * SV1 * D'
          + SV4' * SV3' * SV2 * SV0 * D
          + SV4' * SV3 * SV1 * SV0' * D
          + SV4 * SV2 * SV1 * SV0 * D'
          + SV4 * SV3 * SV2' * SV1 * SV0' * D';
```

10

Figure 7. Minimized Logic Equation File (Part 2 of 2)

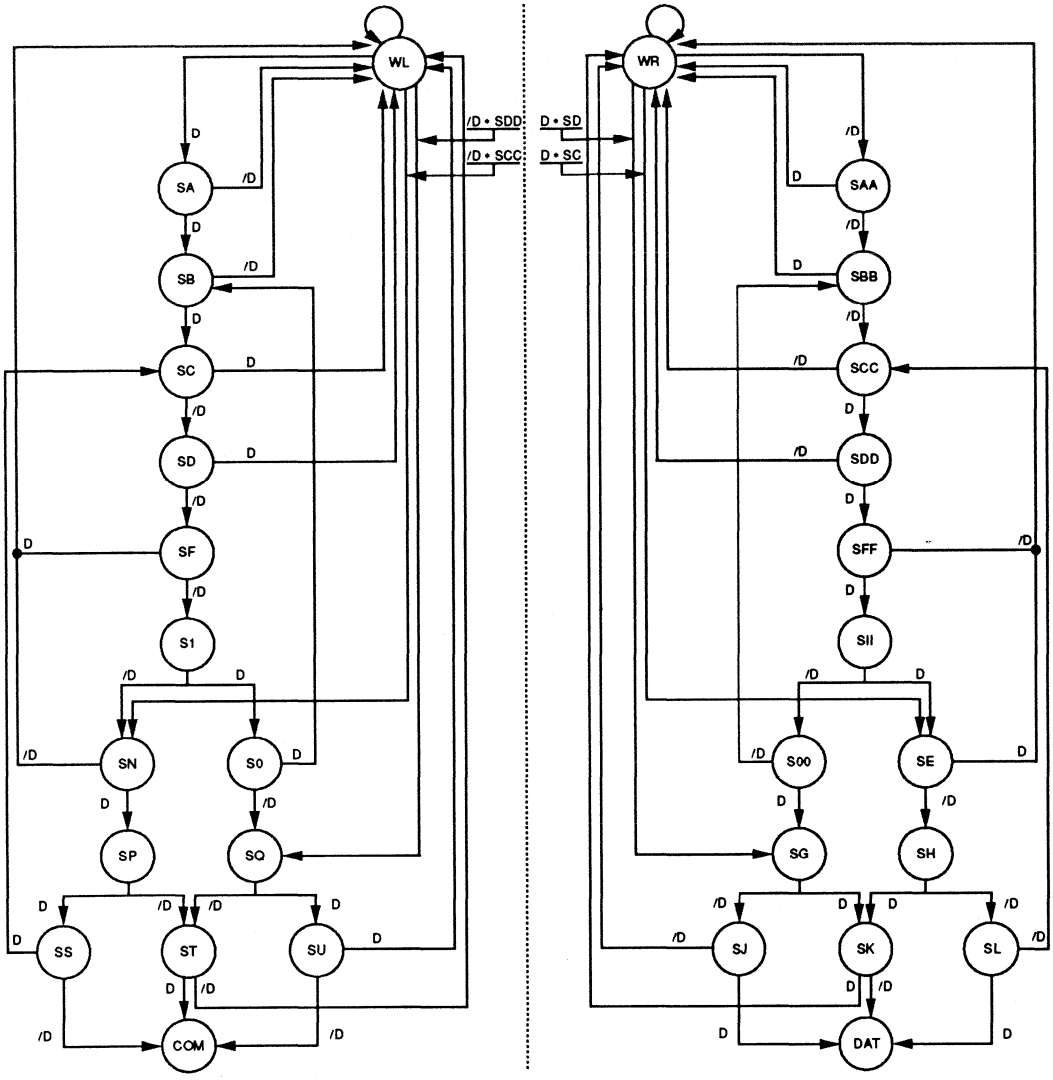
```
SV2.t = SV4 * SV2 * SV1
      + SV3' * SV2 * SV1 * SV0
      + SV4' * SV2' * SV1' * SV0 * D'
      + SV4 * SV3' * SV2 * SV0 * D'
      + SV4' * SV3 * SV2' * SV1' * SV0' * D
      + SV4' * SV3' * SV2 * SV1' * SV0' * D
      + SV4' * SV3 * SV2 * SV1' * SV0' * D'
      + SV4' * SV3' * SV2' * SV1 * SV0' * D
      + SV4 * SV3 * SV2' * SV1' * SV0 * D
      + SV4' * SV3 * SV2 * SV1' * SV0 * D;

SV3.t = SV3 * SV2' * SV1 * SV0'
      + SV4' * SV3 * SV2' * SV0' * D'
      + SV4' * SV3' * SV2' * SV1' * D
      + SV4' * SV3' * SV1 * SV0' * D
      + SV3 * SV2 * SV1' * SV0' * D'
      + SV4 * SV2' * SV1 * SV0' * D'
      + SV4 * SV3 * SV1' * SV0 * D'
      + SV4' * SV3 * SV2 * SV0 * D
      + SV4 * SV3 * SV2 * SV1 * SV0;

SV4.d = SV4' * SV2 * SV1 * D'
      + SV3' * SV2' * SV1' * SV0' * D'
      + SV4 * SV3' * SV2' * SV0' * D'
      + SV4' * SV2' * SV1' * SV0 * D'
      + SV3' * SV2 * SV1' * SV0 * D'
      + SV4 * SV3 * SV2 * SV0' * D'
      + SV3 * SV2' * SV1' * SV0 * D
      + SV4 * SV3 * SV2 * SV1' * D'
      + SV4 * SV3' * SV2' * SV1 * SV0
      + SV4 * SV3' * SV2 * SV1 * D
      + SV4 * SV2 * SV1 * SV0 * D
      + SV4' * SV3' * SV2' * SV1 * SV0' * D
      + SV4' * SV3 * SV2 * SV1' * SV0' * D;

END$
```

The state diagram in Figure 8 and the SMF in Figure 9 show one way to partition the design. Figure 10 shows minimized Boolean equations in the Logic Equation File for the partitioned design. Since the original state **SW** is a hold state (i.e., the conditions for holding are ORed together), it can be partitioned into the idle states **WL** and **WR** to create two separate state machines.

Conclusion

State machine designs often have a small number of large equations. Although partitioning a design creates more equations, the equations have fewer product terms. Since the number of product terms that can be implemented in logic equations is limited, partitioning facilitates product-term implementation. Although partitioning is useful for many state machines, not all state machines benefit from the technique. Some state machines may require several attempts before they are partitioned successfully.

Figure 8. Partitioned Manchester Synchronization Detection Design State Diagram



Partitioning Summary

	Original Machine	Partitioned Machine
Number of states	29	15 + 15 = 30
State variables	5	4 + 4 = 8
Number of product terms	56	50
Maximum product terms per state variable	14	7

10

Figure 9. State Machine File for the Partitioned Design (Part 1 of 2)

```
PART:     EP910J
INPUTS:   D, CLK
OUTPUTS:  % no outputs %
MACHINE:  hd15530_left_machine

    CLOCK: clk
                % state assignments %
     STATES:   [ SVL3 SVL2 SVL1 SVL0  ]
        SWL    [  0    0    0    0    ]
        SA     [  0    1    0    0    ]
        SB     [  1    1    1    1    ]
        SC     [  1    0    1    1    ]
        SD     [  1    0    0    1    ]
        SF     [  1    0    0    0    ]
        SI     [  0    0    0    1    ]
        SN     [  0    1    0    1    ]
        SO     [  1    1    0    1    ]
        SP     [  0    0    1    1    ]
        SQ     [  0    1    1    1    ]
        SS     [  1    1    1    0    ]
        ST     [  0    1    1    0    ]
        SU     [  0    0    1    0    ]
        COM    [  1    0    1    0    ]

                % transitions %
        SWL:    IF  D * SWR     THEN     SA
                IF /D * SCC     THEN     SN
                IF /D * SDD     THEN     SQ
           % ELSE %                      SWL
        SS:     IF D    THEN     SC   % ELSE %     COM
        COM:    IF D    THEN     COM  % ELSE %     COM
        SF:     IF D    THEN     SWL  % ELSE %     SI
        SI:     IF D    THEN     SO   % ELSE %     SN
        SA:     IF D    THEN     SB   % ELSE %     SWL
        SU:     IF D    THEN     SWL  % ELSE %     COM
        SB:     IF D    THEN     SC   % ELSE %     SWL
        SO:     IF D    THEN     SB   % ELSE %     SQ
        SP:     IF D    THEN     SS   % ELSE %     ST
        SD:     IF D    THEN     SWL  % ELSE %     SF
        SN:     IF D    THEN     SP   % ELSE %     SWL
        ST:     IF D    THEN     COM  % ELSE %     SWL
        SC:     IF D    THEN     SWL  % ELSE %     SD
        SQ:     IF D    THEN     SU   % ELSE %     ST

MACHINE:
      hd15530_right_machine

    CLOCK: clk
                % state assignments %
     STATES:   [ SVR3 SVR2 SVR1 SVR0  ]
        SWR    [  0    0    0    0    ]
        SAA    [  0    1    0    0    ]
        SBB    [  1    1    1    1    ]
        SCC    [  1    0    1    1    ]
        SDD    [  1    0    0    1    ]
        SFF    [  1    0    0    0    ]
        SII    [  0    0    0    1    ]
        SE     [  0    1    0    1    ]
```

Figure 9. State Machine File for the Partitioned Design (Part 2 of 2)

```
        S00    [  1     1     0     1     ]
        SH     [  0     0     1     1     ]
        SG     [  0     1     1     1     ]
        SL     [  1     1     1     0     ]
        SK     [  0     1     1     0     ]
        SJ     [  0     0     1     0     ]
        DAT    [  1     0     1     0     ]

               % transitions %
        SWR:   IF  D * SWL     THEN     SAA
               IF /D * SC      THEN     SE
               IF /D * SD      THEN     SG
           % ELSE %                     SWR
        SCC:   IF D    THEN     SDD % ELSE %     SWR
        SG:    IF D    THEN     SK  % ELSE %     SJ
        SK:    IF D    THEN     SWR % ELSE %     DAT
        SDD:   IF D    THEN     SFF % ELSE %     SWR
        SE:    IF D    THEN     SWR % ELSE %     SH
        SH:    IF D    THEN     SK  % ELSE %     SL
        SAA:   IF D    THEN     SWR % ELSE %     SBB
        SBB:   IF D    THEN     SWR % ELSE %     SCC
        DAT:   IF D    THEN     DAT % ELSE %     DAT
        S00:   IF D    THEN     SG  % ELSE %     SBB
        SJ:    IF D    THEN     DAT % ELSE %     SWR
        SFF:   IF D    THEN     SII % ELSE %     SWR
        SII:   IF D    THEN     SE  % ELSE %     S00
        SL:    IF D    THEN     DAT % ELSE %     SCC

END$
```

Figure 10. Minimized Logic Equation File from the Partitioned Design (Part 1 of 2)

```
OPTIONS: TURBO = ON, SECURITY = OFF
PART:    EP910J
INPUTS:  D, CLK
OUTPUTS: SVL3, SVL2, SVL1, SVL0, SVR3, SVR2, SVR1, SVR0

NETWORK:
    clk = INP(clk)
    D = INP(D)
    SVL3 = NOTF(SVL3.t, clk, GND, GND)
    SVL2 = NORF(SVL2.d, clk, GND, GND)
    SVL1 = NOTF(SVL1.t, clk, GND, GND)
    SVL0 = NORF(SVL0.d, clk, GND, GND)
    SVR3 = NOTF(SVR3.t, clk, GND, GND)
    SVR2 = NORF(SVR2.d, clk, GND, GND)
    SVR1 = NOTF(SVR1.t, clk, GND, GND)
    SVR0 = NORF(SVR0.d, clk, GND, GND)

EQUATIONS:
    SVR0.d = SVR3' * SVR2 * SVR1' * D'
           + SVR3' * SVR2' * SVR1' * SVR0
           + SVR3 * SVR2 * SVR1 * D'
           + SVR3 * SVR2 * SVR1' * SVR0
           + SVR3 * SVR2' * SVR1' * SVR0' * D
           + SVR3 * SVR2' * SVR1 * SVR0 * D
           + SVR3' * SVR1' * D' * SVL3 * SVL2' * SVL0;
```

10

Figure 10. Minimized Logic Equation File from the Partitioned Design (Part 2 of 2)

```
    SVR1.t = SVR3' * SVR2 * SVR1' * D'
           + SVR2 * SVR1' * SVR0 * D'
           + SVR3 * SVR2' * SVR1 * SVR0
           + SVR3 * SVR2 * SVR0 * D
           + SVR3' * SVR2' * SVR1 * SVR0' * D'
           + SVR3' * SVR2 * SVR1 * SVR0' * D
           + SVR3' * SVR2' * SVR0' * D' * SVL3 * SVL2' * SVL1' * SVL0;

    SVR2.d = SVR3' * SVR2' * SVR0
           + SVR3 * SVR2 * SVR1' * SVR0
           + SVR3' * SVR1 * SVR0 * D
           + SVR3' * SVR2 * SVR1' * SVR0' * D'
           + SVR3' * SVR2' * SVR1' * D' * SVL3 * SVL2' * SVL0
           + SVR3' * SVR2' * SVR1' * D * SVL3' * SVL2' * SVL1' * SVL0';

    SVR3.t = SVR2' * SVR0 * D'
           + SVR3' * SVR2 * SVR0' * D'
           + SVR3 * SVR2' * SVR1' * SVR0'
           + SVR3 * SVR2 * SVR0 * D
           + SVR3' * SVR2' * SVR1 * SVR0' * D;

    SVL0.d = SVL3' * SVL2' * SVL1' * SVL0
           + SVL3' * SVL2 * SVL1' * D
           + SVL3 * SVL2 * SVL1' * SVL0
           + SVL3 * SVL2 * SVL1 * D
           + SVL3 * SVL2' * SVL1' * SVL0' * D'
           + SVL3 * SVL2' * SVL1 * SVL0 * D'
           + SVL2' * SVL1' * SVL0' * D' * SVR3 * SVR2' * SVR0;

    SVL1.t = SVL3' * SVL2 * SVL1' * D
           + SVL3 * SVL2' * SVL1 * SVL0
           + SVL3 * SVL2 * SVL1' * SVL0
           + SVL3 * SVL2 * SVL0 * D'
           + SVL3' * SVL2' * SVL1 * SVL0' * D
           + SVL3' * SVL2 * SVL1 * SVL0' * D'
           + SVL3' * SVL2' * SVL1' * SVL0' * D' * SVR3 * SVR2' * SVR1' *
             SVR0;

    SVL2.d = SVL3' * SVL2' * SVL0
           + SVL3' * SVL1 * SVL0 * D'
           + SVL3 * SVL2 * SVL1' * SVL0
           + SVL3' * SVL2 * SVL1' * SVL0' * D
           + SVL3' * SVL2' * SVL1' * D' * SVR3 * SVR2' * SVR0
           + SVL3' * SVL2' * SVL1' * D * SVR3' * SVR2' * SVR1' * SVR0';

    SVL3.t = SVL2' * SVL0 * D
           + SVL3 * SVL2' * SVL1' * SVL0'
           + SVL3' * SVL2 * SVL0' * D
           + SVL3 * SVL2 * SVL0 * D'
           + SVL3' * SVL2' * SVL1 * SVL0' * D';

END$
```

EP1810 EPLD as a Bar Code Decoder

October 1990, ver. 2

Application Brief 27

Introduction

This application brief describes a bar code decoder implemented in an Altera EP1810 EPLD. The EP1810 decodes a generic bar code, stores the decoded data byte, and alerts a microprocessor that data is ready. The following information is provided:

- ❒ Definition of a generic bar code
- ❒ Use of state machine syntax for design of control functions
- ❒ Use of TTL macrofunctions
- ❒ Description of a bar code decoder implemented in an EPLD
- ❒ Processing and verifying the design

The application brief also shows several design entry methods used to implement the design. The completed design consists of a mix of design entry formats: state machine design entry specifies an internal controller, while schematic capture and TTL macrofunctions define additional functions.

What is a Bar Code?

A bar code represents binary data or program information. Its flexibility and low cost have made it a popular means of representing this kind of information. For certain applications—particularly when it is important that a first-time read be successful—a bar code yields better results than optical character recognition (OCR). Bar codes are also suitable for applications that cannot use a magnetic stripe or other media to represent information.

Physical Specifications

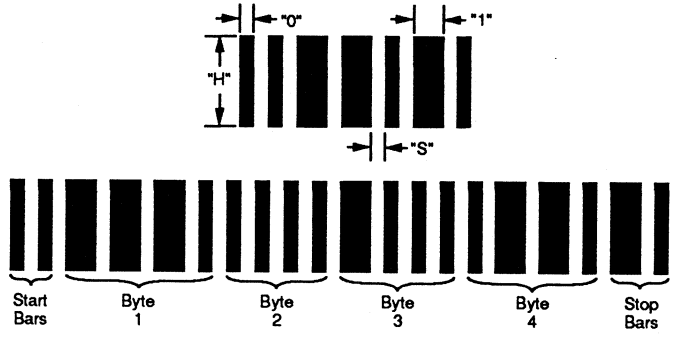
Although there are as many variations of bar code applications, they show enough common features to examine a "generic" case (see Figure 1).

- ❒ All bar codes contain 0, 1, and space characters.
- ❒ The 0 and space characters always have the same width; the 1 is twice that width.
- ❒ All bar codes have a header and a tail.
- ❒ All data follows the header and ends with the tail.
- ❒ All bar codes have maximum limits on the number of data bytes.

To ensure accurate bar code detection, the reader—usually a light pen—

Figure 1. Sample Bar Code

A generic bar code has 0, 1, and space characters. Bar code sequences consist of start and stop bars, data, and checksum bytes. The generic bar code described here has a header consisting of a 00 sequence, followed by a checksum byte. The tail is a 10 sequence.



must scan the bar code at relatively constant speed. Therefore, bar codes should be fairly short, since it is easier to move a light pen at constant speed over a short distance than over a long one. Before reading the bar code, the light pen output is low, indicating that the light pen is inactive (i.e., in a white region). As the light pen reaches the black region of the header, its output goes high. If the light pen's speed varies too much, the data is read incorrectly. Errors of this type are easily detected by comparing checksums.

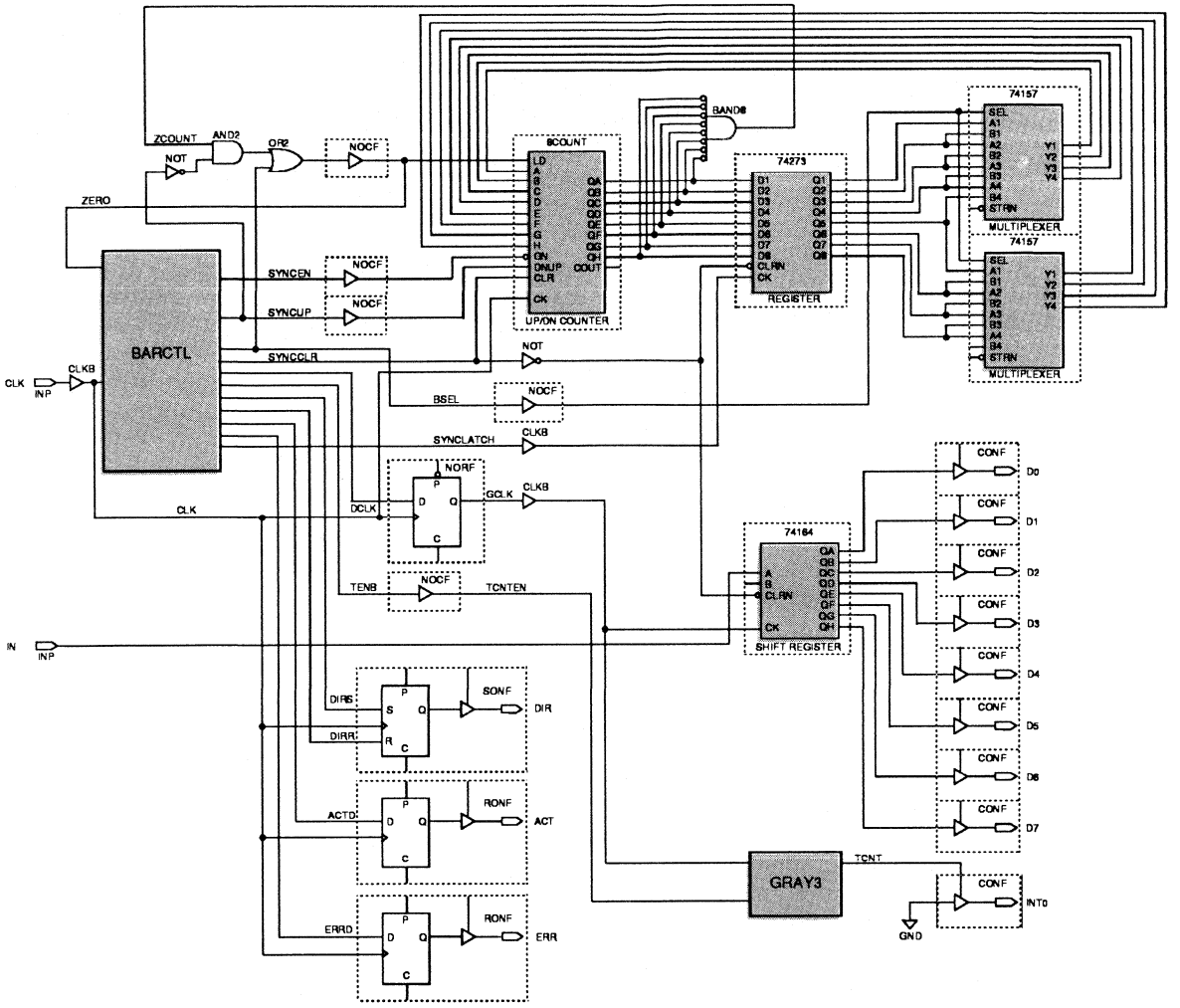
Bar Code Decoder Description

Figure 2 shows a bar code decoder implemented in an EP1810. The bar code data is fed into the EP1810 through the **IN** input. The synchronous counter uses the data to determine the input data read rate, and the state machine needs it to decode incoming data, which is stored in a shift register. Once eight bits of data have been shifted into the shift register, an open collector interrupt to the microprocessor (**INT0**) goes low. Various status outputs (**DIR**, **ACT**, and **ERR**) indicate the current state of the bar code decoder.

The bar code decoder consists of five parts:

- Synchronous Counter
- Byte Counter
- Shift Register
- Status Output
- Controller State Machine

Figure 2. Bar Code Decoder Schematic



Synchronous Counter The bar code decoder uses the bar code header's leading single-width black region to measure the width of a 0. When the light pen passes over the first black region, the synchronous counter begins counting, and stops only when the light pen has completed reading the first black region (i.e., when it reads the beginning of the next white region). The count value thus reflects the amount of time required to read the width of a space or 0 bar. It takes twice the count value to read a 1 bar.

10

The count value samples the bar code data. Since the light pen's scanning speed will vary slightly, the count value is only an approximate measure of the width. Therefore, data should be sampled in the center of the black and white regions rather than on the edges. The first sampling occurs in the middle of the space following the leading black region, when the synchronous counter reaches half of its synchronous value. Sampling is achieved by bit-shifting the latched synchronous counter with a pair of 74157 multiplexers. The counter is reset, and the next sampling occurs when it reaches the full synchronous value.

The values of data samples correspond to the bar-encoded data. If a black-white region is read, the light pen has passed over a single-width black region followed by a single-width white region, indicating a 0. If a black-black-white region is read, the light pen has passed over two adjacent black regions followed by a white region, indicating a 1. If any other combinations starting with black are read, or if two adjacent white regions are read, the code is invalid.

The synchronous counter is implemented with macrofunctions from the A+PLUS TTL MacroFunction Library (one **8COUNT**, two **74157**s, and one **74273**). The associated terminal-count circuitry includes basic gated logic and an **NOCF** primitive.

Byte Counter The byte counter counts the number of bits shifted in. When a complete byte has been read, the byte counter signals the microprocessor. The byte counter is implemented with a Gray-code sequence, in which only one bit changes between any two count transitions, ensuring that the outputs are glitch-free and preventing spurious outputs from inadvertently interrupting the microprocessor. The byte counter has an open-collector output to the microprocessor (**INT0**) that is created by connecting the input of a tri-state driver to ground and selectively enabling the tri-state (**TCNT**). Figure 3 shows the byte counter implemented in a State Machine File (SMF).

Shift Register Input data is stored in a **74164** shift register. The shift register clock is fed from the state controller state machine to ensure that data is latched at the appropriate time. The **74164** outputs must be buffered and connected to the microprocessor bus. The shift register is implemented by a **74164** macrofunction and **CONF** primitives.

Status Outputs Special outputs allow external devices to determine the status of the bar code decoder. The status information consists of the open collector **CONF** output and the **SONF** and **RONF** status flags.

A direction signal, **DIR**, is provided to compensate for backwards scanning of a bar code. If inactive, the **DIR** signal indicates that the tail was read before the header. In this instance, all data and checksum words are

Figure 3. Byte Counter State Machine File

This eight-stage Gray-code counter uses the high-level State Machine File format.

```
% GRAY3  3 BIT GRAY COUNTER %
PART: EP1810J
INPUTS:
OUTPUTS:
NETWORK:
EQUATIONS:

   TCNT = TCNTEN*/G2*/G1*/G0;

MACHINE: GRAY3

  CLOCK: GCLK
  CLEAR: SYNCCLR
  STATES: [ G2 G1 G0 ]
   S0     [  0  0  0 ]
   S1     [  0  0  1 ]
   S2     [  0  1  1 ]
   S3     [  0  1  0 ]
   S4     [  1  1  0 ]
   S5     [  1  1  1 ]
   S6     [  1  0  1 ]
   S7     [  1  0  0 ]

S0:  S1   % STATES MAKE UNCONDITIONAL %
S1:  S2   % TRANSITION TO NEXT STATE  %
S2:  S3
S3:  S4
S4:  S5
S5:  S6
S6:  S7
S7:  S0

END$
```

reversed, and the microprocessor must make the compensating transformations. The ability to read bar codes backwards and forwards also permits them to be read upside down or from right to left.

The **ACT** signal indicates that a bar code is being scanned in. This signal is useful for a variety of error handling or initialization tasks. For example, a microprocessor may poll this signal and enter special routines dedicated to bar code reading.

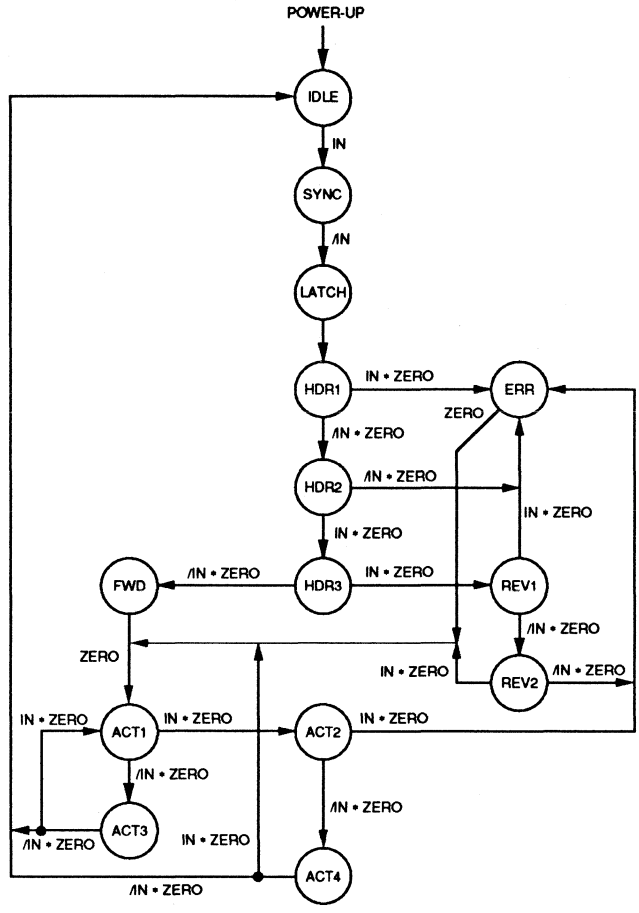
The **ERR** signal indicates that an illegal sequence of white and black regions was encountered during the decoding process; perhaps the bar code was unreadable or the scan rate was not uniform enough. The microprocessor can use **ERR** to dump illegal reads without first computing and comparing a checksum against the checksum byte passed to the microprocessor by the bar code.

Controller State Machine The bar code decoder must determine if a header is valid, and then convert the white and black bar code regions to machine-readable data. It also coordinates activities of the synchronous counter, shift register, byte counter, and status generation circuitry. This

coordination is best achieved with a centralized state controller state machine, shown in Figure 4. (Figure 5 shows the corresponding SMF.)

Figure 4. Bar Code Controller State Diagram

Bar code decoding is coordinated by the state controller. This diagram shows the behavior of the BARCTL portion of the state controller.



The following control states are used:

IDLE The machine idles until the light pen reads a black region; then the sync counter is started.

SYNC The synchronous counter counts while in this state. The machine stays in this state until the light pen reads the first white region. The count corresponds to the width of the first black bar.

LATCH The machine latches the count value into a holding register. For the next count cycle, the control line **BSEL** shifts the count value to the right by one bit. Subsequently, the synchronous count expires on every modulus of the latched count, and triggers reading of the input stream.

HDR1, HDR2, FWD, REV, ERR The machine begins reading the rest of the header bits. Two sequences are possible, depending on whether a 00 (forward) sequence or a 01 (backward) sequence is read. Any improper reads cause the state machine to go to the **ERR** state. Direction status is latched, depending on state **FWD** or **REV**. Once the header information is read, data and checksum bytes are read into the shift register.

ACT1, ACT2, ACT3, ACT4 These four states are the active reading states. The **ACT1-ACT3** loop indicates that a 0 was read. The **ACT1-ACT2-ACT4** loop corresponds to reading of a 1. Reading stops when a long white space is read, indicating the end of the bar code.

The controller is implemented in a State Machine File, shown in Figure 5.

Implementing the Bar Code Decoder

Figure 2 shows a LogiCaps schematic of the bar code decoder. The synchronous counter is implemented by four macrofunctions (one **8COUNT**, two **74157**s, and one **74273**). Terminal-count circuitry consists of logic and an **NOCF** primitive. The byte counter, **GRAY3**, is implemented in the State Machine File **GRAY3.SMF** (Figure 3). The shift register is implemented by combining a **74164** macrofunction with eight **CONF** primitives. The status information module consists of a **CONF** configured as an open collector output, and status flags implemented by **SONF** and **RORF** primitives. The state machine controller is implemented in the State Machine File **BARCTL.SMF** (Figure 5).

10

Figure 5. Bar Code Controller State Machine File

```
% Bar Code Controller %
PART: EP1810J
INPUTS:
OUTPUTS:
NETWORK:
EQUATIONS:
  DIRS = FWD;                DIRR = IDLE;
  ACTD = !IDLE;              ERRD = ERR * !IDLE;
  SYNCCLR = IDLE * /IN;      SYNCUP = SYNC;
  SYNCEN = IDLE;             SYNCLATCH = LATCH;
  DCLK = ACT2 + ACT3;        TENB = ACT2 + ACT3 + ACT4;
  BSEL = LATCH;

MACHINE: BARCTL
  CLOCK: CLK
  STATES: [ Q3 Q2 Q1 Q0 ]
    IDLE  [  0  0  0  0 ]
    SYNC  [  1  1  0  1 ]
    LATCH [  1  1  1  1 ]
    HDR1  [  0  1  1  1 ]
    HDR2  [  1  0  0  0 ]
    HDR3  [  1  0  0  1 ]
    FWD   [  1  0  1  0 ]
    REV1  [  0  1  1  0 ]
    REV2  [  0  1  0  1 ]
    ACT1  [  0  0  1  1 ]
    ACT2  [  0  1  0  0 ]
    ACT3  [  0  0  0  1 ]
    ACT4  [  0  0  1  0 ]
    ERR   [  1  0  1  1 ]
IDLE:
        IF IN THEN SYNC
SYNC:
        IF /IN THEN LATCH
LATCH:
        HDR1
HDR1:
        IF /IN * ZERO THEN HDR2
        IF  IN * ZERO THEN ERR
HDR2:
        IF  IN * ZERO THEN HDR3
        IF /IN * ZERO THEN ERR
HDR3:
        IF IN * ZERO THEN REV1
        IF /IN * ZERO THEN FWD
FWD:
        IF ZERO THEN ACT1
REV1:
        IF IN * ZERO THEN ERR
        IF /IN * ZERO THEN REV2
REV2:
        IF IN * ZERO THEN ACT1
        IF /IN * ZERO THEN ERR
ACT1:
        IF IN * ZERO THEN ACT2
        IF /IN * ZERO THEN ACT3
ACT2:
        IF IN * ZERO THEN ERR
        IF /IN * ZERO THEN ACT4
ACT3:
        IF IN * ZERO THEN ACT1
        IF /IN * ZERO THEN IDLE
ACT4:
        IF IN * ZERO THEN ACT1
        IF /IN * ZERO THEN IDLE
ERR:
        IF ZERO THEN ACT1
END$
```

Design Processing

Three separate files in two different formats are used for design input. LogiCaps generates the **BAR.ADF** file. **BARCTL.SMF** and **GRAY3.SMF** are state machine files. The Altera Design Processor (ADP) of the A+PLUS software processes the designs and links them at the same time, as long as the designer enters all three filenames when the ADP issues the **FILE NAME:** prompt.

The ADP then creates a JEDEC output file, **BAR.JED**, for EPLD programming. The utilization report generated by the ADP indicates that the following resources are used: 41 of the 48 macrocells, 2 of the 17 inputs, and only 34 percent of the available logic.

Design Verification

The A+PLUS Functional Simulator (FSIM) simulates the design to ensure proper operation. In this example, a serial input stream corresponding to a valid bit stream is read by the design. It properly sequences through the states, latches the data, and interrupts a processor. The simulation data is shown in Figure 6.

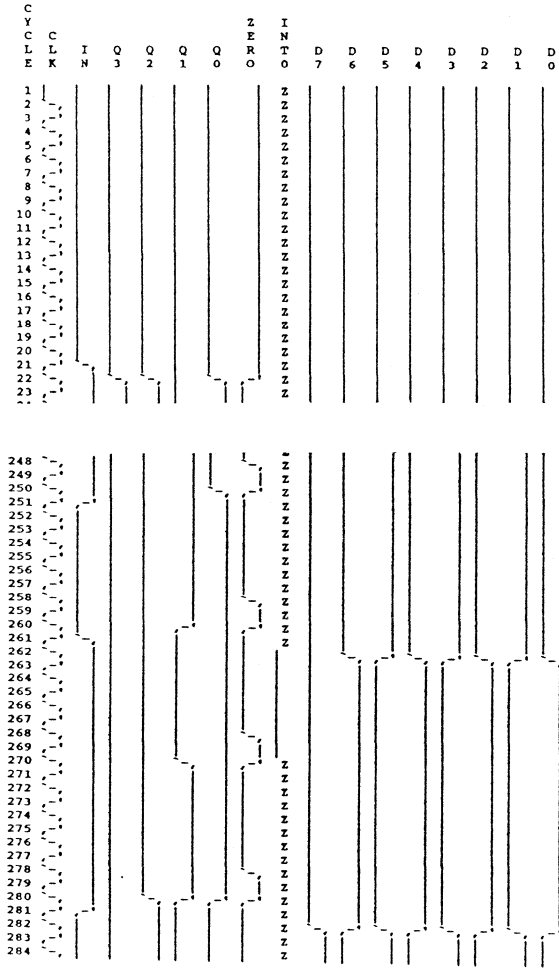
Figure 6. Simulation Data

Simulation is driven with data input that emulates this sequence. Design behavior is monitored in the Functional Simulator.

10

The controller state machine is verified by comparing the **Q0** to **Q3** outputs, and the state table values in Figure 5. The actual simulation waveforms are shown in Figure 7.

Figure 7. A+PLUS Functional Simulator Waveforms



Timing Simulation
for EP-Series EPLDs

October 1990, ver. 4

Application Brief 54

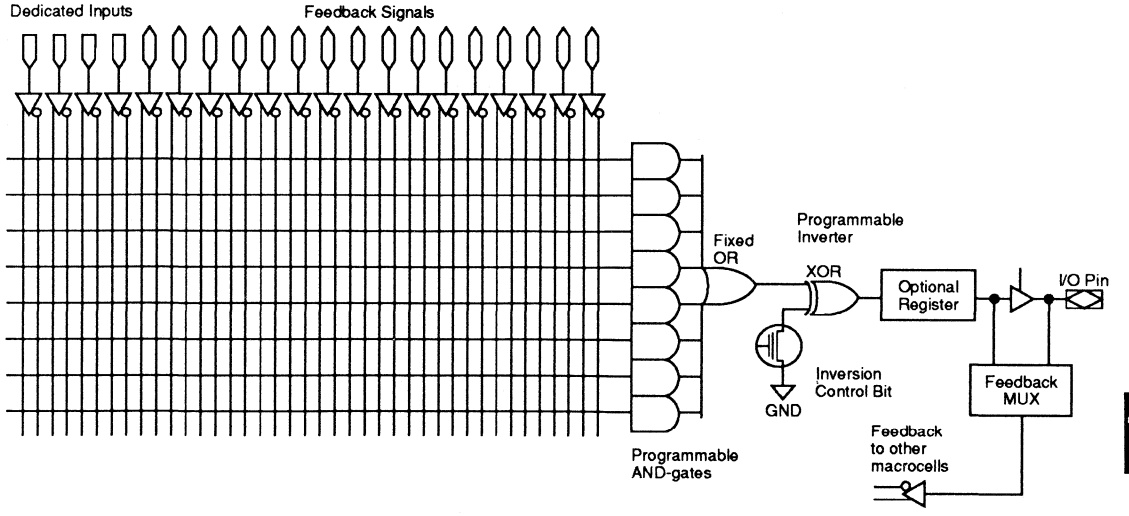
Introduction

EPLDs integrate complex functions into single-chip solutions. After a design is functionally correct, timing analysis should be completed to ensure AC parameter compatibility. This application brief discusses the timing delays in Altera EP-series EPLDs. It models the internal delay paths inherent in every EPLD, shows their relationships to AC specifications (shown in the EPLD data sheets), and presents worst-case values for these parameters. Designers can model and simulate their own logic designs once they understand how the A+PLUS software actually implements logic designs in EPLDs.

Basic EPLD Architecture

To accurately model timing characteristics, a designer must understand how logic is implemented in the EPLD. Most designs targeted for EP-series EPLDs contain basic gates and TTL macrofunctions, which are emulated by the EPLD's general macrocell structure. This macrocell structure consists of an array of logic in a programmable-AND/fixed-OR configuration, with a programmable inverter (XOR), optional flip-flop, and feedback (see Figure 1).

Figure 1. EP-Series EPLD Macrocell



10

When modelling EPLD timing, the concept of "gate delay" is not a useful measure. The EPLD logic array contains *n*-input AND gates called product terms (*n* is the number of connections). Depending on the logic implemented, a single product term may represent one to several gate equivalents.

AND/OR/XOR Structure

The AND portion of the EPLD logic array consists of a group of AND gates, each of which has a very large number of possible inputs that are selected by EPROM bits. The EPROM bits serve as electrical switches. An erased bit passes the input into the AND gate (switch on), while a programmed bit cuts it off (switch off). All bits are initially erased.

The number of possible inputs to an AND gate varies from 36 (EP320) to 88 (EP1810). In EP300-series, EP600-series, and EP900-series EPLDs, every dedicated input, its inversion, every macrocell feedback, and its inversion are possible inputs to the AND gate. In EP1800-series EPLDs, which have local and global busing, not every macrocell feedback is available at every AND gate. (Restricting the number of feedbacks preserves the speed characteristics of the device.) A significant portion of the component delay is in the propagation through the array (described later in more detail).

The AND gates feed a fixed 8-input OR function, so named because the AND functions are hard-wired into the OR gates, and cannot be redistributed if they are left unused.

The OR gate feeds a programmable inverter (XOR), which is controlled by a dedicated EPROM bit.

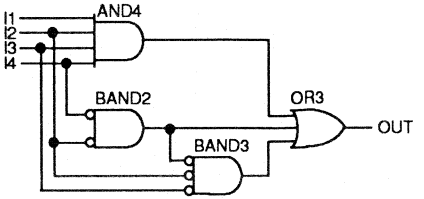
Designing Explicitly for EPLD Architecture

The AND/OR/XOR structure can implement general logic structures in either sum-of-products or product-of-sums form. However, since Altera's A+PLUS software automatically translates the input design file into a minimized Boolean form, designers are not necessarily aware of the intermediate Boolean form. Nevertheless, it is still important for designers to understand how Boolean logic is implemented in the array structure.

Figure 2 shows a single network of SSI functions and its corresponding Boolean equation and Karnaugh map. Figure 3 shows how a sum-of-products equation is formed by blocking adjacent product terms, and ORing them together by programming appropriate connections. (If there are more than 8 product terms, it is not possible to implement the logic in a single AND/OR array of the type shown.) The product terms P4 through P8 do not interfere because they are programmed out (a low logic level results because both the true and complement of every signal are ANDed).

The architecture also supports product-of-sums logic—the ANDing of OR functions—through De Morgan's inversion. Product-of-sums logic is possible because both the true and complement forms of the inputs are

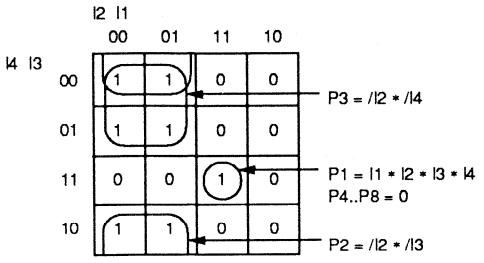
Figure 2. Sample Logic

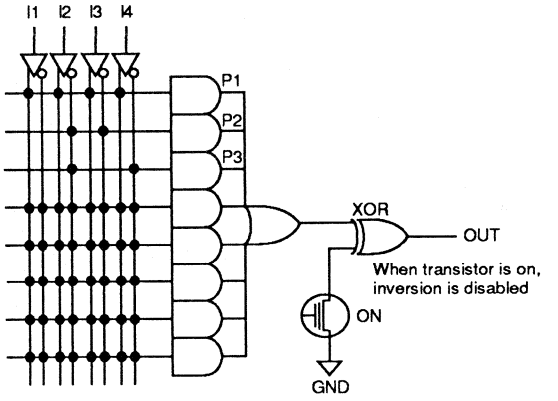


OUT = (I1 * I2 * I3 * I4) + (/I2 * /I4) + (/I2 * /I4) * (/I2 * /I3)

I4 I3 \ I2 I1	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	0	0	1	0
10	1	1	0	0

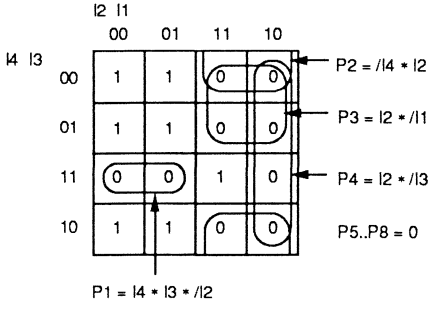
Figure 3. Sum-of-Products Logic

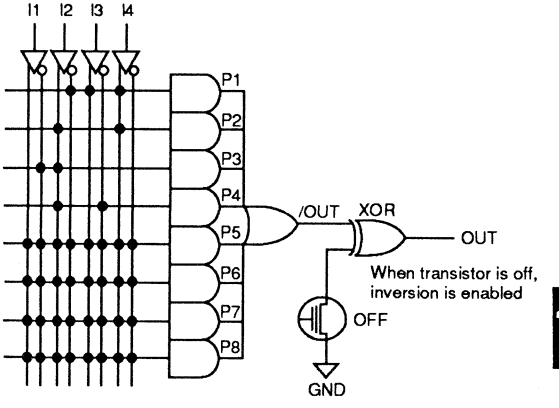




available to the AND array. Figure 4 shows how product terms are formed, by grouping the zeros and setting the invert bit. In this case, there are more product terms than in the sum-of-products form, but sometimes De Morgan's inversion dramatically reduces their number.

Figure 4. De Morgan's Inversion





10

EPLD Delay Parameters

The propagation delay through the logic array (t_{LAD}) is modelled as a constant: A+PLUS logic minimization and the array structure of macrocells make it unnecessary to do otherwise. Other elements in the timing model are akin to those found in conventional logic: input and output delay parameters (t_{IN}, t_{IO}, t_{OD}), register parameters (t_{SU}, t_H, t_{CLR}, t_{ICS}, t_{IC}), and internal connection parameters (t_{FD}). Figure 5 shows the timing model for EP-series EPLDs.

t_{IN} Input pad and buffer delay: the time required for the dedicated input pin to drive the true and complement data input signals into the AND array.

t_{IO} I/O input pad delay: the delay added to t_{IN} for I/O pins that are used as inputs.

t_{OD} Output buffer and pad delay. In registered applications, t_{OD} is the clock-to-output delay of the flip-flop. In combinatorial applications, it is the delay from the output of the array to the output pin.

t_{XZ} Active output to tri-state delay: the time between a high-to-low transition on the enable input of the tri-state buffer to assertion of a high-impedance value at an output pin.

t_{ZX} Tri-state to active output delay: the time between a low-to-high transition on the enable input of the tri-state buffer to assertion of a high or low logic level at an output pin.

t_{LAD} Logic array delay: the delay incurred as an input or feedback travels through the AND/OR/XOR structure.

t_{SU} Register setup time: the internal setup time of the register inside a macrocell, measured from the register data input until the register clock.

t_H Register hold time: the internal hold time of the register inside a macrocell, measured from the register clock to the register data input.

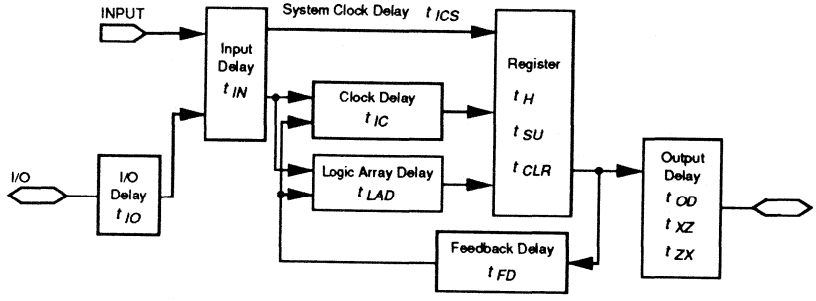
t_{CLR} Asynchronous register clear time: the time required for a low signal to appear at the output of a register after the transition at the input of the logic array.

t_{ICS} System clock delay: the total delay incurred between the output of the input pad and the clock input of the registers for dedicated clock pins.

t_{IC} Clock delay: the total delay incurred between the output of an input pad or I/O pad and the clock input of a register, including the time required to pass through the logic array.

t_{FD} Feedback delay. In registered applications, t_{FD} is the delay from the output of the register to the input of the logic array. In combinatorial applications, it is the delay from the combinatorial feedback to the input of the logic array.

Figure 5. EPLD Timing Model



AC Timing Characteristics

The data sheet for each EP-series EPLD gives timing parameters that characterize the AC operating specifications. These parameters are measured worst-case values, derived from extensive performance measurements and guaranteed by 100% testing. These AC characteristics include t_{PD1}, t_{PD2}, t_{PZX}, t_{PXZ}, t_{CLR}, t_{SU}, t_H, t_{CO1}, t_{CNT}, t_{ASU}, t_{AH}, t_{ACO1}, and t_{ACNT}. Each parameter can be represented by a combination of the internal delay elements described earlier in this application brief (see Figure 6).

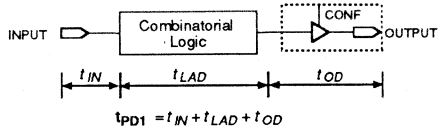
t_{PD1} Propagation delay: the dedicated input to non-registered output delay. t_{PD1} is the time required for any dedicated input to propagate through the combinatorial logic in a macrocell and appear at the external EPLD output pin. This delay is the sum of the input delay (t_{IN}), array delay (t_{LAD}), and output delay (t_{OD}).

t_{PD2} Propagation delay: the I/O pin input to non-registered output delay. t_{PD2} is the time required for any I/O pin input to propagate through the combinatorial logic in a macrocell and appear at the external EPLD output pin. This delay is the sum of the I/O delay (t_{IO}), input delay (t_{IN}), array delay (t_{LAD}), and output delay (t_{OD}).

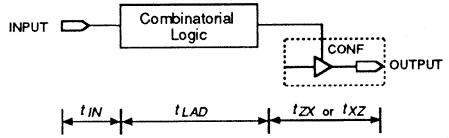
t_{PZX} Tri-state to active output delay: the time required for an input transition to change an external output from a tri-state (high impedance) logic level to a valid high or low logic level. This delay is the sum of the input delay (t_{IN}), array delay (t_{LAD}), and the time to disable the tri-state buffer (t_{ZX}).

10

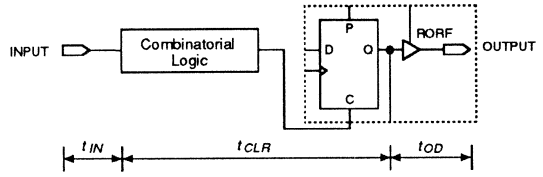
Figure 6. EPLD Timing Equations



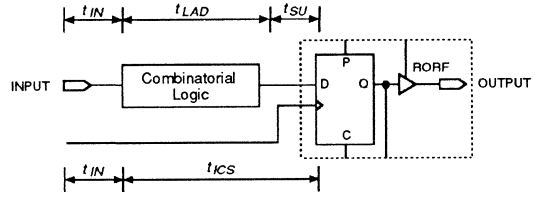
$t_{PD1} = t_{IN} + t_{LAD} + t_{OD}$



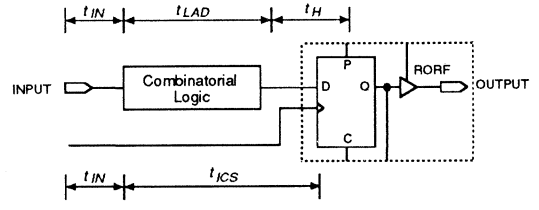
$t_{PZX} = t_{IN} + t_{LAD} + t_{ZX}$
$t_{PXZ} = t_{IN} + t_{LAD} + t_{XZ}$



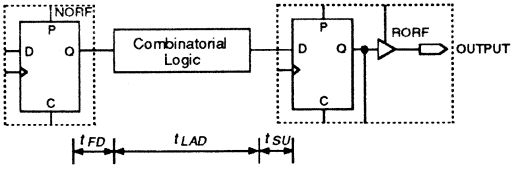
$t_{CLR} = t_{IN} + t_{CLR} + t_{OD}$



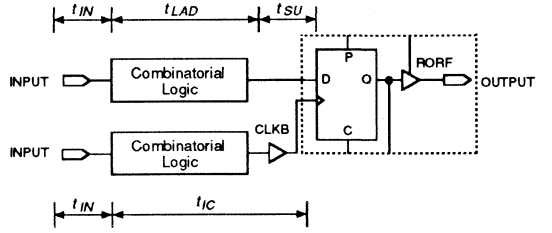
$t_{SU} = (t_{IN} + t_{LAD}) - (t_{IN} + t_{ICS}) + t_{SU}$



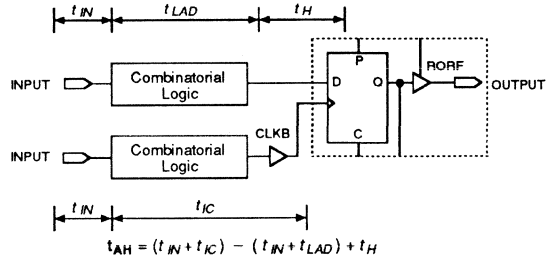
$t_{H} = (t_{IN} + t_{ICS}) - (t_{IN} + t_{LAD}) + t_{H}$



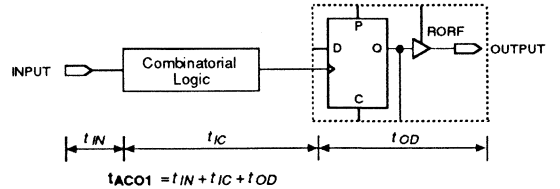
$t_{CNT} = t_{FD} + t_{LAD} + t_{SU}$
$t_{ACNT} = t_{FD} + t_{LAD} + t_{SU}$



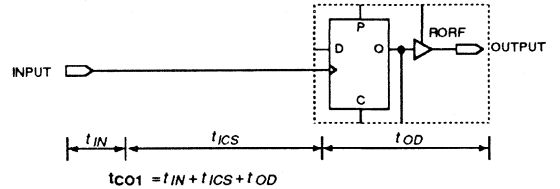
$t_{ASU} = (t_{IN} + t_{LAD}) - (t_{IN} + t_{IC}) + t_{SU}$



$t_{AH} = (t_{IN} + t_{IC}) - (t_{IN} + t_{LAD}) + t_{H}$



$t_{ACO1} = t_{IN} + t_{IC} + t_{OD}$



$t_{CO1} = t_{IN} + t_{ICS} + t_{OD}$

t_{PXZ} Active output to tri-state delay: the time required for an input transition to change an external output from a valid high or low logic level to a tri-state (high-impedance) logic level. This delay is the sum of the input delay (t_{IN}), array delay (t_{LAD}), and the time to enable the tri-state buffer (t_{XZ}).

t_{CLR} Time to clear register delay: the time required for a low signal to appear at the external output, measured from the input transition. This delay is the sum of the input delay (t_{IN}), register clear delay (t_{CLR}), and the output delay (t_{OD}).

t_{SU} Setup time on the register: the time data must be present at the register before the system clock. This value is the difference between the sum of input delay (t_{IN}), array delay (t_{LAD}), and internal register setup time (t_{SU}), and the sum of the input delay (t_{IN}) and the system clock delay (t_{ICS}).

t_{H} Hold time for the register: the time the data must be valid after the system clock. This value is the difference between the sum of the internal input delay (t_{IN}), the system clock (t_{ICS}), and the internal register hold time (t_{H}), and the sum of the input delay (t_{IN}) and logic array delay (t_{LAD}).

t_{CO1} System clock to output delay: the time required to obtain a valid output after the system clock is asserted on an input pin. This delay is the sum of the input delay (t_{IN}), the system clock delay (t_{ICS}), and the output delay (t_{OD}).

t_{CNT} System clocked counter period: the minimum period maintained by a counter. This delay is the sum of the feedback delay (t_{FD}), the logic array delay (t_{LAD}), and the internal register setup time (t_{SU}).

t_{ASU} Asynchronous setup time on the register: the time data must be present at the register before an asynchronous clock. This value is the difference between the sum of the input delay (t_{IN}), array delay (t_{LAD}), and the register setup time (t_{SU}), and the sum of the input delay (t_{IN}) and the clock delay (t_{IC}).

t_{AH} Asynchronous hold time for the register: the time the data must be present after an asynchronous clock. This value is the difference between the sum of the input delay (t_{IN}), the clock delay (t_{IC}), and the hold time (t_{H}), and the sum of the input delay (t_{IN}) and logic array delay (t_{LAD}).

t_{ACO1} Asynchronous clock to output delay: the time required to obtain a valid output after a clock is asserted on a dedicated clock pin. This delay is the sum of the input delay (t_{IN}), the clock delay (t_{IC}), and the output delay (t_{OD}).

t_{ACNT} Asynchronous clocked counter period: the minimum period maintained by a counter when it is asynchronously clocked. This delay is the sum of the feedback delay (t_{FD}), the logic array delay (t_{LAD}), and the register setup time (t_{SU}).

Tables 1 through 4 show the internal delay parameters for all EP-series EPLDs. It is also possible to calculate the internal timing model parameters by using the timing information in individual EPLD data sheets. In some cases, the modeled result differs slightly from the data sheet specification, because different guardbands are used to ensure compliance with the specification when the EPLDs are tested during manufacturing. The model does not account for the extended guardbanding of some data sheet specifications. Although data sheets show maximum and minimum values, the model results are considered to be "typical worst-case" performance.

Table 1. EP300-Series Timing Parameters

Parameter (ns)	EP330-12	EP330-15	EP320-1	EP320-2	EP320
t_{IN}	3	4	4	5	7
t_{IO}	1	1	1	1	1
t_{LAD}	6	7	20	22	26
t_{OD}	3	4	5	7	11
t_{ZX}	3	4	6	8	12
t_{XZ}	3	4	6	8	12
t_{SU}	3	4	8	10	14
t_{H}	0	0	10	10	10
t_{IC}					
t_{ICS}	2	3	7	7	7
t_{FD}	1	1	8	8	10
t_{CLR}					

Table 2. EP600-Series Timing Parameters

Parameter (ns)	EP640-12	EP640-15	EP630-15	EP630-20	EP610-25	EP610-30	EP610-35
t_{IN}	3	4	4	5	5	6	7
t_{IO}	0	0	2	2	2	2	2
t_{LAD}	4	6	6	9	14	17	19
t_{OD}	5	5	5	6	6	7	9
t_{ZX}	5	5	5	6	6	7	9
t_{XZ}	5	5	5	6	6	7	9
t_{SU}	5	6	6	8	8	8	8
t_{H}	5	6	6	8	12	12	12
t_{IC}	4	6	6	9	14	17	19
t_{ICS}	1	2	2	3	4	4	4
t_{FD}	1	1	1	1	3	5	8
t_{CLR}	4	6	6	9	16	17	21

Table 3. EP900-Series Timing Parameters

Parameter (ns)	EP910-30	EP910-35	EP910-40	EP910-45
t_{IN}	7	8	8	8
t_{IO}	3	3	3	3
t_{LAD}	16	19	22	25
t_{OD}	7	9	10	12
t_{ZX}	7	9	10	12
t_{XZ}	7	9	10	12
t_{SU}	10	10	10	12
t_{H}	15	15	15	17
t_{IC}	16	19	22	24
t_{ICS}	4	5	6	6
t_{FD}	4	6	8	10
t_{CLR}	19	22	25	28

Table 4. EP1800-Series Timing Parameters

Parameter (ns)	EP1830-20	EP1830-25	EP1830-30	EP1810-35	EP1810-40	EP1810-45
t_{IN}	5	7	7	7	7	7
t_{IO}	2	3	4	5	5	5
t_{LAD}	9	12	15	19	23	27
t_{OD}	6	6	8	9	10	11
t_{ZX}	6	6	8	9	10	11
t_{XZ}	6	6	8	9	10	11
t_{SU}	8	10	12	10	11	11
t_{H}	8	10	12	15	17	18
t_{IC}	9	12	15	19	23	27
t_{ICS}	4	5	7	4	6	8
t_{FD}	3	3	3	6	6	7
t_{CLR}	9	12	15	24	28	32

Timing Examples

In the following pages, four timing examples are presented to show how internal delay paths are calculated.

Example 1: Simple Combinatorial Circuit

The simple circuit shown in Figure 7 illustrates circuit partitioning into the EPLD internal delay paths. It is a combinatorial design that uses no clock or feedback. The timing restrictions are dependent only on the input, array, and output delay elements.

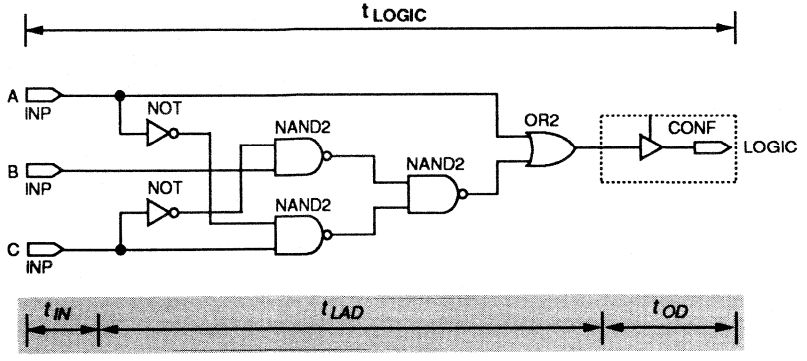
The circuit consists of three inputs, gated logic, and one output that together create three delay paths: input delay, array delay, and output delay. All gated logic is incorporated into the AND/OR array. The maximum time allowed for any input to propagate through the entire circuit (from input pin to output pin) is the propagation delay t_{PD1}. The worst-case t_{PD1} is given in each EPLD data sheet. This propagation delay always applies when combinatorial output logic is used:

$$t_{LOGIC} = t_{IN} + t_{LAD} + t_{OD} = t_{PD1}$$

Example 2: Combinatorial Feedback

Figure 8 shows a simple design that contains combinatorial feedback, i.e., a set of cross-coupled NOR gates. It also shows how the design is implemented with "flattened" primitives and partitioned into appropriate input, output,

Figure 7. Simple Combinatorial Circuit



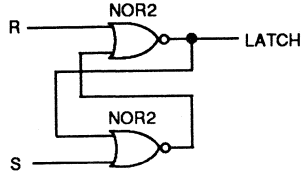
$t_{LOGIC} = t_{IN} + t_{LAD} + t_{OD}$

array and feedback delay paths. The propagation delay is the maximum combinatorial delay from input to output (path 1). If the output of the latch is a logic 0 (**Q** low), and a logic 1 is applied to the Set input (**S** high), the output of the latch will go to a logic 1 (**Q** high) within one propagation delay. The same principle is true if the latch is reset (**R** high, **S** low).

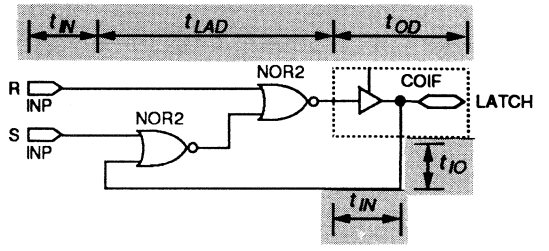
Path 2 is the hold time of the latch. To avoid any unwanted output glitches, the asynchronous setup time of the latch must be observed. The pulse width of either Reset (**R**) or Set (**S**) inputs must be long enough to ensure that the feedback signal can return from the output primitive and stabilize

Figure 8. Cross-Coupled NOR Gates

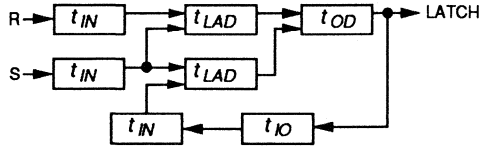
LogiCaps Schematic



Primitive Implementation



Delay Paths



Path 1 $= t_{IN} + t_{LAD} + t_{OD}$

Path 2 $= t_{IO} + t_{IN} + t_{LAD}$

10

the latch. The input pulse width must be greater than the time required for the feedback signal to return to the AND array and propagate through the gated logic. Otherwise, a high-to-low or low-to-high glitch may occur at the output. Thus, input pulse width is greater than the sum of t_{IO}, t_{IN}, and t_{LAD}. In this case, (t_{IO} + t_{IN}) is the feedback delay because the **COIF** feedback comes from the I/O pin.

Example 3: Synchronous Design

Figure 9 shows a synchronous design that uses three inputs, one clock (common to both D flip-flops), gated logic, and one active-low output. The flattened A+PLUS primitive implementation is partitioned to highlight the delay paths. The design contains five inherent delays—clock delay, input delay, array delay, feedback delay, and output delay—as well as setup and hold time requirements on each of the registers. All gated logic, including the inverter, is incorporated into the AND/OR array.

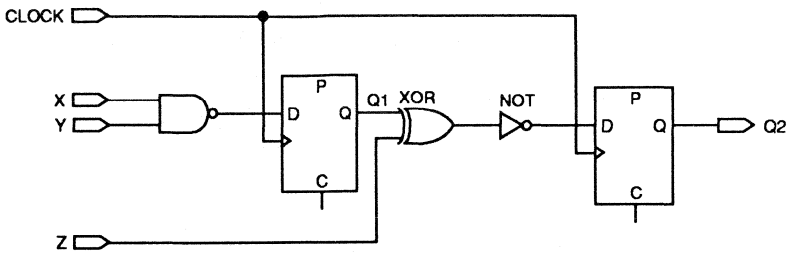
Path 1, which refers to the output register in the flattened schematic (second D flip-flop), is the clock-to-output delay. Assuming the data satisfies the setup time for the register, a rising-edge clock at the clock of the flip-flop causes data residing on the D input to appear at the output pin, after passing through the tri-state output buffer. The clock at the input pin is delayed into the register clock input by a specific time period (t_{IN}+t_{ICS}). This additional delay must be incorporated into the total clock-to-output delay. Thus, the maximum delay is the sum of the input delay (t_{IN}), the clock delay (t_{ICS}), and the output delay (t_{OD}).

The setup time is the time required for the input data to stabilize before the triggering edge of the clock. The setup time for the EPLD internal flip-flops has been modeled from the input, array, and clock delay paths, as well as the actual register setup time. Path 2 specifies the setup time requirement. The requirement for stable data at the flip-flop—the sum of the input, logic array, and setup time—is the first term in the Path 2 equation. The clock is delayed from the input pin to the clock input of the flip-flop by the sum of the input delay and the clock delay. The setup time for the EPLD is the difference between these quantities (as shown in the Path 2 equation). As long as the setup time is obeyed at the external inputs, the circuit will function properly.

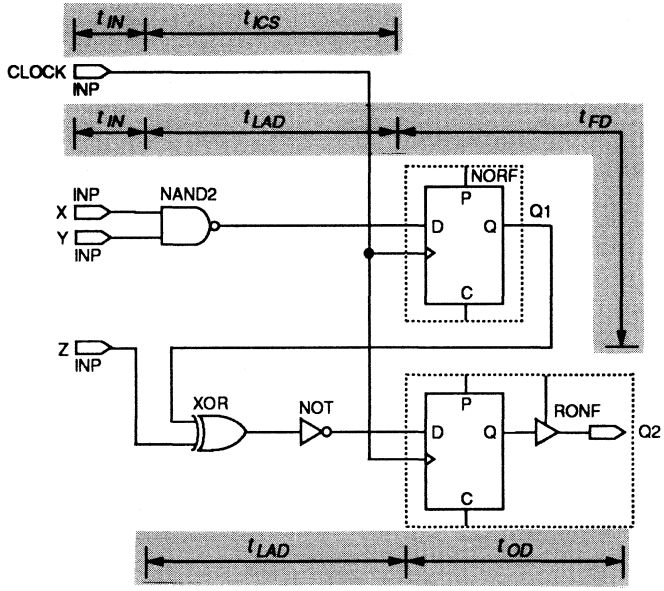
The minimum internal counter period, t_{CNT}, is governed by the internal feedback path. t_{CNT} (shown in Path 3) is the time required for a signal to pass from the upper register to the lower register, and determines the minimum internal clock period limit for the circuit. Once data is triggered into the top register, the signal passes through a feedback delay (t_{FD}) and array delay (t_{LAD}) before reaching the bottom register. It must then meet the setup time of the register (t_{SU}). The sum of these delays represents the minimum practical internal clock period. Therefore, circuits that depend

Figure 9. Synchronous Design

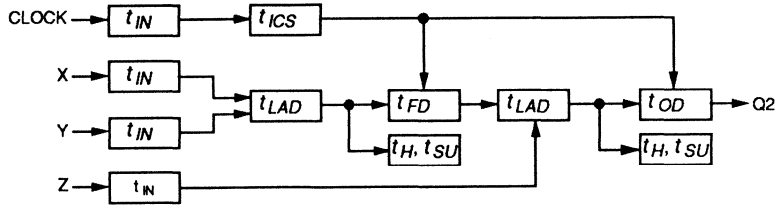
LogiCaps Schematic



Primitive Implementation



Delay Paths



Path 1 = $t_{IN} + t_{ICS} + t_{OD}$

Path 2 = $(t_{IN} + t_{LAD} + t_{SU}) - (t_{IN} + t_{ICS})$

Path 3 = $t_{FD} + t_{LAD} + t_{SU}$

10

only on internal signals can operate at the minimum clock period specified by the t_{CNT} parameter.

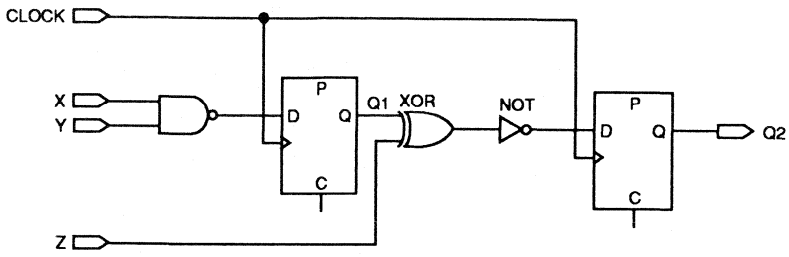
Some circuit functions are dependent on both external and internal signals. In this case, the maximum clock frequency is dependent on the input delay, array delay, output delay, feedback delay, and clock delay.

Example 4: Asynchronous Clock

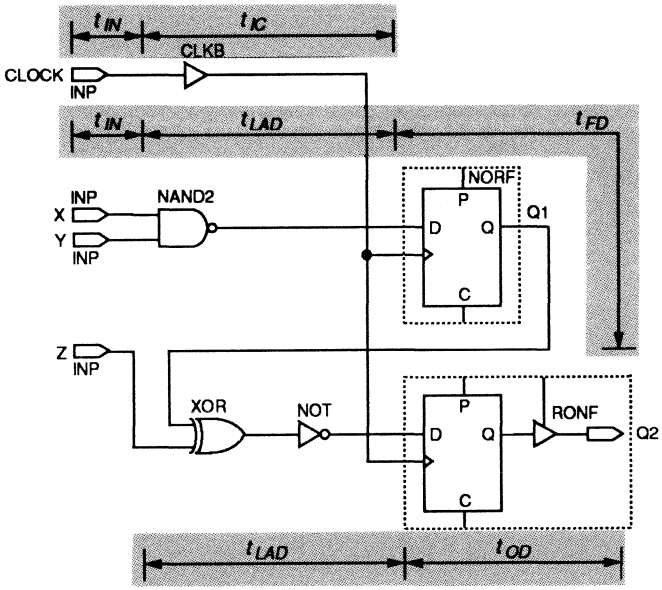
Figure 10 illustrates the asynchronous Clock feature in EP-series EPLDs. This design is similar to Example 3 (Figure 9), with the exception that the asynchronous clock option is implemented with the **CLKB** primitive. Therefore, the t_{IC} parameter is substituted for the corresponding t_{ICS} parameter in Example 3. The timing paths reflect the change: the clock-to-output delay (Path 1) and the setup time parameters (Path 2) are both affected, but the internal feedback path (Path 3) is not.

Figure 10. Asynchronous Clock Design

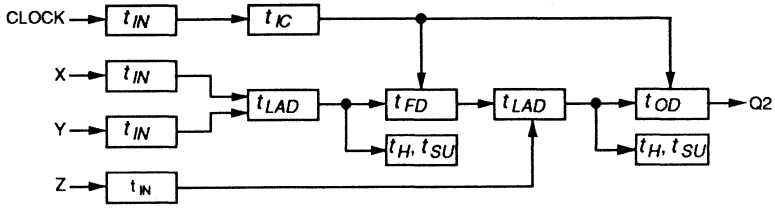
LogiCaps Schematic



Primitive Implementation



Delay Paths



Path 1 = $t_{IN} + t_{IC} + t_{OD}$

Path 2 = $(t_{IN} + t_{LAD} + t_{SU}) - (t_{IN} + t_{IC})$

Path 3 = $t_{FD} + t_{LAD} + t_{SU}$

10

Macrofunction Timing

A+PLUS software automatically flattens macrofunctions into low-level EPLD architectural elements (primitives), which can be analyzed with the timing model and techniques described in this application brief. Worst-case macrofunction timing can be obtained before the design is flattened by applying the timing model to the macrofunction schematics given in the *TTL MacroFunctions* manual.

Conclusion

To understand timing relationships in EP-series EPLDs, the internal delay paths must be broken into meaningful microparameters that model portions of the EPLD architecture. Accurate timing delay information can be obtained by adding appropriate combinations of these microparameters (given in Tables 1 through 4). Architectural information the parameters that apply and how they are implemented is provided in individual EPLD data sheets. The combination of these elements and the techniques described herein allow any timing path within an EPLD to be estimated accurately.

Estimating a Design Fit
for EP-Series EPLDs

October 1990, ver. 2

Application Brief 60

Introduction

This application brief provides an estimation formula and worksheet to determine which EP-series EPLD best suits a logic design. The following steps are suggested for estimating a design fit:

- ❒ Partitioning the design
- ❒ Determining the timing specifications
- ❒ Estimating a fit

Also included in this application brief are suggestions for design entry.

Familiarity with EPLD architecture and device characteristics is assumed. Refer to individual data sheets in this data book for complete descriptions of EP-series EPLDs.

Partitioning the Design

The design must be partitioned into functional blocks. Major functional blocks can be expressed with standard TTL functions for fast integration into the desired EPLD. If the design requires multiple EPLDs, I/O connections between the EPLDs should be minimized. The complete schematic should be structured as a set of subsystems—such as counters, shift registers, comparators, basic gates, and flip-flops—to facilitate design entry.

Determining Timing Specifications

Knowledge of basic clock frequency and critical timing paths is necessary to choose the appropriate EPLD. Critical timing paths are determined on the basis of total propagation delay (t_{PD}), maximum clock frequency (f_{CNT}), set-up time (t_{SU}), and clock-to-output delay (t_{CO1}). Refer to the individual data sheets in this data book for AC specifications. See also *Application Brief 54 (Timing Simulation for EP-Series EPLDs)* in this data book.

Estimating a Fit

The following formula is used to determine which EP-series EPLD will fit a logic design:

$$\mathbf{IP + OP + BFF + MR = TM}$$

where IP = number inputs – total EPLD inputs
OP = number of output pins
BFF = buried flip-flops
MR = macrofunction requirements
TM = total number of macrocells required

10

Input and Output Pins

The number of input pins (IP) is equal to the number of inputs in the schematic minus the total EPLD inputs (shown in the *Product Selection Guide* or individual data sheets). If IP is less than zero, enter zero in the formula. This subtraction is necessary because I/O pins can also be used as inputs.

The number of output pins (OP) is equal to the number of outputs in the schematic.

Buried Flip-Flops

The number of buried flip-flops (BFF) includes D, T, JK, or SR flip-flops that do not drive output pins.

Macrocell Requirements

The lower right-hand corner of each symbol in the A+PLUS TTL MacroFunction Library shows the maximum number of macrocells (MR) needed to build the function. Some macrofunctions have no macrocell specifications. They use only a portion of the logic array, so additional logic may be integrated before the entire macrocell is used.

Since basic combinatorial gates (NAND, OR, XOR, etc.) are implemented in the EPLD logic array, they do not require an entire macrocell, and may be excluded from the estimate.

Total Macrocells

Table 1 shows the number of available macrocells for various EP-series EPLDs. If the total number of macrocells (TM) required by a design is smaller than the number of available macrocells for a given ELPD, the design will probably fit into that EPLD. Macrocell availability is also found in the *Product Selection Guide* and individual data sheets.

Table 1. Total Macrocell Availability

EPLD	Macrocells
EP1830	48
EP1810	48
EP910	24
EP640	16
EP630	16
EP610	16
EP330	8
EP320	8

Design Entry

Design fit is quick and simple with LogiCaps schematic capture and a mouse. The A+PLUS Primitive Library contains functions to support all possible EPLD architectures, including basic gates, flip-flops, and I/O structures. Boolean equations may also be entered directly into the schematic. TTL macrofunctions that allow high-level design entry are available in the A+PLUS TTL MacroFunction Library. LogiCaps features split windows; multiple zoom levels; orthogonal rubberbanding; automatic tag-and-drag editing; and area editing, save, and load functions. Schematics may be printed or plotted for hard-copy documentation.

Helpful Hints

A+PLUS offers the following features to facilitate design entry. For additional information on design entry and guidelines, refer to the *MacroFunction Tutorial* in the ***LogiCaps*** manual.

MacroMunching

The A+PLUS Design Processor (ADP) automatically removes any unused logic within a macrofunction. This feature, called "MacroMunching," allows designers to use macrofunctions without the worry of losing design efficiency. For example, if a 74374 octal register is used in a schematic, but only six of the eight outputs are connected, the ADP automatically removes the logic associated with the two unused outputs from the design. Thus, the total macrocell count for the macrofunction is six rather than eight.

I/O Architecture Compression

A+PLUS automatically compresses the architecture of I/O primitives. Macrofunction outputs that drive I/O pins via a **CONF** primitive are compressed inside the I/O structure of the EPLD. Therefore, any macrofunction output that drives an I/O pin (**CONF**) should not be added to the Macrofunction Requirements section (MR) of the estimation formula.

Input Default Values

Each macrofunction contains intelligent input default values (**VCC** or **GND**). If a macrofunction input is not used, it automatically defaults to either **VCC** or **GND**, so the user will not have to manually connect **VCC** or **GND** to unused inputs. For example, an active-low Clear signal has an input default value of **VCC** that always disables an unused function. Refer to the *TTL MacroFunctions* manual for specific input default values.

Altera-Provided Macrofunctions

In addition to familiar TTL macrofunctions, the A+PLUS MacroFunction Library also provides custom functions optimized for EP-series EPLD architecture. These include counters such as **8COUNT**, which uses T-type

rather than D flip-flops. When implementing counters inside EPLDs, macrofunctions similar to **8COUNT** are recommended for most efficient EPLD resource utilization.

Estimation Example

The target EPLD for the sample circuit shown in Figure 1 is the EP610. When the estimation worksheet shown in Figure 2 is completed, it shows that the design will fit into the EP610 EPLD.

A blank estimation worksheet is provided at the end of this application brief.

Figure 1. Sample Circuit

The target EPLD in this sample circuit is an EP610.

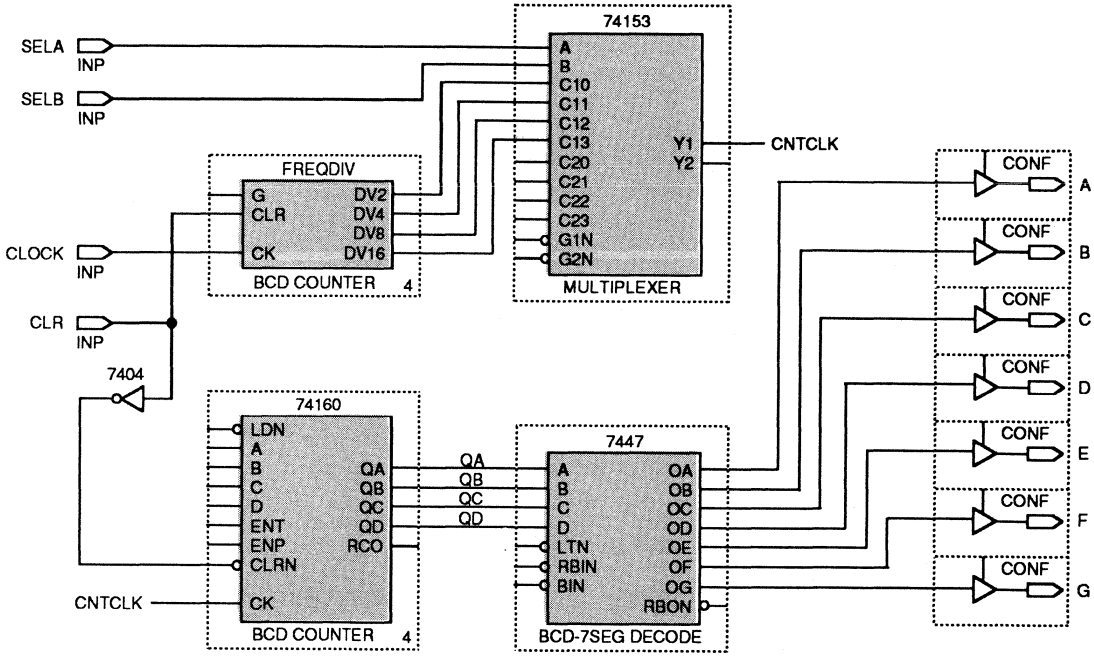


Figure 2. Estimation Worksheet Example

This example uses an EP610 as the target EPLD for the sample circuit shown in Figure 1.

1. INPUT PINS **IP = 4 – 4 = 0**

SELA	SELB	CLOCK	CLR	

2. OUTPUT PINS **OP = 7**

A	B	C	D	
E	F	G		

3. BURIED LOGIC **BFF = 0**

DFF = 0 TFF = 0 JKFF = 0 SRFF = 0

4. TTL LOGIC (MACROFUNCTIONS) **MF = 8**

FREQDIV = 4	74153 = 0			
74160 = 4	7447 = 0			

5. TOTAL MACROCELLS **TM = IP + OP + BFF + MF = 0 + 7 + 0 + 8 = 15**

6. TARGET EPLD **(Circle appropriate EPLD)**

EP1830	EP910	EP640	EP330
EP1810		EP630	EP320
		(EP610)	

10

Estimation Worksheet

1. INPUT PINS **IP =**

_____ _____ _____ _____ _____
_____ _____ _____ _____ _____
_____ _____ _____ _____ _____

2. OUTPUT PINS **OP =**

_____ _____ _____ _____ _____
_____ _____ _____ _____ _____
_____ _____ _____ _____ _____

3. BURIED LOGIC **BFF =**

DFF = _____ TFF = _____ JKFF = _____ SRFF = _____

4. TTL LOGIC (MACROFUNCTIONS) **MF =**

_____ _____ _____ _____ _____
_____ _____ _____ _____ _____

5. TOTAL MACROCELLS **TM = IP + OP + BFF + MF =**

6. TARGET EPLD **(Circle appropriate EPLD)**

EP1830	EP910	EP640	EP330
EP1810		EP630	EP320
		EP610	

Multiway Branching with the EPS448 SAM EPLD

October 1990, ver. 2

Application Brief 63

Introduction

The EPS448 Stand-Alone Microsequencer (SAM) EPLD performs a four-way branch in a single clock cycle. This application brief describes two processes:

- How to perform multiway branching
- How to access more than four product terms per branch

The application brief assumes a working understanding of the EPS448 architecture and the SAM Assembly Language (ASM) or Altera State Machine Input Language (ASMILE).

Beyond Four-Way Branching

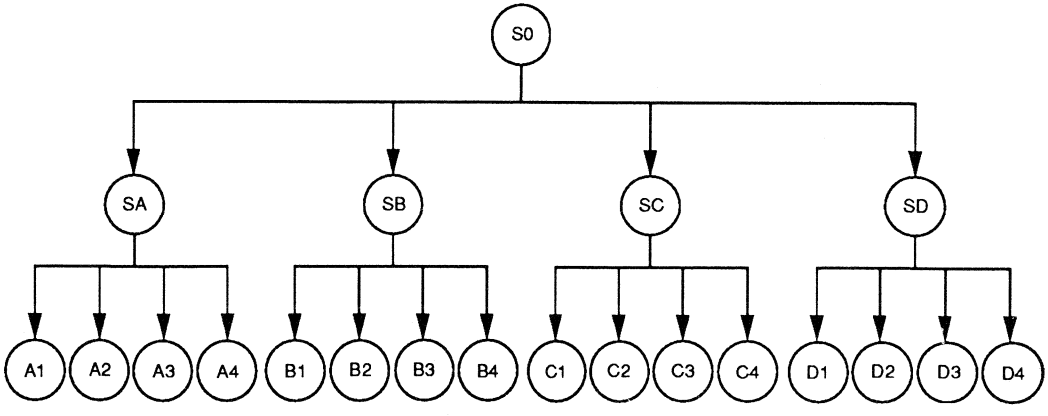
The EPS448 performs multiway branching in three different ways:

- Linked branching
- Dispatch routine
- Counter-conditioned branching

Linked Branching

Linked branching (shown in Figure 1) uses 2 clock cycles to perform branching of up to 16 ways. On the first clock cycle, the machine completes a 4-way branch to 4 intermediate states; on the second clock cycle, it

Figure 1. Linked Branching *Intermediate states can generate a 16-way branch in 2 clock cycles. In the first clock cycle, the machine branches to 1 of 4 intermediate states (SA to SD). On the second clock edge, the machine performs another 4-way branch to 1 of 16 states (A1 to A4, B1 to B4, C1 to C4, D1 to D4).*



10

finishes with a 4-way branch out of each of these intermediate states. The result is a 16-way branch in 2 clock cycles. This method can be implemented with either ASMILE or ASM design entry.

In applications running at low frequency, it may be possible to double the EPS448 clock frequency, use linked branching, and give the appearance of a multiway branch in a single system clock.

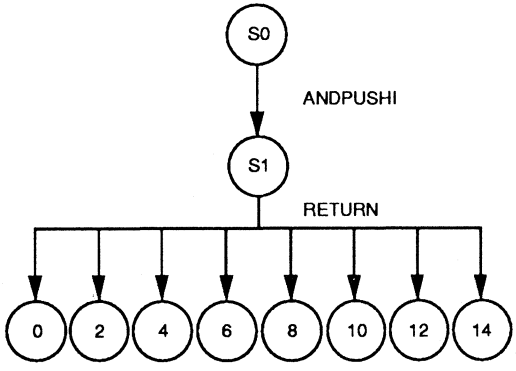
Dispatch Routine

The dispatch routine can perform any number of branches. Inputs are pushed onto the stack in the first clock cycle and are used as the next-state address in the second clock cycle. The ASM commands **PUSHI** or **ANDPUSHI** push the inputs onto the stack, and the **RETURN** command causes a branch to that address.

In the sample code shown in Figure 2, **ANDPUSHI** pushes the inputs onto the stack after first masking them with the binary number **00001110B**, where input **I7** is the most significant bit and input **I0** is the least significant bit. Consequently, an even number between 0 and 14 (decimal) is pushed onto the stack, matching the binary number for **I3 I2 I1 0**.

The **RETURN** command causes the top-of-stack to become the next address. Since this value is an even decimal number between 0 and 14, an 8-way branch to one of these addresses is performed. To complete the branch, the 8 even memory addresses between 0 and 14 must contain the 8 potential next states. Instructions are placed in a particular memory location

Figure 2. Dispatch Routine *The dispatch routine performs an 8-way branch in 2 clock cycles. On the first clock edge, the number represented by the top-of-stack is used as the next address. Since I3, I2, and I1 can be used to represent an even number between 0 and 14, each of these addresses is a potential next state.*




```
S0: [STATE0]  ANDPUSHI  00001110B;
    % Push I3, I2, and I1 onto the stack %

S1: [STATE1]  RETURN;
    % Jump to the top of stack %

0D: [OUT0] JUMP NEXT0;
2D: [OUT2] JUMP NEXT2;
4D: [OUT4] JUMP NEXT4;
6D: [OUT6] JUMP NEXT6;
8D: [OUT8] JUMP NEXT8;
10D: [OUT10] JUMP NEXT10;
12D: [OUT12] JUMP NEXT12;
14D: [OUT14] JUMP NEXT14;
```

by using an absolute label instead of a relative label. Absolute labels must represent a legal and unique memory location within the EPS448 EPLD. Addresses and numbers specified with SAM+PLUS software must start with a digit and end with **H** (hexadecimal), **D** (decimal), or **B** (binary). For example, the instruction **4D: [OUT4] JUMP NEXT4** in Figure 2 is placed in address **4D** of the microcode. It is executed if **/I3*I2*/I1** is true at the end of the **ANDPUSHI** command because input **I2** (high) translates to **4D**.

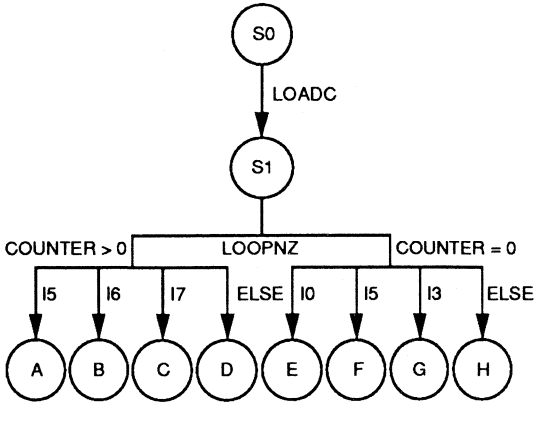
The following list shows examples of absolute, relative, and illegal labels:

Absolute labels:	**10001B, 0F2H, 44D**
Relative labels:	**F2H, 44D**
Illegal labels:	**44, 0F2D, 10001H**

Counter-Conditioned Branching

Counter-conditioned branching uses the counter as a flag and performs a two-way branch with the **LOOPNZ** command, based on the value of the counter. The two branched addresses can be multiway-branch addresses that permit two additional four-way branches. This branching scheme creates an eight-way branch in a single clock cycle, as shown in Figure 3. The code in Figure 3 shows an example of the actual ASM syntax required. The **LOADC** command at the **S0** label sets the counter to 1 or 0, based on the current value of the **I1** input. The next instruction, at label **S1**, performs the branch with **LOOPNZ**. Based on the value in the counter, the next state will come from label **ABCD** or from label **EFGH**. Since both of these labels are in

Figure 3. Counter-Conditioned Branching *An 8-way branch can be performed in a single clock cycle if the counter has been set as a flag. In the code shown here, LOADC sets the counter flag based on the input condition I1. LOOPNZ performs a 2-way branch based on the flag to label ABCD or EFGH, both of which are 4-way branch locations.*




```
  S0: IF I1 THEN [OUTS0] LOADC 1D;
      ELSE [OUTS0] LOADC 0D;

  S1: LOOPNZ ABCD ONZERO EFGH;

ABCD: IF I5 THEN [OUTA] JUMP NEXTA;
      ELSEIF I6 THEN [OUTB] JUMP NEXTB;
      ELSEIF I7 THEN [OUTC] JUMP NEXTC;
      ELSE [OUTD] JUMP NEXTD;

EFGH: IF I0 THEN [OUTE] JUMP NEXTE;
      ELSEIF I5 THEN [OUTF] JUMP NEXTF;
      ELSEIF I3 THEN [OUTG] JUMP NEXTG;
      ELSE [OUTH] JUMP NEXTH;
```

10

the multiway-branch block, the actual next instruction depends on the input values. If the counter is set to 1 and input condition **/I0*I5** is true, then the next command will be **JUMP NEXTF**.

More Than Four Product Terms per Condition

A design that runs out of product terms for a branch condition, or generates the error message **Predicate too long**, can still be fitted after some adjustments. For example, the first branch condition in the following ASMILE code contains five product terms, while the second condition contains only one:

```
START:  IF   I5*I0' +
             I5*I1' +
             I5*I2' +
             I5*I3' +
             I5*I4'          THEN S1
        IF   I2*I1*I2*I3*I4  THEN S0
        S2
```

It is possible to make a tradeoff between the number of branches and the number of product terms. The software automatically partitions the first branch, so that there are actually two branches to **S1**—one branch with four product terms and another with one product term:

```
START:  IF   I5*I0' +
             I5*I1' +
             I5*I2' +
             I5*I3'              THEN S1
             I5*I4'              THEN S1
        IF   I2*I1*I2*I3*I4      THEN S0
        S2
```

This approach reduces the number of possible branches to three. In cases where all four branches are needed, reordering the branches may reduce the number of product terms. Also, reordering may make better use of the built-in prioritization of the EPS448 architecture, since the second branch condition can assume that the first condition has failed. In the previous example, the first condition can be factored into **I5*/(I0*I1*I2*I3*I4)**, and the branch order can be rearranged so that the **S0** branch is considered first. Then the **S1** branch condition can assume that the expression **I0*I1*I2*I3*I4** has failed and need not be tested. The following example shows the previous code reordered, with one product term per condition (a fourth branch may easily be added):

```
START:  IF   I0*I1*I2*I3*I4  THEN S0
        IF   I5              THEN S1
        S2
```

Conclusion

The flexibility of the EPS448 architecture allows the designer to perform multiway branching in a number of ways. If more than four product terms exist per condition, they may be reduced through partitioning and reordering.

Vertical Cascading of EPS448 SAM EPLDs

October 1990, ver. 2

Application Brief 65

Introduction

When an application requires more states or more microcode depth than a single EPS448 EPLD can provide, i.e., more than 448 states or words, multiple EPLDs can be cascaded vertically to accommodate the additional states. This application brief describes how to vertically cascade EPS448 EPLDs by passing control from one EPLD to another. The method that best suits a specific application depends on how the sequencer is most easily partitioned. The following methods are examined:

- ❒ Simple vertical cascading
- ❒ Addressed-branch cascading
- ❒ Vertical subroutine calls
- ❒ Master/slave cascading

This application brief assumes an understanding of the EPS448 EPLD and a working knowledge of Altera's Assembly Language (ASM) syntax.

Simple Vertical Cascading

When EPS448 EPLDs are vertically cascaded, the outputs of all the devices are tied together, forming a tri-state output bus (see Figure 1). One EPLD at a time controls the output bus; all others are disabled by tri-state buffers. The controlling EPLD performs sequencing until it encounters a sequence found in another EPS448. Then it tri-states its outputs, and the other EPS448 takes control of the bus.

The simplest method of vertically cascading EPS448 EPLDs is to use one input pin to signal to a device that it must take control of the output bus (Figure 1). A control line to each EPS448 (e.g., **nGO1**) is pulled up through a 1 K-Ω resistor. When the active EPS448 passes control to another EPS448, it pulls down the appropriate **nGO** signal and jumps to an idle state where its outputs are tri-stated. This configuration allows conditional **JUMP** instructions between EPS448 EPLDs.

10

Figure 1. Vertical Cascading

When EPS448 EPLDs are vertically cascaded, the outputs are tied together. While one EPS448 has control of the output bus, the others are tri-state-disabled. Each EPS448 has one input (e.g., nG01) that enables it onto the bus.

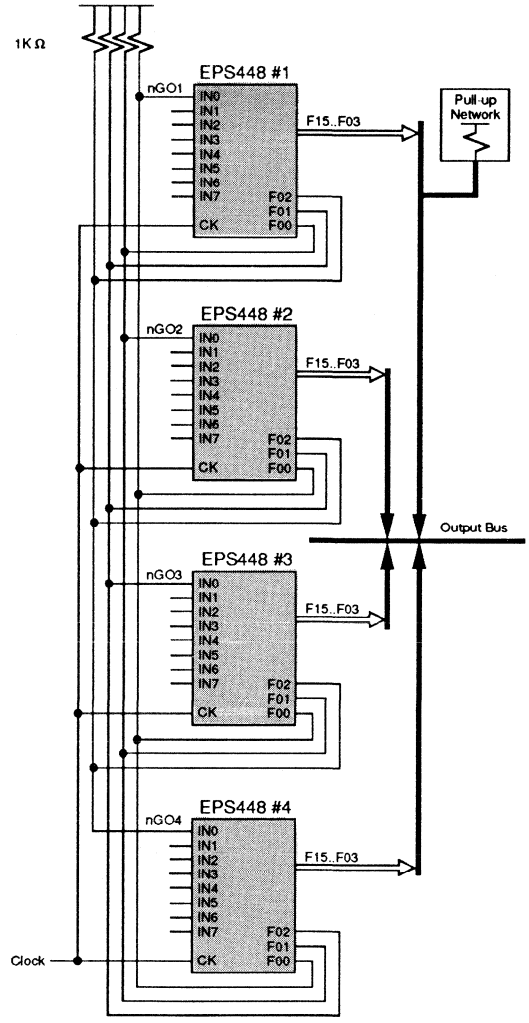
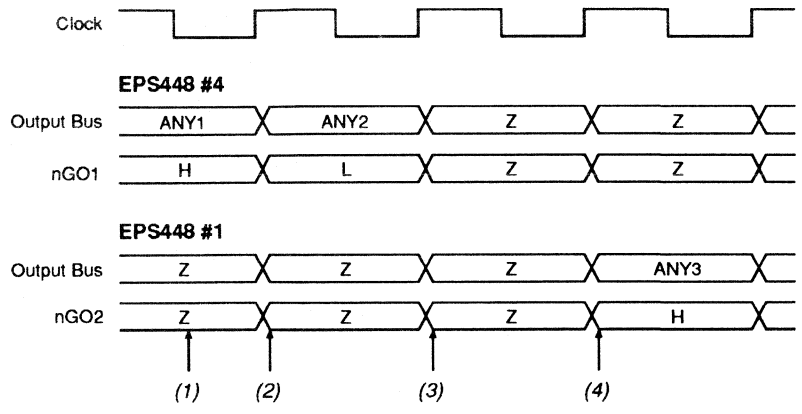


Figure 2 shows the timing associated with this simple vertical cascading. Initially, EPS448 #4 has control of the output bus and EPS448 #1 is idle. EPS448 #1 outputs are tri-stated during this period. To pass control, EPS448 #4 brings the **nGO1** signal low to activate EPS448 #1. On the next clock cycle—called the transition clock period—the outputs of both EPS448 #1 and EPS448 #4 tri-state. EPS448 #1 becomes active and takes control of the output bus. EPS448 #4 becomes idle on the following clock.

Both EPLDs are tri-state-disabled during the transition clock period. This idle period prevents potential glitches (i.e., bus contention with the other EPLDs) during transitions from high-impedance to a valid output, or vice

Figure 2. Passing Control between EPLDs

To pass control between two EPLDs (EPS448 #4 and EPS448 #1), EPS448 #4 starts active (1), then brings nG01 low (2) to pass control. On the next clock period, both EPLDs are disabled (3) before EPS448 #1 takes control of the output bus (4).

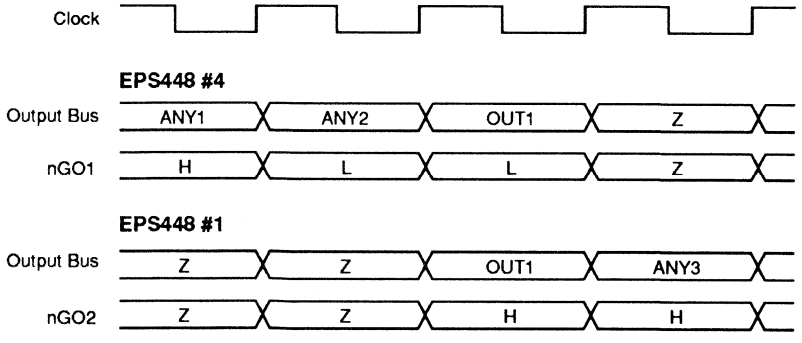


versa. During this clock period, the control lines (**nGO1** to **nGO4**) float unless they are pulled up by the resistors.

In another method, shown in Figure 3, both EPS448 EPLDs drive the same value (**OUT1**) on the output bus for the transition clock period. This method removes the need for pull-up resistors, but it may introduce temporary bus contention, possibly causing a current surge into the EPS448 EPLDs and inducing noise or ground bounce elsewhere on the board. However, bus contention does not impair EPLD operation.

Figure 3. Alternative to Passing Control

To avoid the clock cycle where the output bus is left floating, both EPLDs drive the same value onto the bus for a single clock cycle.



10

Each of the EPS448 EPLDs must have its own design file (with the extension **.ASM** or **.SMF**). A segment of the code used in EPS448 #1 is shown in Figure 4. While the machine is inactive, it remains in **IDLE** with the outputs disabled. When it receives an active-low **nGO1** signal from pin **1NO**, it jumps to the label **START**, takes control of the output bus, and holds **nGO2** to **nGO3** high, preventing the other EPLDs from competing for control of the output bus.

Figure 4. Assembly Language Code for Vertical Cascading

This ASM syntax is for EPS448 #1. This EPLD sits in the IDLE state until the nGO1 signal from pin IN0 goes low. It then jumps to START and takes control of the output bus. When EPS448 #1 is ready to give up control of the outputs, it jumps to QUIT and brings one of the three nGO lines low to activate the next machine. Finally, it jumps back to the IDLE state.

```
MACROS:  % F F F F F F                    F %
         % 0 1 2 3 4 5  .......          15 %
  HOLD = " 1 1 1 "
  GO2  = " 0 1 1 "
  GO3  = " 1 0 1 "
  GO4  = " 1 1 0 "
  OUT1 = " 1 0 1 ........... 0 "     % INSERT DESIRED    %
  ANY3 = " 1 0 0 ........... 1 "     % OUTPUT VALUES     %
  ANY2 = " 0 0 1 ........... 1 "     % FOR THESE STATES  %

PROGRAM:
IDLE:  IF /IN0 THEN [Z] JUMP START;
       ELSE [Z] JUMP IDLE;
START: [HOLD ANY3] CONTINUE;
       .
       .
       .
QUIT:  IF IN3 * IN4 THEN [GO2 ANY2] JUMP IDLE;
       ELSEIF   IN3 THEN [GO3 ANY2] JUMP IDLE;
       ELSE              [GO4 ANY2] JUMP IDLE;
```

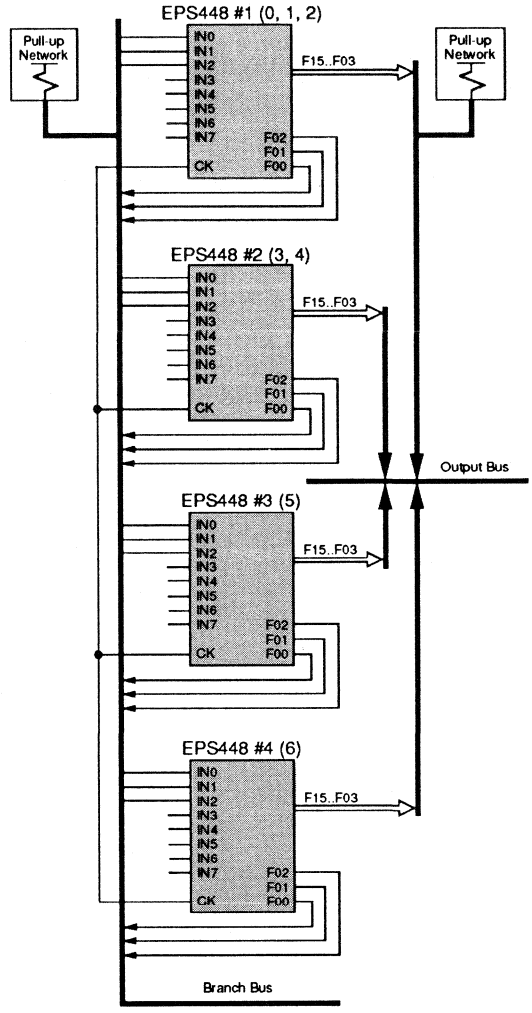
When EPS448 #1 is ready to pass control to another EPS448 EPLD, it jumps to **QUIT** and pulls the appropriate **nGO** control line low. For example, when **GO2** is the output, the EPS448 #1 **F00** output goes low, bringing the signal **nGO2** low and "waking up" EPS448 #2 on the next rising edge of the clock. In this example, **QUIT** is the label of a multiway branch instruction, so any one of the other three EPS448 EPLDs can be the next one activated. After activating the next machine, EPS448 #1 jumps to **IDLE** and waits to be given control again.

Addressed-Branch Cascading

With an increasing number of EPS448 EPLDs, an addressed branching configuration is more convenient (see Figure 5). A branch bus lies alongside the output bus. Each EPS448 examines the branch bus, and stays idle until an address assigned to it appears on the bus. The EPS448 then takes control of both buses. Since each EPS448 device has access to the branch bus, each EPS448 can call any one of the other EPS448 EPLDs.

Figure 5. Addressed-Branch Cascading

In addressed-branch cascading, each EPS448 EPLD wakes up when it sees one of its assigned addresses on the branch bus. Each EPLD can recognize more than one address. For example, EPS448 #1 takes control if it sees addresses 0, 1, or 2 on the branch bus.



A single EPS448 EPLD can have several addresses assigned to it, with each address corresponding to a different starting location in its microcode. Thus, a single EPLD can contain more than one block of independent states. In such a case, each address on the branch bus corresponds to a block of states within an EPLD.

The branch bus in Figure 5 consists of three output bits (**F00**, **F01**, and **F02**) that are passively pulled up by 1 K-Ω resistors and controlled by the active EPS448 EPLD. Values on the branch bus serve as branch addresses that access discrete blocks of states. EPS448 #1 has been assigned branch addresses 0, 1, and 2, each of which corresponds to a different sequence of states within the EPLD. While EPS448 #1 has control of the output bus, it

10

drives the branch bus with the address of its current sequence so that other machines are not activated. Only when it is ready to pass control, does it drive a new value onto the branch bus. For example, when EPS448 #1 outputs the branch address 3, EPS448 #2 assumes control.

Figure 6 shows a portion of the ASM code required for EPS448 #1. This code is similar to the code for simple cascading. While EPS448 #1 is idle, it must look for any of its three possible branch addresses (**BA0**, **BA1**, and **BA2**) to become valid. If one of these values appears on the branch bus, EPS448 #1 jumps to the start of the corresponding sequence (**START0**, **START1**, or **START2**).

Figure 6. Assembly Language Code for Addressed-Branch Cascading

The addressed cascading shown in Figure 5 uses three input lines (IN0 to IN2) to address different routines within a single EPLD. By specifying different addresses at the QUIT label, any routine in any other device can be called.

```
MACROS:  % F  F  F  F . . . . .  F  %
         % 0  1  2  3 . . . . .  15 %

   GO0 = "0  0  0"
   GO1 = "0  0  1"
   GO3 = "0  1  0"
   GO3 = "0  1  1"
   GO4 = "1  0  1"
   GO5 = "1  0  1"
   GO7 = "1  1  1"
  ANY2 = "1  . . . . . . 0"
  ANY3 = "0  . . . . . . 1"

EQUATIONS:                          % INPUT ADDRESS DECODES %
          BA0 = /IN2*/IN1*/IN0;     % BRANCH ADDRESS 0 %
          BA1 = /IN2*/IN1*IN0 ;     % BRANCH ADDRESS 1 %
          BA2 = /IN2*IN1*/IN0 ;     % BRANCH ADDRESS 2 %

PROGRAM:

IDLE:     IF BA0 THEN [Z] JUMP START0;
          ELSEIF BA1 THEN [Z] JUMP START1;
          ELSEIF BA2 THEN [Z] JUMP START2;
          ELSE [Z] JUMP IDLE;

START0:   [GO0 ANY3] CONTINUE;

START1:   [GO1 ANY3] CONTINUE;

START2:   [GO2 ANY3] CONTINUE;
          .
          .
          .
QUIT:     IF IN5 * IN6 THEN [GO3 ANY2] JUMP IDLE;
          ELSEIF  IN6  THEN [GO4 ANY2] JUMP IDLE;
          ELSEIF  IN7  THEN [GO5 ANY2] JUMP IDLE;
          ELSE  [GO7 ANY2]  JUMP IDLE;
```

When EPS448 #1 is ready to give up control, it jumps to **QUIT** until it calls the next address. Each branch calls a different routine by specifying a different value on the branch bus. (Note that the **G00** to **G07** macros correspond to binary branch addresses, i.e., **G03** = **011** .)

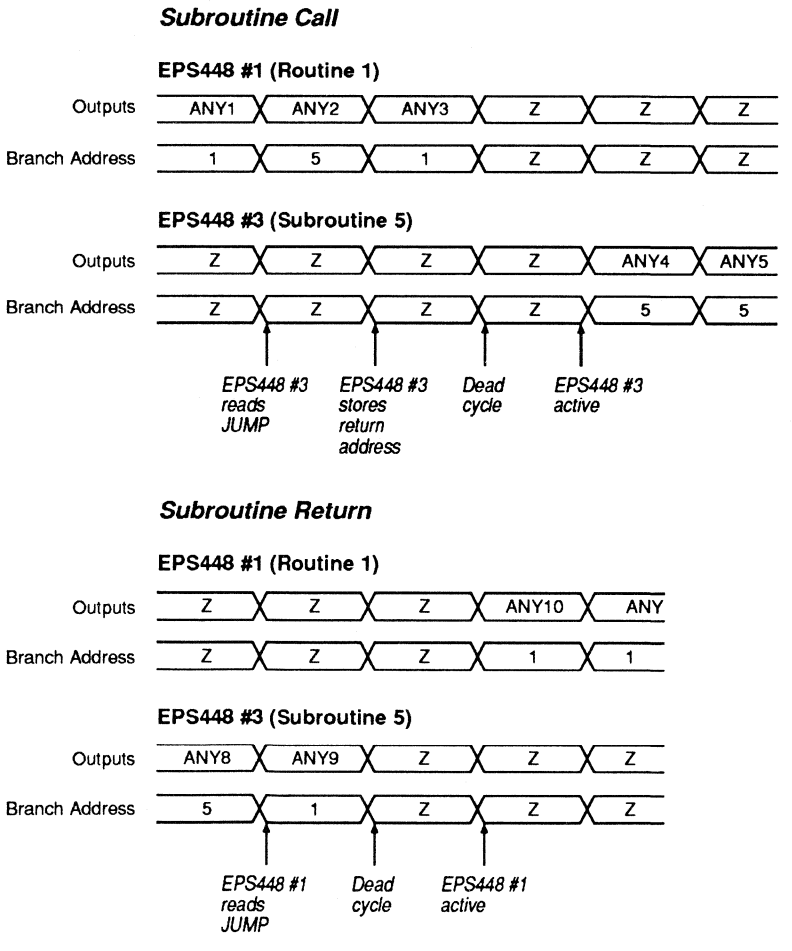
Vertical Subroutine Calls

The configurations discussed so far have used a **JUMP** function between EPS448 EPLDs. However, it is possible to perform subroutine calls between EPLDs. A subroutine call differs from a jump in that it must eventually jump back to the machine that called it. The subroutine must be able to recall the return address of the calling EPLD. The return address can be stored on the EPS448 internal stack. Subroutines are most easily implemented with the branch bus configuration (Figure 5).

The time required for passing control is illustrated in Figure 7. Before the subroutine call, EPS448 #1 is in control of the bus and is executing the block

Figure 7. Timing of a Vertically Cascaded Subroutine

In a sample cascaded subroutine call, EPS448 #1 calls subroutine 5 from EPS448 #3 by placing 5 on the branch bus. Next, EPS448 #1 places its own address (1) on the branch bus to tell the subroutine where to return. Finally, EPS448 #1 tri-states, and on the next clock period, EPS448 #3 takes control of the output bus. When EPS448 #3 (address 5) is ready to return, it puts the return value (1) on the branch bus and then idles.



10

of states starting with 1 (branch address 1). It drives the branch bus with the value 1 until it is ready to call a subroutine (branch address 5) within another EPLD (EPS448 #3). To start the subroutine call, EPS448 #1 specifies the address of subroutine 5 on the branch bus, which wakes up EPS448 #3. EPS448 #1 then applies return address 1 so that EPS448 #3 can store the return value on its internal stack. EPS448 #1 tri-states both the branch and output buses on the next clock cycle. A "dead" clock cycle follows, after which EPS448 #3 takes control of the output bus.

The syntax required for EPS448 #3 is shown in Figure 8. Note that after EPS448 #3 reads its address, it still leaves its outputs tri-stated for two more clock cycles. The **ANDPUSHI** opcode is used to store the return address in the stack by masking off inputs **IN3-IN7**. See *Application Brief 63 (Multiway Branching with the EPS448 SAM EPLD)* for information on how to use **ANDPUSHI** with dispatch routines.

Figure 8. Assembly Language Code for Vertically Cascaded Subroutine

In this example, EPS448 #3 is used as a subroutine at the branch address 5. After it is called, it stores the return address on the stack with the ANDPUSHI command. The RETURN command jumps to an absolute address, which puts the correct return address onto the branch bus.

```
MACROS:    % OUTPUT VALUES FOR BRANCH BUS %
           GO0 = "000"    GO1 = "001"    GO2 = "010"
           GO3 = "011"    GO4 = "100"    GO5 = "101"
           GO6 = "110"    GO7 = "111"

           % OUTPUT BUS VALUES %
           ANY3 = "10...1", ANY8 = "00...1", ANY9 = "01...0"

EQUATIONS:% BRANCH ADDRESS INPUTS %
           BA5 = IN2*/IN1*IN0;       % BRANCH ADDRESS 5%

PROGRAM:

% IDLE OR START THE SUBROUTINE %
IDLE:     IF BA5 THEN [Z] ANDPUSHI 7H GOTO START6;
          ELSE [Z] JUMP IDLE;

START5:   [Z] CONTINUE;                 % DEAD CYCLE %
          [GO5 ANY3] CONTINUE;          % TAKE CONROL OF %
                                        % OUTPUT BUS %
          . . .
QUIT5:    [GO5 ANY8] RETURN;            % SUBROUTINE RETURN %

% JUMP TABLE (RESERVED LOCATIONS) %
0D: [GO0 ANY9] JUMP IDLE;               % ALSO THE POWER-UP STATE %
1D: [GO1 ANY9] JUMP IDLE;
2D: [GO2 ANY9] JUMP IDLE;
3D: [GO3 ANY9] JUMP IDLE;
4D: [GO4 ANY9] JUMP IDLE;
5D: [GO5 ANY9] JUMP IDLE;
6D: [GO6 ANY9] JUMP IDLE;
7D: [GO7 ANY9] JUMP IDLE;               % ERROR CONDITION %
```

To return from the subroutine, EPS448 #3 applies the return address (branch address 1) on the branch bus for one clock cycle (see Figure 7). EPS448 #1 reads this address and takes control of the output bus by jumping to the address at the top-of-stack. The **RETURN** opcode at the **QUIT** label accomplishes this task. This address must be between 0 and 7 because the inputs **IN3** to **IN7** are masked off when the **ANDPUSHI** opcode pushes the return address onto the stack. In this example, the top-of-stack is 1, and a jump to address **1D** (1 decimal) is performed. Address **1D** has the correct output specification to signal EPS448 #1 to take control of the buses. Finally, EPS448 #3 jumps back to **IDLE**.

With this approach, branch address 7 is unusable because it corresponds to the default condition of all branch-address bits pulled up. If address **7D** is reached, an error has occurred and all cascaded EPS448 EPLDs must be reset.

During power-up, the EPLD starts in state **0D**. All machines jump to their idle state on the next clock cycle, and the EPS448 that initiates control after power-up jumps to its correct starting state.

After making the subroutine call, EPS448 #1 must go into a high-impedance state and wait for the routine to finish. It can take this action in one of two ways: it can return to its **IDLE** starting state; or it can stay in a separate loop. In the first case, EPS448 #1 regains control by recognizing its address, effectively restarting. In the second case, EPS448 #1 returns to the state it was in when the call was made.

Master/Slave Cascading

Another vertical cascading method is the master/slave configuration. See Figure 9. A single EPS448 (the master) controls the overall sequencing by calling routines within other EPS448 EPLDs (the slaves). The slaves are cascaded with all but one of the outputs connected to the output bus. The single output (**F00**), called **nDone** in Figure 9, is passively pulled up through the resistor. When an active slave EPLD has finished executing its routine, it pulls the **nDone** line low. This action alerts the master that it should jump to the next routine.

Any number of control lines can run from the master to a given slave depending on the number of routines within the slave. For n control lines the slave can have $2n-1$ routines. When all control lines are high, the slave is idle. In this example, Slave #1 has two inputs and thus can contain up to three routines. When both **CONTROL0** and **CONTROL1** are high, Slave #1 is idle. There is no limit to the number of slaves that can be added to this system. Furthermore, unlike the branch bus method, the number of inputs to the other EPLDs does not increase as more routines are added.

Figure 9. Master/Slave Cascading

In a master/slave configuration, one EPS448 EPLD (the master) controls sequencing. It calls routines from any number of slaves. The slaves signify that they are finished by pulling the nDONE line low.

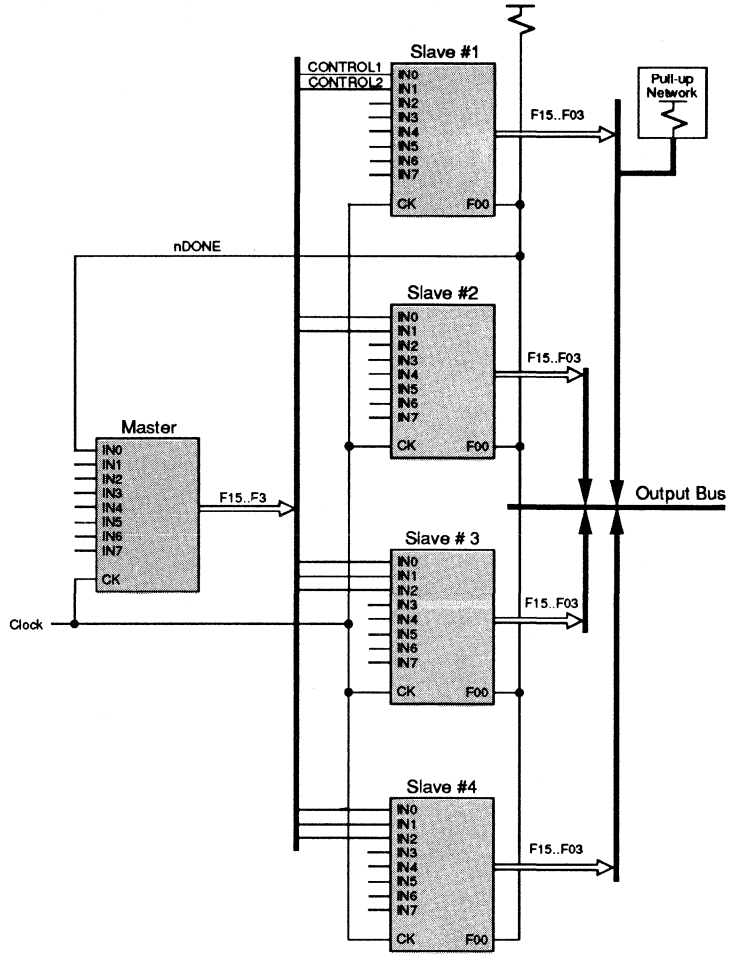
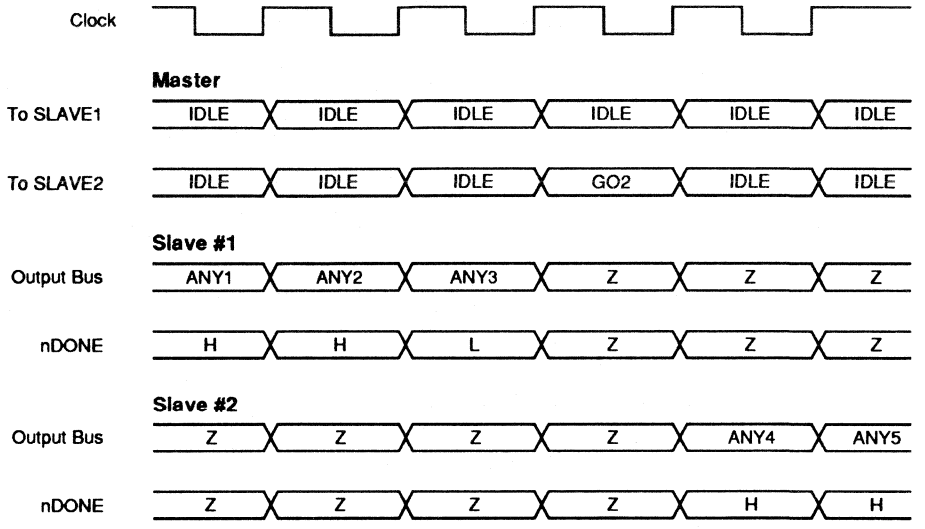


Figure 10 shows the timing associated with the master/slave configuration. Slave #1 originally has control of the output bus. It signifies that it is done by pulling **nDONE** low. The Master responds by initiating the Slave #2 machine, which then takes over the output bus. (The output bus is undefined for a single clock cycle.)

Figure 10. Timing of Master/Slave Cascading

In the example of master/slave cascading shown in Figure 9, Slave #1 starts with control of the output bus. It signifies it is done by bringing nDONE low. The Master then initiates Slave #2 by sending it an active starting address.



Final Tips on Vertical Cascading

- Each technique for vertically cascading EPS448 EPLDs consumes output and input pins from each device to accommodate control passing. If needed, additional outputs can be created by horizontally cascading EPLDs (see the ***SAM+PLUS Reference Guide*** for more information). Additional inputs can be generated by multiplexing several signals and reducing them to the eight inputs available with the EPS448 EPLD. See *Application Brief 66 (Input Reduction for the EPS448 SAM EPLD)* for more information on input reduction.

- Partitioning—i.e., the process of dividing the code or state segments among the multiple EPS448 EPLDs—is the most important step in creating a vertically cascaded sequencer. Most large sequential applications have natural divisions that indicate where partition borders should fall. The general goal in partitioning is to create blocks of sequences that fit within a single EPS448 EPLD, and to minimize control transfers from one EPLD to another.

- All Clock pins should be tied together, as should all Reset pins. Inputs to different EPS448 EPLDs need not be tied together. In fact, it may be desirable for different sections or routines of the overall sequencer to branch on the basis of different input signals.

- In vertical cascading, the output lines from the resulting sequencer originate at a tri-state output bus. The output bus signals should never be connected to system Clock or Latch Enable inputs. As in any tri-statable bus, the output bus of a cascaded EPS448 EPLD is not guaranteed to be glitch-free during transitions from high-impedance to output valid, or vice versa.

- Only one EPLD may become active on power-up. All other EPLDs should jump to **IDLE**, and therefore should have the following line in their source files: **ØD: [Z] JUMP IDLE**.

- If every EPLD goes **IDLE** without first calling another EPLD (or bringing **nDONE** low), the system is deadlocked in an idle state. This problem can be avoided with proper coding, or with a watchdog timer implemented in one of the EPS448 EPLDs. The watchdog timer counts to a final count value that corresponds to a fixed time period. During normal operation, the other EPS448s periodically clear the watchdog counter, preventing it from reaching its final count value. Therefore, if the watchdog reaches its final count value, a system error has occurred. The watchdog timer should then reset the system or generate the appropriate system error status.

Input Reduction
for the EPS448 SAM EPLD

October 1990, ver. 2

Application Brief 66

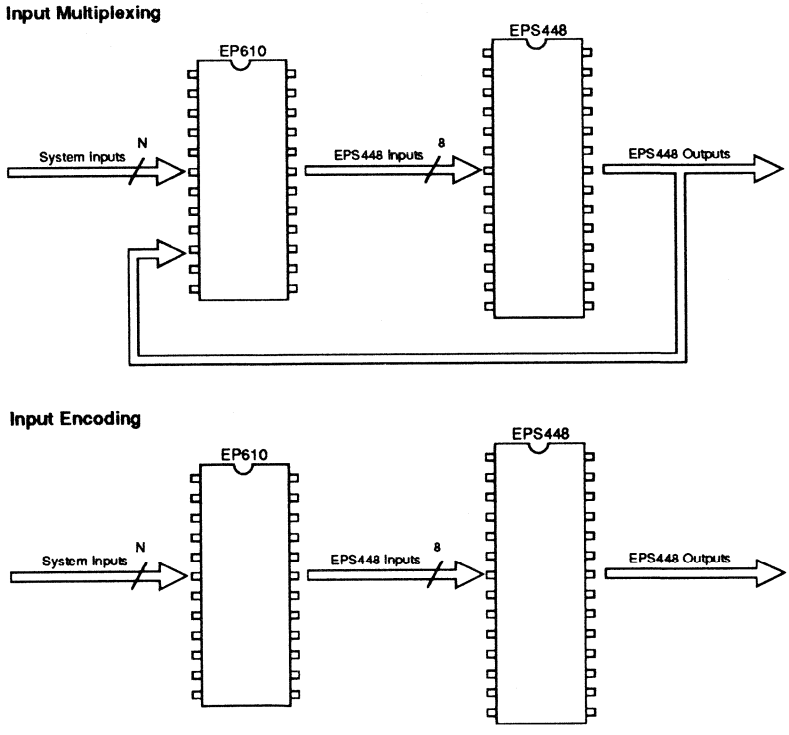
Introduction

The EPS448 SAM EPLD provides eight dedicated inputs. Applications that require more than eight inputs can use external logic to reduce the number of signals while retaining the same sequencing function. This application brief describes two techniques for reducing the number of inputs:

- ❒ Input multiplexing
- ❒ Input encoding

Both methods use an EP-series EPLD in front of the EPS448 device to reduce the number of inputs to a maximum of eight. Input multiplexing reduces the inputs based on the current state of the machine, while input encoding does not use the current state at all (see Figure 1).

Figure 1. Input Reduction Methods *Input count can be reduced through multiplexing and encoding.*



10

Input Multiplexing

Input multiplexing is the most flexible approach to input reduction. A multiplexer transforms the system signals into eight outputs that connect to the EPS448 inputs and are used for branch-control decisions. The EPS448 EPLD controls the multiplexer select lines based on the current state. Thus, each state selects the set of inputs required to make the subsequent branch.

Figure 2 shows a sample state machine. It has 12 inputs (A to L), but only 8 (or fewer) are needed for a single 4-way branch.

Figure 2. Input Multiplexing Example

This state machine requires 12 inputs, but never more than 8 for a single 4-way branch. Input multiplexing performs 12-to-8 multiplexing to achieve the input reduction.

```
S0: S1

S1: IF A */B THEN S1
    IF A */C THEN S2
    IF A */D THEN S3
    S4

S2: IF A * E * /F THEN S4
    IF B * F * /G THEN S1
    IF C * G * /H THEN S0
    S2

S3: IF H * I * J * K * L THEN S4
    IF A * H * I * J * K THEN S2
    IF A * B * H * I * J THEN S0
    S3

S4: IF C * B * /H * J THEN S0
    IF A * D * /I * K THEN S2
    IF A + C + H + I  THEN S3
```

Table 1 shows the inputs required by each of the branching states. This table maps each state against the system inputs required to determine the next state. Requirements are divided into three sets; each set contains fewer than eight inputs. One set at a time is routed to the input pins of the EPS448 EPLD.

Table 1. Inputs Required for Each State

Set	State	System Inputs											
		A	B	C	D	E	F	G	H	I	J	K	L
Set 1	S1	X	X	X	X								
Set 1	S2	X	X	X		X	X	X	X				
Set 2	S3	X	X						X	X	X	X	X
Set 3	S4	X	X	X	X				X	X	X		

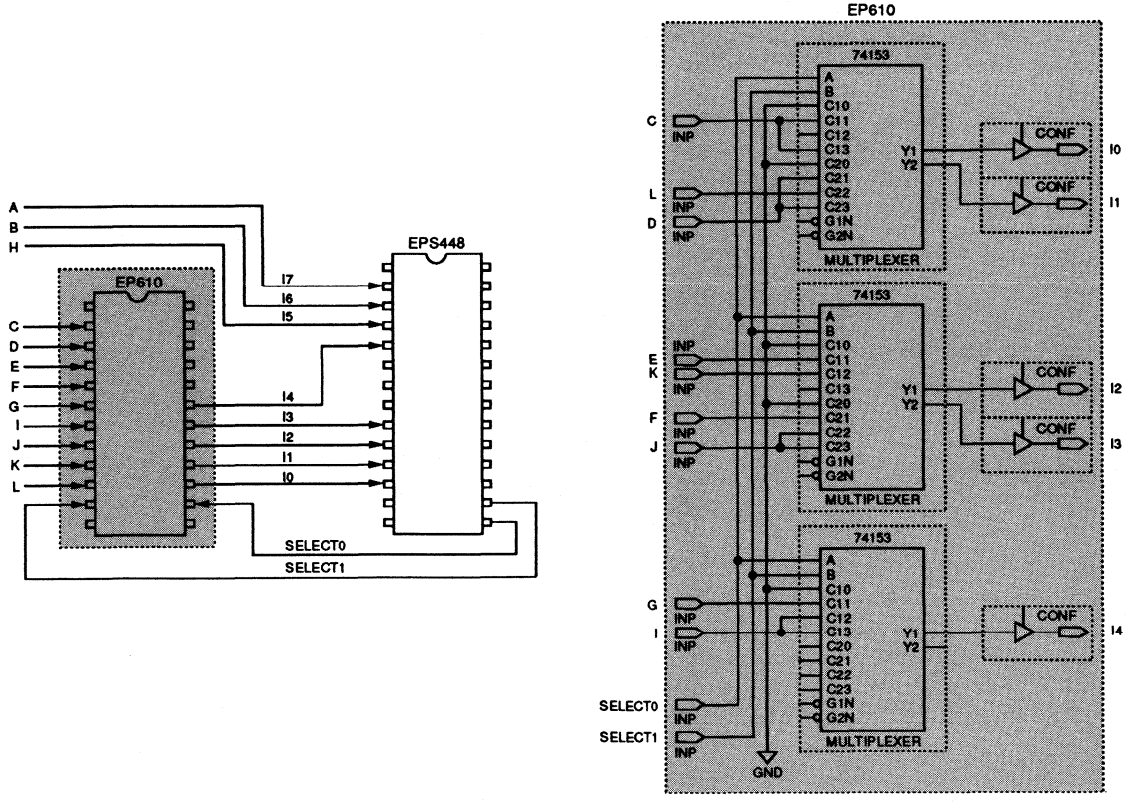
State **S1** requires four inputs to determine the next state. **S2**, **S3**, and **S4** each require seven inputs. **S1** and **S2** together still only require eight inputs (**A** to **H**). State **S0** does not require any inputs because it performs an unconditional branch.

The input requirements, shown in Table 1, can be divided into three sets. Set 1, required by states **S1** and **S2**, consists of inputs **A**, **B**, **C**, **D**, **E**, **F**, **G**, and **H**. Set 2, required by state **S3**, consists of inputs **A**, **B**, **H**, **I**, **J**, **K**, and **L**. Set 3, required by state **S3**, consists of inputs **A**, **B**, **C**, **D**, **H**, **I**, and **J**. The current state determines which set of inputs is routed to the EPS448's inputs.

Figure 3 illustrates input multiplexing. Since inputs **A**, **B**, and **H** are included in all three of the input sets, they run directly into the EPS448

Figure 3. Input Multiplexing

This input reduction method requires multiplexing the many system signals (A to L) down to the 8 input pins of the EPS448 EPLD (I0 to I7). Inputs A, B, and H run directly into the EPS448 device. An EP610 EPLD takes the remaining 9 inputs and multiplexes them down to I0 to I4. The multiplexers within the EP610 are entered with LogiCaps. SELECT0 and SELECT1, which select the inputs needed for the next transition, come from the EPS448 EPLD.



10

EPLD (**I5** to **I7**). The other nine inputs are routed through an EP610 EPLD and multiplexed down to the five remaining input pins (**I4** to **I0**). Two outputs from the EPS448 EPLD (**SELECT0** and **SELECT1**), which depend on the current state, feed the multiplexer select inputs on the EP610 and direct the correct set of system signals to the EPS448 inputs.

The EP610 multiplexing circuitry can be created with A+PLUS TTL macrofunctions and LogiCaps schematic capture software. The nine system signals (**C, D, E, F, G, I, J, K,** and **L**) enter from the left, and the five inputs for the EPS448 EPLD (**I0** to **I4**) leave on the right.

Table 2 shows the resulting truth table for the inputs to the EPS448 EPLD.

Table 2. Input Multiplexing Truth Table

SELECT1	SELECT0	I0	I1	I2	I3	I4	I5	I6	I7
0	0	0	0	0	0	0	H	B	A
0	1	C	D	E	F	G	H	B	A
1	0	0	L	K	J	I	H	B	A
1	1	C	D	0	J	I	H	B	A

The State Machine File (SMF) for the EPS448 EPLD (Figure 4) must contain references that map the 12 system signals to the appropriate EPS448 input pin. The Equations Section performs this mapping by making each system signal equal to a pin name. For example, **L** and **D** are both equal to input pin **I1**, because—depending on the current state—the multiplexer places one of these two signals at pin **I1**.

The select lines for the EP610 multiplexer, **SELECT0** and **SELECT1**, are defined as outputs in the **STATES** section, together with all of the general-purpose outputs. For example, **S3** has outputs **SELECT 1 = 1** and **SELECT0 = 0**, which corresponds to input set 2.

Input Encoding

Like input multiplexing, input encoding also uses an EP-series EPLD to map system signals to the eight input pins available with the EPS448 EPLD. Unlike the multiplexing method, however, input encoding completes the input reduction without using the current state of the machine.

Input encoding is most frequently used for address decoding to enable a sequencer to make a branch decision. This decision is based on the current value of an address. For example, the EPS448 EPLD may need to sit in an idle state until an eight-bit address reaches 0F hex. If all eight address bits were to go into the EPS448 device, no other input pins would be left for additional branch conditions. Therefore, it is more efficient to run the eight address bits into an EP610 device, which encodes the address down to a single bit that notifies the EPS448 EPLD when the address equals 0F hex. Only one EPS448 input is consumed, leaving seven inputs for other uses.

Figure 4. Modified EPS448 State Machine File

The modified SMF for the EPS448 EPLD includes an Equations Section that maps the system inputs to the actual input pin used. The SELECT0 and SELECT1 outputs cause the proper inputs to be routed to the pins for the next branch.

```
PART:     EPS448

INPUTS:   I0, I1, I2, I3, I4, A, B, H

OUTPUTS:
SELECT1, SELECT0, F02, F03, F04, F05, F06, F07, F08, F09, F10, F11, F12, F13, F14, F15
% The select lines go to the EP610 %

EQUATIONS:
% The Equations Section is used to map the system input pins %
% to the actual EPS448 input pins                           %

C = I0;
D = I1;
L = I1;
E = I2;
K = I2;
F = I3;
J = I3;
G = I4;
I = I4;

MACHINE:  EXAMPLE2

STATES:   [SELECT1 SELECT0 F02 F03 F04 F05 F06 F07 F08 F09 F10 F11 F12 F13 F14 F15]
S0        [   0       0     X   X   X   X   X   X   X   X   X   X   X   X   X   X ]
S1        [   0       1     X   X   X   X   X   X   X   X   X   X   X   X   X   X ]
S2        [   0       1     X   X   X   X   X   X   X   X   X   X   X   X   X   X ]
S3        [   1       0     X   X   X   X   X   X   X   X   X   X   X   X   X   X ]
S4        [   1       1     X   X   X   X   X   X   X   X   X   X   X   X   X   X ]

S0: S1

S1: IF A */B THEN S1
    IF A */C THEN S2
    IF A */D THEN S3
    S4

S2: IF A * E * /F THEN S4
    IF B * F * /G THEN S1
    IF C * G * /H THEN S0
    S2

S3: IF H * I * J * K * L THEN S4
    IF A * H * I * J * K THEN S2
    IF A * B * H * I * J THEN S0
    S3

S4: IF C * B * /H * J THEN S0
    IF A * D * /I * K THEN S2
    IF A + C + H + I  THEN S3
    S4

END$
```

Input encoding, however, can do much more than address decoding. Anytime a group of inputs represents a single piece of information—such as a board-level command, interrupt lines, or carry-out signals—input encoding should be considered. In particular, input encoding is advised if a group contains an illegal input combination, or an input combination with several "don't care" values.

Table 3 shows the truth table for a state machine that issues a series of commands to the EPS448 EPLD. The EPS448 executes a given sequence of states for each command. The state machine contains 19 commands, each of which is represented by a unique combination of 13 signals labeled **A** through **M**. The truth table shows the 13 signals and the corresponding commands. If **A** is 0, for example, then the command to the EPS448 EPLD is called **IDLE**, regardless of the other 12 input values.

Table 3. Input Truth Table

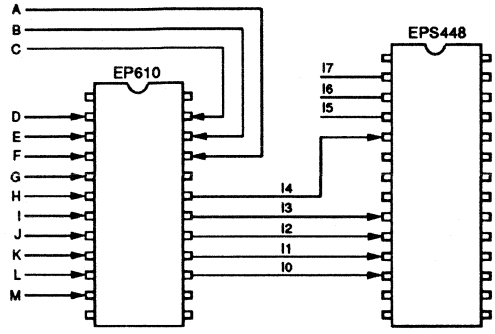
A	B	C	D	E	F	G	H	I	J	K	L	M	System Command	Row # (I4 to I0)
0	X	X	X	X	X	X	X	X	X	X	X	X	IDLE	0
1	0	X	X	X	X	X	X	X	X	X	X	X	ERROR	1
1	1	0	X	X	X	X	X	X	X	X	X	X	RESET	2
1	1	1	0	0	0	X	X	X	X	X	X	X	BACK1	3
1	1	1	0	0	0	1	X	X	X	X	X	X	RIGHT1	4
1	1	1	0	0	1	0	X	X	X	X	X	X	LEFT1	5
1	1	1	0	0	1	1	X	X	X	X	X	X	FORWD1	6
1	1	1	0	1	X	X	0	0	X	X	X	X	BACK2	7
1	1	1	0	1	X	X	0	1	X	X	X	X	RIGHT2	8
1	1	1	0	1	X	X	1	0	X	X	X	X	LEFT2	9
1	1	1	0	1	X	X	1	1	X	X	X	X	FORWD2	10
1	1	1	1	0	X	X	X	X	0	0	X	X	BACK3	11
1	1	1	1	0	X	X	X	X	0	1	X	X	RIGHT3	12
1	1	1	1	0	X	X	X	X	1	0	X	X	LEFT3	13
1	1	1	1	0	X	X	X	X	1	1	X	X	FORWD3	14
1	1	1	1	1	X	X	X	X	X	X	0	0	BACK4	15
1	1	1	1	1	X	X	X	X	X	X	0	1	RIGHT4	16
1	1	1	1	1	X	X	X	X	X	X	1	0	LEFT4	17
1	1	1	1	1	X	X	X	X	X	X	1	1	FORWD4	18

Since the truth table contains only 19 rows, 5 bits (**I0**, **I1**, **I2**, **I3**, and **I4**) can uniquely define the current command. These 5 bits are the only inputs that the EPS448 EPLD needs to make the branch decisions. With the resulting input encoding, 5 inputs—instead of 13 as shown in the table—are required, and the design easily fits into the EPS448 EPLD, with 3 inputs to spare.

Figure 5 shows how an EP610 accomplishes the input encoding. The 13 inputs (**A** to **M**) go into the EP610 and are converted to five outputs (**I0** to **I4**) that connect to the EPS448 EPLD.

Figure 5. Input Encoding

With input encoding, an EP610 EPLD compresses 13 system inputs (A to M) down to 5 inputs to the EPS448 EPLD (I0 to I4). Three input pins (I5 to I7) are still available for general-purpose use.



Altera's state machine entry language (provided with the A+PLUS documentation) is the most convenient language for entering the EP610 truth table. Figure 6 shows the SMF that describes the truth table. The Network Section indicates that the outputs (**I0** to **I4**) are combinatorial outputs with no feedback (**CONF**). The Truth Table Section of the file lists the 13 inputs (**A** to **M**) and the corresponding outputs (**I0** to **I4**) for each command. A+PLUS software translates the SMF into a standard JEDEC file for programming the EP610 EPLD.

To further simplify design entry, the Equations Section of the EPS448 design file can define each of the commands in terms of the inputs **I0** to **I4** as shown by the three sample equations here:

```
ERROR  = /I4 */I3 */I2 */I1 * I0;
BACK   = /I4 */I3 */I2 */I1 * I0;
FORWD1 = /I4 */I3 */I2 */I1 */I0;
```

With these equations, branching within the EPS448 device can then be expressed as follows:

```
S0: IF ERROR THEN S1
    IF BACK1 THEN S2
    IF FORWARD1 THEN S3
    S6
```

10

Figure 6. Truth Table Entry for the EP610

Truth table entry is easy: the 13 inputs (A to M) are listed on the left, and the 5 outputs (I0 to I4) are listed on the right.

```
PART: EP610

INPUTS:
A, B, C, D, E, F, G, H, I, J, K, L, M

OUTPUTS:
I0, I1, I2, I3, I4

NETWORK:

I0 = CONF(I0,)
I1 = CONF(I1,)
I2 = CONF(I2,)
I3 = CONF(I3,)
I4 = CONF(I4,)

T_TAB:

%            SYSTEM INPUTS                        OUTPUTS          COMMAND   %

A  B  C  D  E  F  G  H  I  J  K  L  M : I4 I3 I2 I1 I0 ;

0  X  X  X  X  X  X  X  X  X  X  X  X : 0  0  0  0  0  ;  %    IDLE     %
1  0  X  X  X  X  X  X  X  X  X  X  X : 0  0  0  0  0  ;  %    ERROR    %
1  1  0  X  X  X  X  X  X  X  X  X  X : 0  0  0  0  0  ;  %    RESET    %
1  1  1  0  0  0  0  X  X  X  X  X  X : 0  0  0  0  0  ;  %    BACK1    %
1  1  1  0  0  0  1  X  X  X  X  X  X : 0  0  0  0  0  ;  %    RIGHT1   %
1  1  1  0  0  1  0  X  X  X  X  X  X : 0  0  0  0  0  ;  %    LEFT1    %
1  1  1  0  0  1  1  X  X  X  X  X  X : 0  0  0  0  0  ;  %    FORWD1   %
1  1  1  0  1  X  X  0  0  X  X  X  X : 0  0  0  0  0  ;  %    BACK2    %
1  1  1  0  1  X  X  0  1  X  X  X  X : 0  0  0  0  0  ;  %    RIGHT2   %
1  1  1  0  1  X  X  1  0  X  X  X  X : 0  0  0  0  0  ;  %    LEFT2    %
1  1  1  0  1  X  X  1  1  X  X  X  X : 0  0  0  0  0  ;  %    FORWD2   %
1  1  1  1  0  X  X  X  X  0  0  X  X : 0  0  0  0  0  ;  %    BACK3    %
1  1  1  1  0  X  X  X  X  0  1  X  X : 0  0  0  0  0  ;  %    RIGHT3   %
1  1  1  1  0  X  X  X  X  1  0  X  X : 0  0  0  0  0  ;  %    LEFT3    %
1  1  1  1  0  X  X  X  X  1  1  X  X : 0  0  0  0  0  ;  %    FORWD3   %
1  1  1  1  1  X  X  X  X  X  X  0  0 : 0  0  0  0  0  ;  %    BACK4    %
1  1  1  1  1  X  X  X  X  X  X  0  1 : 0  0  0  0  0  ;  %    RIGHT4   %
1  1  1  1  1  X  X  X  X  X  X  1  0 : 0  0  0  0  0  ;  %    LEFT4    %
1  1  1  1  1  X  X  X  X  X  X  1  1 : 0  0  0  0  0  ;  %    FORWD4   %

END$
```

Final Tips

- Truth tables used for input encoding must not conflict with specified output values. Such conflicts occur when an input combination satisfies the input conditions for two rows in the table, but each specifies different output values. Such conflicts are usually introduced by injudicious use of don't care (**X**) values in truth table definitions.

  Table 3 shows an example in which **X** values create contradictory rows. If the input combination (**/A * B * C**) were applied to the truth table, it would satisfy the first row, which specifies that **OUT1** should go high, as well as the second row, which specifies that **OUT1** should go low. Because **OUT1** has conflicting values for this input combination, the truth table is invalid.

  Table 3. Invalid Truth Table

A	B	C	OUT1	OUT2
0	X	X	1	0
0	1	X	0	1
X	0	1	1	1

- If an invalid truth table is used in an input-encoded EPS448 design, undefined state transitions may occur. The SAM Design Processor treats **X** entries as true (for minimization purposes), and therefore does not check whether rows are free from output conflicts. Consequently, the designer must verify that the truth table is valid and contains only mutually exclusive input conditions. Otherwise, inadequately defined outputs may cause incorrect state branching.

- Input encoding is more suitable than input multiplexing for speed-critical applications because it does not require any feedback from the EPS448 EPLD. Input multiplexing, on the other hand, requires that outputs leave the EPS448 device, propagate through an EP-series EPLD, and still make the setup time of the EPS448 before the next clock edge.

- If inputs to an EPS448 SAM EPLD are asynchronous, they must be synchronized by placing registers in front of the inputs. Since input reduction typically places an EP-series EPLD in front of the EPS448, synchronization is easily accomplished by using registered rather than combinatorial outputs from the EPLD.

- Some applications may need to combine input encoding and input multiplexing. For example, an address decode signal can be used as one of the inputs to the multiplexers in Figure 5. When the proper state arrives, the decode signal is routed to the input of the EPS448 EPLD.

Notes:

A+PLUS Boolean Equation Design Entry

October 1990, ver. 2

Application Brief 71

Introduction

Boolean equations are a standard design entry method for low-density programmable logic. Altera's A+PLUS software offers a Boolean equation design entry method with simple syntax rules, standard logic operators, and added enhancements that take advantage of the advanced architectures of Altera EPLDs.

ADF Syntax

Boolean design files are created in the Altera Design File (ADF) syntax with an ASCII text editor (in non-document mode). The ADF is divided into sections, many of which are defined by a keyword. A sample ADF is shown in Figure 1.

Figure 1. Sample Altera Design File (ADF)

```
Header Section
Your Name                    % COMMENTS ARE ENCLOSED IN PERCENT SYMBOLS %
Your Company
10/1/90
1.00
B
EP610

Beverage Dispenser Controller

Options Section
OPTIONS:  TURBO = OFF, SECURITY = OFF

Part Section
PART:  EP610                 % TURBO & SECURITY DEFAULT %
                             % TO 'OFF'                 %

Inputs Section
INPUTS:  ENABLE@7, COINDROP, CUPFULL, CLOCK

Outputs Section
OUTPUTS:  DROPCUP@15, STROBE, POURDRNK

Network Section
NETWORK:                     % USER-ASSIGNED PIN NUMBER  %
ENABLE = INP (ENABLE)        % CLOCK, CLEAR, AND OUTPUT  %
                             % ENABLE OF ALL FLIP-FLOPS  %
COINDROP = INP (COINDROP)    % CAN BE CONTROLLED WITH    %
CUPFULL = INP (CUPFULL)      % BOOLEAN-DERIVED SIGNALS.   %
CLOCK = INP (CLOCK)

DROPCUP, DROPCUP = RORF (DROPCUPd, CLOCK, NEWCYCLE, GND, ENABLE)

POURDRNK, POURDRNK = RORF (POURDRKd, CLOCK, NEWCYCLE, GND, ENABLE)

STROBE = CONF (STROBEc)

Equations Section
EQUATIONS:                   % A KEYWORD MUST BE ENTERED %
                             % AT THE BEGINNING OF THE   %
DROPCUPd = COINDROP * /DROPCUP * CUPFULL;   % THE LINE   %

CUPREADY = DROPCUP * /POURDRNK;   % INTERMEDIATE EQUATION %
                                  % OPTION                %
POURDRKd = CUPREADY + /CUPFULL * TEMP;

STROBEc  = CUPREADY + TEMP;
NEWCYCLE = DROPCUP * POURDRNK;

TEMP = /DROPCUP * POURDRNK;

End Statement
END$
```

ADFs can be processed individually or combined with state machine, truth table, schematic, and netlist files. The Altera Design Processor (ADP) checks the syntax, performs logic minimization, fits the design into an appropriate EPLD, and produces a JEDEC file for device programming.

Header Section

The optional Header Section of the ADF includes all "bookkeeping" information, such as design title, revision number, designer, and any other desired information. It has no effect on design processing. This section has no keyword.

Options Section

The Options Section (keyword **OPTIONS:**) controls the Turbo Bit and Security Bit.

Part Section

The Part Section (keyword **PART:**) specifies the target EPLD. If the **AUTO** option is used, A+PLUS automatically selects the best EPLD for the design. If pin assignments are specified in the ADF, the **AUTO** option is not available, since the assigned pins must necessarily correspond to a particular EPLD and package.

Inputs Section

The Inputs Section (keyword **INPUTS:**) specifies all EPLD inputs for the design. Pin numbers can be assigned by appending an "at" symbol (**@**) plus the pin number to the end of the pin name.

Outputs Section

The Outputs Section (keyword **OUTPUTS:**) declares all output and bidirectional pins used in the design. Output pin numbers are assigned in the same way as input pins. (Bidirectional pins are never declared in the Inputs Section.)

Network Section

The Network Section (keyword **NETWORK:**) defines the programmable I/O architecture of the EPLD and provides access to all advanced I/O architecture features (e.g., programmable flip-flops).

Equations Section

The Equations Section (keyword **EQUATIONS:**) contains the Boolean equations, which use standard operators (AND = * or &; OR = + or #; NOT = /, ', or !) to implement the desired function. Equations need not be in sum-of-products form; parentheses are used to indicate grouping and establish precedence. Intermediate equations are also supported to simplify design entry.

End Statement

All Altera Design Files must end with the **END$** statement.

Additional Features

The ADF format can implement dual I/O feedback and active-low inputs and outputs.

Dual Feedback

The dual-feedback capability of the EP1800-series global macrocells is implemented by declaring the buried logic in the Outputs Section, and the dedicated input in the Inputs Section. The following example shows a buried register and input pin implemented with the same macrocell:

```
PART:   EP1810
INPUTS:   DUAL1@10
OUTPUTS:   DUAL2@10
NETWORK:
   DUAL1 = INP(DUAL1)
   DUAL2 = NORF(DUAL2d,CLK,CLR,GND)
           .
           .
           .
EQUATIONS:
   DUAL2d = A * B + C;
           .
           .
           .
END$
```

If Output Enable control is required, the following syntax is used:

```
PART:   EP1810
INPUTS:   DUAL1
OUTPUTS:   DUAL1
NETWORK:
   DUAL1 = INP(DUAL1)
   DUAL1, DUAL1f = RORF(DUAL1d,CLK,CLR,GND,OE)
           .
           .
           .
EQUATIONS:
   DUAL1d = A * B + C;
   OE     = D * E;
           .
           .
           .
END$
```

10

Active-Low Inputs

Active-low inputs are specified by inverting the signal in either the Network or Equations section.

❒ Active-low input specified in the Network Section:

```
INPUTS: /X
% The "/" character has no logical meaning in the %
% INPUT section %
NETWORK:
   Xn = INP(/X)
   X  = NOT(Xn)
% X is the internal active-low signal %
```

❒ Active-low input specified in the Equations Section:

```
INPUTS: /Y
NETWORK:
   Yn = INP(/Y)
EQUATIONS:
   Y = /Yn;
% Y is true when input signal /Y is low %
```

Active-Low Outputs

Active-low outputs are specified by inverting the left-hand side of the equation. If active-low outputs are required by a sequential function, the feedback nodes must also be inverted. For example:

```
PART:  EP630
INPUTS:  CLOCK
OUTPUTS:  /QC, /QB, /QA, /RCO
% The "/" character has no logical meaning in the %
% OUTPUT section %
NETWORK:
   CK          = INP(CLOCK)
   /QA, QAf    = RORF(QAd,CK,,,)
   /QB, QBf    = RORF(QBd,CK,,,)
   /QC, QCf    = RORF(QCd,CK,,,)
   /RCO        = CONF(RCOc,)
% Unspecified arguments in "RORF(.....)" assume %
% default values %
EQUATIONS:
   QA = /QAf;      % Invert feedback nodes %
   QB = /QBf;
   QC = /QCf;
     RCOc' = QC * QB * QA;
% Invert left-hand side of equation %
   QCd' = QC * QA' + QC * QB' + QC'* QA * QB;
   QBd' = QB * QA' + QB'* QA;
   QAd' = QA';
END$
```

EPM5000-Series MAX EPLD Timing

October 1990, ver. 2

Application Brief 75

Introduction

The advanced technology of Altera's Multiple Array MatriX Erasable Programmable Logic Devices (MAX EPLDs) and the sophisticated MAX+PLUS Development System have made CMOS programmable logic much easier to use. Complex logic designs can be implemented with a single EPM5000-series EPLD. In addition, the MAX+PLUS Simulator and delay prediction feature can identify critical delay paths and find each path's worst-case delays, which are the sum of discrete component delays inherent in the EPM5000-series architecture.

This application brief discusses these internal delay paths, their relationships to AC specifications (shown in the MAX EPLD data sheets), and the calculated timing delays generated by MAX+PLUS. In addition, timing models for analyzing delays calculated by the MAX+PLUS, and equations that are used to calculate the delays are discussed.

MAX EPLD Architecture Basics

To accurately model timing characteristics, a designer must understand how logic is implemented in a MAX EPLD. EPM5000-series architecture is based on a flexible Logic Array Block (LAB), which consists of three parts: the macrocell array, the expander product-term (expander) array, and the I/O control block. The number of macrocells, expanders, and I/O control blocks varies, depending on the device used. Large EPM5000-series EPLDs contain multiple LABs that are interconnected by a Programmable Interconnect Array (PIA). The PIA is fed by macrocell and I/O pin feedback, and globally routes signals within devices containing more than one LAB.

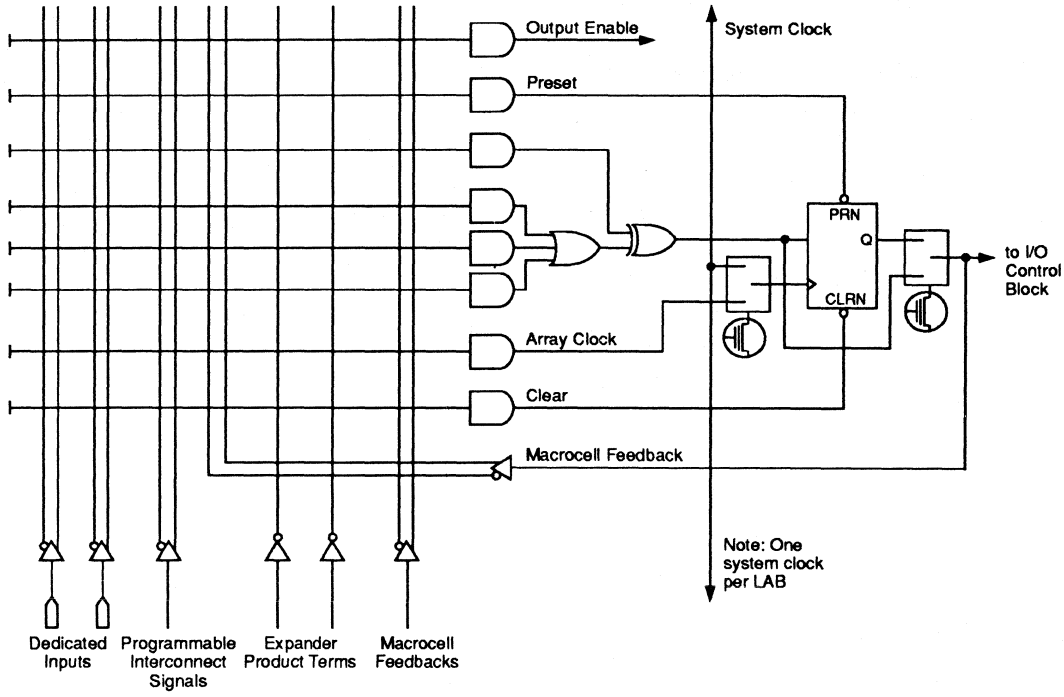
All gated and registered logic is implemented in macrocells and expanders. The expander product-term array and the macrocells contain product terms that are *n*-input AND gates (where *n* represents the number of connections). Depending on the implemented logic, a product term may be logically equivalent to one or more gates. (Refer to the *EPM5016 to EPM5192: High-Speed, High-Density MAX EPLDs Data Sheet* in this data book for more information.)

Macrocell Array

The macrocell structure of EPM5000-series EPLDs, shown in Figure 1, has been optimized to handle variable product-term requirements. Each macrocell consists of a programmable-AND/fixed-OR array, an XOR gate, a configurable register, and a number of inputs (varying from 80 to 152). Together, they allow EPM5000-series EPLDs to integrate complex logic.

Combinatorial logic is implemented in the macrocell with three product terms ORed together that feed the XOR gate. The second input to the XOR

Figure 1. Macrocell Array



gate is also controlled by a product term, providing the ability to control active-high or active-low logic. MAX+PLUS uses this gate to implement complex, mutually exclusive-OR logic, or to apply De Morgan's inversion, reducing the number of product terms required to implement a function. The macrocell also includes additional product terms (secondary product terms) that are used for Output Enable, Preset, Clear, and Clock logic. If more product terms are required to implement a function, they can be added to the macrocell from the expander product-term array.

The macrocell's AND array is an EPROM array; each product term generated by the AND array is a function of the EPROM bits within the array. The EPROM bits, initially erased, serve as electrical switches for the array inputs. An erased bit enables an input to reach a product term, while a programmed bit prohibits an input from reaching a product term. A product term is thus a function of inputs connected by unprogrammed, non-erased EPROM bits.

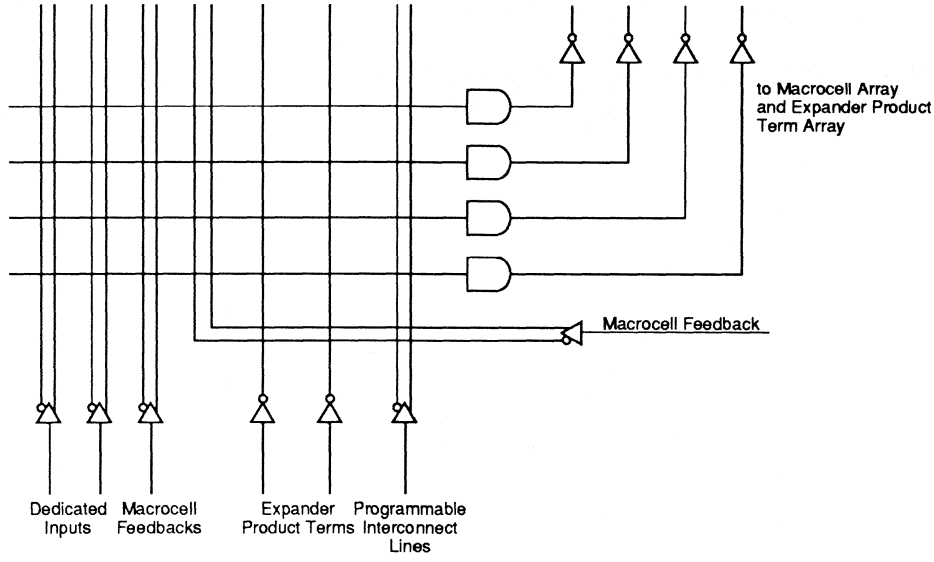
The configurable register within a macrocell can be programmed for D, T, JK, SR, or flow-through operation with independent, programmable Clock, Reset, and Preset options. It can also be bypassed entirely for purely combinatorial logic.

Expander Product-Term Array

Additional product terms, called expanders, may feed the macrocell array's product terms to generate very complex Boolean logic. The expander product-term array, shown in Figure 2, is a programmable AND array with inversion. Expanders are fed by the dedicated input bus, the PIA, the macrocell feedback, expanders themselves, and the I/O pin feedbacks. The outputs of the expanders then go to each product term in the macrocell array. Since these expanders also feed the secondary product terms of each macrocell (Preset, Clear, Clock, and Output Enable), complex register-control functions can be implemented without using another macrocell.

Figure 2. Expander Product-Term Array

Expander product terms are unallocated resources that can be used for registered or combinatorial logic.



These expanders are used and shared by the macrocells, allowing complex functions to be easily implemented in a single macrocell. Expanders may also feed other expanders, so that complex multi-level logic and input latches can be implemented. As illustrated in Figure 2, the expander product-term array contains an AND array followed by inversion. It may supply from 32 to 64 additional product terms per LAB. Large MAX EPLDs (EPM5192, EPM5130, EPM5128, and EPM5064) provide up to 32 expanders per LAB; smaller MAX EPLDs (EPM5032 and EPM5016) provide 64 and 32 expanders, respectively. Inputs into the expander product-term array also vary from 144 to 80.

I/O Control Block

In the LAB, the I/O control block is separate from the macrocell array. It contains programmable tri-state buffers and I/O pins with optional feedbacks. Each I/O pin can be configured for dedicated input, dedicated output, or bidirectional operation. The input of the tri-state buffer comes from a macrocell within the associated LAB. The feedback path from the I/O pin can feed other macrocells within the LAB, as well as the PIA.

Programmable Interconnect Array

The EPM5192, EPM5130, EPM5128, and EPM5064 MAX EPLDs have multiple LABs connected by a Programmable Interconnect Array (PIA) that globally routes all signals. The PIA avoids interconnect limitations by routing only the signals needed by each LAB, effectively solving any routing problems that may arise in a design. In addition, the PIA has a fixed delay from point to point. The PIA delay is constant because each signal that feeds into the PIA has its own dedicated metal line with multiple taps, one for each LAB. Undesired skews between logic signals, which may cause glitches in internal or external logic, are eliminated.

EPLD Delay Parameters

Internal delays within an EPLD are described by a number of AC parameters (called microparameters) that refer to the actual internal delay within the device. Figure 3 shows the timing model for single-LAB devices.

Figure 3. EPM5032 / EPM5016 Timing Model

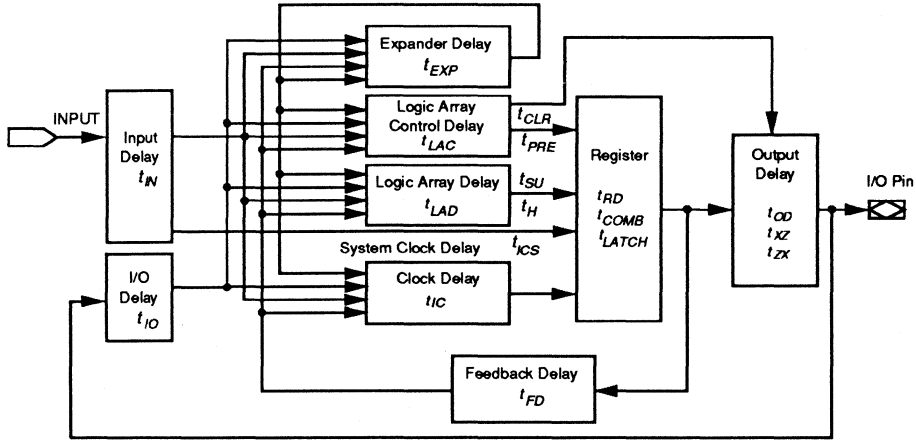
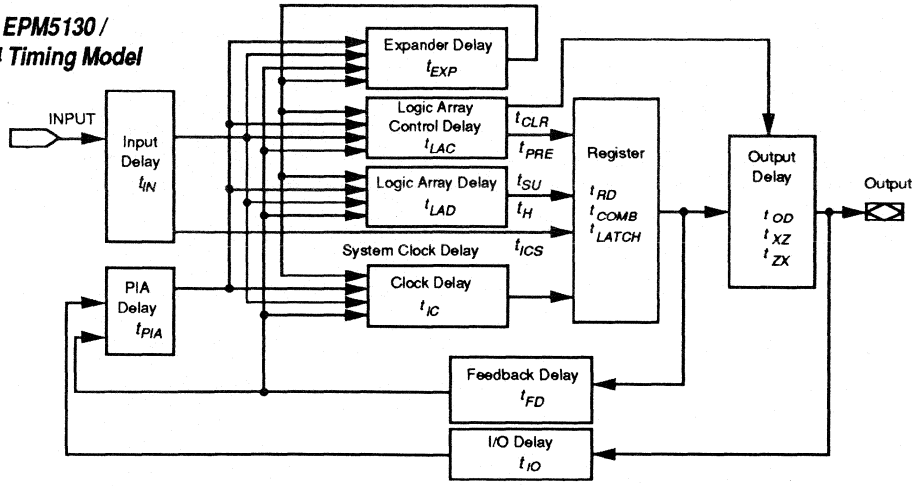


Figure 4 shows the timing model for multiple-LAB devices. Following is a list of these microparameters and examples of how to predict timing delays with equations that use them.

Figure 4. EPM5192 / EPM5130 / EPM5128 / EPM5064 Timing Model



t_{IN} Input pad and buffer delay. This delay directs the true and complement input signals from the dedicated input pin into the LAB. Within the LAB, the signals may propagate to any of four arrays: expander product-term array, logic array, logic array control, and clock array.

t_{IO} I/O input pad and buffer delay for I/O pins used as inputs. The t_{IO} delay value must be substituted for t_{IN} for the EPM5032 and EPM5016. When an I/O pin is used as an input, the t_{IO} delay value must also be added to t_{PIA} to obtain the total delay from the I/O pin to the LAB for the EPM5192, EPM5130, EPM5128, and EPM5064.

t_{EXP} Expander product-term array delay. This is the delay through the AND-NOT structure of the expander product-term array. It is added to the delay already present in the four arrays when expanders are used, or added to itself when an expander feeds another expander.

t_{LAC} Logic array control delay. This is the delay through the AND array by Clear, Preset, and Output Enable signals, representing the time required to propagate through the AND array to the **CLRN** and **PRN** inputs to the register, and the **OE** signal to the tri-state buffer.

t_{CLR} Asynchronous register clear time, which represents the time required to reset a register output to a logical low. It is the time the register **CLRN** input is asserted low to the time the register output stabilizes at logical low.

t_{PRE} Asynchronous register preset time. This delay represents the amount of time required to set a register output to a logical high. It is the time the register **PRN** input is asserted low to the time the register output stabilizes at logical high.

10

t_{LAD} Logic array delay. This is the time a signal requires to propagate through a macrocell's EPROM AND array, the three-input OR gate, and the two-input XOR gate.

t_{ICS} System clock delay. This is the delay from the dedicated clock pin to a register's clock input.

t_{IC} Clock delay. This is the delay through a macrocell's clock product term to the register clock input.

t_{FD} Feedback delay. For the EPM5032 and EPM5016, this delay is the propagation time from macrocell output to any of the LAB's four arrays. For the EPM5192, EPM5130, EPM5128, and EPM5064, it is the propagation time from a macrocell output to any of the LAB's arrays, or the propagation time from a macrocell output to a PIA input or other macrocells in the LAB.

t_{SU} Setup time required for a signal to be stable at the register input before the clock's rising edge.

t_{H} Hold time required at the register input after the register clock's rising edge to ensure that the register stores the input data.

t_{RD} Delay from the register clock's rising edge to the time that output appears at the register output.

t_{COMB} Combinatorial buffer delay, which is used only for combinatorial logic. This is the delay from the time the logic array's XOR output bypasses the programmable register to the time it becomes available for the macrocell output.

t_{LATCH} Propagation delay through the latch from latch input to output.

t_{OD} Output pad and buffer propagation delay from the macrocell output through the tri-state output buffer to the output pin.

t_{XZ} Delay required for high impedance to appear at the output pin after the output buffer's active-high enable control is brought logically low.

t_{ZX} Delay required for the macrocell output to appear at the output pin after the output buffer's active-high enable control is brought logically high.

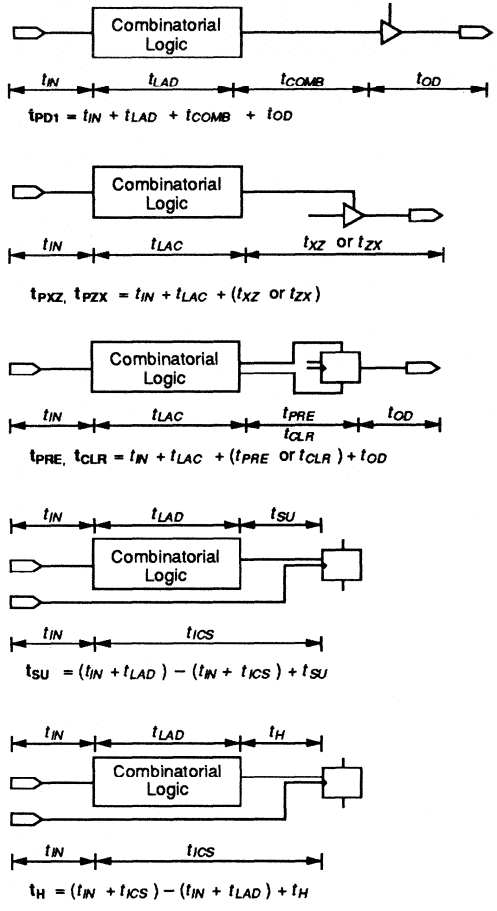
t_{PIA} Programmable Interconnect Array delay for multiple-LAB devices. This delay is used for EPM5192, EPM5130, EPM5128, and EPM5064 designs that use the PIA for routing. The PIA delay path starts where the macrocell feedback or I/O delay path ends, and ends where it enters the LAB and reaches any of its four arrays.

Critical pin-to-pin delays (denoted by a boldface **t**) are shown in Figure 5.

Figure 5. Critical Pin-to-Pin Delays

Notes:

1. *If an I/O pin is used for an input:*
   a. *for EPM5032 and EPM5016, substitute t_{IO} for t_{IN}.*
   b. *for EPM5192, EPM5130, EPM5128, and EPM5064, substitute $t_{IO} + t_{PIA}$ for t_{IN}.*
2. *If expanders are used for complex logic, add t_{EXP} to the delay path.*



Critical pin-to-pin delay calculations are shown in Figure 6. The calculations used to derive these values from the microparameters listed in the MAX EPLD data sheets are shown for each path. These calculations assume that a dedicated input pin is used.

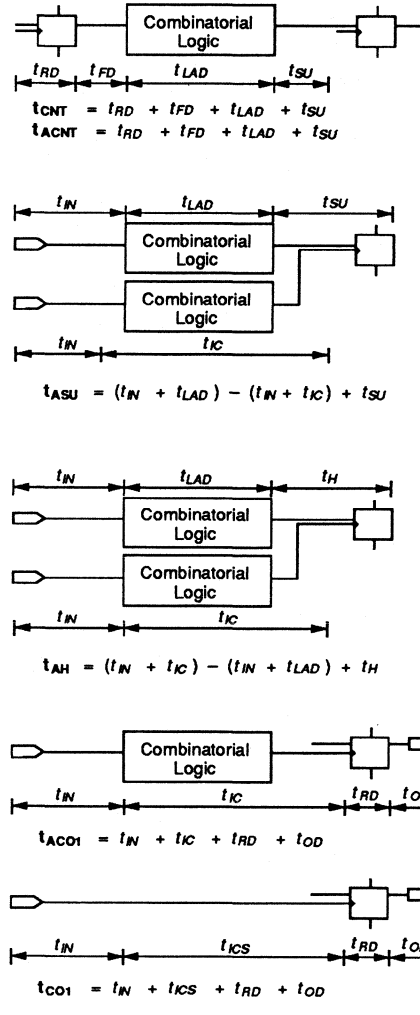
If the input comes from an I/O pin, t_{IO} is substituted for the t_{IN} value. For multiple-LAB EPLDs that use an I/O pin as an input, $t_{IO} + t_{PIA}$ is substituted for the t_{IN} value. If an expander is used in the path at any time, the t_{EXP} value must also be added to the total delay path.

10

Figure 6. Critical Pin-Delay Calculations

Notes:

1. *If an I/O is used for an input:*
   a. *for EPM5032 and EPM5016, substitute t_{IO} for t_{IN}.*
   b. *for EPM5192, EPM5130, EPM5128, and EPM5064, substitute $t_{IO} + t_{PIA}$ for t_{IN}.*
2. *If expanders are used for complex logic, add t_{EXP} to the delay path.*



Examples

The MAX+PLUS Simulator and the delay prediction commands in the MAX+PLUS Graphic Editor and Text Editor can identify timing delays for any circuit. These delays may be separated into the microparameters already described and are provided in the MAX EPLD data sheets.

Example 1: 4-to-1 Multiplexer

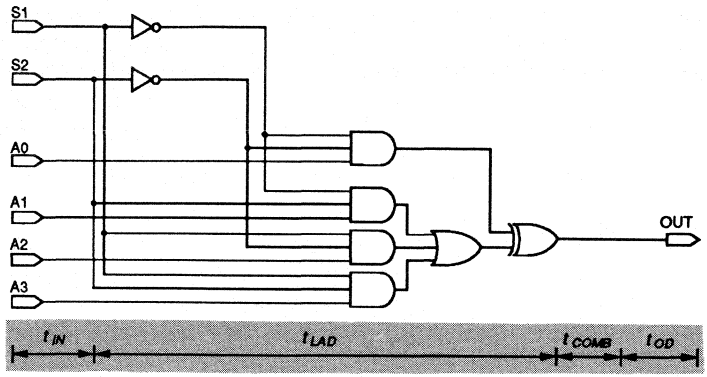
The design shown in Figure 7 represents a 4-to-1 multiplexer, which is a combinatorial circuit. The circuit consists of four data inputs, two select controls, and one output. It is partitioned into four delay paths: input delay, logic array delay, combinatorial buffer delay, and output delay.

Figure 7. 4-1 MUX Logic Timing

This circuit can be partitioned into 4 delay paths: input delay, logic array delay, combinatorial buffer delay, and output delay.

Input delay will be t_{IN} if only dedicated inputs are used. If I/O pins are used, the input delay will be t_{IO} for the EPM5032 and EPM5016, and $t_{IO} + t_{PIA}$ for the EPM5192, EPM5130, EPM5128, and EPM5064.

The propagation delay for combinatorial logic that bypasses the programmable register is t_{COMB}. For output data, the output pad and buffer delay is t_{OD}. The overall propagation delay is t_{PD1} for all dedicated inputs and t_{PD2} when any I/O pin is used for input.



The propagation delay from input pin to I/O pin is the sum of the input delay, logic array delay, combinatorial buffer delay, and output delays: $t_{IN} + t_{LAD} + t_{COMB} + t_{OD}$. This worst-case delay is also known as t_{PD1} (from input pin to output pin) or as t_{PD2} (from I/O pin to output pin). The maximum t_{PD1} and t_{PD2} values are shown in the individual MAX EPLD data sheets, or may be determined with the following equations.

For EPM5000-series EPLDs without a PIA:

$$t_{PD1} = t_{IN} + t_{LAD} + t_{COMB} + t_{OD}$$
$$t_{PD2} = t_{IO} + t_{LAD} + t_{COMB} + t_{OD}$$

For EPM5000-series EPLDs with a PIA:

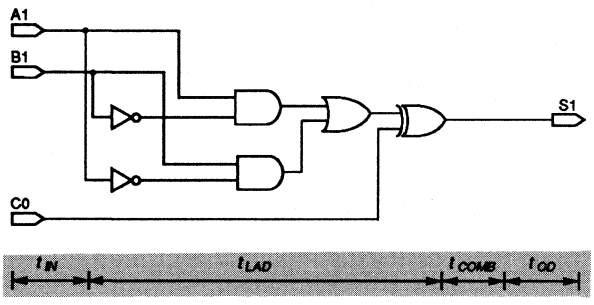
$$t_{PD2} = t_{IO} + t_{PIA} + t_{LAD} + t_{COMB} + t_{OD}$$

Example 2: 7483 TTL Macrofunction

The timing delays for macrofunctions subjected to logic synthesis can also be analyzed. The synthesized logic equations can be obtained from the "long" version of the Report File and have been structured so that a designer can quickly determine the logic configuration. (Refer to *Report File* in the ***MAX+PLUS User Guide***.) For example, the equations for **S1**, the least significant bit of the **7483** TTL macrofunction (a 4-bit full adder), are:

```
S1      =   OUTPUT (_MC021 , VCC);
_MC021  =   MCELL (_EQ026 $ C0);
_EQ026  =   B1 & A1'
        #   B1' & A1 ;
```

Figure 8. Adder Logic Timing



where **S1** is the output of macrocell 21 (**_MC021**), which contains combinatorial logic. The combinatorial logic, **MCELL(_EQ026 $ C0)**, represents the **XOR** of the intermediate equation (**_EQ026**) and the carry-in (**C0**). In turn, **_EQ026** represents logic equivalent to the **XOR** of inputs, **B1** and **A1**. See Figure 8.

Therefore, the timing delay for **S1** is $t_{IN} + t_{LAD} + t_{COMB} + t_{OD}$, which is equal to t_{PD1}.

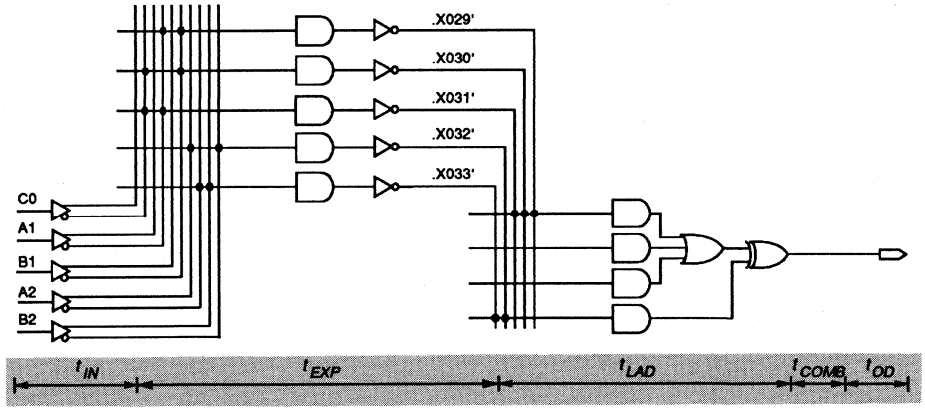
Example 3: S2 Adder Bit

For complex logic that requires expanders (represented here by **_X**...), the expander-array delay, t_{EXP}, is added to the delay element. For instance, **S2**, the second bit of the full adder, requires expanders. The equations are:

```
S2      = _MC019;
_MC019  = MCELL(  _EQ023 $  _EQ024 );
_EQ023  = _X029 & _X030 & _X031;
_X029   = EXP( !B1 & !A1 );
_X030   = EXP( !B1 & !C0 );
_X031   = EXP( !A1 & !C0 );
_EQ024  = _X032 & _X033;
_X032   = EXP( !B2 &  A2 );
_X033   = EXP(  B2 &  A2 );
```

Using these equations, the logic structure can be mapped onto the MAX architecture, as shown in Figure 9.

Figure 9. Adder Equation Mapped to EPM5000-Series MAX Architecture



The overall delay for **S2** is equivalent to $t_{IN} + t_{EXP} + t_{LAD} + t_{COMB} + t_{OD}$, which is equal to $t_{EXP} + t_{PD1}$.

Example 4: Asynchronous 4-Bit Counter

The example shown in Figure 10 evaluates an asynchronous 4-bit counter. The counter has one logic-controlled clock input (**CLK**) and the following outputs: **RCO**, **QD**, **QC**, **QB**, and **QA**. In addition, it has five inherent delays associated with registered logic (clock delay, input delay, array delay, feedback delay, and output delay) as well as setup time and hold time requirements for each register.

The propagation delay from input pin to the clock input of the register for the least significant bit **QA** is $t_{IN} + t_{IC}$ (see Figure 10). If an I/O pin is used for input, the propagation delay from the I/O pin to the register's clock input is $t_{IO} + t_{IC}$, or $t_{IO} + t_{PIA} + t_{IC}$ for MAX EPLDs with multiple LABs. Since the delay from register to output pin is $t_{RD} + t_{OD}$, the total clock to output delay is $t_{IN} + t_{IC} + t_{RD} + t_{OD}$ for dedicated input to output; $t_{IO} + t_{IC} + t_{RD} + t_{OD}$ for I/O pin to output for EPM5000-series EPLDs with one LAB; or $t_{IO} + t_{PIA} + t_{RD} + t_{OD}$ for I/O pin to output for EPM5000-series EPLDS with multiple LABs.

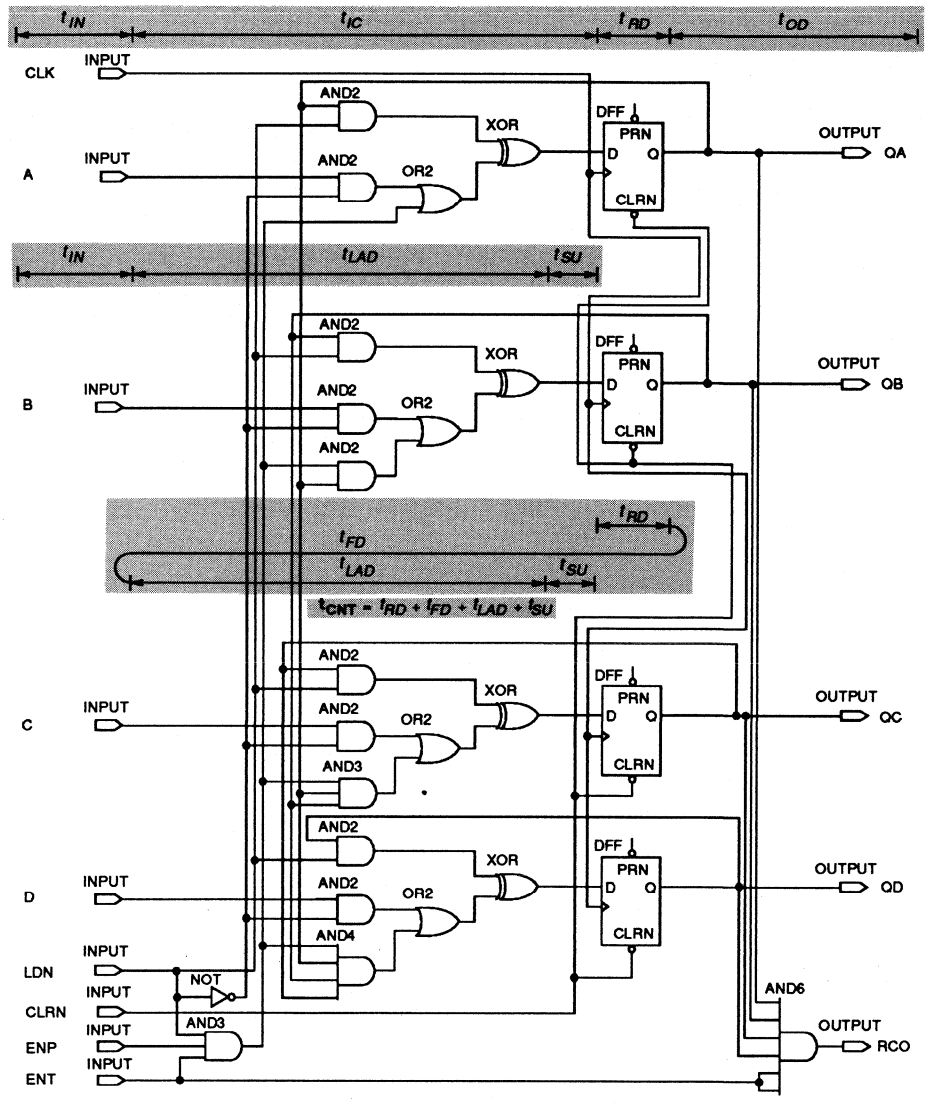
In addition, data input to the register must meet both setup and hold time requirements. The internal setup time is the time needed for the input data to stabilize before the triggering edge of the clock appears at the register input. The external setup time is the difference between the sum of the input, logic array, and setup time, and the sum of the input and clock delay: $(t_{IN} + t_{LAD}) - (t_{IN} + t_{IC}) + t_{SU}$. When expanders are used, t_{EXP} must be added, and when I/O pins are used, t_{IO} or $(t_{IO} + t_{PIA})$ must be added. As long as the external setup time is met at the inputs, the counter functions properly.

The maximum internal counter frequency (f_{CNT}) is the inverse of t_{CNT}, which is the worst-case delay for internal feedback. This frequency is the minimum internal clock period at which the counter can operate correctly. The delay is the sum of delay paths that the register feedback must traverse before reaching a register input and meeting the internal setup time: $(t_{RD} + t_{FD} + t_{LAD} + t_{SU})$. Once the clock triggers **QB**, data takes t_{RD} delay prior to appearing at the register output. The signal feeds back (t_{FD}) and flows through the logic array (t_{LAD}). Finally, the signal reaches the register **QC** and meets the setup time of the register (t_{SU}).

When expanders are used, t_{EXP} must be added as the signal passes through the expander array before reaching the logic array. The t_{CNT} delay represents only internal circuit delays, while a circuit that depends on external and internal signals must also account for input and I/O delays.

10

Figure 10. Timing Analysis of 74161 Counter



Conclusion

MAX+PLUS development tools offer the ability to determine timing delays within MAX EPLDs. Delay prediction and simulation allow the designer to analyze the delays for a given design. The designer may also wish to hand-calculate these paths before entering the design. To understand timing relationships in MAX EPLDs, the designer must think of the total delay path in terms of the microparameters listed in the target EPLD data sheet. From these AC values, it is easy to determine accurate timing delay information by adding up the appropriate combination of microparameters.

Using Expanders

to Build Registered Logic in EPM5000-Series MAX EPLDs

October 1990, ver. 2

Application Brief 76

Introduction

EPM5000-series MAX EPLDs are high-density, user-configurable devices with up to 192 macrocells. Each macrocell contains one register that can be programmed for registered functions or bypassed for combinatorial functions. Since applications sometimes require more registers than are available in the macrocells, extra registers can be built with expander product terms (expanders). Expanders are unallocated product terms used to build complex combinatorial functions or latches and registers. This application brief explains how and when to use expander latches and registers, and describes timing considerations, specifically for an SR latch, a transparent D latch, and a synchronous register. Familiarity with EPM5000-series MAX architecture and timing models is assumed.

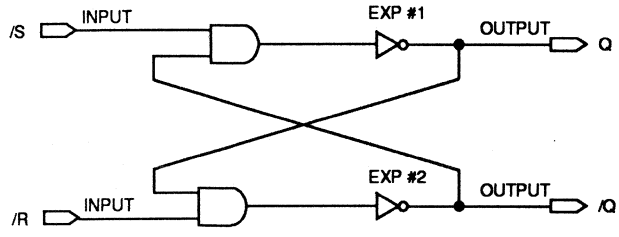
Expander Product Terms

EPM5000-series architecture provides expanders with inverted outputs that feed the logic array. Each expander is fed by the same inputs that feed the macrocell: a global bus, macrocell feedbacks, other expanders, and I/O feedbacks. Since expanders feed themselves, they can be used to build latches and registers. Two expanders (**EXP** primitives) can be cross-coupled to generate an SR latch, three can be used to build a transparent D latch, and six can be used to build a synchronous D flip-flop with asynchronous Preset and Clear. The expander circuits described here have been built into macrofunctions to optimize performance. Each function is built with AND gates and **EXP** primitives to optimize fitting.

Asynchronous SR Latch

Figure 1 shows an asynchronous SR latch implemented with two expanders. Since expanders are product terms with inverted outputs, the latch is a NAND implementation that makes the Set and Reset terms active low. If both inputs are simultaneously low, both outputs will become logic low until one or both of the outputs go high again.

Figure 1. SR Latch Implemented with Expanders



10

The functional output of an SR latch is shown in Table 1. To implement the latch with active-high inputs, as in a NOR latch, the inputs are inverted with **NOT** primitives. Asynchronous SR latches are often used to debounce input-switching circuits or to detect edges in switching circuits.

Table 1. Functional Output of SR Latch

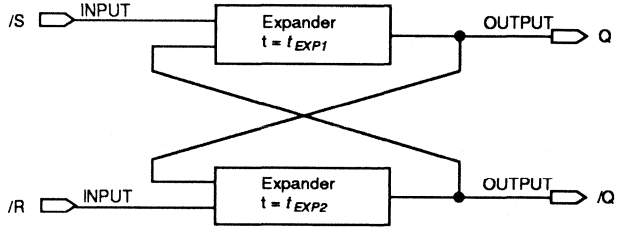
/S	/R	Q
H	H	Q_0
L	H	H
H	L	L
L	L	H *(1)*

Note:
(1) /S and /R low causes both Q and /Q to be high.

SR Latch Timing

Each expander has a timing delay defined as t_{EXP}. When implementing latches and registers with an expander, it is important to be aware of the timing requirements to ensure that the expander functions properly. The hold time for the SR latch shown in Figure 2 is t_H. A low signal at the **S** input must remain low long enough to propagate through Expander 1 and Expander 2 to latch the input. The propagation delay from the latch input to the latch output is t_{CO}. A low at the **S** input must travel through Expander 1 and Expander 2 before the outputs **Q** and **/Q** become valid.

Figure 2. SR Latch Timing Model



SR Latch Timing Equations

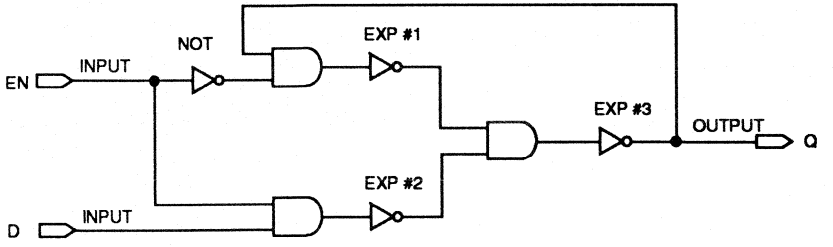
$$t_H = t_{EXP1} + t_{EXP2} = 2 * t_{EXP}$$

$$t_{CO} = t_{EXP1} + t_{EXP2} = 2 * t_{EXP}$$

Transparent D Latch

The transparent asynchronous D latch with **EN** (Enable) is implemented with three expanders, as shown in Figure 3. This latch is comparable to a 74LS373, and is transparent while the **EN** signal is at a logic high. When **EN** goes low, the input is latched until **EN** goes high again. This latch is especially applicable for latching inputs from a bus. The functional output of this latch is shown in Table 2.

Figure 3. Transparent D Latch



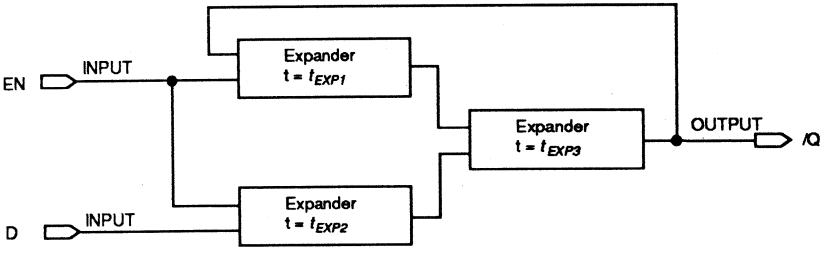
Transparent D Latch Timing

The timing paths for the transparent asynchronous D latch are shown in Figure 4. The t_H value for this circuit is 0 ns because the paths from the **D** input and the **EN** input have delays equal to those of Expander 3, which latches the result. The setup time, t_{SU}, requires the **D** input to go through Expander 2 and Expander 3 to reach Expander 1 before it can be latched. The delay through Expander 2 and Expander 3 to the output is t_{CO}.

Table 2. Functional Output of D Latch

EN	D	Q
L	L	Q_0
L	H	Q_0
H	L	L
H	H	H

Figure 4. Transparent D Latch Timing Model



Transparent D Latch Timing Equations

$t_H = 0$

$t_{SU} = t_{EXP2} + t_{EXP3} = 2 * t_{EXP}$

$t_{CO} = t_{EXP2} + t_{EXP3} = 2 * t_{EXP}$

10

Synchronous D Register

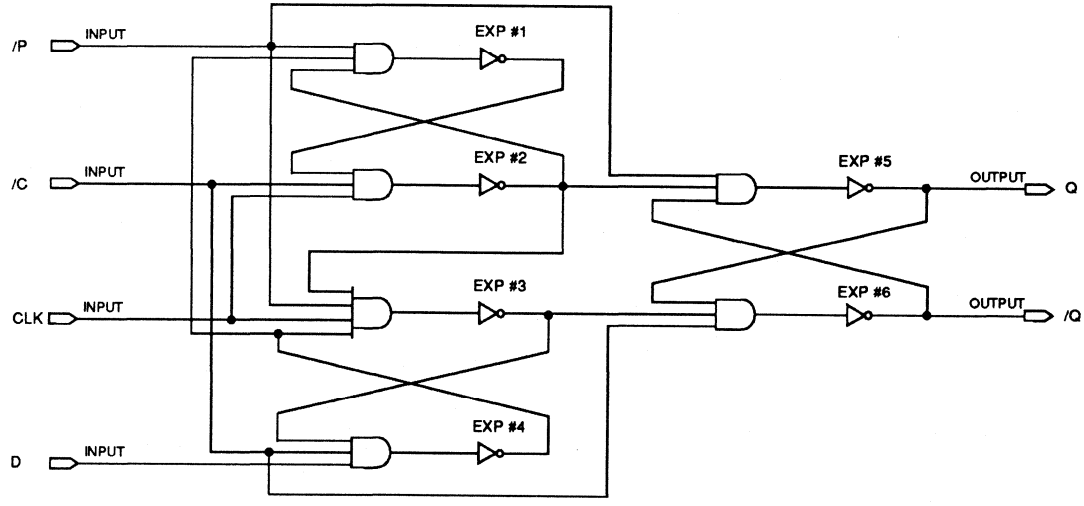
The function outputs for the synchronous D register are shown in Table 3. A synchronous D register with asynchronous Preset and Clear (/**P** and /**C**) can be built with six expanders, as shown in Figure 5.

Table 3. Function Outputs for Synchronous Register

/P	/C	D	CLK	Q	/Q
H	H	L	⌐	L	H
H	H	H	⌐	H	L
H	H	X	H	Q_0	$/Q_0$
H	H	X	L	Q_0	$/Q_0$
L	H	X	X	H	L
H	L	X	X	L	H
L	L	X	X	H	H

The state at the **D** input is clocked into the latch with a rising edge at the clock input. Both the true and complement signals are available at the output. This output remains latched until the next rising edge clock or until the Preset or Clear is activated with a logical low signal. The Preset and Clear can be made active high by placing **NOT** primitives in front of the two signals. Using expanders as registers increases the total register count by 31%. For example, the EPM5192 can have up to 60 registers implemented with expanders, which gives it a total capacity of 252 registers.

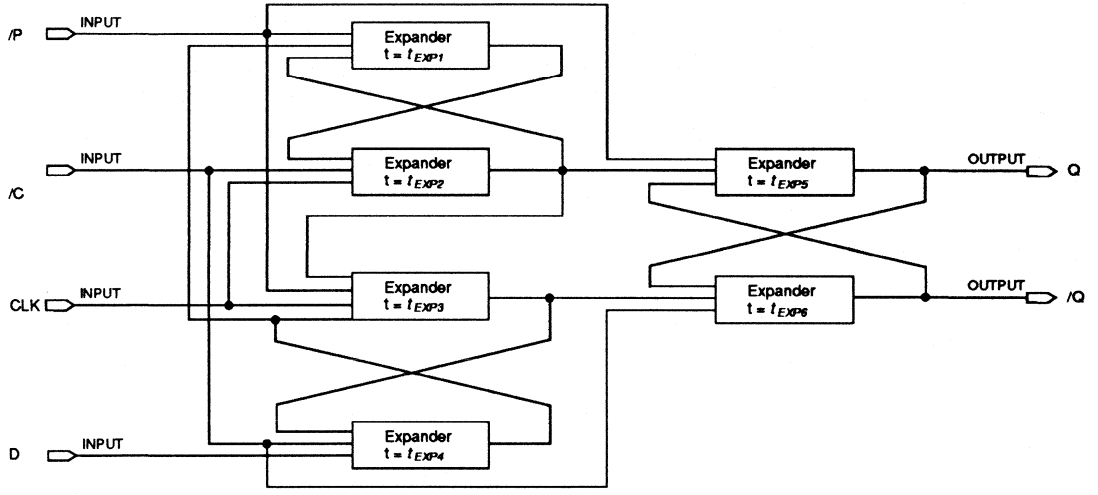
Figure 5. Synchronous D Register with Preset and Clear



Synchronous D Register Timing

The timing paths for the synchronous D register are shown in Figure 6. Two different timing paths are determined by the D input, depending on whether the input is high or low. The worst-case path is described here. Before the input may be clocked, t_H is one expander delay through Expander 3. A valid path through Expander 4 and Expander 1 at the input of Expander 2 is the worst case for t_{SU}. The time required for clock-to-output (called t_{CO1}) has a worst-case path through Expander 3, Expander 6, and Expander 5 for both Q and /Q to produce valid outputs. Both t_{CLR} and t_{PRE} have a delay path through Expander 5 and Expander 6 to produce a valid output. The minimum time in which the register can be clocked in a pipeline register is t_{CNT}. In this case, t_{CNT} is simply $t_{CO1} + t_{SU}$. The worst-case timing for the synchronous register is five expander delays.

Figure 6. Timing Model for Synchronous D Register



Synchronous D Register Latch Timing Equations

$t_H = t_{EXP2}$ or $t_{EXP3} = t_{EXP}$

$t_{SU} = t_{EXP1}$ or $t_{EXP1} + t_{EXP4} = 2 * t_{EXP}$

$t_{CO1} = t_{EXP3} + t_{EXP6} + t_{EXP5}$ or $t_{EXP2} + t_{EXP5} + t_{EXP6} = 3 * t_{EXP}$

$t_{CLR} = t_{EXP6} + t_{EXP5} = 2 * t_{EXP}$

$t_{PRE} = t_{EXP5} + t_{EXP6} = 2 * t_{EXP}$

$t_{CNT} = t_{CO1} + t_{SU} = 5 * t_{EXP}$

10

Fitting Expanders into MAX Designs

Expander latches and registers should be placed strategically within a design to optimize fitting. Expanders are fed by three sources: inputs, macrocell outputs, and other expanders. Complex logic should not feed the inputs to expander registers, because other expanders are the only source of this logic. Instead, registers driven by complex logic should be placed in macrocells. On the other hand, registers that require additional logic after the register should use expander registers, since the output feeds directly into the logic within the macrocells.

Reference

All of the macrofunctions defined in this application brief are available via modem from Altera's Electronic Bulletin Board Service (see the *Electronic Bulletin Board Service Data Sheet* in this data book) or on disk from Altera representatives. The files are named as follows: the SR latch is called **NANDLTCH.GDF**, the transparent D latch is called **EXPLATCH.GDF**, and the synchronous D register is called **EXPDFF.GDF**.

Design Guidelines
for EPM5000-Series MAX EPLDs

October 1990, ver. 2

Application Brief 77

Introduction

MAX+PLUS software uses a combination of advanced logic synthesis techniques and a heuristic fitter to efficiently map designs into MAX EPLDs. The MAX+PLUS Compiler typically synthesizes even the most complex designs in less than five minutes.

However, certain designs are more difficult to fit. These designs contain very complex combinatorial logic or require more macrocells than are available in the target EPM5000-series EPLD. The MAX+PLUS Compiler uses logic synthesis techniques specifically developed to quickly fit these designs. In most cases, MAX+PLUS can automatically fit even these complex designs.

Sometimes designs require subtle modifications to enable the Compiler to perform logic synthesis and obtain a fit. This application brief discusses some of the synthesis techniques used to fit these complex designs and explains the most common Compiler error messages.

Logic Synthesis

The MAX+PLUS Compiler includes a Balancer program, which synthesizes designs that require too many of a certain type of resource. The Balancer is responsible for balancing the logical resources required by a design.

If a design contains too many macrocells for the specified MAX EPLD, the Balancer can transform buried combinatorial macrocells into expander product terms (expanders). It also resynthesizes designs containing too many expanders by transferring logical expressions implemented on expanders into macrocells. Up to three expanders may be transferred into each macrocell. By balancing resource usage, the Compiler can fit designs that originally require too many instances of a particular logical resource. In this manner, most complex designs are automatically fitted.

Compiler Messages

If a design is too large or complex, the Compiler generates an error message specifying the problem and, if applicable, indicating the error location. The most common Compiler error messages relating to synthesis and fitting of designs generally take one of the following forms:

- **Design requires too many <#/#> macrocells**
  This message indicates that the current design contains too many macrocells for the specified MAX EPLD. The ratio <#/#> is the ratio of macrocells in the design to the macrocells available on the EPLD.

- **Logic too complex: for <node name> [<text>]**
  This message indicates that an equation for a node, in its current form, is too complex for a MAX EPLD. Logic synthesis may generate this message while processing very complex combinatorial expressions.

- **Design requires too many <#/#> expanders for <text>**
  This message indicates that the overall design is too complex and requires too many expanders for the target MAX EPLD. The ratio **<#/#>** is the ratio of expanders in the design to the available expanders. The **<text>** portion of the message specifies the particular macrocell or Logic Array Block (LAB) that requires too many expanders. This message is typically generated by the Compiler's Fitter module.

Once the MAX+PLUS Compiler generates an error message, simple design modifications often achieve a fit. The following sections provide suggestions on how to obtain or improve fitting results.

Fitting a Design with Too Many Macrocells

The MAX+PLUS Compiler generates an error message when a design contains too many macrocells for the specified EPLD, and the Balancer has already unsuccessfully tried to transfer enough buried macrocells onto expanders. Thus, macrocells must be eliminated from the design. The following steps may help reduce the number of macrocells in a design.

- Any **MCELL** or **SOFT** buffers that have been manually placed into combinatorial logic may be eliminated.

- **SOFT** buffers from Altera-provided combinatorial TTL macrofunctions may be removed. **SOFT** buffers are placed in complex combinatorial macrofunctions to partition the logic into multiple macrocells. Usually, logic synthesis removes the **SOFT** buffers that are not required inside these macrofunctions. However, macrocell count can sometimes be reduced by removing one or more buffers. Table 1 lists the macrofunctions with and without **SOFT** buffers. For example, the **7485** macrofunction contains four **SOFT** buffers. One or more buffers may be removed, particularly if the inputs or outputs are single product terms. The edits should be made in a copy of the macrofunction that is in the same directory as the user's design, not to the Altera-provided macrofunction installed in the **\MAXPLUS\MAXLIB** directory.

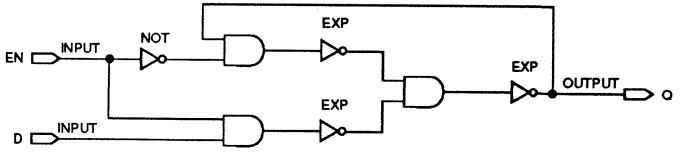
Table 1. Complex Combinatorial TTL Macrofunctions

Function	Macrofunctions with Soft Buffers	Macrofunctions without Soft Buffers
Arithmetic Function	8FADD, MULT4, 7480, 7483, 74181	MULT2, MULT24, 7482, 74183
Comparator	8MCOMP, 7485, 74518	
Converter		74184, 74185
Decoder		7442, 7443, 7444, 7445, 7446, 7447, 7448, 7449, 74138, 74139, 74154 74155, 74156
Latch	74116, 74259, 74279	
Multiplexer		74151, 74153, 74157, 74158, 74298
Parity Checker	74180, 74280	

- Buried registers or latches can be implemented with expanders to reduce the macrocell count. Expanders can be cross-coupled to form a variety of register and latching functions for use as buried logic. Figure 1 shows a transparent D latch placed on cross-coupled expanders. Tips on building registers with expanders are presented in *Application Brief 76 (Using Expanders to Build Registered Logic in EP5000-Series MAX EPLDs)*. Expander-based macrofunctions are also available in the MAX+PLUS TTL MacroFunction Library.

Figure 1. Transparent D Latch Implemented with Cross-Coupled Expanders

If a design contains too many macrocells, buried register functions can be placed on cross-coupled expanders.



If none of these techniques yields the desired fit, contact the Altera Applications Department for assistance at 1 (800) 800-EPLD. As a last resort, logic may have to be removed from the design to achieve a fit into the target device, or the design may have to be partitioned into two MAX EPLDs. Call Altera's Applications Department for more information.

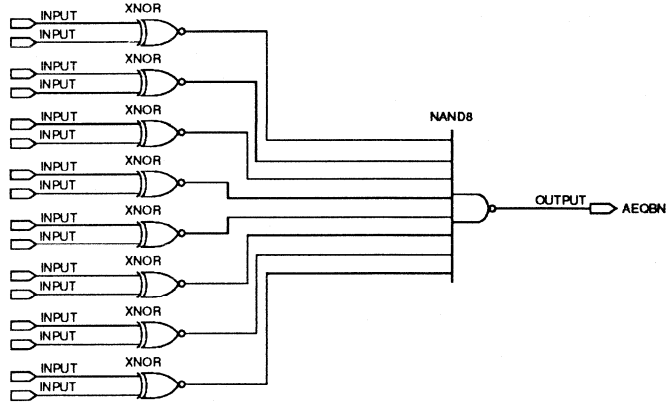
Designing Complex Combinatorial Functions

The MAX+PLUS Compiler generates error messages when a design's combinatorial logic is too complex. This complexity can result from cascading several combinatorial macrofunctions, such as adders and comparators, or from heavy use of **XOR** functions. Figure 2 shows complex combinatorial logic caused by heavy use of the **XNOR** function.

To resolve Compiler errors related to complex combinatorial logic, one or more **SOFT** or **MCELL** buffers can be inserted into a design to separate complex combinatorial expressions. Placing a **SOFT** or **MCELL** buffer consumes a macrocell to implement a portion of a complex logic expression. The complex expression is then distributed over two or more macrocells and simplified. Thus, inserting **SOFT** or **MCELL** buffers in a design can affect the timing of a design. However, proper placement of **SOFT** buffers can significantly simplify a design, and therefore reduce the number of expanders and the number of macrocells required by a design.

Figure 2. Complex Combinatorial Logic

This complex function results from heavy use of the XNOR function.



Placing SOFT and MCELL Buffers

The best location for a **SOFT** buffer may not be obvious. Although the Compiler error message specifies an error location, the identified node name indicates the output of the complex expression, which is usually not the correct location for a **SOFT** buffer. To find the best location, the logic feeding the node must be analyzed to determine the cause of the expression's complexity.

A **SOFT** buffer can be used to minimize the complexity of a combinatorial expression. In Figure 3, the Compiler has flagged **NODE_A** as an expression that is too complex. Since **NODE_B** has the same structure (and thus the same complexity) as **NODE_A**, a **SOFT** buffer can be inserted at **NODE_C** to simplify the complex expressions of both **NODE_A** and **NODE_B**.

Figure 3. SOFT Buffer Minimizing Multiple Complex Expressions

Inserting a SOFT buffer at location NODE_C reduces the complexity of the logic for both NODE_A and NODE_B.

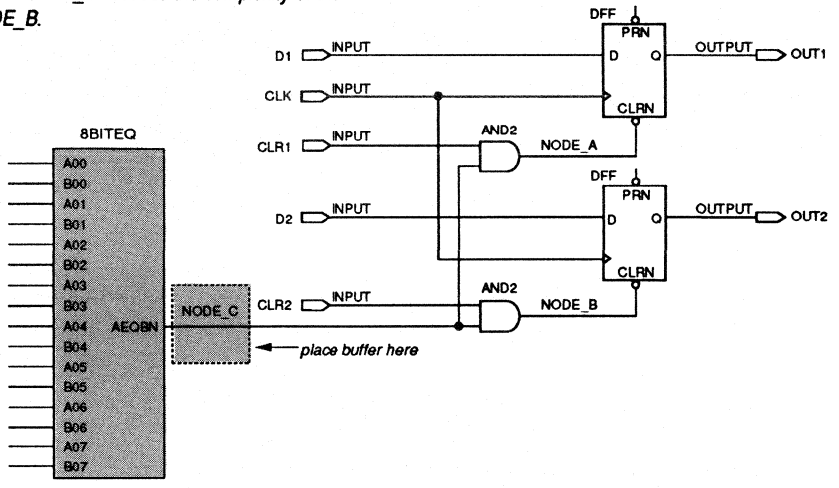
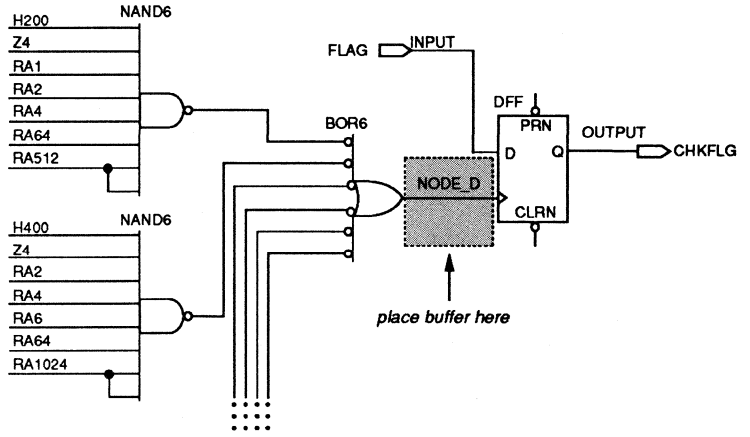


Figure 4 shows an example where the **SOFT** buffer is best placed at the location **NODE_D**, as specified by the Compiler. The logic expression for **NODE_D** is an AND-OR function feeding the Clock input of a D register. The Clock input has a single product term dedicated to it, and thus would use a great number of expanders to implement the logic. By placing a **SOFT** buffer at **NODE_D**, the AND-OR function is efficiently implemented in a macrocell, and the output of the macrocell feeds the Clock input of the D register.

Figure 4. SOFT Buffer Minimizing Complex Expression that Feeds a DFF Clock Input

The control functions for flip-flops have a single dedicated product term. Therefore, complex functions on the flip-flop Clock, Preset, and Clear terms should be simplified with a SOFT buffer. In this example, the SOFT buffer should be inserted at NODE_D.



10

The following suggestions for placing **SOFT** buffers should be used to prevent and resolve compilation problems. Not only can fitting problems be avoided if these suggestions are followed, but often a more efficient fit will result, allowing additional logic to be integrated into the target EPLD. The following four steps should be taken in order:

1. The *Design Guidelines* section of the *MAX+PLUS Graphic Editor* manual contains basic guidelines for designing efficiently with the MAX architecture. This section should be read before any other steps are taken.

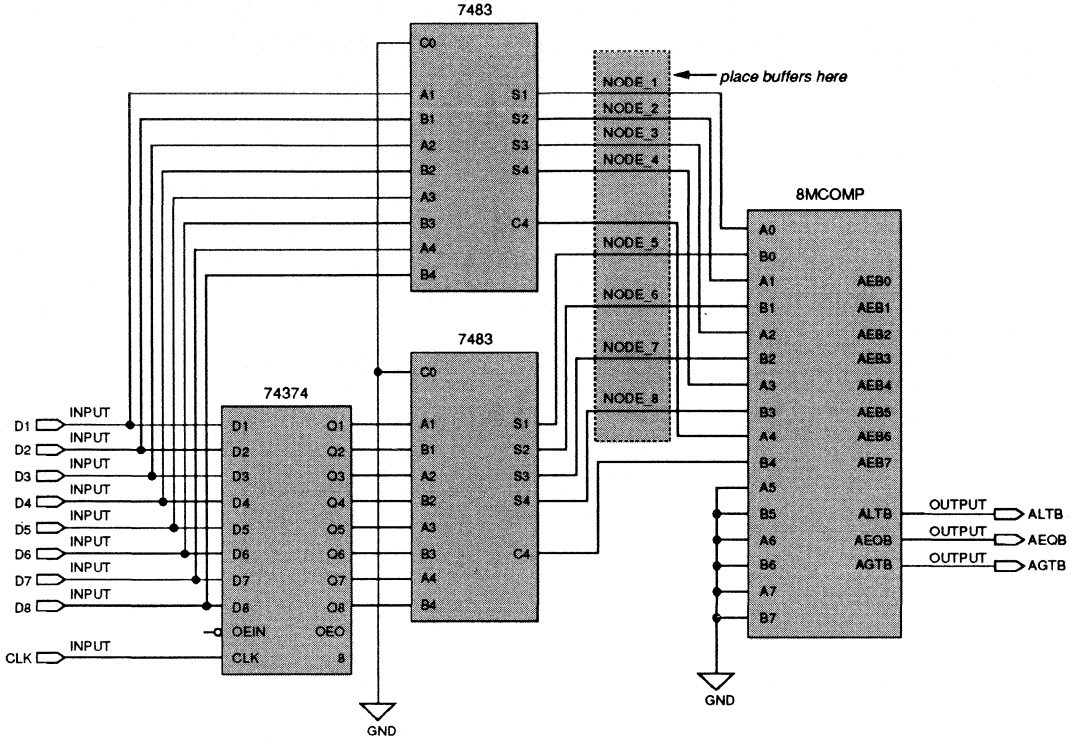
2. Complex combinatorial expressions feeding flip-flop controls and tri-state buffers may be good locations for **SOFT** buffers. The Output Enable tri-state input and the Preset, Clear, and Clock inputs to flip-flops all have a single product term associated with them. If the expression feeding a control input is complex or feeds multiple locations, it may require a large number of expanders. Placing a **SOFT** buffer will reduce the number of expanders used.

3. Complex combinatorial outputs of macrofunctions may be good locations for **SOFT** buffers if they do not feed a flip-flop input or an I/O pin. These complex expressions may need to be isolated before being routed into another macrofunction. Inputs to macrofunctions may also be good locations for **SOFT** buffers if they are being fed by complex expressions.

   Figure 5 shows the outputs of two **7483** adders feeding an 8-bit magnitude comparator. Both the adder and the comparator contain complex logic, but placing **SOFT** buffers at **NODE_1** through **NODE_8** significantly reduces the complexity of the overall function.

Figure 5. SOFT Buffers Minimizing Complex Expressions that Feed an 8-Bit Magnitude Comparator

Cascading complex combinatorial macrofunctions may sometimes require SOFT buffers. In this example, NODE_1 through NODE_8 require SOFT buffers.



4. Complex combinatorial expressions that feed many different locations may be good locations for **SOFT** buffers. If a complex expression feeds multiple Logic Array Blocks (LABs), the logic associated with the expression is duplicated in each LAB fed by the expression, and the number of expanders is increased. Placing a **SOFT** buffer at the complex combinatorial output reduces the number of expanders. The Compiler's Report File contains a list of all duplicated expanders.

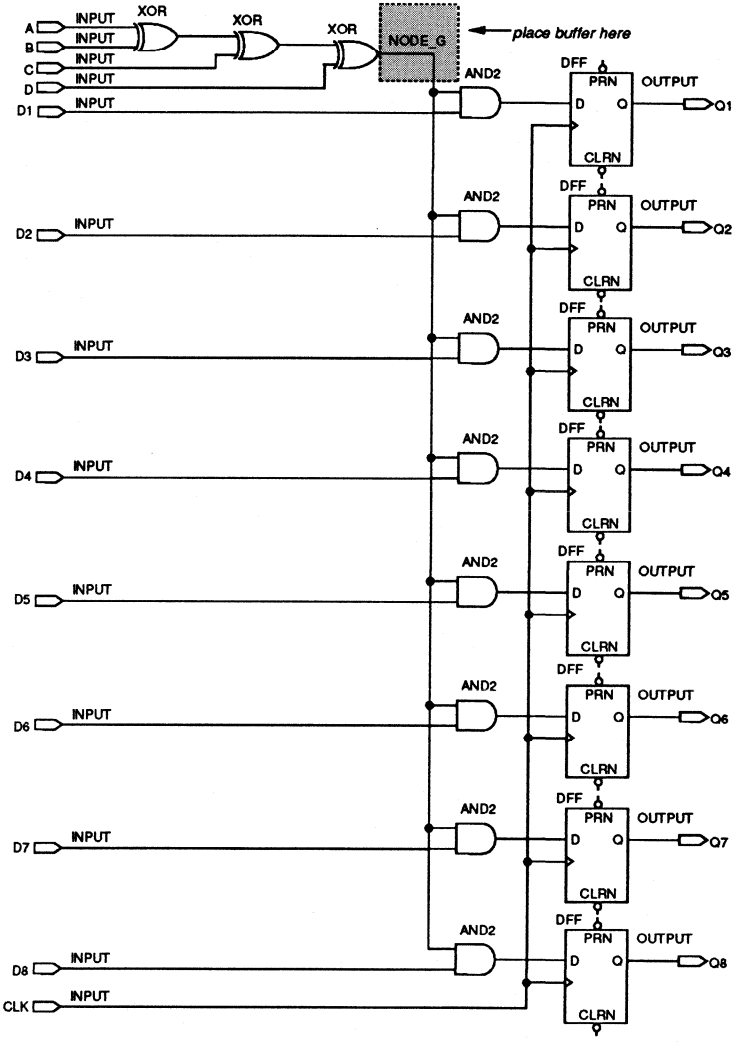
   Figure 6 shows a design that has 1 output in each of the 8 LABs of an EPM5128. The design consumes 5 expanders in each LAB, a total of 40 expanders. Inserting a **SOFT** buffer at **NODE_G**, however, reduces the number of expanders used to only 3.

10

The *MAX+PLUS User Guide* explains all Compiler error messages. Additional help can be obtained by calling the Altera Applications Department at 1 (800) 800-EPLD.

Figure 6. SOFT Buffer Minimizing a Complex Expression that Feeds Multiple LABs

If a logical expression feeds multiple LABs, it may be a good place for a SOFT buffer. In this example, inserting a SOFT buffer at NODE_G reduces the number of expanders from 40 to 3.



Conclusion

When Compiler error messages indicate that logic synthesis or design fitting are not possible, the designer can use a variety of methods to help achieve a fit. By adding or removing strategic **MCELL** and **SOFT** buffers or implementing registers and latches with expanders, complex designs can be modified to fit into an EPM5000-series MAX EPLD.

Optimizing Memory for MAX+PLUS Software

October 1990, ver. 2

Application Brief 78

Introduction

Altera's MAX+PLUS Development System contains a Compiler and a Simulator that efficiently utilize the memory resources available in PC-based (DOS) computer systems. If a system does not contain enough memory, MAX+PLUS may issue out-of-memory error messages when implementing large designs. A computer that contains more than 1 Mbyte of RAM may also cause these error messages if the memory is not configured for optimum use by MAX+PLUS. This application brief discusses the different types of memory supported by DOS computers, how to determine the memory configuration of a computer, and how to configure a computer to allow even the most complex designs to be processed by MAX+PLUS.

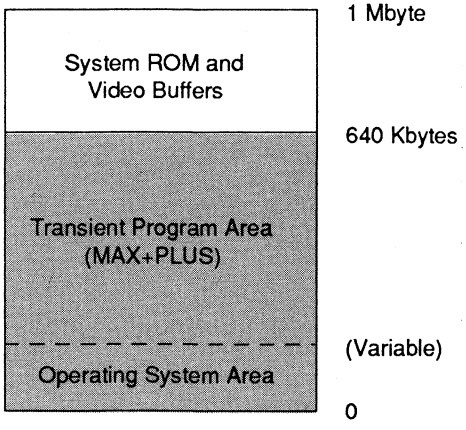
Memory Types

DOS-based computers may have three different types of memory: conventional, expanded, and extended.

Conventional Memory

Conventional memory consists of the 1 Mbyte of memory directly addressable by the 80286, 80386, or 80486 microprocessor running in real mode (also referred to as 8086-emulation mode). Figure 1 shows how DOS allocates conventional memory.

Figure 1. Allocation of Conventional Memory in DOS-Based Computer Systems



Memory ranging from 640 Kbytes up to 1 Mbyte is used for system ROM (device handlers) and expansion board buffers. DOS and programs that run under DOS (e.g., MAX+PLUS) occupy the lower 640 Kbytes of conventional memory. This area is further divided into an operating system area and a transient program area (TPA).

The operating system area contains the memory-resident portion of DOS and all installed device drivers specified in the **CONFIG.SYS** file. The amount of memory this area consumes depends on the version of DOS, the number of disk buffers, and the cumulative size of the installed drivers. Minimizing the size of this area will increase the memory available to the TPA, and therefore to MAX+PLUS. The size of the operating system area, when MAX+PLUS runs at peak efficiency, is typically 60 to 70 Kbytes.

10

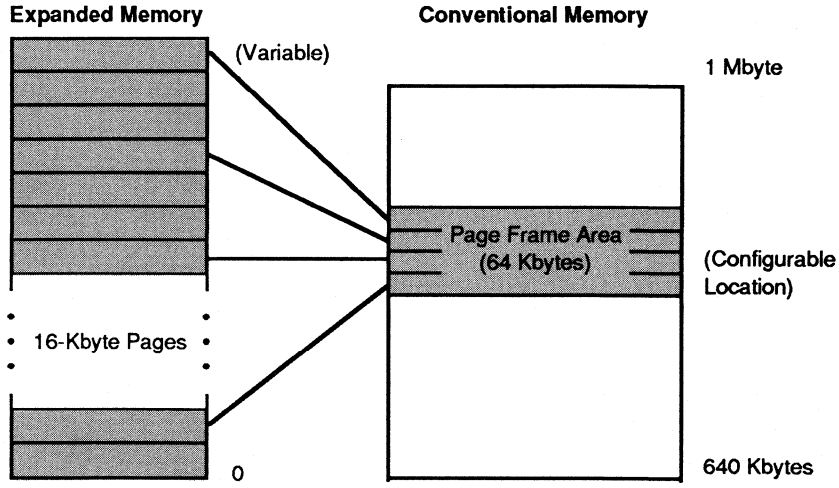
The TPA starts immediately above the operating system area and extends up to the the installed RAM or the 640-Kbyte boundary, whichever is smaller. All programs running under DOS are loaded into this area for execution. For peak efficiency, MAX+PLUS requires 570 Kbytes of RAM for the TPA. This area may also contain programs that run simultaneously with MAX+PLUS. These programs, commonly referred to as background, memory-resident, or TSR (terminate-and-stay resident) programs, occupy memory that could otherwise be used by MAX+PLUS.

Expanded Memory

Expanded memory in 80286/80386/80486 DOS computers provides access to additional memory beyond the 1 Mbyte of conventional memory. The required configuration for MAX+PLUS includes a minimum of 1 Mbyte of expanded memory. An expanded memory subsystem consists of expanded memory hardware and a resident driver program. The hardware is either a memory expansion board plugged into an expansion slot or RAM located on the computer motherboard. The driver program, called the expanded memory manager (EMM), is installed via a device directive (**device = EMM.SYS**) in the **CONFIG.SYS** file. A specific EMM, supplied by the expansion board manufacturer, must be used to ensure proper operation. Figure 2 shows how the EMM makes expanded memory available to application software.

Figure 2. Expanded Memory Mapping

The expanded memory manager maps at least four 16-Kbyte pages at any one time into a contiguous 64-Kbyte page frame area in conventional memory.



The EMM dynamically maps at least four 16-Kbyte pages of expanded memory into a contiguous 64-Kbyte page-frame area in conventional memory. The exact location of the page frame is user-configurable to avoid hardware conflicts, and is typically placed in the upper 384 Kbytes of conventional memory (system memory). Placing the page frame in the lower 640 Kbytes will decrease the size of the TPA. Since MAX+PLUS requires 570 Kbytes of TPA for peak efficiency, the page frame must be placed in the upper 384 Kbytes to obtain the optimum configuration for MAX+PLUS.

The 80286, 80386, and 80486 machines use expanded memory differently. Memory-mapping capabilities built into the 80386 and 80486 processors allow them to use RAM as expanded memory with no additional hardware. The 80286 does not have this capability, and requires expanded memory hardware for the memory mapping function. Therefore, some extended memory boards may not be compatible with 80286 machines. The user should consult the board manufacturer for system compatibility information.

Extended Memory

Extended memory consists of RAM located above 1 Mbyte that can be linearly addressed by 80286/80386/80486 computers running in protected mode. DOS does not support extended memory except for ROM BIOS routines, which allow extended memory to be used as RAM disks via the IBM VDISK software. Since extended memory cannot be used to load and execute programs, it also cannot be used to increase the usable memory for MAX+PLUS. Methods for converting extended memory to expanded memory are discussed later under the heading "Increasing Expanded Memory for MAX+PLUS."

Determining Memory Configuration

The MAX+PLUS Altera Field Diagnostics (AFD) utility can determine the memory configuration of a computer. This utility also tests Altera programming hardware. AFD may be invoked by typing **afd <Enter>** from the **MAXPLUS** directory. The computer's memory configuration is displayed, as shown in Figure 3. The following four items appear under **Memory Configuration**:

- **Normal** refers to the amount of conventional memory installed for use with DOS and programs under DOS.
- **Available** refers to the amount of normal memory in the TPA available to MAX+PLUS.
- **Expanded** refers to the amount of expanded memory available to MAX+PLUS.
- **Extended** refers to the amount of extended memory in the computer.

If **Available** *is below 570 Kbytes or* **Expanded** *is below 1024 Kbytes, MAX+PLUS may run out of memory.*

10

Figure 3. AFD Memory Configuration Screen

```
ALTERA Applications Engineering Diagnostics
Version 7.0   Nov 06 1989   09:17:09
Copyright (C) 1986-1989 ALTERA Corporation

MS-DOS Version 3.30
Computer type: AT or equivalent.
BIOS release date: '10/1/90'
Memory Configuation:
Normal     :  640k bytes.
Available  :  566k bytes.
Expanded   : 2048k bytes.
Extended   : 1084k bytes.
BIOS revision: 0.
MicroChannel not available.
system configuration: model: 252 sub-model: 1
```

Increasing Available Memory for MAX+PLUS

Available memory for MAX+PLUS can be increased by removing all background programs and unnecessary device drivers. Removing background programs frees the entire TPA for use by MAX+PLUS, while removing device drivers decreases the size of the operating system area, thereby maximizing TPA size. The following procedure describes how to obtain the optimum configuration for MAX+PLUS and quickly switch between configurations:

1. Save the current configuration by copying the **AUTOEXEC.BAT** file to **ORIGINAL.BAT** and the **CONFIG.SYS** file to **ORIGINAL.SYS**.

2. Edit **AUTOEXEC.BAT** and **CONFIG.SYS** to remove background programs and unneeded device drivers (EMM must *not* be removed).

3. Reboot the system and run AFD again to verify that available memory has increased.

4. Copy the edited **AUTOEXEC.BAT** to **MAX.BAT** and **CONFIG.SYS** to **MAX.SYS** to save the MAX+PLUS configuration that maximizes available memory.

5. Create a DOS batch file (**SWITCH.BAT**) to provide a means of switching between the MAX+PLUS configuration and the original configuration.

Figure 4 shows **SWITCH.BAT** and possible **MAX.BAT** and **MAX.SYS** configuration files.

Figure 4. SWITCH.BAT, MAX.BAT, and MAX.SYS

SWITCH.BAT

```
c:
cd \
Copy %1.bat AUTOEXEC.BAT
Copy %1.sys CONFIG.SYS
```

MAX.BAT

```
Path = c:\;c:\maxplus;c:\dos;
prompt $p$g
```

MAX.SYS

```
Device = EMM.SYS AT D000 258 ND
Files = 20
Buffers = 20
```

SWITCH.BAT is used to switch automatically to a desired configuration. By typing **switch max <Enter>**, **MAX.BAT** is copied to **AUTOEXEC.BAT** and **MAX.SYS** is copied to **CONFIG.SYS**. The system may then be rebooted to load the MAX+PLUS configuration. Typing **switch original <Enter>** followed by a reboot switches back to the original configuration.

Note that **MAX.BAT** and **MAX.SYS** are examples; actual files may differ.

Increasing Expanded Memory for MAX+PLUS

The amount of expanded memory available to MAX+PLUS may be increased in four ways:

- Configure MAX+PLUS to use all of the computer's expanded memory.
- Reconfigure extended memory to expanded memory.
- Emulate expanded memory with extended memory (80386 or 80486 only).
- Add RAM to the computer.

During installation, MAX+PLUS is set to use all expanded memory present in the computer. The **OE (Options : Expanded memory)** command described in *Appendix B—MAX+PLUS Configuration File* of the ***MAX+PLUS User Guide*** displays the current expanded memory available to MAX+PLUS. If this value differs from the expanded memory reported by AFD, the value may be increased to make all expanded memory in the computer available to MAX+PLUS.

10

If the computer contains extended memory, it may be possible to reconfigure this memory to be expanded. Some plug-in memory expansion cards or system motherboards allow the memory to be divided between conventional, expanded, and extended memory. Typically, the memory on plug-in is configured via on-card dip switches or a setup program supplied with the card. The system motherboard is usually configured with the setup program supplied with the computer. If the computer contains extended memory, the card or computer documentation may provide information on configurability.

If the computer has extended memory that is not directly configurable, it may be possible to emulate expanded memory with an appropriate device driver. The memory-mapping capabilities of the 80386 and 80486 allow generic device drivers to convert extended memory to expanded memory. Two such drivers for 80386 machines are 386MAX from Qualitas, Inc. and QEMM from Quarterdeck Office Systems. Because the interface to expanded memory is hardware-dependent, however, generic drivers for 80286 machines are not available. The user should consult the computer manufacturer for an appropriate device driver for an 80286 machine with expanded memory capability on the system motherboard.

If the computer does not contain memory that can be used to provide 1 Mbyte of expanded memory, the user must add RAM in one of the methods described in this application brief, and the associated EMM must be installed to add expanded memory to the computer.

After increasing expanded memory, AFD should be run again to verify that expanded memory has been installed properly. Entering the **OE** (**Options : Expanded memory**) command and updating the amount of memory available then gives MAX+PLUS access to any newly installed expanded memory.

Conclusion

To avoid out-of-memory messages when running MAX+PLUS, a computer must be configured to provide 570 Kbytes of available program memory and 1 Mbyte of expanded memory. Batch files, such as the ones shown in this application brief, allow quick loading of this configuration for operating MAX+PLUS. If out-of-memory error messages persist, contact Altera's Applications Department at 1 (800) 800-EPLD for assistance.

Simulating Internal Nodes with MAX+PLUS Software

October 1990, ver. 2

Application Brief 79

Introduction

The MAX+PLUS Simulator and Waveform Editor allow designers to graphically verify logic, both at the EPLD pins and at nodes internal to the device. This application brief discusses a variety of methods used to simulate internal nodes with the MAX+PLUS software. First, it describes the four ways of naming internal nodes. Next, it shows how to locate nodes within state machines and combinatorial nodes. Finally, it describes how to identify appropriate nodes and incorporate them into an input file to perform a quick and thorough simulation.

Logic Synthesis Affects Simulation

During compilation, the advanced algorithms in MAX+PLUS minimize logic and perform logic synthesis to fit designs into MAX architecture. This process implements the functionality of the design, but it may eliminate some of the original nodes that defined the circuit. Consequently, not all of the original nodes can be simulated. The nodes that can be simulated are device inputs and outputs, **MCELL** buffers, and all of the nodes on registers and latches. **SOFT** buffers and **TRI** buffers can also be simulated if they are not eliminated during logic synthesis.

Before simulating a MAX design, it is helpful to generate a History File that contains a list of all nodes present in the synthesized design:

1. Invoke the Simulator and load the netlist for the design.
2. Select **FHC** (**File : History : Create**) and type the name of the file. Press **<Enter>** to create the History File (with the extension **.HST**).
3. Select **CNL** (**Control : Node : List**), type *, and press **<Enter>**. The wildcard character (*) is used to list all nodes in the design that can be simulated.
4. Quit the Simulator and MAX+PLUS.
5. Print the History File (**.HST**).

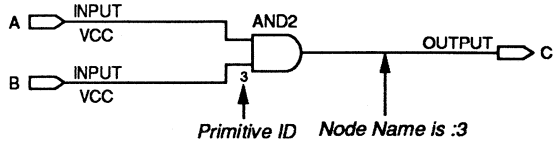
10

Nodes That Can Be Simulated

All nodes in the History File are assigned a name and defined to be of type IN, I/O, or OUT. Nodes defined as IN correspond to EPLD inputs from device pins; they have the same node names as the pins they represent. Nodes defined as I/O are the I/O pins on the EPLD; they have the same names as their corresponding input, output, or bidirectional pins. Nodes defined as OUT are internal to the device; they have no direct pin connection. Internal node names specify the location of a node within a design and refer to primitives only. The section titled *MAX+PLUS Primitives* in the *MAX+PLUS Graphic Editor* manual shows the complete list of primitives.

All primitives, except **TITLE** and **OUTPUT**, have a single output. Figure 1 shows a simple design with a single internal node fed by **AND2**, and lists all nodes that can be simulated.

Figure 1. Nodes Available for Simulation



Type	Name
I/O	C
OUT	:3
IN	A
IN	B

The MAX+PLUS Graphic Editor assigns a unique identification number to every symbol in a schematic. (This ID is located at the lower left-hand corner of the symbol.) The only internal node in Figure 1 is **:3**, defined as type **OUT**. This node corresponds to the output of the **AND** primitive with symbol ID **3**.

Node Names in Levels of Hierarchy

Complex designs with multiple hierarchical levels use node names that incorporate the hierarchical path of the node to define the location of an internal node. Internal node names may take four forms, and the user may use more than one of these forms for a particular internal node. However, the History File lists each node only once using the highest-priority node name found. The four kinds of node names, listed from highest to lowest priority, are as follows:

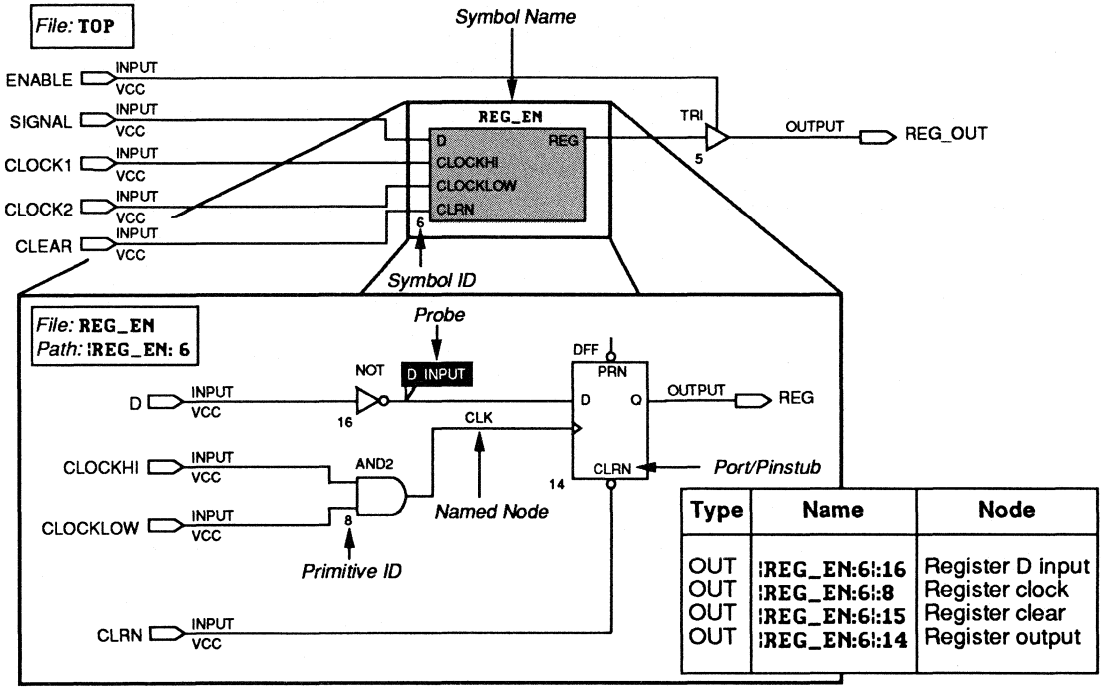
Priority 1: Probes
Priority 2: Named nodes
Priority 3: Ports/pinstubs
Priority 4: Hierarchical pathnames

All internal nodes have hierarchical pathnames, and most may also be specified with a port (stub) name. Nodes can be named and probes can be assigned by the user to further simplify the process of locating and

simulating the nodes. For an overview of internal node name syntax, see "Hierarchical Node Names & Probes" in the *Simulator Reference* section of the *MAX+PLUS Simulator* manual.

Figure 2 shows a hierarchical design example with two levels. The title of each schematic file in a hierarchical design indicates which symbol it defines. This design has four accessible internal nodes that feed the **CLK**, **CLRN**, and **D** inputs to the register, and the register output.

Figure 2. Node Names in a Hierarchical Design



Priority 4: Hierarchical Pathnames

All internal nodes can be identified by a hierarchical pathname, which traces the path of the node through multiple levels of hierarchy with a name of the following format:

¦ <symbol name> : <symbol ID> . . . ¦ : <symbol ID>

The pipe (¦) indicates that the symbol name following it is the name of a file in a lower level of the hierarchy. The colon (:) precedes the symbol ID, which provides further distinction between common symbols in the same level of hierarchy. Thus, the hierarchical pathname for the clock of the register in

10

Figure 2 is **!REG_EN:6!:8**. This name defines the **AND2** primitive (with symbol ID **8**) driving the **CLK** input, which is inside the symbol **REG_EN** (with symbol ID **6**). Designs with multiple levels of hierarchy follow this pattern, with a pipe (!) separating each level of hierarchy.

Priority 3: Ports/ Pinstubs

Internal nodes connected to primitives that convert directly into "hard" nodes (i.e., nodes that will not be eliminated by logic synthesis) use ports as their internal node names. A port is an extension to the hierarchical pathname that defines the specific input or output of the primitive. These ports represent pins at a lower level of the design hierarchy. Ports consist of a period (.) and a port name that follow a hierarchical pathname. For example, the **D** input to the register in Figure 2 is **!REG_EN:6!:14.D**. Table 1 shows all port names associated with the register in Figure 2. A list of all primitives with ports can be found under the heading "Primitives" in *AHDL Elements* in the *MAX+PLUS AHDL* manual.

Table 1. Port Names for the Register Shown in Figure 2

Type	Name	Node
OUT	**:5.OE**	Tri-state control
OUT	**!REG_EN:6!:14.CLK**	Register clock
OUT	**!REG_EN:6!:14.CLRN**	Register clear
OUT	**!REG_EN:6!:14.D**	Register D input
OUT	**!REG_EN:6!:14.Q**	Register output
OUT	**!REG_EN:6!:15.IN**	MCELL input
OUT	**!REG_EN:6!:15.OUT**	MCELL output

Priority 2: Named Nodes

Hierarchical pathnames can be enhanced by using named nodes. Nodes are named in a schematic by inserting text above a node, which causes the symbol ID to be replaced with the text. For example, the node feeding the Clock in Figure 2 has the text **CLK** inserted above it. This name replaces the symbol ID **8**, changing the name of this internal node to **!REG_EN:6!CLK**.

Figure 3. TDF with Named Nodes

```
TITLE "4-Bit Counter With A 74161";

FUNCTION
74161(A,B,C,D,LDn,ENP,ENT,CLRn,CLK)
     RETURNS (QA,QB,QC,QD,RCO);

DESIGN IS 4_bit

SUBDESIGN 4_bit
( ... )

VARIABLE
     counter : 74161;

BEGIN
...
END;
```

Named nodes are automatically created in AHDL Text Design Files (TDFs) when variables are assigned to nodes that can be simulated. Figure 3 shows part of a TDF with the variable **counter** assigned to a **74161** counter in the **VARIABLE** section. The default node name of the first register's output is **!74161:33!QA**. The node name after making the variable assignment is **!COUNTER!QA**.

Priority 1: Probes

Internal node names can be fully customized with probes in a logic schematic. Probes are entered on primitives at any level of hierarchy. The probe automatically connects to the output of the primitive to which it is assigned. The following steps describe how to place a probe on the appropriate primitive to connect to the clock node of the register in Figure 2.

1. Open the **REG_EN** file.
2. Position the cursor on the **AND2** symbol and type **SPE** (**Symbol : Probe : Enter**).
3. Type a name for the probe and press **<Enter>**.

A pointer with the assigned probe name is attached to the primitive output. During compilation, the probe name is put into the Simulator Netlist File (SNF) in place of the hierarchical pathname. For example, in Figure 2 a probe called **D_INPUT** placed on the **NOT** gate feeding the **D** input to the register will produce the node name **D_INPUT** in the SNF, thus eliminating all the hierarchy information for the node name.

MAX+PLUS allows the user to name internal nodes in a variety of ways. The Simulator lists each node only once, using the simplest name for the node. If the node has a probe attached to it, the node is listed under the probe name. If not, the node is listed with a node name, or under a port name if no node name exists. Finally, if no other higher-priority name exists, the node is listed under its hierarchical pathname. Table 2 shows the four ways of naming the D input to the register in Figure 2.

Table 2. Four Possible Names for D Input

Internal Node Name	Type of Name	Description
D_INPUT	Probe	Probe named **D_INPUT** placed on **NOT**
!REG_EN:6!D_INPUT	Named node	Node feeding **D** input **D_INPUT**
!REG_EN:6!:14.D	Port/Pinstub	**D** input port
!REG_EN:6!:16	Hierarchy pathname	Hierarchical pathname of **NOT** primitive feeding **D** input

10

Simulating State Machines

Since state machines often control an entire design, simulation is especially useful for debugging state machines. Four items in a state machine may need to be simulated: inputs, decoded outputs, state register bits, and present states.

State Machine Inputs

State machine input lines feed combinatorial logic, which, in turn, feeds the **D** inputs to state register bits. However, state machine inputs can only be simulated when they are fed by EPLD input pins or by hard nodes within a design. To allow simulation of state machine inputs when these inputs are fed by combinatorial logic, an **MCELL** buffer should be placed in front of all inputs to a state machine. The **MCELL** is a hard node; therefore, its outputs (i.e., the inputs to the state machine) can be simulated. Note, however, that inserting an **MCELL** affects the timing of the state machine, and **MCELL** buffers should therefore be removed after simulation. This technique is used to verify functionality, not timing.

To make the **MCELL** more accessible during simulation, a probe with the same name as the input to the state machine may be attached to the **MCELL** in the Graphic Editor.

Decoded Outputs

Decoded outputs are also combinatorial, and therefore may not allow simulation. If decoded outputs do not feed hard nodes, such as **MCELL** buffers or output pins, then they cannot be simulated. As with state machine inputs, decoded outputs may simulated if an **MCELL** with a probe on it is placed after the decoded outputs.

State Register Bits

State register bits are assigned names in a state machine declaration in the **VARIABLE** section of an AHDL file. The syntax of this declaration is:

```
<name> : MACHINE OF BITS ( <state registers> )
         WITH STATES     ( <state assignments> )
```

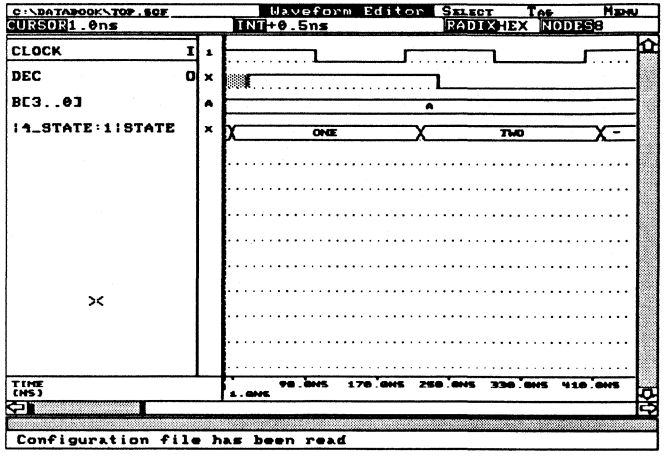
The Simulator extracts the state register names from the TDF to define the register bits. A state machine in a lower level of a hierarchical design incorporates the state machine name into the node name to distinguish it from state machines at the same level. Figure 4 shows a TDF with a state machine. If this machine is loaded into the top level of a schematic design, the state register outputs for this design are **!4_STATE:n!Q0** and **!4_STATE:n!Q1** (where **:n** is the ID for the symbol representing the state machine).

Figure 4. Text Design File with State Machine

```
TITLE "A Simple State Machine";
DESIGN IS 4_state;

SUBDESIGN 4_state
(
    clk, BUS[3..0]  : INPUT;
    CNT[3..0], decode   : OUTPUT;
)

VARIABLE
    state : MACHINE OF BITS (Q[1..0])
                WITH STATES ( ONE, TWO, THREE,
FOUR );

BEGIN
    state.clk = clk;
    CNT[ ] = Q[ ];

    CASE (state) IS
        WHEN ONE =>
            state = TWO;
            decode = (BUS[] == h"A");
        WHEN TWO =>
            state = THREE;
        WHEN THREE =>
            state = FOUR;
        WHEN FOUR =>
            state = ONE;
            decode = (BUS[] == h"7");
    END CASE;
END;
```

AHDL state machines provide automatic state assignments. When this feature is used, it is more useful to monitor the state name rather than the individual state registers. The state name can be monitored easily in the Waveform Editor by entering waveforms for the individual state register bits and combining them into a bus with the same name as the state machine. The waveforms have the associated state names displayed in a text format inside the bus. Figure 5 shows the waveforms for the TDF in Figure 4.

Figure 5. State Names in the Waveform Editor



Present States

There are three ways to set up the Waveform Editor to show the present state of a state machine. The easiest is to use the default channel file that automatically places the state machine bus in the Simulator Channel File (**.SCF**). Another method is to enter the state machine name in an ASCII Vector File (**.VEC**) by entering the state machine name (including its hierarchy information) in the Outputs Section. The third method, entering the state machine into a Simulator Channel File, requires these steps:

1. Enter the waveforms for the state register bits individually into the Waveform Editor, with each node corresponding to a state register bit.
2. Group the nodes together and make the group name the same as the name of the state machine.

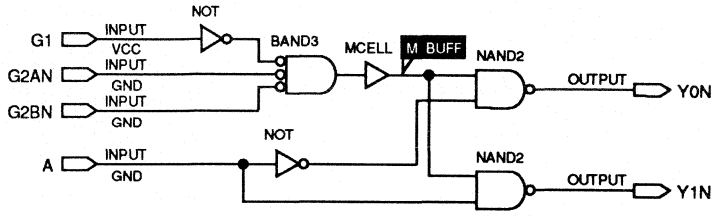
Whichever method is used, the appropriate state information will appear in the Simulator Channel File after simulation.

Simulating Inaccessible Combinatorial Nodes

Some combinatorial nodes cannot be directly simulated. To simulate buried combinatorial logic, the node should be fed into an **MCELL** buffer, then a probe should be placed on the **MCELL** and named. Next, the design should be recompiled to generate an updated Simulator Netlist File. The **MCELL** should be removed once the node is shown to be functionally correct.

Figure 6 shows a combinatorial circuit for a decoder with three enable lines called **G1**, **G2AN**, and **G2BN**. To activate the outputs, all three of the enable lines feeding the **BAND3** gate must be active. To simulate the combinatorial logic in this design, an **MCELL** may be placed in the circuit between the **BAND3** primitive and the **NAND2** primitive. (The timing of the circuit will change.) This technique should be used to verify functionality, not timing.

Figure 6. Placing an MCELL for Simulation of Combinatorial Logic



Finding Nodes

Once all nodes that can be simulated have been identified, it may be useful to find their locations in the schematic or text file. The **LNF (Line : Node : Find)** command in both the Graphic Editor and the Text Editor helps locate nodes that have hierarchical node names:

1. Type **LNF (Line : Node : Find)**.
2. Type the internal node name and press **<Enter>**.

The schematic or file containing the node is then loaded and the node is highlighted.

A quick way to find a node is to use the **LNF (Line : Node : Find)** command within the schematic that contains the node. For example, if **LNF (Line : Node : Find)** is selected while the file **REG_EN** is displayed, the prompt line displays **:REG_EN:6**, which is the hierarchy path that leads to the current file. Typing **<End> :8 <Enter>** then highlights the **AND2** gate. If a node is identified with a probe, only the probe name must be entered.

Using Internal Nodes in Simulation

Nodes can quickly be placed into a Simulator Channel File or Vector File for simulation. Internal nodes are added to the Outputs Section of a Vector File by importing the names from the node list in the History File. Figure 7 shows an example of a Vector File created from a History File node list with I/O pins and internal nodes grouped together to form buses.

The Waveform Editor automatically creates a default Simulator Channel File that contains the device I/O pins, internal nodes with probes, and nodes associated with state machines. Additional nodes that need to be simulated can be appended to this file.

10

Figure 7. Creating Groups in a Vector File from an Internal Node List

```
--- NODE LIST ---

 Type        Name
 ----        ----
 IN          CLOCK
 IN          DIR
 IN          S
 IN          D0
 IN          D1
 IN          D2
 IN          D3
 I/O         B0
 I/O         B1
 I/O         B2
 I/O         B3
 OUT         :5
 OUT         :6
 OUT         :7
 OUT         :8
 OUT         |21MUX:9|:5
 OUT         |21MUX:12|:5
OUT          |21MUX:10|:5
 OUT         |21MUX:11|:5
```

```
--- VECTOR FILE ---

GROUP CREATE B_BUS   = B3 B2 B1 B0;

GROUP CREATE D_BUS   = D3 D2 D1 D0;

GROUP CREATE INT_BUS = |21MUX:9|:5
                       |21MUX:12|:5
                       |21MUX:10|:5
                       |21MUX:11|:5;

OUTPUTS B_BUS INT_BUS;
```

Conclusion

The MAX+PLUS Simulator incorporates many features. With the Graphic Editor's probe capability and the integrated Altera Hardware Description Language (AHDL), all nodes that can be simulated are easily identified. Additionally, both AHDL Text Design Files and Graphic Design Files use a parallel method to identify internal nodes, making simulation quick and simple.

Emulating Internal Buses in General-Purpose EPLDs

October 1990, ver. 2

Application Brief 82

Introduction

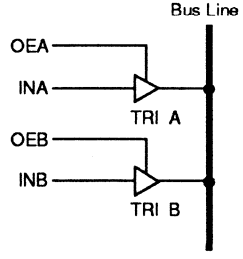
Altera's general-purpose (EP- and EPM5000- series) EPLDs allow internal buses to be emulated by using logic to replace tri-state functions. A series of simple 2-to-1 multiplexers can create buses with two sets of input signals. 4-to-1 (and larger) multiplexers can create buses with three or more sources. Multiplexing also saves device resources and helps to eliminate timing and loading problems. This application brief describes how to use multiplexers for different bus configurations and explains the benefits of this approach.

Two-Source Bus Configurations

Figure 1 shows the simplest bus configuration, a one-bit bus created by connecting the outputs of two tri-state buffers to a single line. The function table shows the possible states of the bus. When tri-state buffer A is enabled, the input to that buffer (**INA**) appears on the bus. When tri-state buffer B is enabled, the input to that buffer (**INB**) appears on the bus. If neither buffer is enabled, the bus is in a high-impedance, or floating, state. Note that buses are often tied high with a pull-up resistor to prevent them from floating.

Figure 1. One-Bit Bus

Two tri-state buffers can create a simple bus.

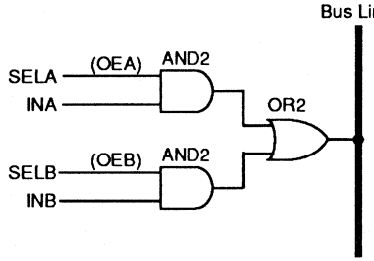


INA	OEA	INB	OEB	BUS LINE
X	0	X	0	Z or 1
X	0	0	1	0
X	0	1	1	1
0	1	X	0	0
1	1	X	0	1

Figure 2 shows two AND gates and an OR gate that emulate the tri-state functions of Figure 1. Each AND gate has a data input (**INA** or **INB**), and a select input (**SELA** or **SELB**) that represents the original Output Enable control. The function table shows that the AND/OR logic exactly emulates the original tri-state functions if one of the two outputs is always selected. If neither output is selected, the output of the AND/OR logic is low.

10

Figure 2. AND/OR Logic Emulating Tri-Stating Functions

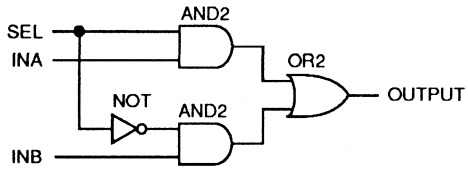


INA	SELA	INB	SELB	BUS LINE
X	0	X	0	0
X	0	0	1	0
X	0	1	1	1
0	1	X	0	0
1	1	X	0	1

The select controls are mutually exclusive, since only one input is ever enabled onto a bus at any given time. Therefore, they can be encoded into a single input by making **SELA** the common select input, and then feeding the inverse of this signal into the previous **SELB** input. Figure 3 shows that these steps transform the AND/OR logic into a typical 2-to-1 multiplexer. The multiplexer functions in the same way as the AND/OR logic in Figure 2, except that the two select signals have been encoded into a single line.

Figure 3. Multiplexer Created with AND/OR Logic and Select Controls

Encoded select lines transform the AND/OR logic from Figure 2 into a multiplexer.



INA	INB	SEL	OUTPUT
X	0	0	0
X	1	0	1
0	X	1	0
1	X	1	1

Additional 2-to-1 multiplexers, all controlled by a common select signal, can create wider buses. One multiplexer is necessary for each bit of the bus. For example, Figure 4 shows eight 2-to-1 multiplexers emulating a byte-wide bus.

Buses with Three or More Sources

Larger multiplexers with multiple select inputs can emulate buses with more than two sources. Figure 5 shows how a 4-to-1 multiplexer can create a bus with up to four sources. Figure 5 also includes a truth table with the proper encoding for the select inputs. This type of multiplexer can also implement buses with two or three sources.

Additional multiplexers with shared select lines can create buses with nearly any width. For example, five 4-to-1 multiplexers can create a 5-bit-wide bus with two, three, or four sets of inputs.

Figure 4. Eight 2-to-1 Multiplexers Emulating a Byte-Wide Bus

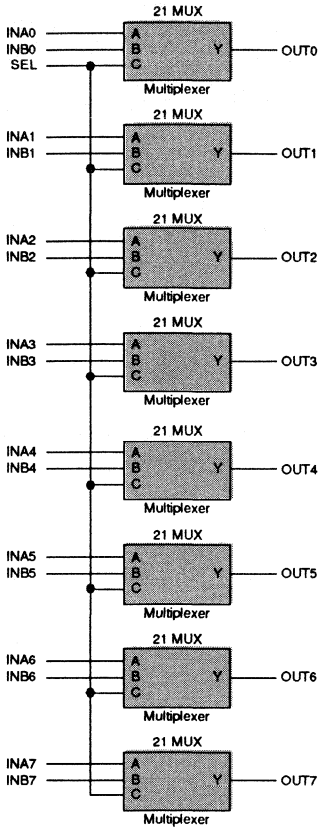
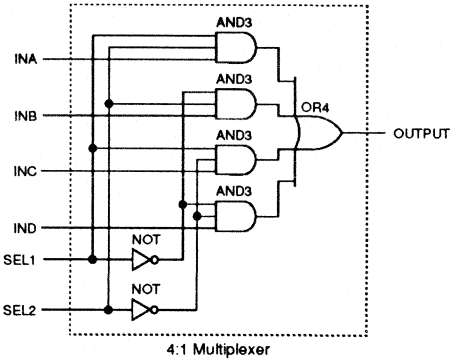


Figure 5. 4-to-1 Multiplexer Implementing a Bus with Up to Four Sources



INA	INB	INC	IND	SEL2	SEL1	OUTPUT
X	X	X	0	0	0	0
X	X	X	1	0	0	1
X	X	0	X	0	1	0
X	X	1	X	0	1	1
X	0	X	X	1	0	0
X	1	X	X	1	0	1
0	X	X	X	1	1	0
1	X	X	X	1	1	1

Implementing Bus Functions with AHDL

For MAX+PLUS users, the Altera Hardware Description Language (AHDL) provides a quick alternative to graphic schematic entry for implementing bus functions with multiplexing. AHDL files can generate buses with nearly any number of inputs and of nearly any width.

For example, Figure 6 shows the lines of AHDL code required to create an eight-bit bus with three sources. The data inputs are **A7** to **A0**, **B7** to **B0**, and **C7** to **C0**. The two select inputs, **SEL1** and **SEL2**, can be treated as an encoded group in AHDL. These select lines control which set of input signals is connected to the outputs through a series of simple **IF-THEN** statements.

10

Figure 6. AHDL Implementation of Eight-Bit Bus with Three Sources

```
SUBDESIGN BUSMUX (

   A[7..0],
   B[7..0],
   C[7..0],
   SEL[1..0]: INPUT;
   OUT[7..0]: OUTPUT; )
BEGIN
   IF (SEL[]==0) THEN OUT[]=A[]; END IF;
   IF (SEL[]==1) THEN OUT[]=B[]; END IF;
   IF (SEL[]==2) THEN OUT[]=C[]; END IF;

END;
```

By adding data inputs (e.g., **D[7..0]**), this file can be easily modified to create a bus with more sources. For multiplexers with more than four data inputs, one more bit must also be added to the **SEL** group (e.g., **SEL[2..0]**) for each factor-of-two increase in the number of data inputs. For example, a seven-input multiplexer requires three select bits.

The width of the bus can be varied by changing the input and output group widths. For example, the declaration **A[5..0]** creates a 5-bit-wide set of **A** inputs.

For more information on AHDL syntax, refer to *Application Note 22 (Designing with AHDL)*.

Why Emulate Tri-State Functions?

Using multiplexing to emulate tri-state functions saves macrocells and I/O pins for applications that would otherwise require a bus external to the EPLD. Figure 7 shows a 4-to-1 multiplexer in a single macrocell that emulates a bus line with four sources. With conventional tri-stating techniques, the same function requires four macrocells and I/O pins, as shown in Figure 8. Multiplexing saves three macrocells and I/O pins if the switching functions are implemented with the product terms inside the macrocell, instead of with tri-state buffers and I/O pins external to the macrocell.

Figure 7. 4-to-1 Multiplexer in a Single Macrocell

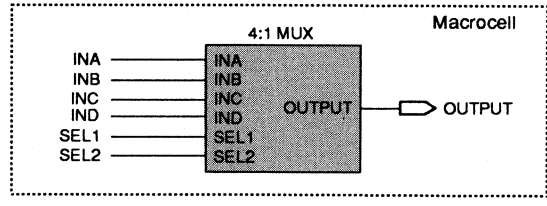
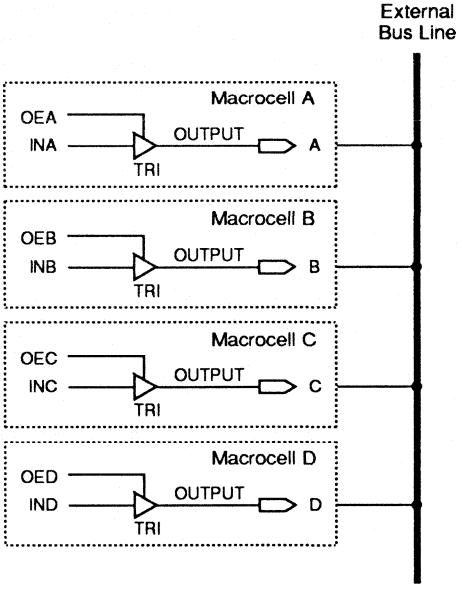


Figure 8. 4-to-1 Multiplexer Implemented with Traditional Tri-State Logic

A four-source bus that uses tri-stating requires four macrocells and I/O pins.



Emulating tri-stated buses with logic eliminates timing hazards such as bus contention, which occurs when two or more tri-state outputs are simultaneously enabled onto a single bus line. This condition (usually unintended) can cause an unpredictable logic level to propagate if multiple buffers are driving high and low at the same time. The select controls for simple AND/OR logic (shown in Figure 2) can both be enabled at the same time, but the result will be a known logic level. The select controls can never be enabled at the same time if they are encoded, as in the true multiplexer configurations (Figure 3).

The only potential timing hazard for a multiplexer configuration is output glitching caused by input signal skew. EPLD architecture minimizes skew difficulties and glitching is seldom a problem in actual designs. However, the designer must exercise care when driving edge-sensitive logic from multiplexer outputs.

Replacing tri-stated buses with logic reduces capacitive loading limitations. High fan-outs to traditional buses create high capacitive loads that slow bus bandwidth. Macrocells and feedback paths in EPLDs have constant delays, regardless of the number of signals entering the macrocell. If control logic is implemented with multiplexers, internal bus loading is not a problem.

Conclusion

Although Altera's general-purpose EP- and EPM5000-series EPLDs do not have internal tri-state capabilities, the tri-state function can be emulated with multiplexing. The multiplexing technique allows designers to create buses of nearly any size in the EPLD. Multiplexing also provides the additional benefits of saving device resources, and eliminating timing and loading problems.

Programmable Frequency Divider with the EP630 EPLD

October 1990, ver. 2

Application Brief 83

Introduction

Altera's EP630 EPLD combines the industry-standard EP600-series architecture with advanced 1.0-micron CMOS EPROM technology to produce a high-speed (t_{PD} = 15 ns) 24-pin EPLD. This application brief compares the EP630 EPLD with the 22V10 device, focusing on architecture and design support. An example that illustrates important EP630 features is also included.

Architecture

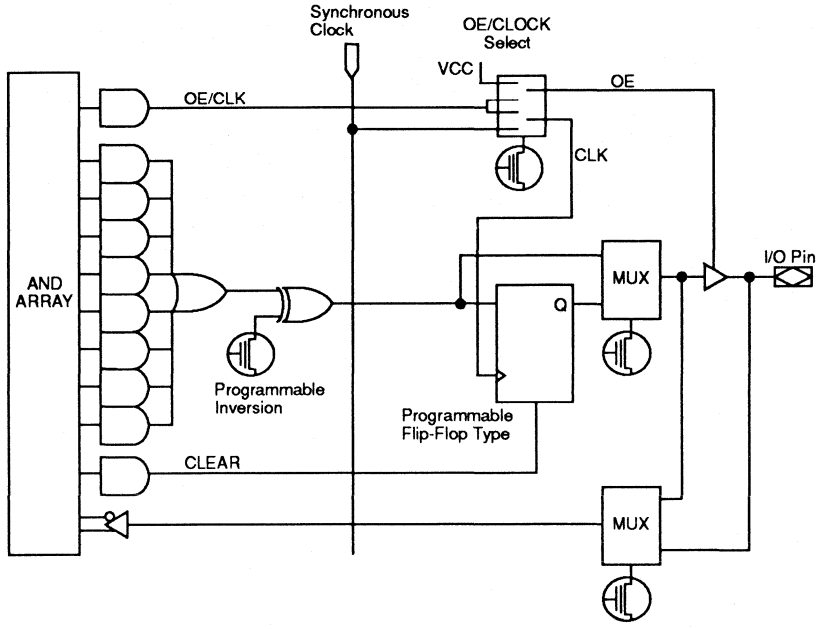
The EP630 EPLD and the 22V10 device have several features in common: both are available in a 24-pin package, both generate Boolean logic expressions with a sum-of-products array, and both process Boolean functions through macrocells (see Figure 1).

However, the two devices differ in important ways:

- The most obvious difference is the number of macrocells offered: the EP630 EPLD contains 16 macrocells, the 22V10 contains only 10.
- All EP630 macrocells contain a programmable flip-flop that can be configured for D, T, JK, or SR operation, thus substantially increasing its flexibility. In contrast, the 22V10 has a fixed D flip-flop architecture. For example, a 10-bit counter implemented in a 22V10 requires 10 product terms to generate the tenth bit, while the same counter in an EP630 EPLD requires only a single product term for each bit when the internal registers are configured for T flip-flop emulation.
- The EP630 registers are individually configured to clock from a global or asynchronous clock, while all of the 22V10 registers are clocked from a single global clock source.
- The EP630 EPLD has 128 equally distributed product terms, 8 per macrocell. In addition, individual product terms are added to the Clear and Output Enable inputs of each EP630 macrocell, giving the designer much greater flexibility. The 22V10 has variable product-term distribution and contains anywhere from 8 to 16 product terms per macrocell.
- Maximum clock frequency for the EP630 EPLD is 83 MHz; for a CMOS 22V10, it is 80 MHz.
- An EP630 requires 90 mA current; a 22V10 requires 130 mA current.

Figure 1. Macrocells in the EP630 EPLD and the 22V10 Device

EP630 Macrocell



22V10 Macrocell

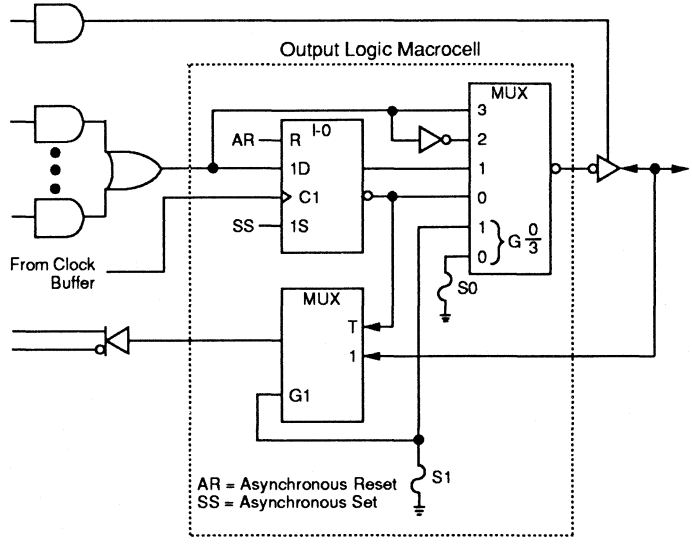


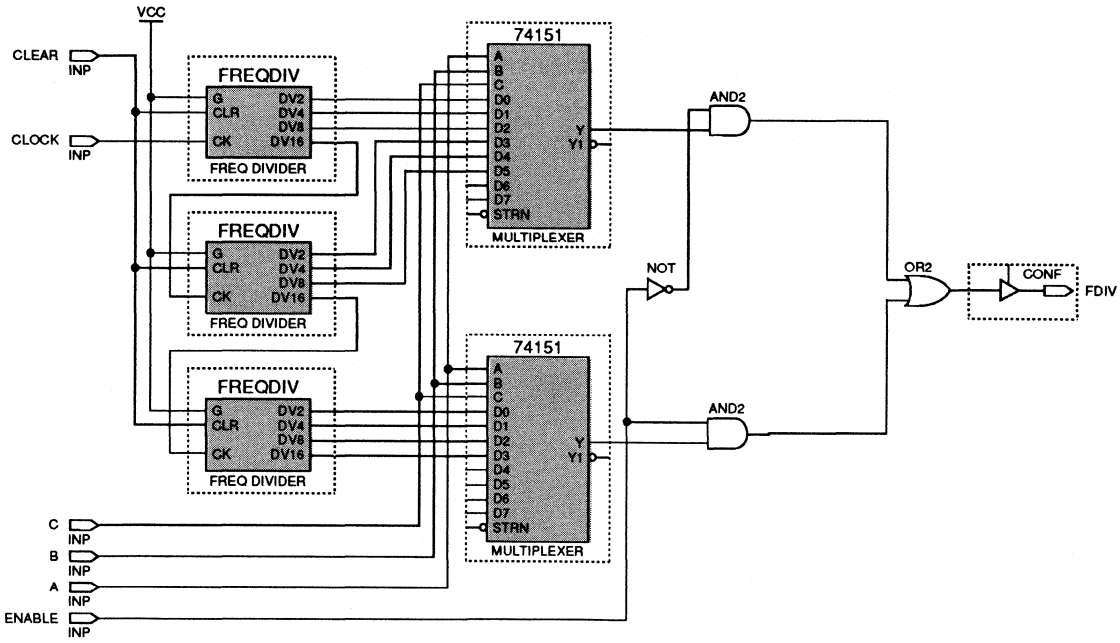
Table 1 summarizes these features.

Table 1. EP630 EPLD vs. 22V10

Features	EP630	22V10
Speed:		
t_{PD}	15 ns	15 ns
f_{CNT}	83 MHz	80 MHz
Macrocells	16	10
Programmable Clocks	Yes	No
Programmable Flip-Flops	Yes	No
Zero Power	Yes	No

The EP630 architecture provides greater register density and flexibility than the 22V10, allowing more logic to be incorporated. For example, Figure 2 shows a programmable frequency divider, a common application for PLDs.

Figure 2. Programmable Frequency Divider



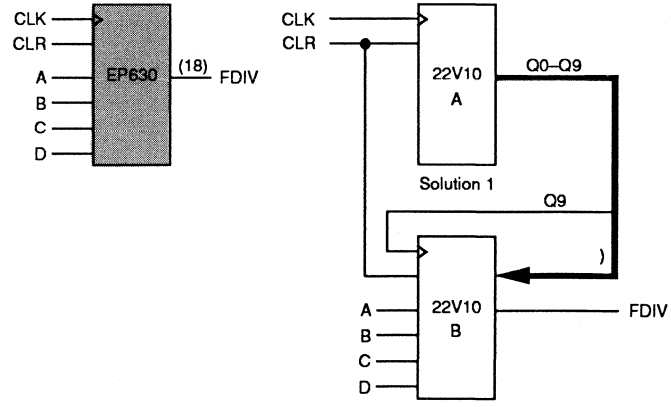
This frequency divider accepts a frequency, divides it by powers of two (2^1, 2^2, 2^3,...), then produces a selected output frequency based upon frequency-select inputs. Frequency division is accomplished by three cascaded frequency dividers, each receiving a clock from either an input or the previous frequency divider. The divider frequencies are connected to two 74157 multiplexers, each of which routes a selected clock—defined by the select address **(A,B,C)**—to its output. Finally, a single frequency is chosen by the 2-to-1 multiplexer, which is implemented with gates.

The EP630 Solution

The EP630 is a better choice than the 22V10, both for hardware implementation and for software design entry.

As Figure 3 shows, the EP630 EPLD easily integrates the programmable frequency divider into a single device to produce the selected frequency **FDIV**. On the other hand, the 22V10 device is not well suited to this application. Since it contains only 10 macrocells, two 22V10 devices are needed to implement the 12-bit counter. Device A must function as a 10-bit counter; its outputs **Q0** to **Q9** must supply part of the input to device B. Device B is clocked by **Q9** forming a ripple counter to create the two most significant bits of the 12-bit counter function. Frequency-select inputs to device B control the clock output multiplexer that ultimately produces the selected frequency, **FDIV**.

Figure 3. EPB630 vs. 22V10 as Programmable Frequency Divider



The logic design for the EP630 EPLD is entered, compiled, verified, and programmed with the Altera Programmable Logic User System (A+PLUS). Design entry is simple with LogiCaps schematic capture. LogiCaps provides access to over 100 Altera-supplied TTL macrofunctions (see Figure 2), which speed up design entry for designs such as this programmable frequency divider, allowing the designer to enter and connect the appropriate TTL macrofunctions. The Altera Design Processor provides

algorithms that automatically fit the programmable frequency divider into the EP630, and produces a standard JEDEC file to program the device. During compilation, A+PLUS automatically performs logic reduction, configures the data paths within the macrocells, and chooses the best flip-flop configuration (D, T, JK, or SR). The result is a working design produced quickly and efficiently.

The 22V10 design, on the other hand, is entered with Texas Instruments' Prologic PLD compiler (shown in Figure 4). Since Prologic is a text-based entry language, each counter and multiplexer must be expressed as a separate Boolean function. This is a time consuming, error-prone task.

Conclusion

Together, the EP630 EPLD and A+PLUS software offer the ideal solution for implementing a programmable frequency divider in a single device.

Figure 4. Frequency Divider File Created with Prologic (Part 1 of 4)

```
Listing 1:  Device A


title (  Device:   PAL22V10
     Application:  12 Bit Programmable Frequency Divider:  Device A
     Source:       Designer Name      9/89   )

include   p22v10:       /* specify that target device is PAL22V10 */

define    CLK = pin1   : /* define input pins */
define    CLR = pin2   :

define    Q0  = pin14 : /* define output pins */
define    Q1  = pin15 :
define    Q2  = pin16 :
define    Q3  = pin17 :
define    Q4  = pin18 :
define    Q5  = pin19 :
define    Q6  = pin20 :
define    Q7  = pin23 :
define    Q8  = pin22 :
define    Q9  = pin21 :

/* define equations to implement lower 10 bits of counter */

Q0.d = ! (Q0.q) & !CLR ;

Q1.d = (!Q0.q & Q1.q # Q0.q & Q1.q) & !CLR ;

Q2.d = (!Q0.q & Q2.q # !Q1.q & Q2.q # Q0.q & Q1.q & !Q2.q) & !CLR ;

Q3.d = (!Q0.q & Q3.q
      # !Q1.q & Q3.q
      # !Q2.q & Q3.q
      # Q0.q & Q1.q & Q2.q & !Q3.q) & !CLR ;

Q4.d = (!Q0.q & Q4.q
      # !Q1.q & Q4.q
      # !Q2.q & Q4.q
      # !Q3.q & Q4.q
      # Q0.q & Q1.q & Q2.q & Q3.q & !Q4.q) & !CLR ;
```

10

Figure 4. Frequency Divider File Created with Prologic (Part 2 of 4)

```
Q5.d = (!Q0.q & Q5.q
       ! !Q1.q & Q5.q
       ! !Q2.q & Q5.q
       ! !Q3.q & Q5.q
       ! !Q4.q & Q5.q
       ! Q0.q & Q1.q & Q2.q & Q3.q & Q4.q & !Q5.q) & !CLR ;

Q6.d = (!Q0.q & Q6.q
       ! !Q1.q & Q6.q
       ! !Q2.q & Q6.q
       ! !Q4.q & Q6.q
       ! !Q3.q & Q6.q
       ! !Q5.q & Q6.q
       ! Q0.q & Q1.q & Q2.q & Q3.q & Q4.q & Q5.q & !Q6.q) & !CLR ;

Q7.d = (!Q0.q & Q7.q
       ! !Q1.q & Q7.q
       ! !Q2.q & Q7.q
       ! !Q3.q & Q7.q
       ! !Q4.q & Q7.q
       ! !Q5.q & Q7.q
       ! !Q6.q & Q7.q
       ! Q0.q & Q1.q & Q2.q & Q3.q & Q4.q & Q5.q & Q6.q & !Q7.q) & !CLR ;

Q8.d = (!Q0.q & Q8.q
       ! !Q1.q & Q8.q
       ! !Q2.q & Q8.q
       ! !Q4.q & Q8.q
       ! !Q3.q & Q8.q
       ! !Q5.q & Q8.q
       ! !Q6.q & Q8.q
       ! !Q7.q & Q8.q
       ! Q0.q & Q1.q & Q2.q & Q3.q & Q4.q & Q5.q & Q6.q & Q7.q & !Q8.q) &
         !CLR ;

Q9.d = (!Q0.q & Q9.q
       ! !Q1.q & Q9.q
       ! !Q2.q & Q9.q
       ! !Q4.q & Q9.q
       ! !Q3.q & Q9.q
       ! !Q5.q & Q9.q
       ! !Q6.q & Q9.q
       ! !Q7.q & Q9.q
       ! !Q8.q & Q9.q
       ! !Q0.q & Q1.q & Q2.q & Q3.q & Q4.q & Q5.q & Q6.q & Q7.q & Q8.q &
         !Q9.q) & !CLR ;

/* permanently enable all counter outputs */
Q0.oe = 1;     Q1.oe = 1;     Q2.oe = 1;     Q3.oe = 1;     Q4.oe = 1;
Q5.oe = 1;     Q6.oe = 1;     Q7.oe = 1;     Q8.oe =1;      Q9.oe = 1;

/* define outputs as active high */
Q0 = q;        Q1 = q;        Q2 = q;        Q3 = q;        Q4 = q;
Q5 = q;        Q6 = q;        Q7 = q;        Q8 = q;        Q9 = q;
```

Figure 4. Frequency Divider File Created with Prologic (Part 3 of 4)

```
/* define some test vectors to verify that counter is working properly */
test_vectors ( /*    CLK     CLR     Q3      Q2      Q1      Q0      */
                     pin1    pin2    pin17   pin16   pin15   pin14;
                     c       1       L       L       L       L       ;  /*RESET*/
                     c       1       L       L       L       L       ;  /*RESET*/
                     c       0       L       L       L       H       ;  /*  1  */
                     c       0       L       L       H       L       ;  /*  2  */
                     c       0       L       L       H       H       ;  /*  3  */
                     c       0       L       H       L       L       ;  /*  4  */
                     c       0       L       H       L       H       ;  /*  5  */
                     c       0       L       H       H       L       ;  /*  6  */
                     c       0       L       H       H       H       ;  /*  7  */
                     c       0       H       L       L       L       ;  /*  8  */
                     c       0       H       L       L       H       ;  /*  9  */
                     c       0       H       L       H       L       ;  /*  10 */
                     c       0       H       L       H       H       ;  /*  11 */
                     c       0       H       H       L       L       ;  /*  12 */
                     c       0       H       H       L       H       ;  /*  13 */
                     c       0       H       H       H       L       ;  /*  14 */
                     c       0       H       H       H       H       ;  /*  15 */
                     c       0       L       L       L       L       ;  /*  0  */
                     c       1       L       L       L       L       ;  /*RESET*/
```

Listing 2: Device B

```
title (  Device:          PAL22V10
         Application:     12 Bit Programmable Frequency Divider:  Device B
                          2MSB and Multiplexing Control
         Source:                                          9/89   )

include   p22v10;          /* specify that target device is PAL22V10 */

define    CLK = pin1  ;    /* clock input is from Q9 on device A     */
define    CLR = pin2  ;    /* clear goes to pin 2 on both devices    */

define    A = pin3 ;       /* select inputs for multiplexing outputs */
define    B = pin4 ;       /*    Select Input     Q Output           */
define    C = pin5 ;       /*      ABCD = 0     divided by 2         */
define    D = pin6 ;       /*             1               4         */
                           /*             2               8 ..etc   */

define    Q0 = pin7  ;     /* define counter inputs from device A    */
define    Q1 = pin8  ;
define    Q2 = pin9  ;
define    Q3 = pin10 ;
define    Q4 = pin11 ;
define    Q5 = pin13 ;
define    Q6 = pin14 ;
define    Q7 = pin15 ;
define    Q8 = pin16 ;
define    Q9 = pin17 ;

define    Q10 = pin19 ;    /* define 2 MSB of 12 bit counter */
define    Q11 = pin20 ;

define    FDIV = pin18 ;   /* divided output */

/* define equations to implement 2 MSB of counter */

     Q10.d = !(Q10.q) & !CLR ;

     Q11.d = (!Q10.q & Q11.q | Q10.q & Q11.q) & !CLR ;
```

10

Figure 4. Frequency Divider File Created with Prologic (Part 4 of 4)

```
/* define equations to control multiplexing of outputs */

     FDIV =    Q0   &  !A  &  !B  &  !C  &  !D
          |    Q1   &  !A  &  !B  &  !C  &   D
          |    Q2   &  !A  &  !B  &   C  &  !D
          |    Q3   &  !A  &  !B  &   C  &   D
          |    Q4   &  !A  &   B  &  !C  &  !D
          |    Q5   &  !A  &   B  &  !C  &   D
          |    Q6   &  !A  &   B  &   C  &  !D
          |    Q7   &  !A  &   B  &   C  &   D
          |    Q8   &   A  &  !B  &  !C  &  !D
          |    Q9   &   A  &  !B  &  !C  &   D
          |  Q10.q &   A  &  !B  &   C  &  !D  /* ".q"was used on terms  */
          |  Q11.q &   A  &  !B  &   C  &   D  /* are internal feedbacks */

/* permanently enable all outputs */
 Q10.oe = 1;  Q11.oe = 1;   FDIV.oe = 1;

/* define outputs as active high */
 Q10 = q;     Q11 = q;
```

DMA Controller with the EPM5064 MAX EPLD

October 1990, ver. 1

Application Brief 84

Introduction

A Direct Memory Access (DMA) controller can increase the performance of a peripheral subsystem by coordinating data transfer between peripheral and subsystem memory. A DMA controller is useful when a design requires data transfer rates that are too fast for a microprocessor. DMA can be used by a disk-drive controller to quickly transfer large blocks of data, by high-speed serial subsystems to maintain communications link bit rates, and by dedicated graphics processors that update images stored within video memory.

DMA controllers can be implemented in several ways. Standard off-the-shelf DMA controllers are available (e.g., Am9517A/8237A), providing a single-chip, application-specific solution. However, these controllers may be unsuitable due to speed, power consumption, or protocol requirements. For example, the Am9517A transfers data at 2.5 Mbytes per second and requires a specific set of control commands, which is normally not available with standard DMA controllers.

If higher performance or custom functions are required, a DMA controller can be implemented with discrete TTL components (e.g., the 7400 series). This approach supports custom DMA protocols and tailored performance, but has a number of drawbacks: TTL logic consumes high power, requires a large amount of PC board space, and the high component count and power dissipation reduces overall system reliability.

Erasable Programmable Logic Devices (EPLDs) offer the ideal solution, integrating the advantages of the standard off-the-shelf DMA controller with the higher performance and flexibility of the TTL approach. This application brief illustrates how a DMA controller implemented in an Altera EPM5064 EPLD achieves data transfer rates of up to 20 megawords per second between peripheral and subsystem memory.

EPM5064 Overview

The EPM5064 is a user-configurable, high-performance, high-density MAX EPLD. It offers a fast 25-ns input-to-output combinatorial delay and 50-MHz 16-bit counter frequencies. This EPLD is offered in 44-pin windowed ceramic and plastic J-lead chip carrier packages that have 8 dedicated inputs and 28 bidirectional I/O pins. Commercial, industrial, and military temperature-range versions are available.

The EPM5064 contains 64 macrocells, each with a register that can be programmed for D, T, JK, SR, or flow-through latch operation, or bypassed entirely for purely combinatorial functions. All macrocell registers also

include asynchronous Clear and Preset controls. The macrocells are grouped into 4 Logic Array Blocks (LABs), each containing 16 macrocells and 32 expander product terms. Expander product terms are freely allocatable product terms that can be used and shared by any macrocell in the LAB.

A Programmable Interconnect Array (PIA) routes signals between the various LABs. The PIA, which is fed by the 28 I/O pins and 64 macrocell feedbacks, provides the resources necessary to ensure 100% signal routability. A fixed interconnect delay across the PIA eliminates skew and provides consistent, predictable performance. For more information on device architecture, refer to the *EPM5016 to EPM5192: High-Speed, High-Density MAX EPLDs Data Sheet* in this data book.

DMA Process

Figure 1 shows a DMA controller that supports the Microprocessor Unit (MPU), a peripheral, and the memory block of a peripheral subsystem.

Figure 1. Sample Peripheral Subsystem with an EPM5064 DMA Controller

A peripheral subsystem can be designed for any custom bus protocol with a custom EPLD DMA controller.

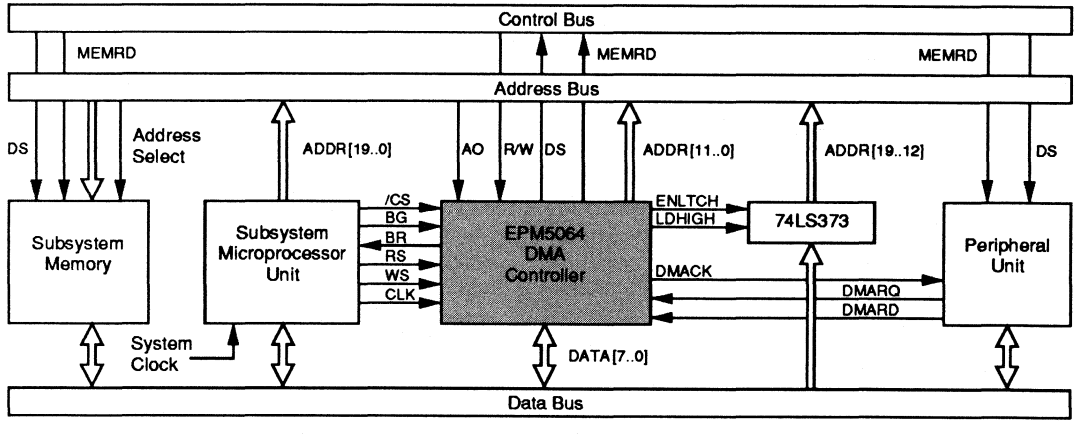
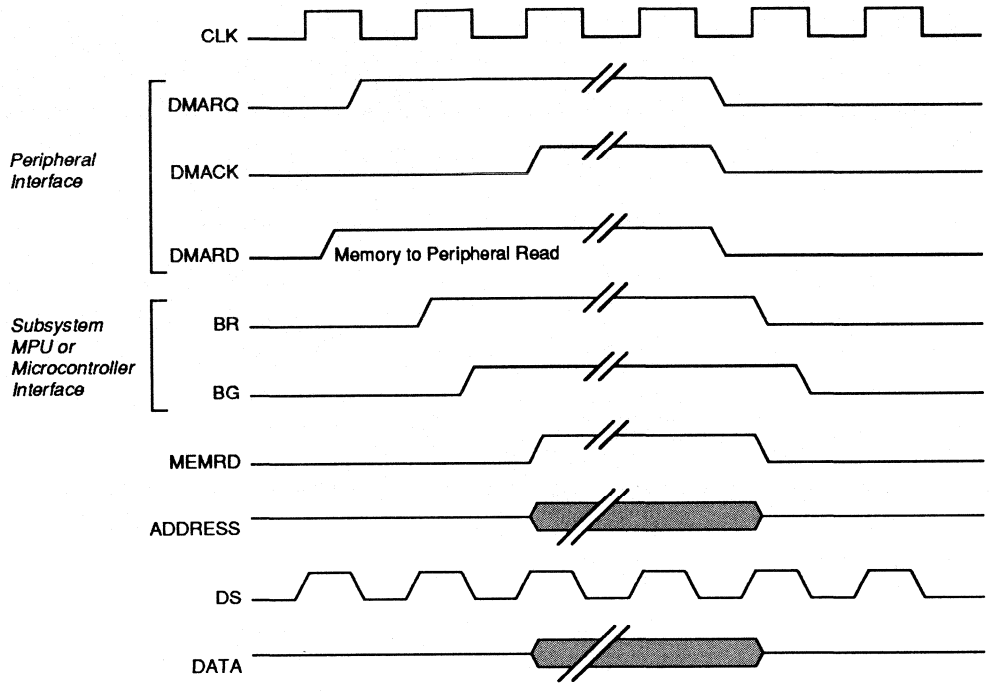


Figure 2 shows the timing associated with the DMA control signals. A custom-designed DMA controller in an EPM5064 can improve performance of the peripheral subsystem by providing faster data transfers than are possible with the MPU alone. Using a DMA controller also frees the MPU to perform other tasks.

Figure 2. Generic DMA Timing



To start the DMA process, the MPU must first select and initialize the DMA controller; then it must write the starting and ending addresses and the control information to the controller. The control data indicates whether the transfer addresses should be incremented or decremented.

After initialization, the peripheral starts the DMA transfer by asserting the **DMARQ** (DMA request) input to the controller. **DMARD** (DMA read), another input from the peripheral, indicates the direction of the DMA transfer. When **DMARD** is high, the current DMA cycle is a memory-read cycle; when it is low, it is a memory-write cycle.

After **DMARQ** and **DMARD** are asserted, the DMA controller asserts the **BR** (bus request) input to the MPU . The MPU then drives **BG** (bus grant) high and releases control of the buses. The DMA controller asserts **DMACK** (DMA acknowledge) to inform the peripheral that it controls the buses, and the DMA cycle can begin.

10

To perform the DMA transfer, the control and address buses must transfer data between peripheral and subsystem memory. **DS** (data strobe) is a control bus signal, driven by the DMA controller, that strobes the data between the peripheral and subsystem memory during the DMA transfer. The DMA controller simultaneously drives **MEMRD** (memory read) with the same logic level as **DMARD** and the address bus with the desired memory address. **MEMRD** indicates whether a transfer is a read (**MEMRD** high) or write (**MEMRD** low) operation.

On each clock cycle during the DMA transfer, the peripheral writes a new data word on the bus or reads a data word off the bus until all data words have been transferred. On every rising edge of **DS**, the DMA controller drives a new address, thus transferring a single data word. This process is called the "burst" mode of DMA transfer.

After the transfer is complete, the DMA controller bus signals are tri-stated, and bus control is returned to the MPU. The controller negates **BR** to inform the MPU that it is finished with the bus. Simultaneously, the address (**A[11..0]**), **DS**, and **MEMRD** signals are tri-stated. When the MPU regains bus control, it resumes execution from its previous state.

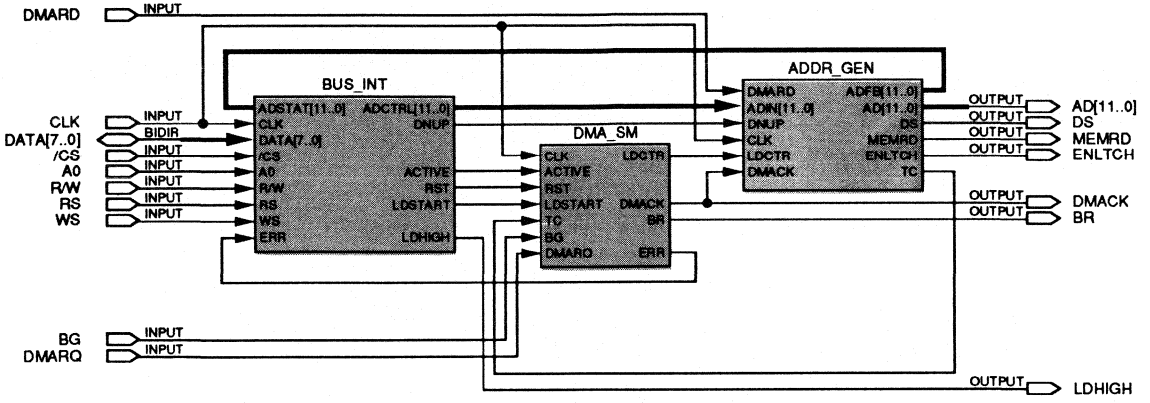
EPM5064 DMA Controller

The DMA controller subsystem is implemented with an EPM5064 EPLD and an external 74LS373 byte-wide address latch. This configuration supports a 20-bit address bus; wider buses can be supported with additional external address latches. The 74LS373 latches the upper 8 bits of the DMA transfer address from the subsystem data bus, while the lower 12 bits are generated by the EPM5064. The lower 12 bits allow DMA transfers of up to 4 K words without processor intervention. The EPM5064 controls the external address latch from the **LDHIGH** (load high-order address) and **ENLTCH** (enable address latch) signals. **LDHIGH** loads the upper address bits from the data bus into the address latch. **ENLTCH** enables the latch to drive its contents onto the address bus.

Figure 3 shows the functional block diagram of the EPM5064 as a DMA controller. The design consists of three basic blocks: a bus interface unit (**BUS_INT**), a DMA control state machine (**DMA_SM**), and an address generator (**ADDR_GEN**). Both **BUS_INT** and **ADDR_GEN** are hierarchical and contain other lower-level functions.

MAX+PLUS, Altera's software development tool for MAX EPLDs, supports hierarchical design entry. MAX+PLUS allows designs to be entered as schematics with the Graphic Editor or as text files with the Text Editor and Altera Hardware Description Language (AHDL). The MAX+PLUS package also includes a powerful compiler and timing simulator to provide a complete CAE system for the most complex designs. The EPM5064 DMA controller design, created in MAX+PLUS, consists of Graphic Design Files (GDFs) and Text Design Files (TDFs). For example, **BUS_INT** and **ADDR_GEN** are GDFs, while **DMA_SM** is a TDF.

Figure 3. DMA Controller Block Diagram



Bus Interface

Figure 4 shows **BUS_INT**, the portion of the design that implements the bus interface unit. The decoder, shown in the upper left corner of the figure, uses MPU signals to control data flow within the design. The 74244B, which is the functional equivalent of a TTL 74244 octal buffer, is an I/O buffer that supports the bidirectional data bus **DATA[7..0]**. The input/control registers receive all inputs from the data bus; the output/status registers store information to be read onto the data bus. **REG_SEL** selects data from one of the two output/status registers to appear at the I/O buffer. Together, the resources in this design support the following operations:

- Initialization of the EPM5064 DMA controller
- Write and Read operations

Initialization

The subsystem MPU communicates with the bus interface unit to select and initialize the EPM5064. The MPU must first address the DMA controller by driving **/CS** (chip select) low. A write operation is started when **A0** and **R/W** select one of the input/control registers and **WS** (write strobe) is asserted. For example, when **A0** is zero, **R/W** is zero, and **WS** has a rising edge, the eight bits of the data bus are written into input/control Register 1.

10

Figure 4. Bus Interface Unit Diagram

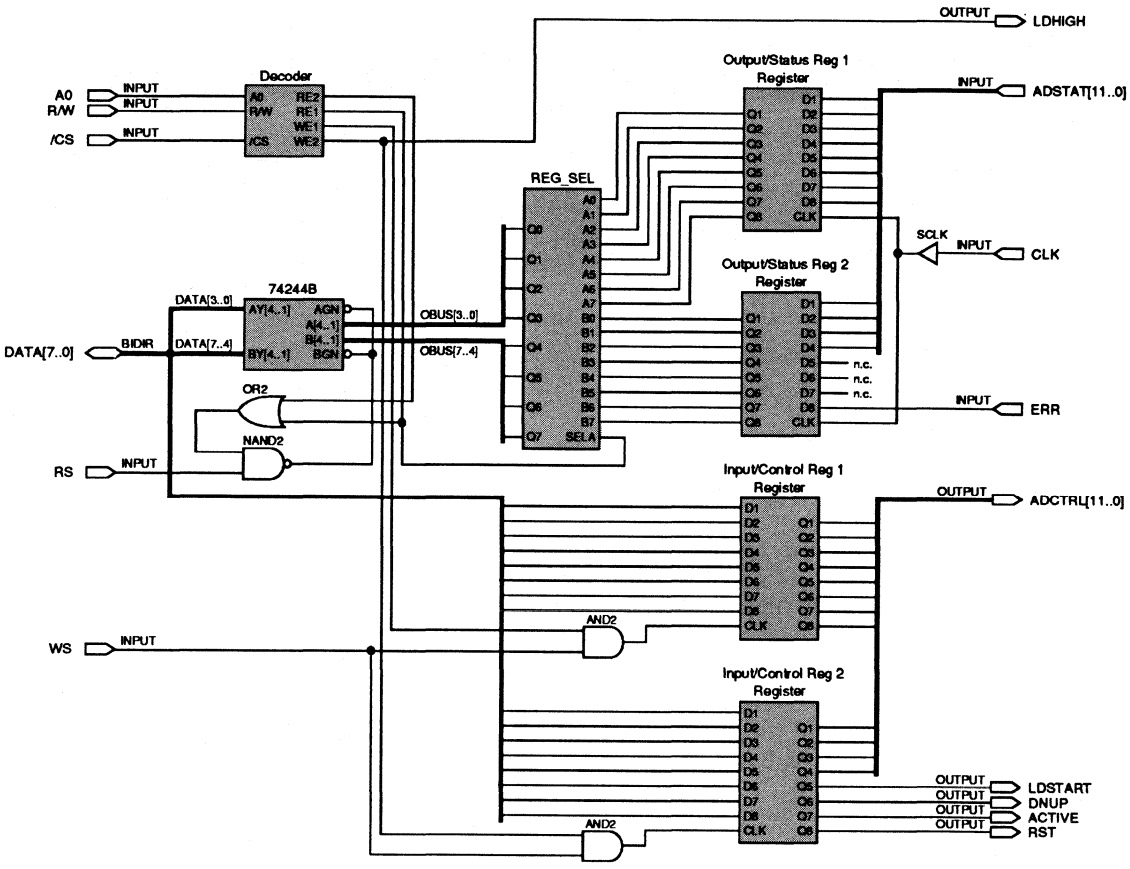


Table 1 shows the bus interface decoding scheme for the MPU signals.

Table 1. Bus Interface Decoding Scheme

RS	WS	/CS	R/W	A0	Action
⌐	L	L	H	H	Read output/status register 1
⌐	L	L	H	L	Read output/status register 2
L	⌐	L	L	H	Write input/control register 1
L	⌐	L	L	L	Write input/control register 2

Write Operation

After selecting the proper register and asserting **WS**, the MPU writes the least significant byte of the starting address, **A[7..0]**, into input/control Register 1. Then **A[11..8]** and the MPU control signals **ACTIVE** (DMA active), **LDSTART** (load starting address), **DNUP** (down up), and **RST** (reset) are written into input/control Register 2. **ACTIVE** and **LDSTART**, the input signals to the state machine, indicate that the starting address is to be transferred to the **ADDR_GEN** block. The **DNUP** signal specifies whether the DMA address will be decremented (**DNUP** high) or incremented (**DNUP** low). **RST**, the final MPU signal, reverts the DMA control state machine to the initial state.

Read Operation

During the DMA process, the MPU may read either of the output/status registers for the current DMA address or DMA controller status. The least significant byte of the DMA address, stored in output/status Register 1, is fed by the **ADDR_GEN** block. Output/status Register 2 receives the most significant nibble, **A[11..8]**, and the status signal, **ERR** (error). The **ERR** signal, which comes from the DMA control state machine, indicates an error condition during a DMA cycle. The **REG_SEL** function selects which output/status register feeds the I/O buffer. The I/O buffer is controlled by **RS** (read strobe) and provides the tri-stating necessary to ensure proper bidirectional operation.

DMA Control State Machine

Figure 5 shows the state diagram of **DMAC_SM**, the DMA control state machine. The state machine consists of five states: **INIT**, **LOAD**, **START**, **TRANSFER**, and **ERROR**. The state machine design file entered with AHDL, **DMAC_SM**, is shown in Figure 6. AHDL supports Boolean equations, truth tables, **IF-THEN** statements, and **CASE** statement constructs, that facilitate state machine design. **DMAC_SM.TDF** uses an **IF-THEN** statements to describe the state transitions.

INIT

DMAC_SM is in the **INIT** state at power-up. While in this state, the MPU enables the DMA controller. After the **ACTIVE** and **LDSTART** inputs are asserted, the machine proceeds to the **LOAD** state.

LOAD

When the **LOAD** state is entered, **LDCTR** (load counter) drives the starting address into **ADDR_CNT**. The MPU deasserts **LDSTART** after it has written the ending address to the DMA controller. The peripheral then activates **DMARQ**. When **LDSTART** is low and **DMARQ** is high, the machine enters the **START** state. **DMARQ** must remain asserted until the transfer is complete.

Figure 5. DMA Control State Diagram

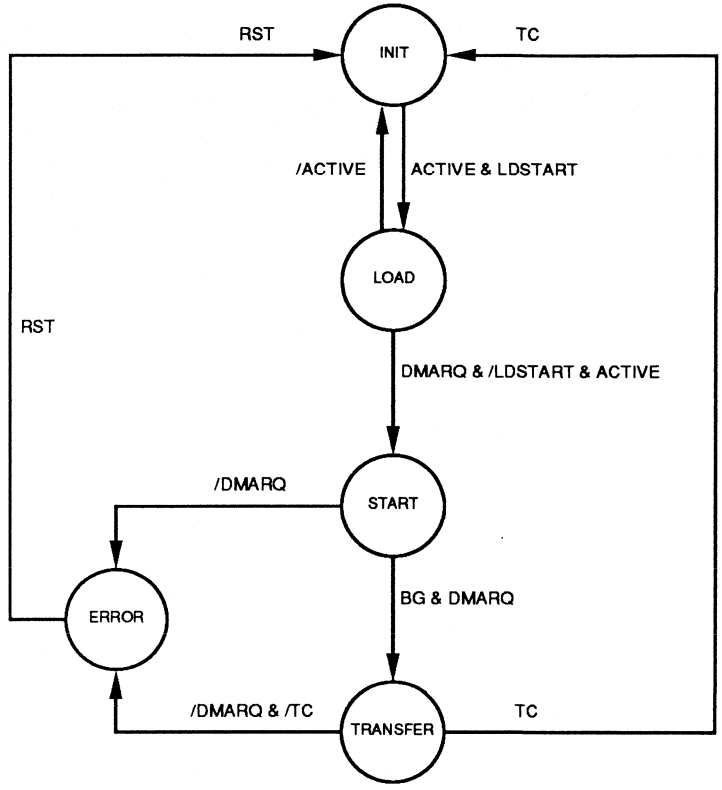


Figure 6. DMA_SM.TDF (Part 1 of 2)

```
TITLE   EPM5064 DMA Control State Machine;

SUBDESIGN   dma_sm

(
        %  State Machine Inputs  %

    CLK      :INPUT;     % subsystem clock                    %
    BG       :INPUT;     % bus grant                          %
    DMARQ    :INPUT;     % DMA request                        %
    ACTIVE   :INPUT;     % activate initialization sequence   %
    RST      :INPUT;     % reset error condition              %
    LDSTART  :INPUT;     % load starting address              %
    TC       :INPUT;     % terminal count                     %

        %  State Machine Outputs  %

    ERR      :OUTPUT;    % error condition %
    DMACK    :OUTPUT;    % DMA acknowledge %
    BR       :OUTPUT;    % bus request     %
    LDCTR    :OUTPUT;    % load counter    %
)
```

Figure 6. DMA_SM.TDF (Part 2 of 2)

```
VARIABLE

    SYSCLK  :NODE;          % declare system clock          %


        %  define machine cycle with 4 state bits q[3..0] %

    cycle   :MACHINE
    OF BITS ( q[3..0] )
            WITH STATES (INIT        = B"0000",
                         LOAD        = B"0001",
                         START       = B"1000",
                         TRANSFER    = B"1100",
                         ERROR       = B"0010"  );


BEGIN

    SYSCLK = SCLK(CLOCK);   % use system clock              %

    cycle.CLK   = SYSCLK;   % state machine clock           %
    cycle.RESET = RST;      % state machine reset           %

    BR      = q3;           % use state bits as outputs     %
    DMACK   = q2;           % outputs of state  machine     %
    ERR     = q1;
    LDCTR   = q0;


        % define state transitions %

    CASE cycle IS
        WHEN INIT =>
            IF (ACTIVE & LDSTART) THEN cycle=LOAD;
            END IF;

        WHEN LOAD =>
            IF (!ACTIVE) THEN cycle=INIT;
            ELSIF (DMARQ & !LDSTART) THEN cycle=START;
            END IF;

        WHEN START =>
            IF (!DMARQ) THEN cycle=ERROR;
            ELSIF (BG) THEN cycle=TRANSFER;
            END IF;
        WHEN TRANSFER =>
            IF (TC) THEN cycle=INIT;
            ELSIF (!DMARQ) THEN cycle=ERROR;
            END IF;

        WHEN ERROR =>
            IF (RST) THEN cycle=INIT;
            END IF;

    END CASE;
END;
```

START

In the **START** state, **DMAC_SM** outputs **BR** to acquire control of the subsystem buses from the MPU. The MPU then sends **BG** to the state machine in response to **BR**. If the peripheral deasserts **DMARQ** while in this state, the machine makes a transition to the **ERROR** state. Otherwise, **DMAC_SM** proceeds to the **TRANSFER** state when **BG** goes high.

TRANSFER

DMACK is asserted in the **TRANSFER** state, indicating that the **ADDR_GEN** block will begin generating the DMA transfer addresses. The state machine is reset after receiving the **TC** (terminal count) signal from **ADDR_GEN**. If **DMARQ** is deasserted before **TC** is true, **DMAC_SM** again enters the **ERROR** state. This condition may occur, for example, if a system power failure disrupts the DMA process.

ERROR

The **ERROR** state occurs when a **DMARQ** low input is received in the **START** or **TRANSFER** state. The **RST** input of the machine must be asserted to clear the error condition. The state machine then returns to the **INIT** state.

DMA Address Generation

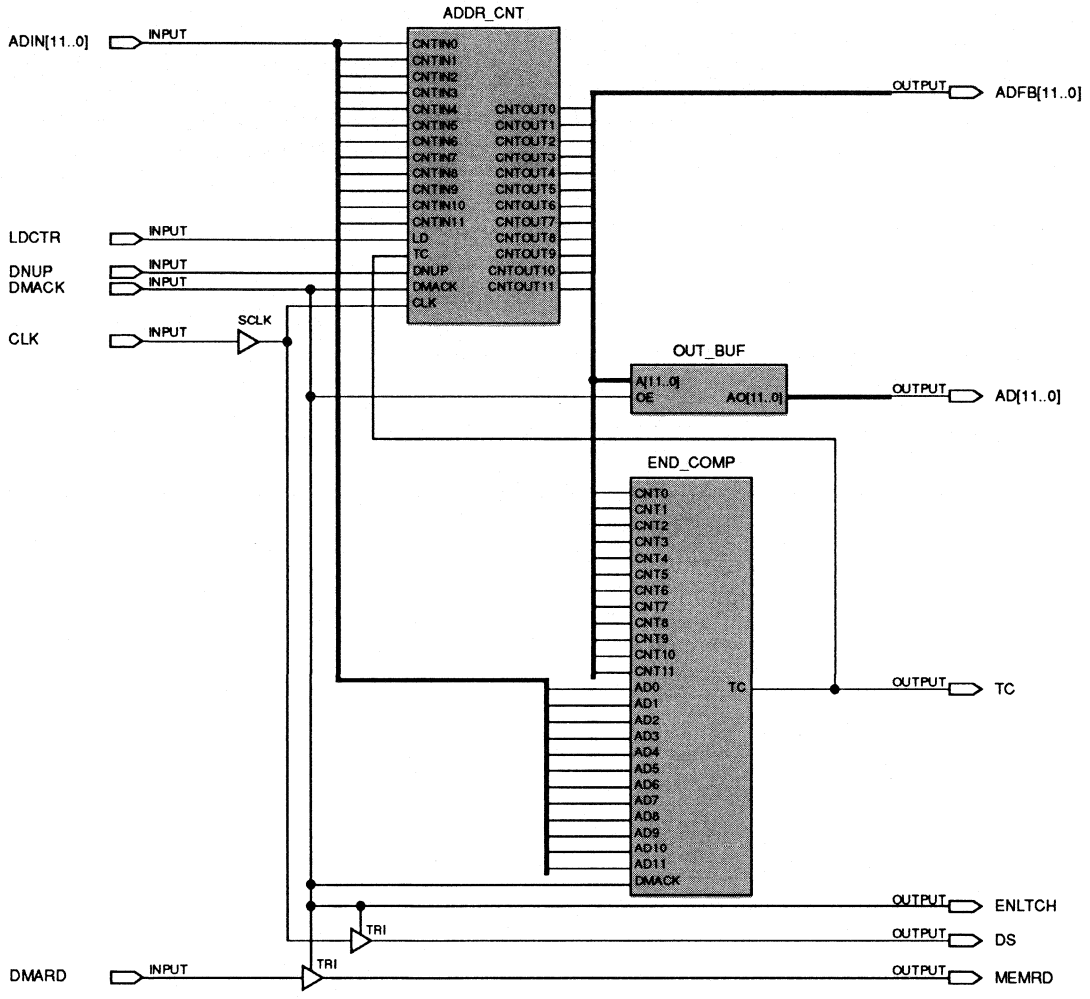
Figure 7 shows the schematic block diagram of **ADDR_GEN**, which contains three lower-level functions: **ADDR_CNT**, **END_COMP**, and **OUT_BUF**. **ADDR_CNT** is a loadable counter that generates the lower 12 bits of the DMA transfer addresses. **END_COMP** compares the current address with the ending address and terminates the DMA transfer when the last address is reached. **OUT_BUF** enables the current DMA transfer address onto the address bus.

The address range is determined by the starting and ending addresses read from the MPU. Depending on the system convention, the addresses may be generated in increasing or decreasing order. With 12 bits, variable address ranges up to 4 K words are achievable. When the last address has been reached, the state machine is reset and the DMA controller may be initialized for a new DMA transfer.

Control Signals

LDCTR, **DMACK**, and **DNUP** are control inputs from the DMA control state machine. When **LDCTR** is asserted and **CLK** has a rising edge, the 12-bit starting address is loaded into **ADDR_CNT**. The **ADDR_CNT** counter begins counting when **LDCTR** is deasserted and **DMACK** is asserted. **DNUP** is from the bus interface unit AND determines the count direction. When **DNUP** is high, the current address is incremented; when **DNUP** is low, the address is decremented.

Figure 7. ADDR_GEN



Output Signals

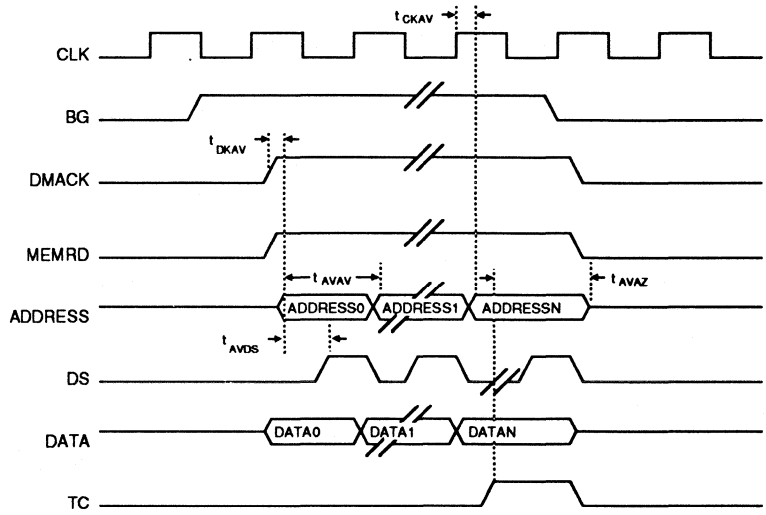
MEMRD, **ENLTCH**, **DS**, and **TC** are outputs of **ADDR_GEN**. The **MEMRD** signal, which indicates the direction of DMA transfer, is driven onto the control bus when DMA begins. **ENLTCH** enables the external latch driving the high-order address onto the address bus. **DS** provides the gating signal that the peripheral uses to strobe the next valid address. During a DMA transfer, **DS** follows the subsystem clock; otherwise, it is tri-stated. When the required addresses have been generated, the **TC** signal becomes active, signifying that the DMA transfer is complete.

10

Performance

Figure 8 shows the critical timing parameters for the DMA controller: t_{DKAV} (**DMACK** to first valid address), t_{AVAZ} (last address valid to next address valid), t_{AVAV} (address valid to next address valid), t_{CKAV} (clock to address valid), and t_{AVDS} (address valid to data strobe). The associated table lists parameter values. The speed at which the EPM5064 can generate successive addresses is shown by t_{AVAV}. This parameter determines the maximum clock rate at which the DMA controller can operate (20 MHz).

Figure 8. Critical Timing Diagram



Critical Timing Parameters

Symbol	Parameter	EPM5064
t_{DKAV}	DMACK to first address valid	28 ns
t_{AVAZ}	Last address valid to bus tri-state	33 ns
t_{AVAV}	Address valid to next address valid	50 ns
t_{CKAV}	Clock to address valid	15 ns
t_{AVDS}	Address valid to data strobe	12 ns

Conclusion

The EPM5064 is a generic EPLD that can implement complete peripheral interfaces. The integration density, power, speed, and flexibility of the EPM5064 make it an excellent choice for a peripheral subsystem DMA controller. The DMA controller described in this application brief can transfer data at 20 megawords per second. The design fits in a single EPM5064 EPLD, requiring about 1/2 square inch of board space.

The design files for this EPM5064 DMA controller are available from the Altera Applications Department. Call 1 (800) 800-EPLD for information.

DRAM Controller with the EP1830 EPLD

October 1990, ver. 1

Application Brief 85

Introduction

The performance levels and integration densities required by microprocessor-based systems push the limits of today's technology. Advances in processing technology, architecture, and circuit techniques have allowed microprocessor manufacturers to create devices that operate faster than 50 MHz. As a result, designers are challenged to create subsystem logic that keeps pace with these high-performance processors, and to achieve higher levels of logic integration in increasingly specialized systems. The development of high-performance, high-density Erasable Programmable Logic Devices (EPLDs) introduced a convenient and efficient approach to overcome many of these problems. Designers can now use these fast, dense EPLDs to customize peripheral support logic for high-performance microprocessor-based systems.

Designers must first design a suitable memory configuration to match the system processor. When choosing memory, they must consider integration densities, speed, and cost. Static RAMs (SRAMs) offer very high speeds, but are expensive, especially at higher densities. On the other hand, dynamic RAMs (DRAMs) offer high integration densities at lower cost, but sacrifice speed and require refresh logic. Therefore, most designers use SRAMs for high-speed functions, such as caching, and DRAMs for main-system memory.

When using DRAMs, a designer must address the complex issues of refresh and mode control. Standard DRAM controllers often restrict memory configurations and may result in a poor design solution. However, high-density programmable logic offers the flexibility of a custom controller, often resulting in a denser and more cost-effective method of controlling DRAMs.

In this application brief, Altera's EP1830 EPLD serves as a DRAM controller in a 68030-based system. It generates the appropriate **RAS** (Row Address Strobe) and **CAS** (Column Address Strobe) signals, multiplexes the addresses, and refreshes the DRAM banks. This application brief also describes the Altera Programmable Logic User System (A+PLUS) CAE tools that are used to create the design and fit it into an EP1830 EPLD.

EP1830 Overview

Altera's EP1830 EPLD is a high-performance pin- and JEDEC-compatible version of the EP1810 device. It offers enhanced performance with internal counter frequencies of up to 50 MHz, and input-to-output combinatorial delays of 20 ns. A single EP1830 EPLD is equivalent to over 7 standard 20-

pin PAL devices. It is available in 68-pin windowed ceramic J-lead (JLCC) and windowed ceramic Pin Grid Array (PGA) package configurations, as well as in a lower-cost, 68-pin one-time-programmable (OTP) plastic J-lead (PLCC) package.

Figure 1 shows a block diagram of the EP1830 EPLD. It has 16 dedicated input pins; 48 I/O pins that can be programmed for input, output, or bidirectional operation; and 48 macrocells that can be individually programmed for combinatorial or registered operation. Each macrocell has a programmable-AND/fixed-OR structure that implements logic with up to 8 product terms. The AND/OR function feeds a programmable inverter that can be used to create either active-low or active-high signals or, to reduce logic, using De Morgan's Inversion. The output of the programmable inverter feeds into the macrocell register, which can be programmed for D, T, JK, or SR operation, or bypassed entirely for combinatorial operation. A programmable clocking structure allows clocking from a dedicated Clock pin or product term and can be programmed on a macrocell-by-macrocell basis. Each macrocell also has a dedicated product term that asynchronously clears the macrocell register. For more information on EP1830 architecture and timing, see the *EP1800-Series EPLDs Data Sheet* in this data book.

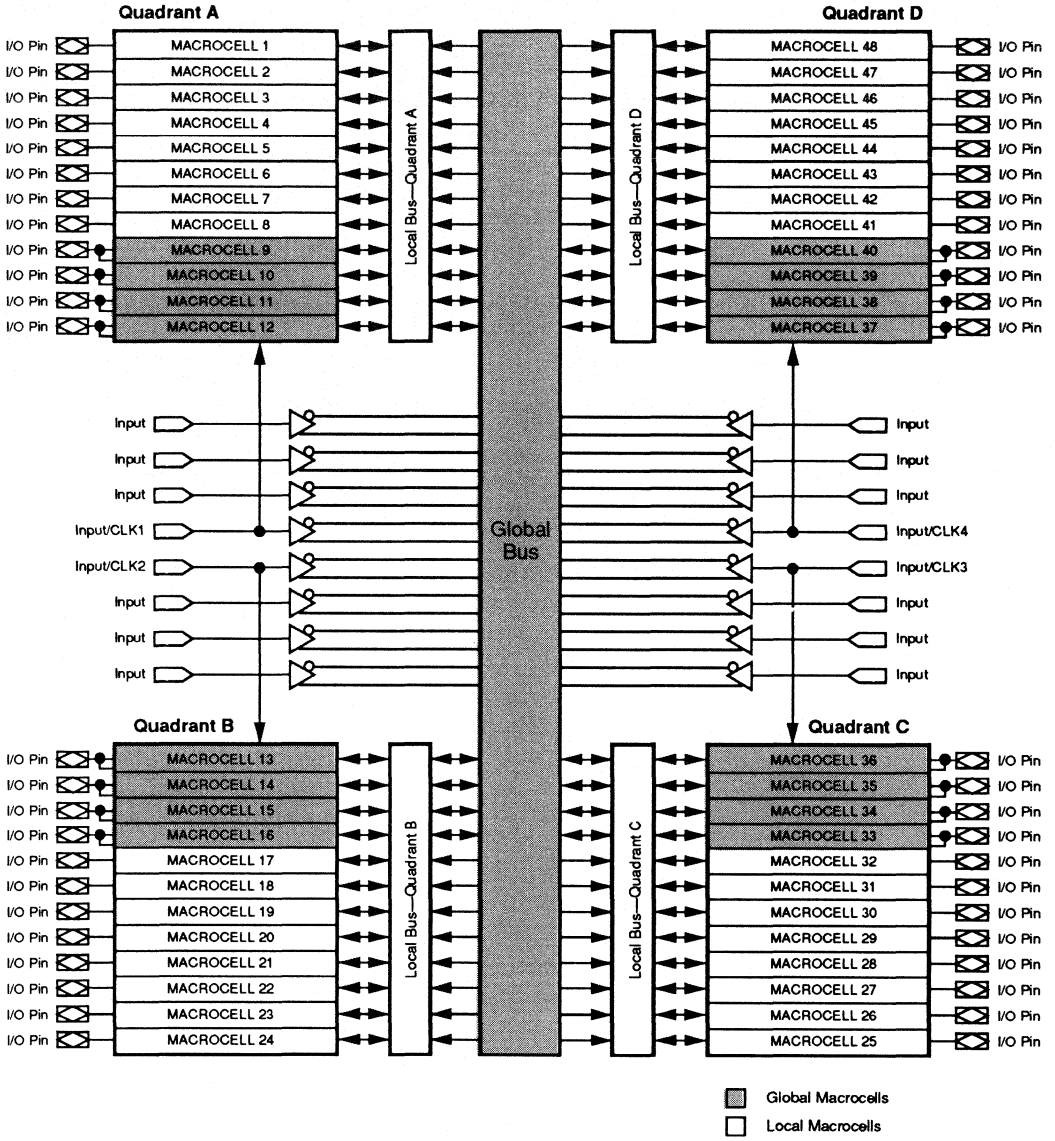
Logic is implemented in EP1830 EPLDs with Altera's A+PLUS Development System. A+PLUS is an advanced CAE system that offers multiple design entry methods, including schematic capture, Boolean equation, state machine, truth table, and netlist design entry. The Altera Design Processor (ADP) features advanced minimization techniques and automatic design fitting and generates an industry-standard JEDEC file for device programming. A+PLUS also includes a Functional Simulator (FSIM) for design verification that supports table and waveform outputs. The EPLD can be programmed in minutes at the designer's desktop to create custom working silicon.

These development tools and the flexible EP1830 architecture support rapid design and debug cycles, allowing a design to go from conception to working silicon in a single day. In addition, extensive third-party support exists for design entry, design processing, and device programming.

DRAM Overview

DRAM devices provide a cost- and space-efficient solution to high-density memory system design. However, DRAMs require control circuitry to properly interface to microprocessor systems. Each memory location in a DRAM device has an associated row and column address. The devices are addressed with multiplexed row and column address pins, which are internally routed to the row and column decoders for each location. Although these devices have fewer pins and require less space, external logic is required to multiplex the system address lines.

Figure 1. EP1830 Block Diagram



Addressing DRAM through multiplexed address pins requires two steps:

1. The row address is placed at the DRAM address pins and /**RAS** is asserted, latching the row address of the desired memory location.
2. The column address is placed on the DRAM address pins and /**CAS** is asserted, latching the column address of the desired memory location.

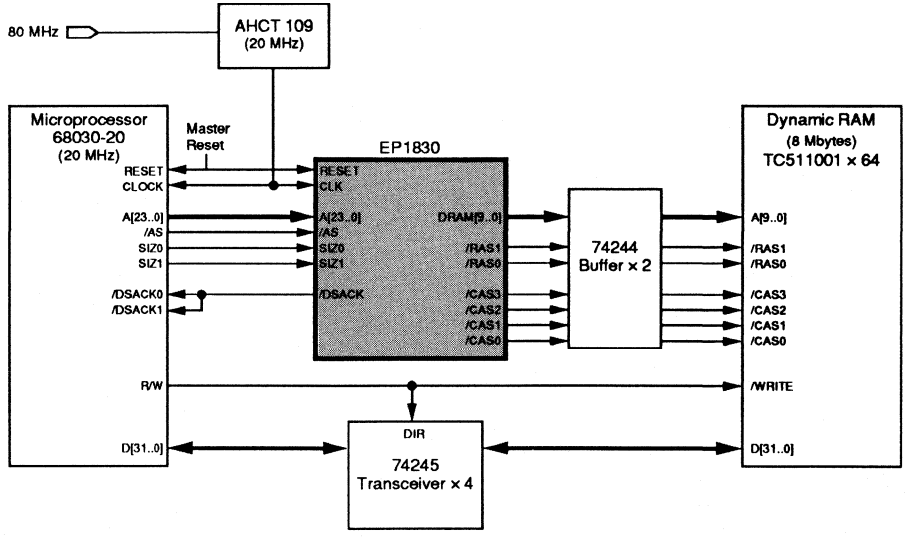
The slash character (/) indicates that the signals are active-low. The memory location with the latched row and column addresses can then be written to or read from. When the /**RAS** and /**CAS** pins of the DRAM are deasserted, the memory transfer is completed.

Logic is required to control the memory transfer and refresh cycles in DRAM designs. The EP1830 design described in this application brief implements the logic necessary to control the multiplexing of the row, column, and refresh addresses; to generate the /**RAS** and /**CAS** signals; and to initiate the required refresh cycles. Each location must be periodically refreshed to ensure memory integrity. The **RAS**-only refresh method, which is used for this application, places a refresh address on the DRAM address pins and asserts /**RAS**.

68030 DRAM Subsystem

Figure 2 shows a block diagram of a 68030-20-based DRAM subsystem. The microprocessor accesses memory with a programmable DRAM controller implemented with an EP1830 EPLD. The subsystem contains 8 MBytes of DRAM. A 32-bit bus transceiver controls the direction of the data bus, and an octal bus driver buffers the DRAM control signals.

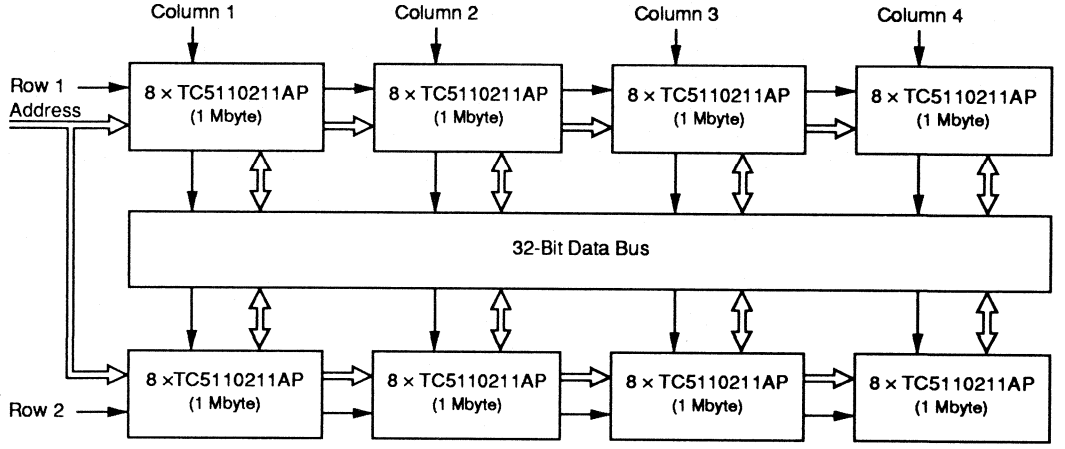
Figure 2. 68030-20-Based DRAM Subsystem



DRAM Configuration

Sixty-four Toshiba TC511001AP 1-Mbyte DRAMs provide the memory in this system. The organization of the DRAM devices on the 32-bit bus of the 68030 microprocessor is shown in Figure 3. The memory is broken into 2 rows and 4 columns, requiring 2 /RAS and 4 /CAS signals for control. Each row and column location has 8 DRAM devices, each providing 1 Mbyte of DRAM. The DRAM controller generates /RAS and /CAS signals to address individual DRAM locations. One memory-transfer operation can address 1, 2, 3, or 4 bytes of information by selecting the appropriate number of columns.

Figure 3. DRAM Memory Organization



DRAM Operation

DRAM is accessed when the microprocessor issues a memory request or when the DRAM controller initiates a refresh operation. Both operations are controlled by the DRAM controller implemented with the EP1830. The DRAM controller provides the DRAM and the 68030 with the following signals.

Microprocessor-Initiated Memory Transfer Cycle

The 68030 initiates a memory request by placing a DRAM address on the address bus A[31..0] and asserting /AS (Address Strobe). The size of the data transfer is indicated by the SIZ1 and SIZ0 control signals. The decoded signals are shown in Table 1.

10

Table 1. Data Transfer Size

SIZ1	SIZ0	Transfer Size (Bits)
L	H	8
H	L	16
H	H	24
L	L	32

The DRAM controller responds to the memory request by placing the row address on the DRAM address lines **A[9..0]**; then the appropriate row address strobe **/RAS***x* is asserted. Next, the column address is placed on the same address lines, and from up four **/CAS***x* column address strobes are generated. The DRAM controller indicates that the DRAM access is completed by asserting one or both of the data select acknowledge signals (**/DSACK1** and **/DSACK0**). These signals also indicate the width of the data port being accessed. The decoded signals are shown in Table 2.

Table 2. Port Width

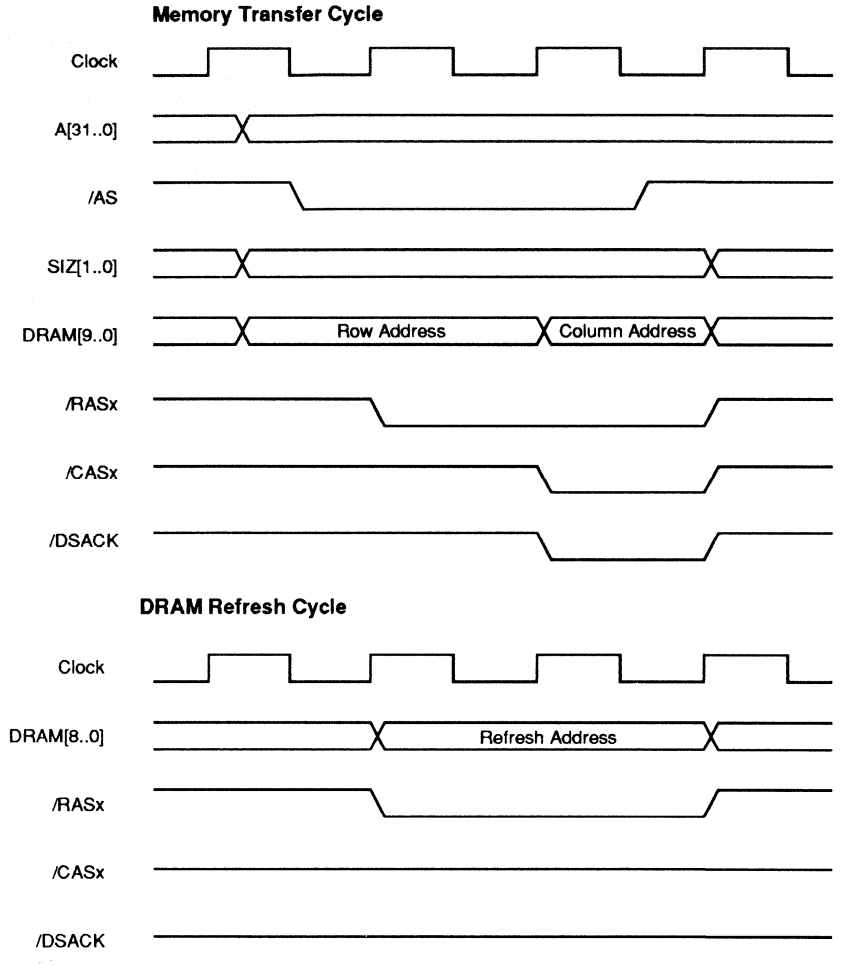
/DSACK1	/DSACK0	Accessed Port Width (in bits)
H	H	Insert Wait States
H	L	8
L	H	16
L	L	32

Because this DRAM subsystem has a 32-bit port, the DRAM controller can drive **/DSACK1** and **/DSACK0** with a common signal, **/DSACK**. The microprocessor deasserts **/AS** to indicate that the memory transfer is complete. The DRAM controller then deasserts **/RAS**, **/CAS**, and **/DSACK**. The timing for this memory access operation is shown in Figure 4.

DRAM Controller-Initiated Refresh Cycle

Each address location in a DRAM device must be refreshed (within a period of time specified by the DRAM manufacturer) to ensure data integrity. When a refresh cycle is required, the DRAM controller places the refresh address on **A[8..0]** and asserts all **/RAS***x* signals, refreshing all address locations with a row address equal to the refresh address. (Only 9 of the 10 address lines are needed for refresh cycles because each DRAM device contains 512 rows.) After the **/RAS***x* lines have been asserted for at least the specified minimum time, they are deasserted. This **RAS**-only refresh cycle is the method used by the DRAM controller in this application brief. The timing for the refresh operation is shown in Figure 4.

Figure 4. Memory Transfer Cycle and DRAM Refresh Cycle



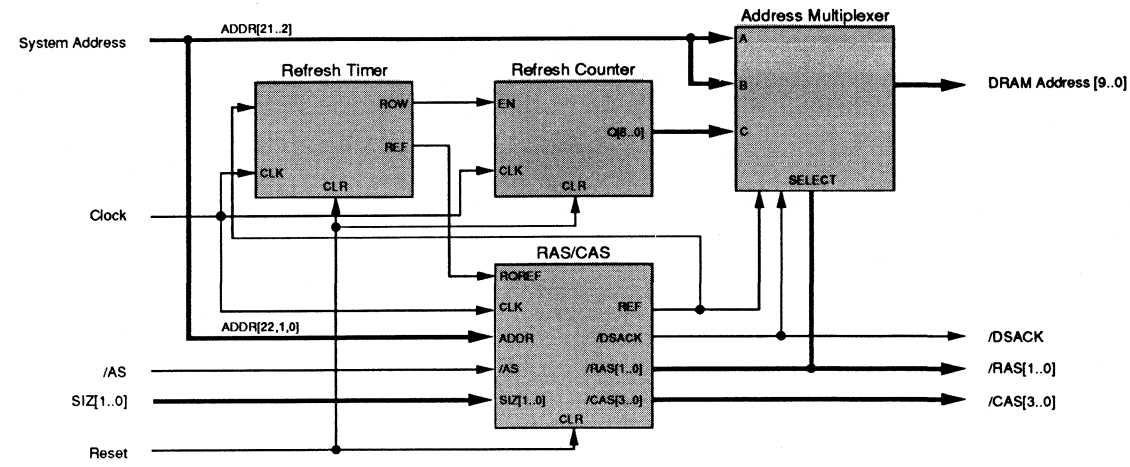
The DRAM controller generates all signals necessary to control these operations. If a memory and refresh request are received at the same time, the refresh operation is given priority. If a refresh request is received during a memory cycle, the cycle is completed before the refresh operation is finished. The EP1830 EPLD can integrate the entire DRAM controller while operating at the speeds of the microprocessor and DRAM devices.

10

DRAM Controller

Figure 5 shows a block diagram of the DRAM controller, implemented in the EP1830 with four functional blocks: the RAS/CAS generator (state machine), the refresh timer, the refresh address counter, and the address multiplexer. Each function is designed separately, and the symbols are

Figure 5. 8-Mbyte DRAM Controller



automatically generated by the A+PLUS software. The operation of each functional block is described here.

RAS/CAS Generator

The RAS/CAS generator is a state machine that generates the DMA controller's **/RAS**, **/CAS**, and **/DSACK** signals (Figure 6). The generator uses the State Machine File (SMF) format that integrates control logic in Altera's EP-series EPLDs. This state machine has 18 states and 9 state bits. The value of the state bits for each state is shown in the States Subsection of the design. The States Subsection is followed by the Transitions Subsection, which defines state machine transitions with **IF-THEN** statements.

Figure 6. RAS/CAS Generator State Machine File (Part 1 of 3)

```
PART: MACRO

INPUTS:
    A22 A1 A0
    SIZ1 SIZ0
    AS
    CLK
    RQREF

OUTPUTS:
    RAS1 RAS0 RAS
    CAS3 CAS2 CAS1 CAS0
    DSACK REF

NETWORK:
    RAS0 = CONF(RAS0,)
    RAS1 = CONF(RAS1,)
```

Figure 6. RAS/CAS Generator State Machine File (Part 2 of 3)

```
EQUATIONS:
    RAS0' = (RAS & A22') + REF';
    RAS1' = (RAS & A22 ) + REF';

MACHINE:    memory
  CLOCK:    CLK

 STATES:    [ CAS0 CAS1 CAS2 CAS3 RAS DSACK REF Q1 Q0 ]
   BEGIN    [  0    0    0    0    0    0     0   0  0  ]
    IDLE    [  1    1    1    1    1    1     1   1  1  ]
   RASx0    [  1    1    1    1    0    1     1   0  0  ]
   RASx1    [  1    1    1    1    0    1     1   0  1  ]
   RASx2    [  1    1    1    1    0    1     1   1  0  ]
   RASx3    [  1    1    1    1    0    1     1   1  1  ]
RASxC0x1    [  0    1    1    1    0    0     1   1  1  ]
RASxC0x2    [  0    0    1    1    0    0     1   1  1  ]
RASxC0x3    [  0    0    0    1    0    0     1   1  1  ]
RASxC0x4    [  0    0    0    0    0    0     1   1  1  ]
RASxC1x1    [  1    0    1    1    0    0     1   1  1  ]
RASxC1x2    [  1    0    0    1    0    0     1   1  1  ]
RASxC1x4    [  1    0    0    0    0    0     1   1  1  ]
RASxC2x1    [  1    1    0    1    0    0     1   1  1  ]
RASxC2x2    [  1    1    0    0    0    0     1   1  1  ]
RASxC3x1    [  1    1    1    0    0    0     1   1  1  ]
    REF0    [  1    1    1    1    0    1     0   1  1  ]
    REF1    [  1    1    1    1    0    1     0   1  0  ]


BEGIN:  IDLE

IDLE:
    IF RQREF           THEN REF0
    IF AS & A1' & A0' THEN RASx0
    IF AS & A1' & A0  THEN RASx1
    IF AS & A1  & A0' THEN RASx2
    IF AS & A1  & A0  THEN RASx3
    IDLE

RASx0:
    IF SIZ1' & SIZ0  THEN RASxC0x1
    IF SIZ1  & SIZ0' THEN RASxC0x2
    IF SIZ1  & SIZ0  THEN RASxC0x3
    RASxC0x4

RASx1:
    IF SIZ1' & SIZ0  THEN RASxC1x1
    IF SIZ1  & SIZ0  THEN RASxC1x2
    RASxC1x4

RASx2:
    IF SIZ1' & SIZ0  THEN RASxC2x1
    RASxC2x2

RASx3:
    RASxC3x1

RASxC0x1: IF AS THEN IDLE
    RASxC0x1
RASxC0x2: IF AS THEN IDLE
    RASxC0x2
```

Figure 6. RAS/CAS Generator State Machine File (Part 3 of 3)

```
RASxC0x3: IF AS THEN IDLE
RASxC0x3
RASxC0x4: IF AS THEN IDLE
RASxC0x4

RASxC1x1: IF AS THEN IDLE
RASxC1x1
RASxC1x2: IF AS THEN IDLE
RASxC1x2
RASxC1x4: IF AS THEN IDLE
RASxC1x4

RASxC2x1: IF AS THEN IDLE
RASxC2x1
RASxC2x2: IF AS THEN IDLE
RASxC2x2

RASxC3x1: IF AS THEN IDLE
RASxC3x1

REF0:   REF1
REF1:   IDLE

END$
```

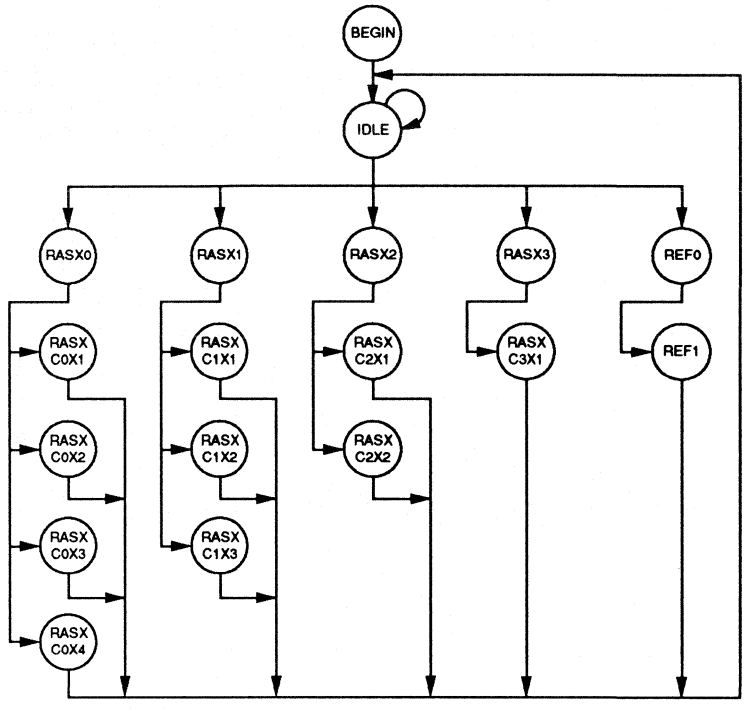
Figure 7 shows a state transition diagram for this state machine. The state machine powers up in the state **BEGIN**, i.e., with all registers low. During the first clock transition, the machine goes to the **IDLE** state.

When a memory request is initiated (i.e., **/AS** goes low), the starting column of the memory transfer is determined from the value of the two least significant address lines, **A1** and **A0**. The row of the transfer is determined from **A22**, the most significant address line. The machine goes to **RAS***xn*, where *n* represents the beginning column and is the decimal representation of **A[1..0]**. The **RAS** state bit is asserted during this state. **RAS** is further decoded with **A22** to assert **/RAS1** if **A22** is high, or **/RAS0** if **A22** is low.

The next clock cycle determines the size of the transfer from the **SIZ1** and **SIZ0** signals. The machine makes a transition to state **RAS***x***C***nxm*, where *n* is the beginning column and *m* indicates the number of bytes requested ($n + m$ is never > 4). The **RAS** signal remains asserted during this cycle. The **/CAS** signals asserted are based on the starting column and the size of the transfer. **/DSACK** is also asserted during this cycle, indicating that the data is valid. Once **/AS** is deasserted, the state machine goes back to the **IDLE** state.

When the refresh timer sends a refresh request, the state machine goes from **IDLE** to **REF0**. The **/RAS1** and **/RAS0** signals are asserted to refresh both rows of the memory configuration. During the next clock cycle, the

Figure 7. RAS/CAS Generator State Diagram

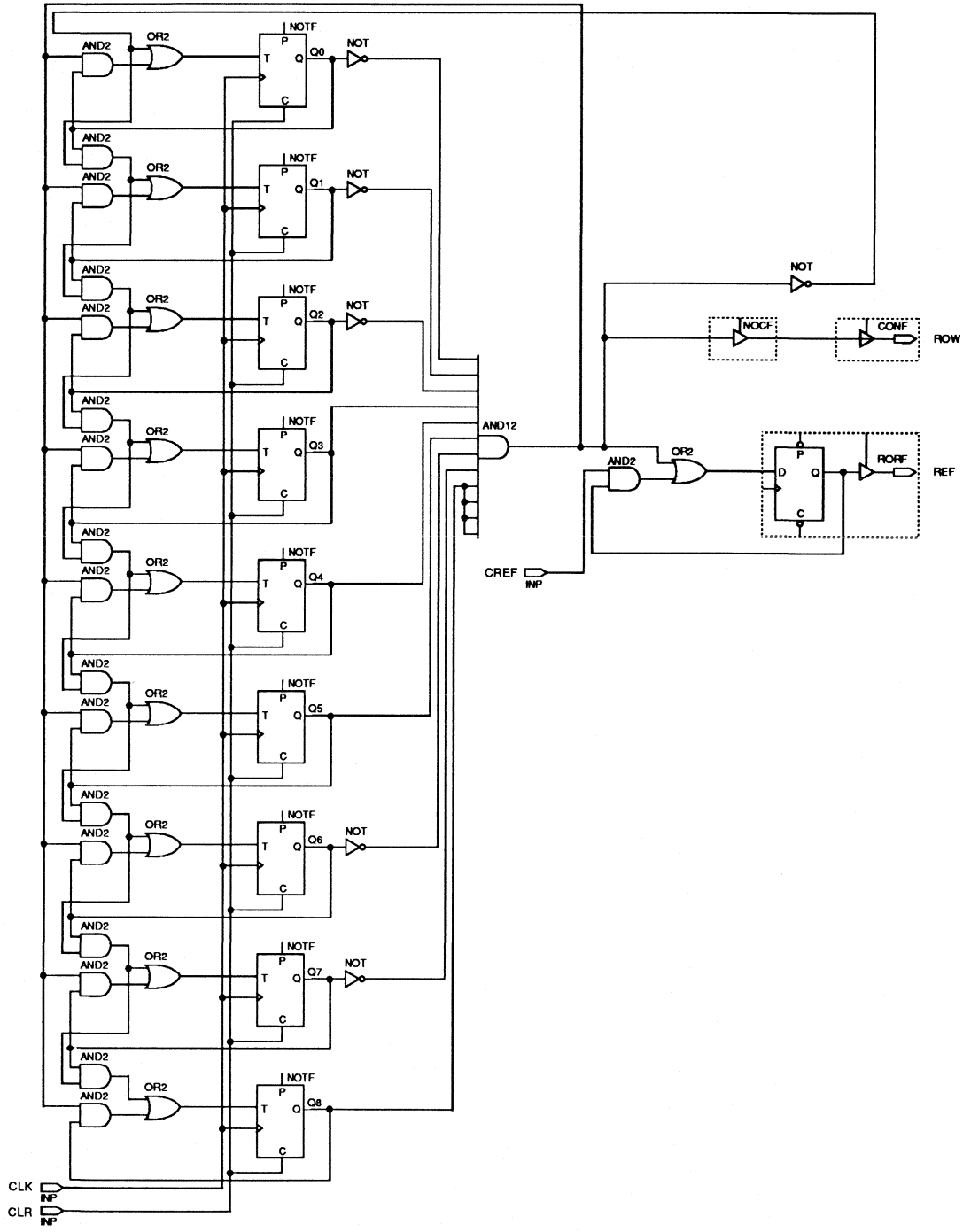


state machine goes to state **REF1**, while **/RAS0** and **/RAS1** remain asserted. This state ensures the 90-ns minimum **RAS** period during the **RAS**-only refresh cycle. The machine then goes to the **IDLE** state at the next clock cycle, and the **/RAS***x* signals are deasserted. The state bits **Q0** and **Q1** are used to distinguish states with otherwise equivalent state values.

Refresh Timer

Figure 8 shows the refresh timer. Each of the DRAM devices described in this application brief has 512 rows that must be refreshed at least once every 8 ms. Since each request refreshes one row, a request must be issued once every 15.6 µs (8 ms/512 rows). With a clock frequency of 20 MHz, a refresh request should be issued every 312 clock cycles (15.6 µs × 20 MHz). The refresh timer is a free-running counter that counts from 0 to 311 and back to 0. Two signals are generated when a refresh is required: the first is an unlatched signal that is true whenever the counter is at 311; the second sets a latch that represents a refresh request to the RAS/CAS generator whenever the counter is at 311. The latch is cleared when the RAS/CAS generator acknowledges the request with the **REF** state bit.

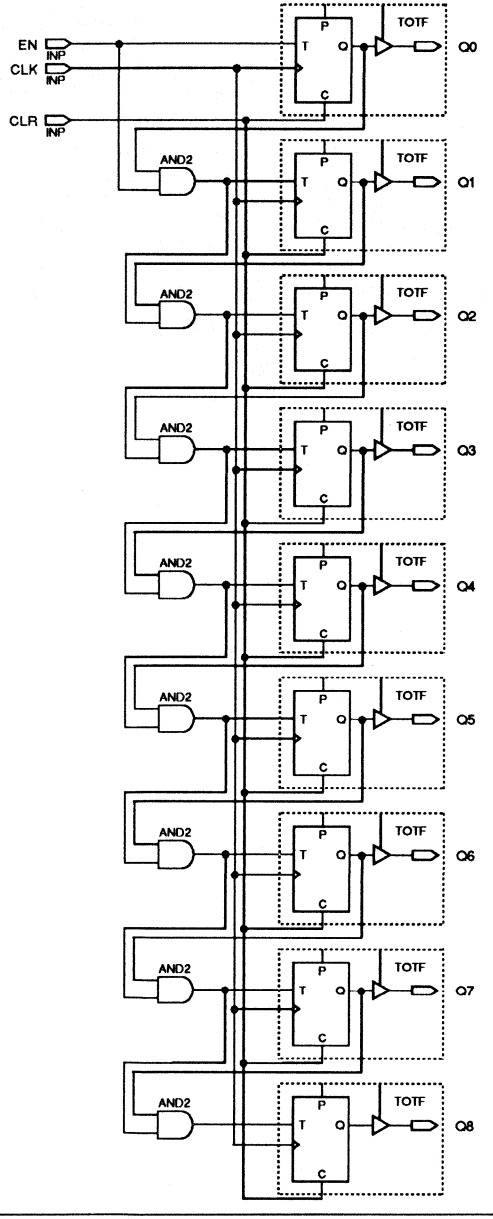
Figure 8. Refresh Timer



Refresh Address Counter

The refresh address counter increments through the 512 rows that need to be refreshed. It is a 9-bit up-counter with Enable. The counter is enabled by the refresh timer when a refresh is required. Each refresh cycle refreshes the row address equal to the value of the counter. See Figure 9.

Figure 9. Refresh Address Counter



Address Multiplexer

The address multiplexer decodes the state of the RAS/CAS generator state bits to determine the address for the DRAM devices. During the **IDLE** and **RAS***xn* states, the row address (**A[21..12]**) is selected. During the **RAS***n***C***xm* states, the column address (**A[11..2]**) is selected. During the **REF0** and **REF1** states, the refresh address from the refresh address counter is selected.

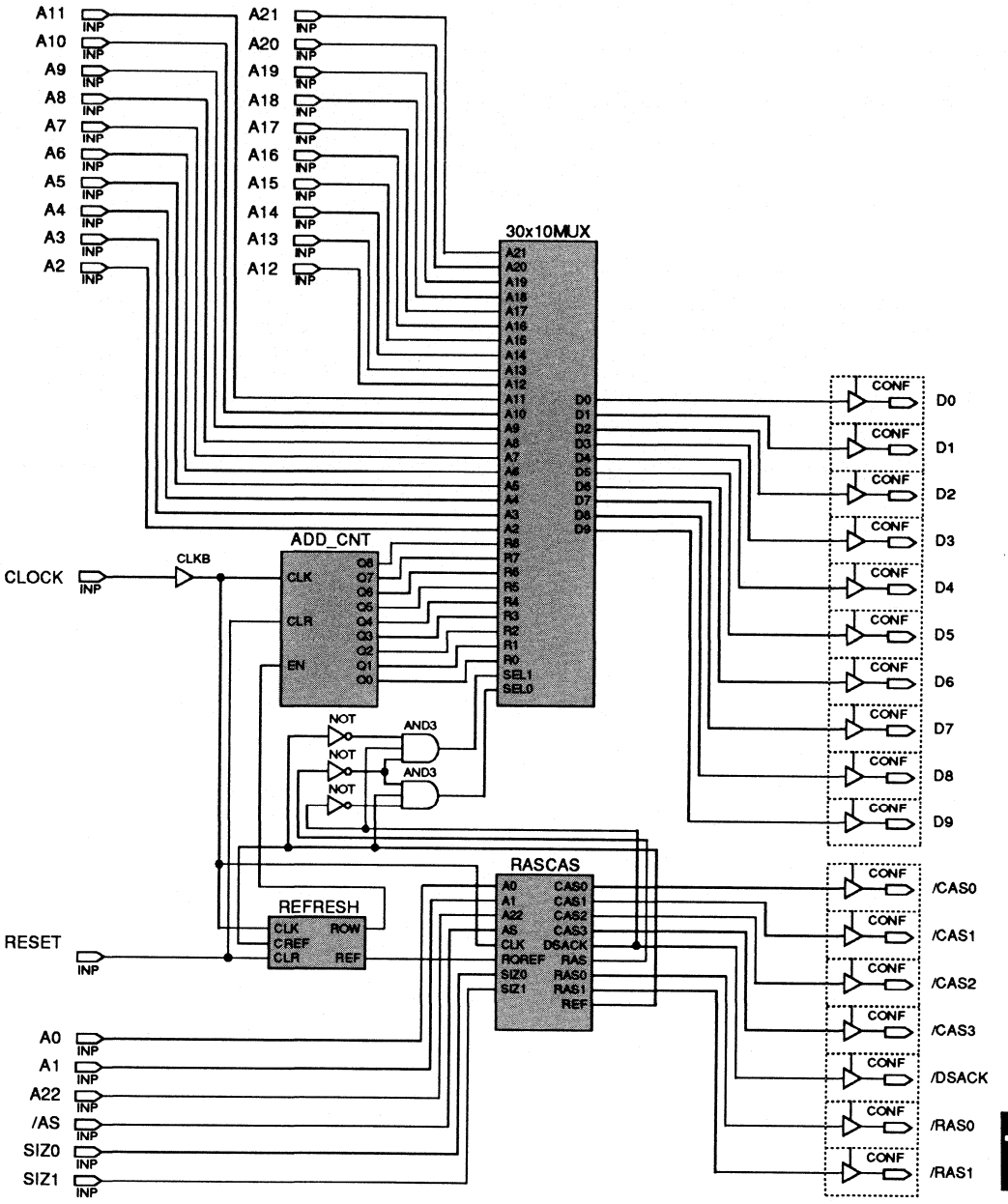
Top-Level Schematic

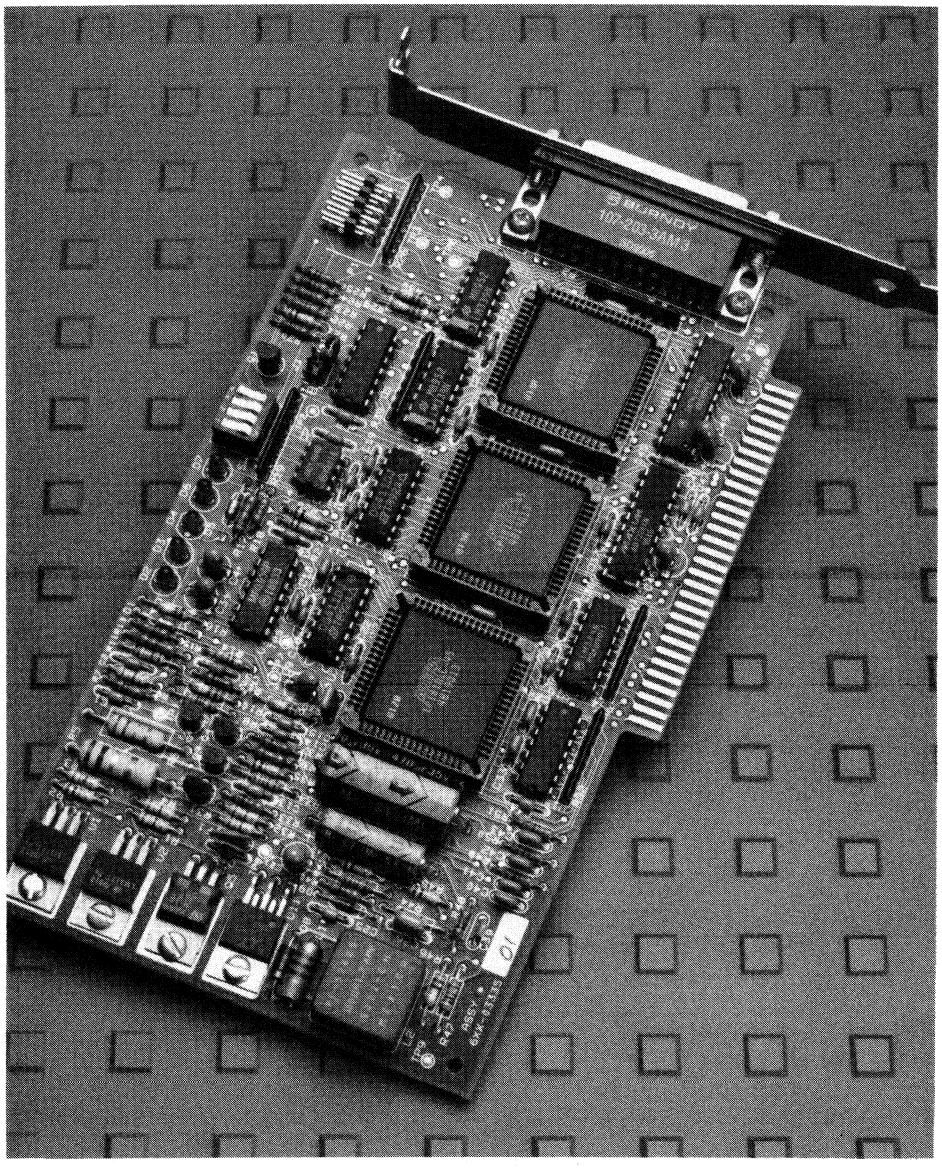
The four separate designs are integrated in the schematic (see Figure 10). A+PLUS allows text- and schematic-based designs to be combined into a single design, enabling the user to represent each major function of the design in the most intuitive form. The design is then processed by the Altera Design Processor (ADP) and simulated with the A+PLUS Functional Simulator (FSIM). Thus, the entire process of entering, processing, verifying, and programming a design is completed with Altera's A+PLUS Development System.

Conclusion

The EP1830 EPLD is ideal for integrating custom subsystem functions or other PLD devices, including the custom DRAM controller described in this application brief. The DRAM controller application only requires a clock speed of 20 MHz, although the EP1830 is capable of running counters at 50 MHz. Together with A+PLUS design tools, the EP1830 effectively and quickly reduces board space, lowers power, and increases system reliability.

Figure 10. DRAM Controller Schematic





Contents

October 1990

Section 11 General Information

11

Electronic Bulletin Board Service

October 1990, ver. 1

Introduction

Altera provides an Electronic Bulletin Board Service (BBS) for continuous access to up-to-date EPLD and development tool information, electronic application notes and briefs, data sheet updates, customer newsletters, and useful utility programs. The BBS also supports file transfers to and from the Altera Applications Engineering Department. Owners of A+PLUS and MAX+PLUS software may refer to their user manuals for detailed information on using the BBS.

Modem Number:
(408) 249-1100

The telephone number for the BBS is (408) 249-1100. To connect to the BBS via modem, the following equipment and configuration are required:

- ❒ Baud rate of 1200 or 2400
- ❒ Bell Standard 212A or compatible modem
- ❒ Data format: 8 data bits, 1 stop bit, no parity

The following file transfer protocols are supported:

- ❒ Xmodem (Checksum)
- ❒ Xmodem-CRC (CRC)
- ❒ Ymodem (1K-Xmodem)
- ❒ Ymodem-G (1K-Xmodem-G)
- ❒ ASCII (Non-Binary)
- ❒ Kermit

Logging On

After the BBS connection has been established, the user may choose between graphic (for EGA or VGA displays) or non-graphic display mode. The user is then prompted for his or her name; a new user may also choose a password. Each name and password are recorded for future log-ons.

A series of screens appears automatically: the Altera News screen, the Personal Mail screen, and the Settings screen. The Main Menu, from which all functions are accessed, appears next. The most commonly used functions are: **F)ile Directories, U)pload a File** and **D)ownload a File**. On-line help is available with the **H)elp Functions** command. The **F)ile Directories** command displays a list of directories containing files that can be downloaded. (Uploaded files are stored in a private directory.)

File Uploading

The File Upload service is available for uploading files that require analysis or correction by an Altera Applications Engineer. All files that are uploaded to the Altera BBS are automatically stored in a private directory. When a file is uploaded, the file description should include the name of the Altera Applications Engineer who has been asked to examine the file.

11

File Downloading

Files may be copied from the following six directories:

1. From-Altera File Directory

This directory is a general directory for downloading files from Altera Applications, for example, after a problem or question has been analyzed.

2. Engineering Application Briefs

This directory contains Electronic Application Briefs (EABs) and Notes (EANs), which provide up-to-date information on using Altera EPLDs effectively.

3. Engineering Application Utilities

This directory contains Electronic Application Utilities (EAUs) that complement Altera software and aid EPLD design. Three commonly used utilities are described below. A full description of all available utilities is given in *Application Brief 73 (Software Utility Programs)* in this data book.

PLD2EQN The PLD2EQN utility converts common PAL/GAL/PLA JEDEC files to Altera Hardware Description Language (AHDL) files that are compatible with the MAX+PLUS software. This utility can also produce an Altera Design File (ADF) that is compatible with A+PLUS software.

JEDSUM The JEDSUM utility calculates the EPROM data checksum, file transmission checksum, and the number of programmed architecture bits contained in an EPLD JEDEC file.

AVEC The AVEC utility adds functional test vectors to EP-series EPLD JEDEC files. AVEC translates the table output files generated by the A+PLUS Functional Simulator's functional vectors. Third-party programmers (e.g., Data I/O 29B and UniSite 40 machines) have built-in hardware drivers that can apply these vectors to a programmed EPLD.

4. Altera Customer Newsletters

This directory contains Altera newsletters that provide current news on EPLDs and development tools, and "Question and Answer" pages that answer many common questions asked by Altera customers.

5 & 6. A+PLUS and MAX+PLUS Macrofunction Exchange Libraries

These directories are used to publicly exchange A+PLUS and MAX+PLUS macrofunctions. Customers may download any macrofunctions in this directory. Macrofunctions that have been uploaded to be shared with other users are also placed here by the system operator ("Sysop").

Ordering Information

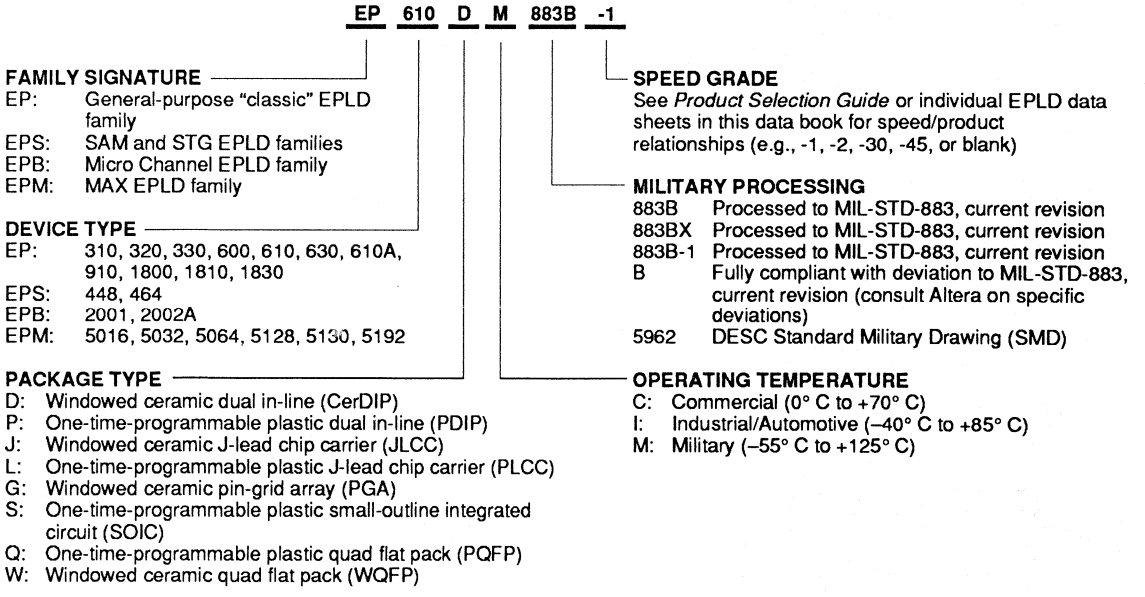
October 1990, ver. 1

Ordering EPLDs

Figure 1 shows how an EPLD part number is constructed. For information on specific package, grade, and speed combinations, refer to individual EPLD data sheets or the *Product Selection Guide* in this data book, or telephone the Altera Marketing Department at (408) 984-2800.

MIL-STD-883-compliant product specifications are provided in Military Product Drawings (MPDs) that are available on request from Altera Marketing. These MPDs should be used for the preparation of Source Control Drawings (SCDs).

Figure 1. EPLD Package Ordering Codes



Examples:

EP1810GI-40	EP1810 in a windowed ceramic pin-grid array package, industrial temperature range, -40 speed grade (t_{PD1} = 40 ns).
EPM5032DM883B	EPM5032 in a windowed ceramic dual in-line package, MIL-STD-883B-qualified.
EPS448LC-25	EPS448 in a plastic J-lead chip carrier package, commercial temperature range, -25 speed grade (f_{MAX} = 25 MHz).

11

Ordering Software & Hardware

Altera development systems, software, and hardware should be ordered by the designations given in the *Product Selection Guide* in this data book. Table 1 lists the part numbers for programming adapters. Refer to individual data sheets in this data book for detailed information on software and hardware products.

Table 1. EPLD Adapter Support

EPLD	Package	Part Number
EP330	DIP	PLED330
	J-Lead	PLEJ330
	SOIC	PLES330
EP600/610	DIP	PLED610
	J-Lead	PLEJ610
EP600/610/630/610A	DIP	PLED630
	J-lead	PLEJ630
	SOIC	PLES630
EP900/910	DIP	PLED910
	J-lead	PLEJ910
EP1800/1810	J-lead	PLEJ1810
	PGA	PLEG1810
EP1800/1810/1830	J-lead	PLEJ1830
	PGA	PLEG1830
EPS448	DIP	PLED448
	J-lead	PLEJ448
EPM5016	DIP	PLED5016
	J-lead	PLEJ5016
	SOIC	PLES5016
EPM5032	DIP	PLED5032
	J-lead	PLEJ5032
	SOIC	PLES5032
EPM5064	J-lead	PLEJ5064
EPM5128	J-lead	PLEJ5128
	PGA	PLEG5128
EPM5130	PGA	PLEG5130
	QFP	PLEQ5130
EPM5192	J-lead	PLEJ5192
	PGA	PLEG5192
	QFP	PLEQ5192
EPB2001	J-lead	PLEJ2001

EPLD Package Outlines

October 1990, ver. 1 — **Data Sheet**

Introduction

This data sheet provides package outlines for all Altera EPLDs. Table 1 shows the type of packages, lead materials, and lead finishes available.

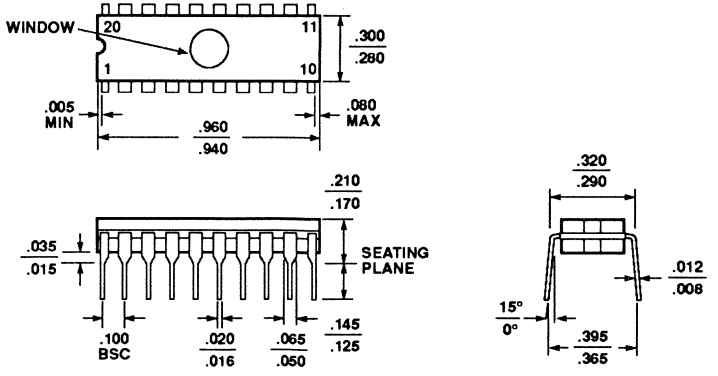
Table 1. EPLD Packages

Package Type	Package Code	Lead Material	Lead Finish
Ceramic dual in-line	D	Alloy 42	Solder dip over tin flash (Military) Matte tin plate
Plastic dual in-line	P	Copper	Solder dip (60/40)
Ceramic J-lead	J	Alloy 42	Solder dip (60/40)
Plastic J-lead	L	Copper	Solder plate (60/40)
Ceramic pin-grid array	G	Alloy 42	Gold over nickel plate
Plastic small-outline IC	S	Copper	Solder plate (60/40)
Ceramic quad flat pack	W	Alloy 42	Matte tin plate
Plastic quad flat pack	Q	Copper	Solder plate (60/40)

Package outlines are listed here in ascending size order. The dimensions shown are nominal with a tolerance of ± 0.020 in. (0.51 mm) unless otherwise indicated. Maximum lead coplanarity is 0.004 in. (0.10 mm).

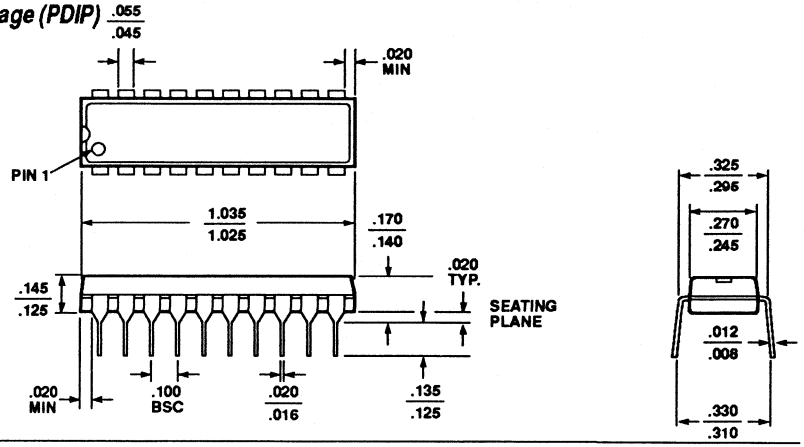
20-Pin Ceramic Dual In-Line Package (CerDIP)

For military-qualified product, see case outline D-8 in Appendix C of MIL-M-38510.

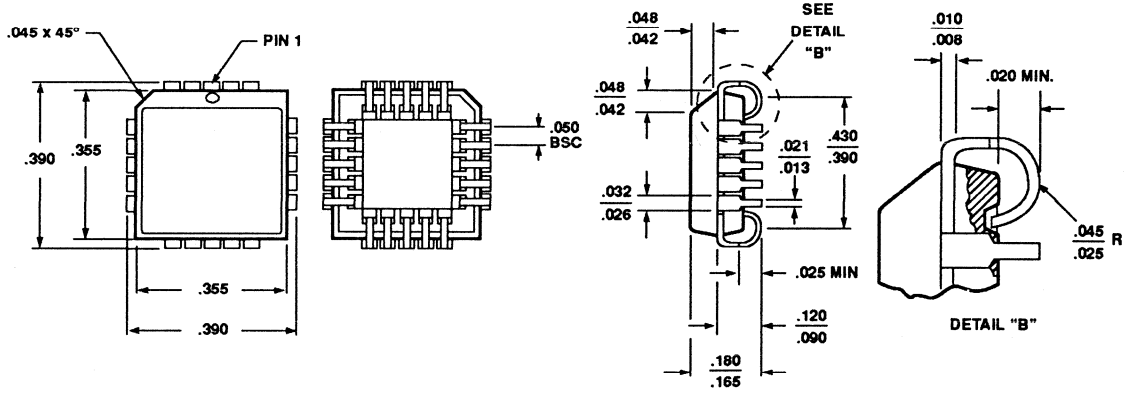


11

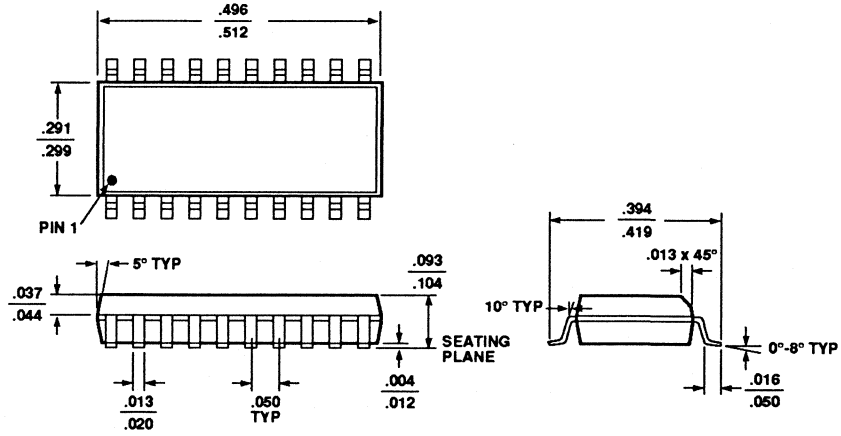
20-Pin Plastic Dual In-Line Package (PDIP)



20-Pin Plastic J-Lead Chip Carrier (PLCC)

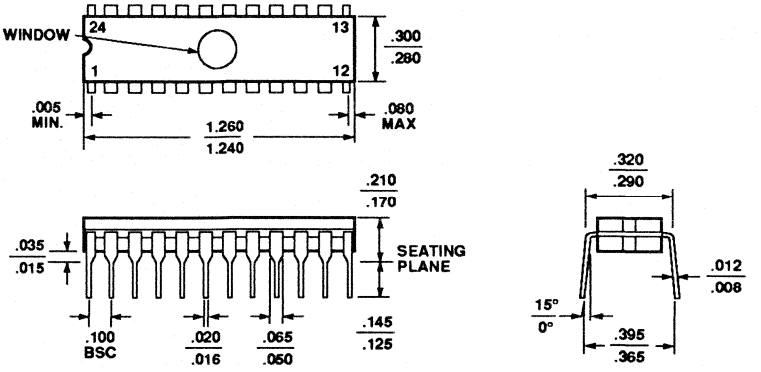


20-Pin Plastic Small-Outline IC (SOIC)

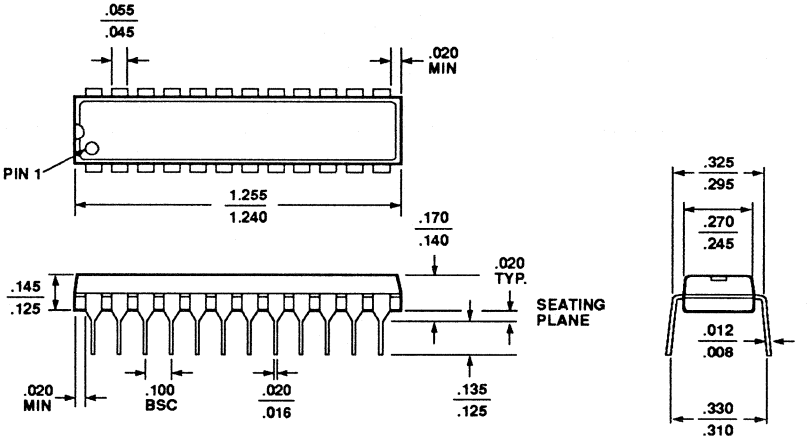


24-Pin Ceramic Dual In-Line Package (CerDIP)

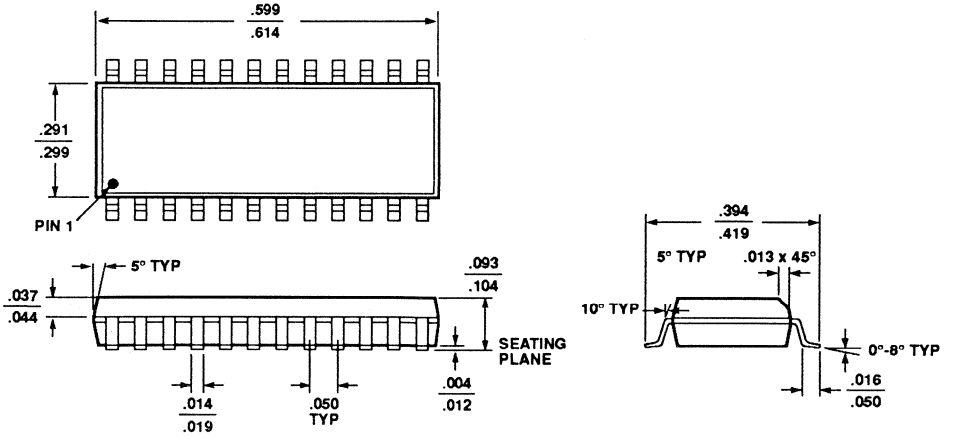
For military-qualified product, see case outline D-9 in Appendix C of MIL-M-38510.



24-Pin Plastic DIP

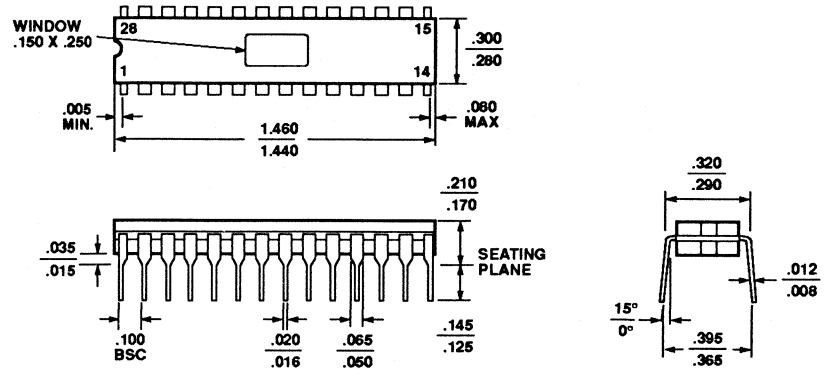


24-Pin Plastic Small-Outline IC (SOIC)

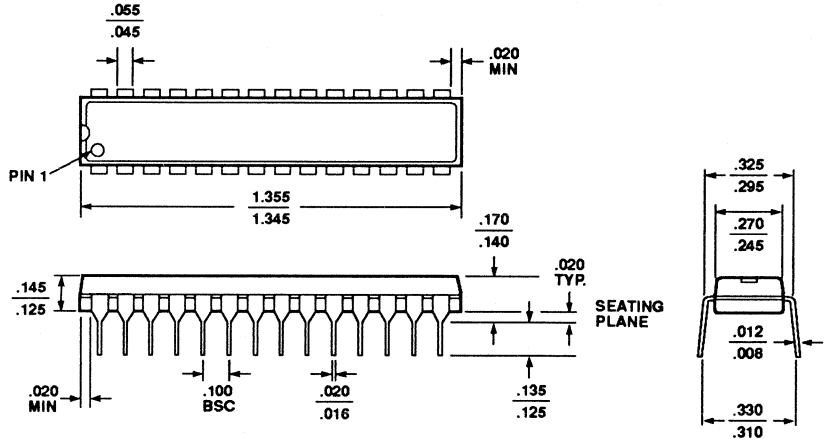


11

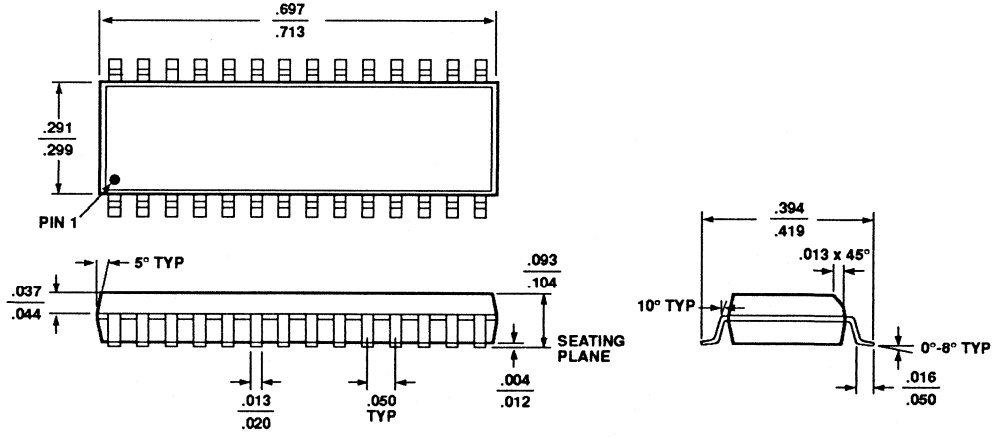
28-Pin Ceramic DIP



28-Pin Plastic DIP

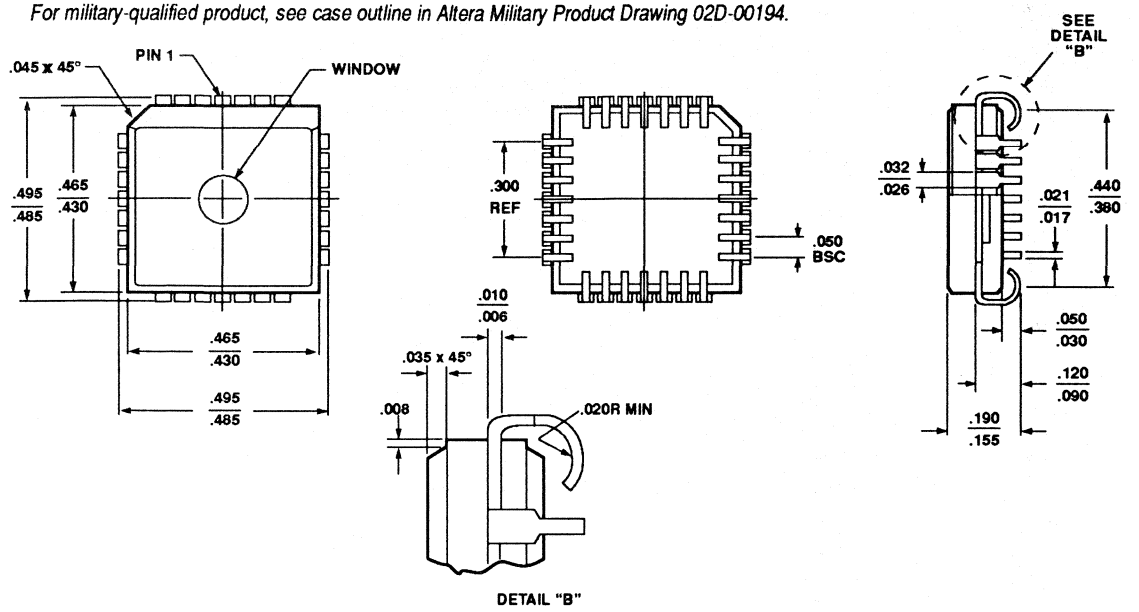


28-Pin Plastic Small-Outline IC (SOIC)

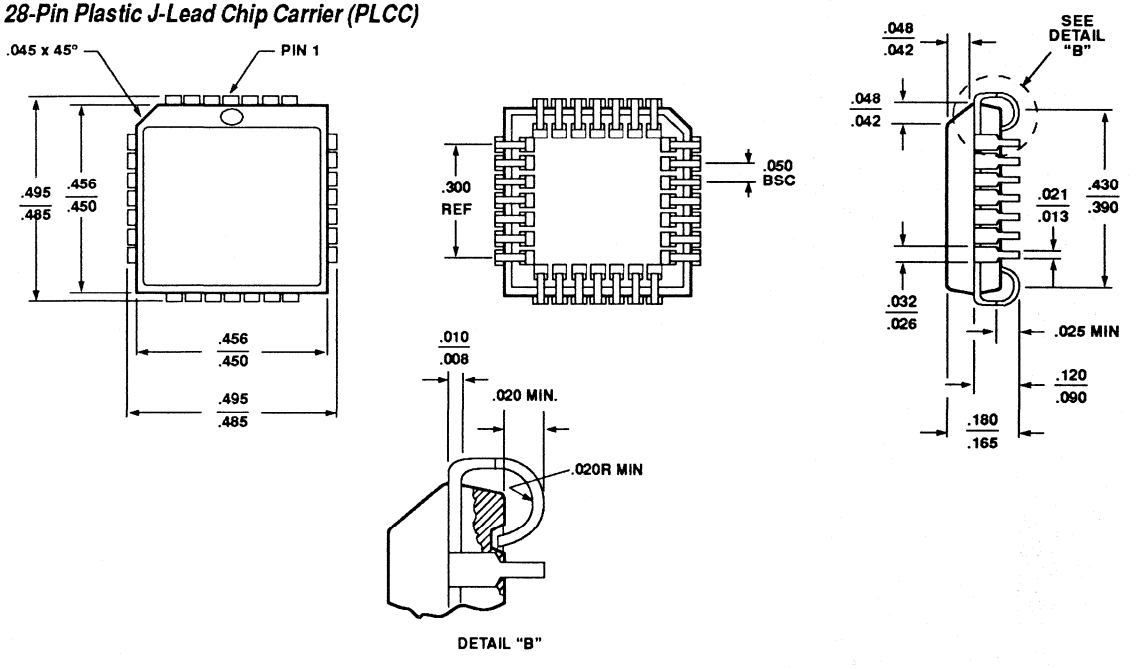


28-Pin Ceramic J-Lead Chip Carrier (JLCC)

For military-qualified product, see case outline in Altera Military Product Drawing 02D-00194.



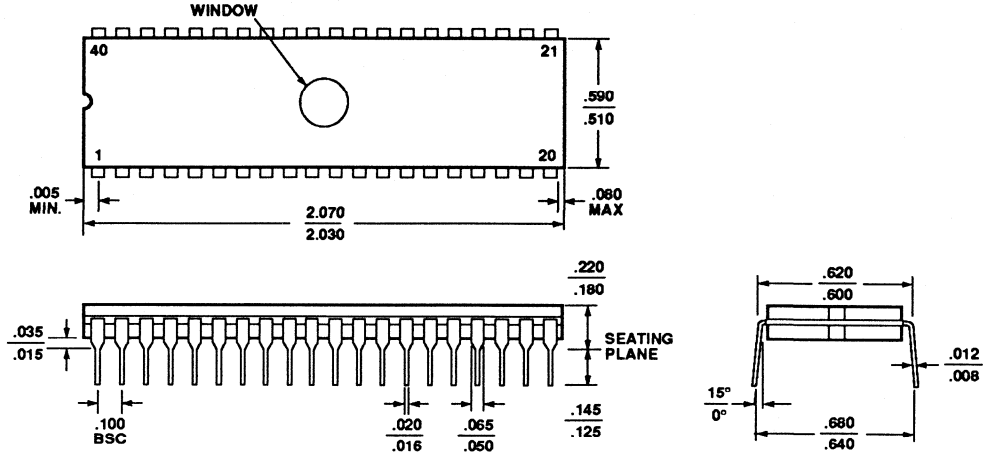
28-Pin Plastic J-Lead Chip Carrier (PLCC)



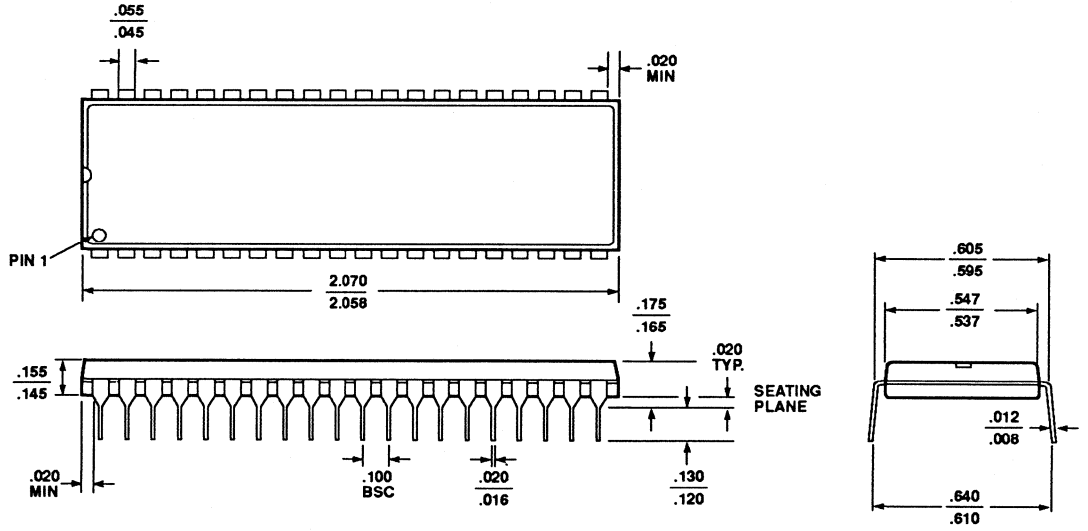
11

40-Pin Ceramic Dual In-Line Package (CerDIP)

For military-qualified product, see case outline D-5 in Appendix C of MIL-M-38510.

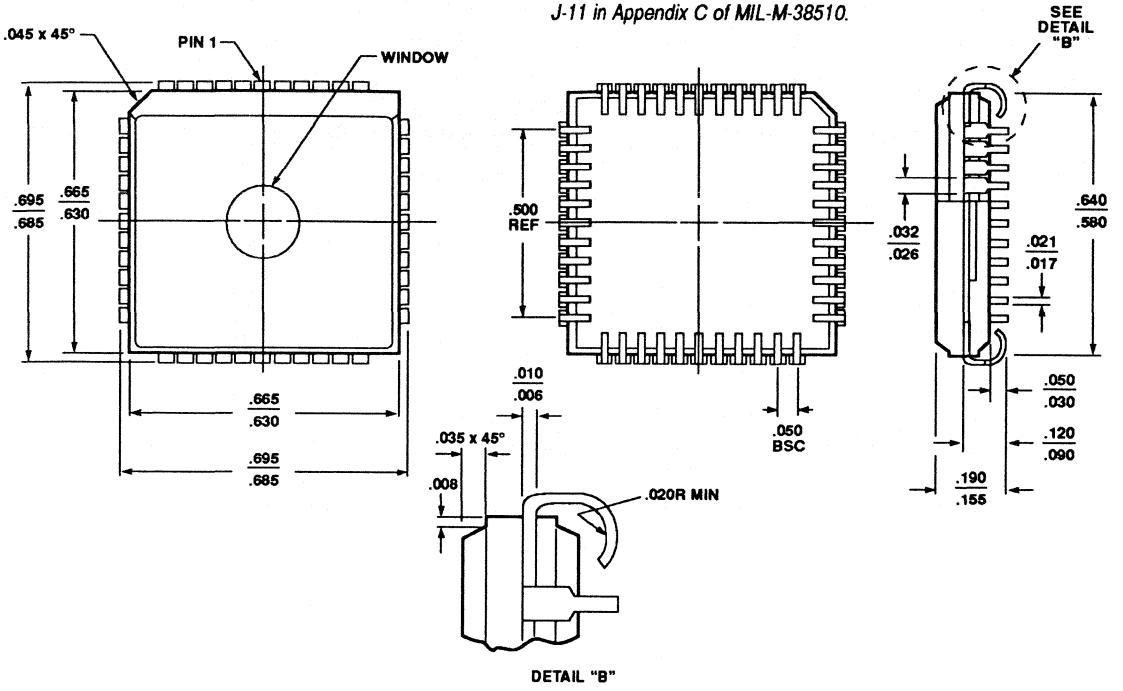


40-Pin Plastic Dual In-Line Package (PDIP)

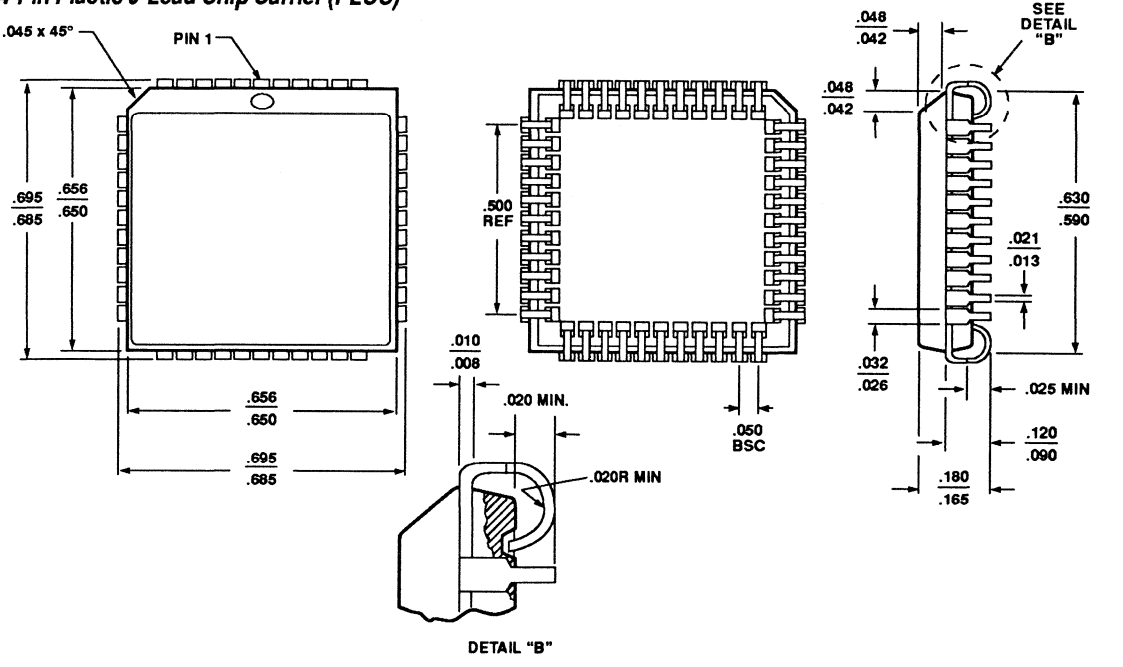


44-Pin Ceramic J-Lead Chip Carrier (JLCC)

For military-qualified product, see case outline J-11 in Appendix C of MIL-M-38510.



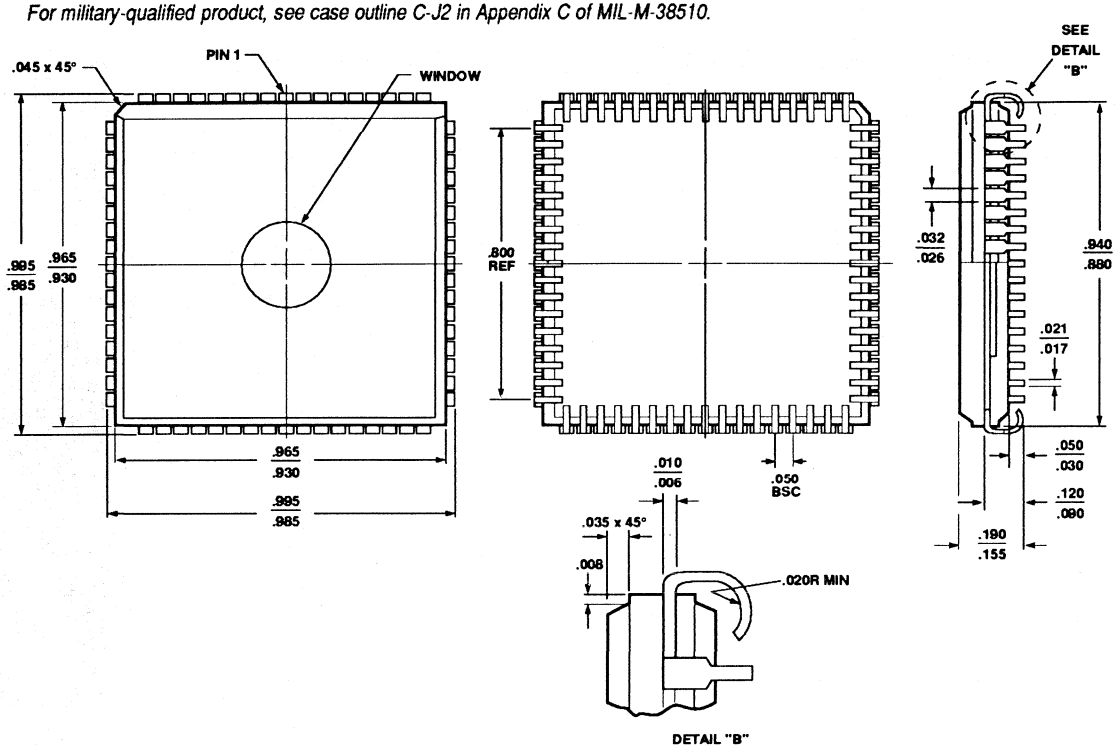
44-Pin Plastic J-Lead Chip Carrier (PLCC)



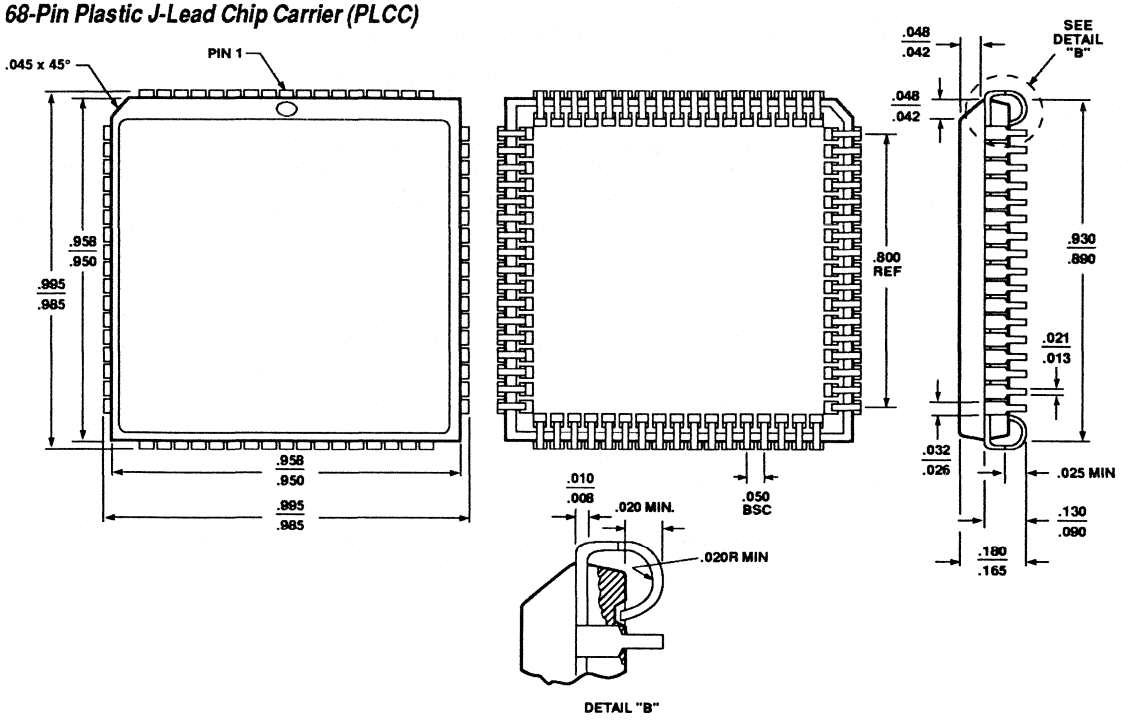
11

68-Pin Ceramic J-Lead Chip Carrier (JLCC)

For military-qualified product, see case outline C-J2 in Appendix C of MIL-M-38510.

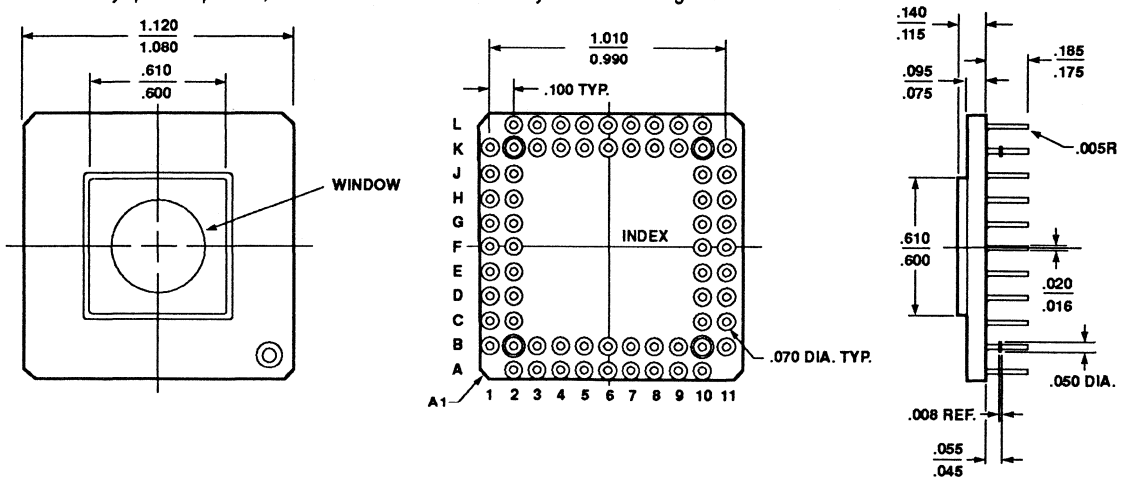


68-Pin Plastic J-Lead Chip Carrier (PLCC)



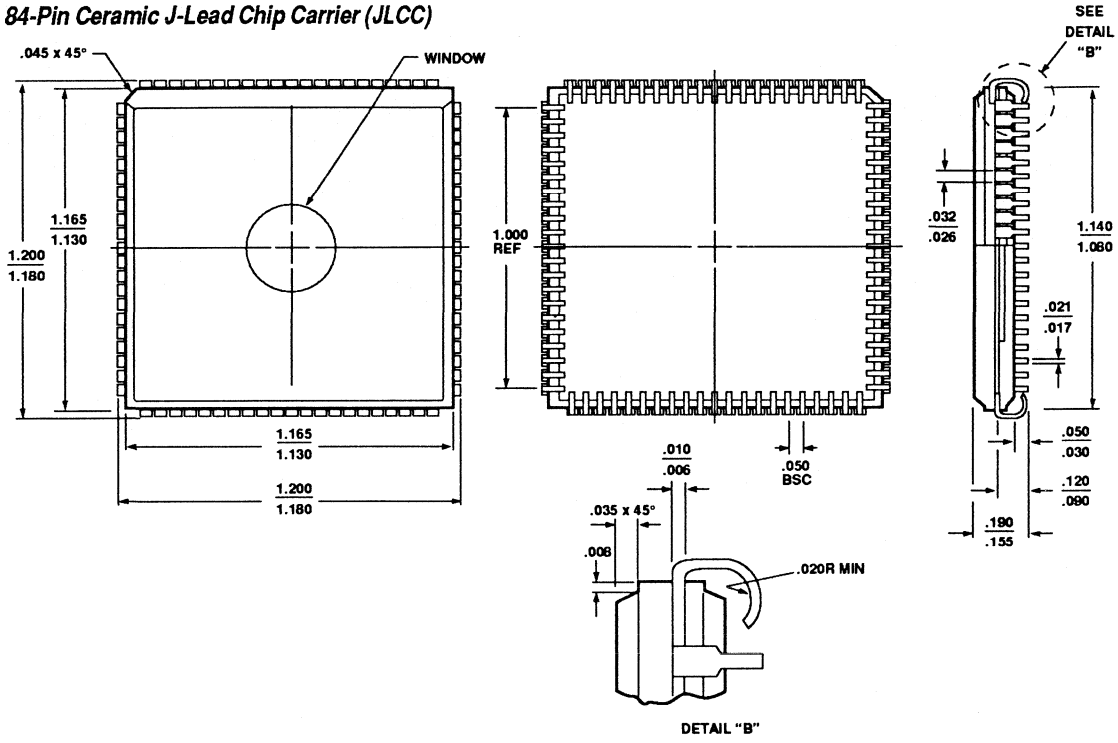
68-Pin Ceramic Pin-Grid Array (PGA)

For military-qualified product, see case outline in Altera Military Product Drawing 02D-00205.

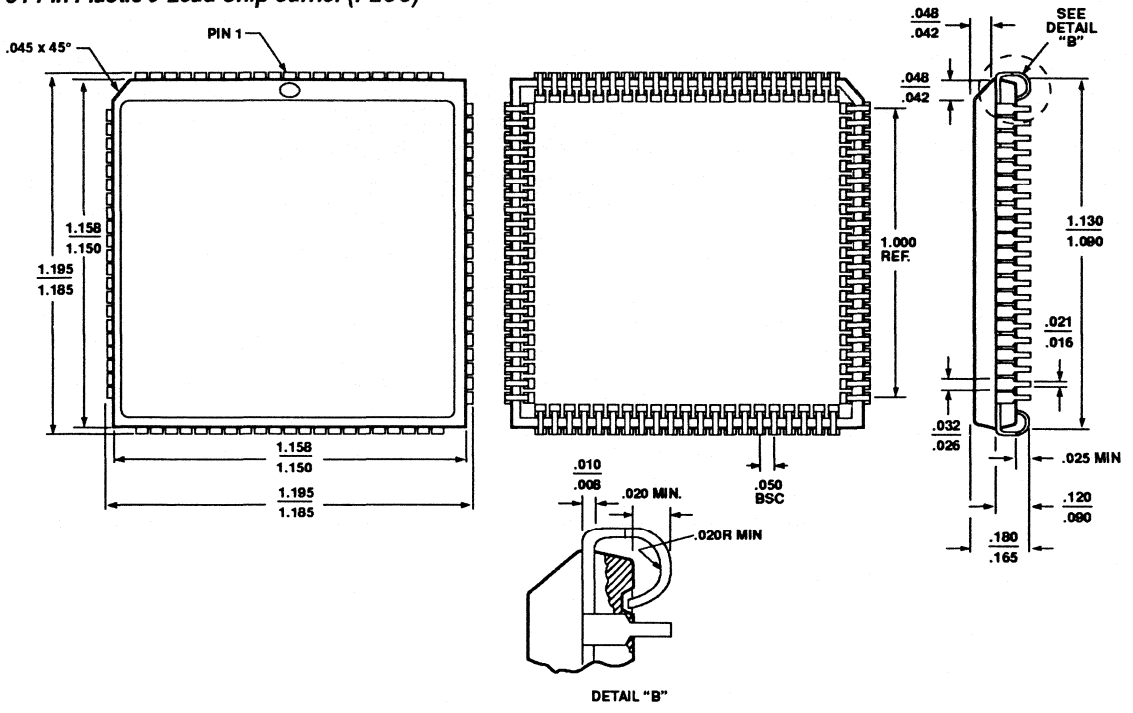


11

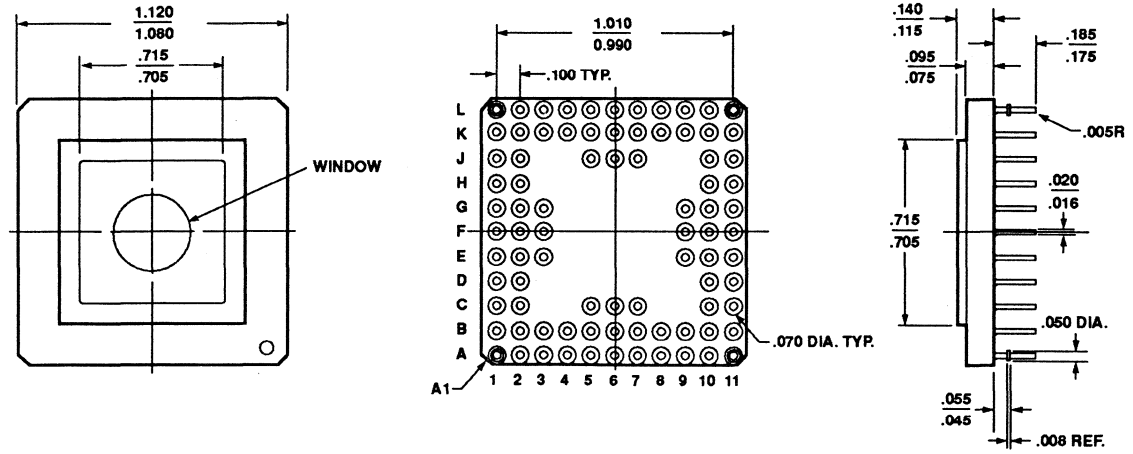
84-Pin Ceramic J-Lead Chip Carrier (JLCC)



84-Pin Plastic J-Lead Chip Carrier (PLCC)

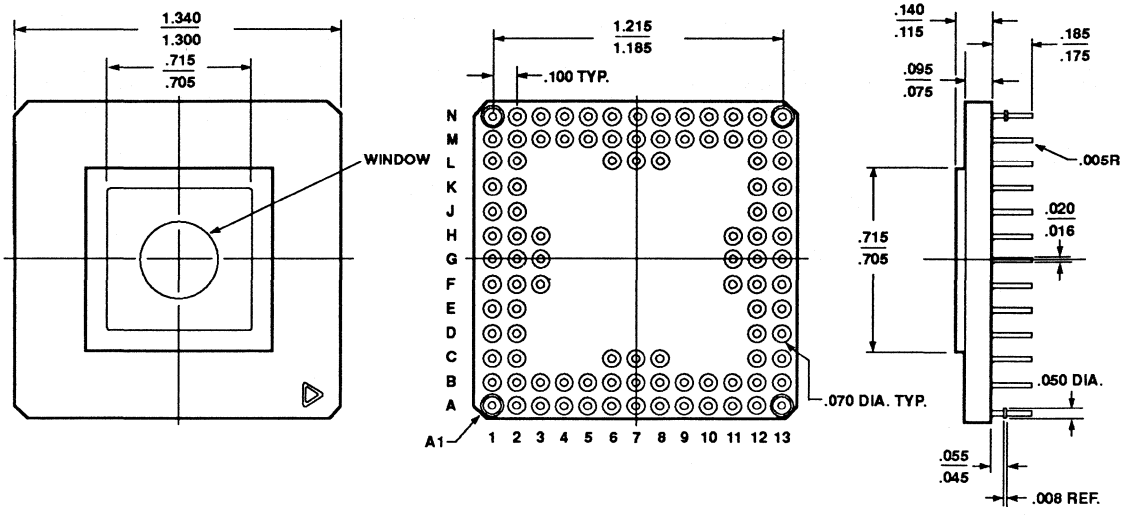


84-Pin Ceramic Pin-Grid Array (PGA)

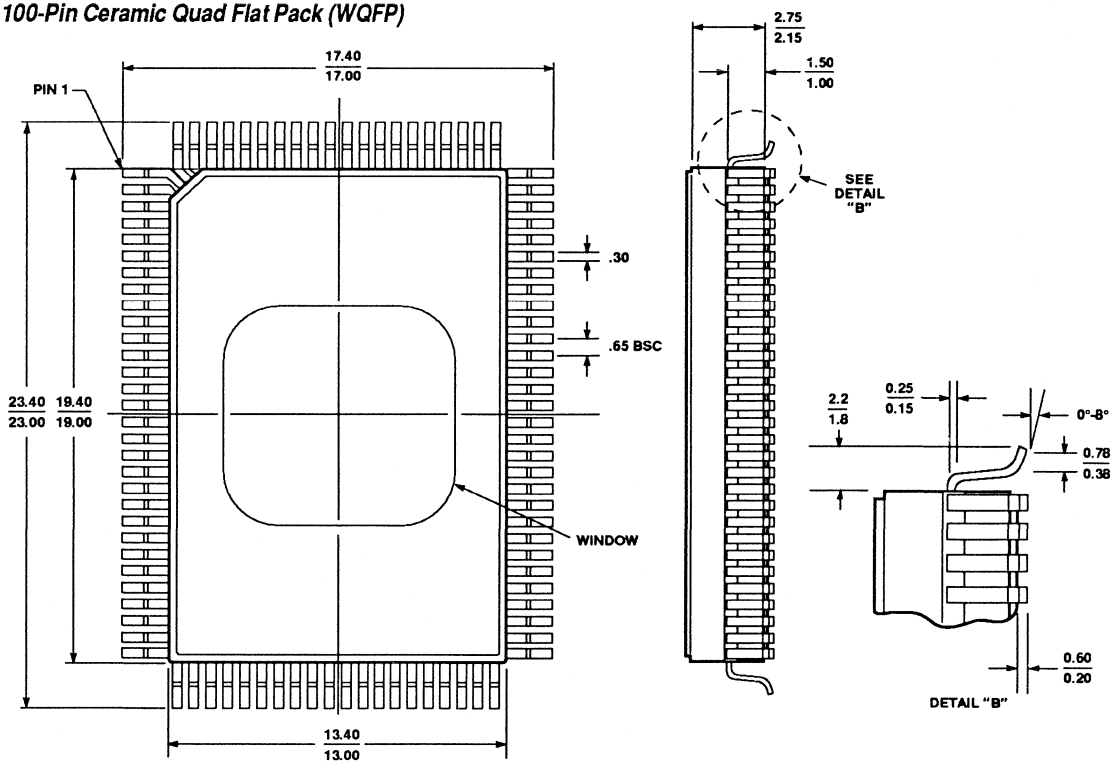


11

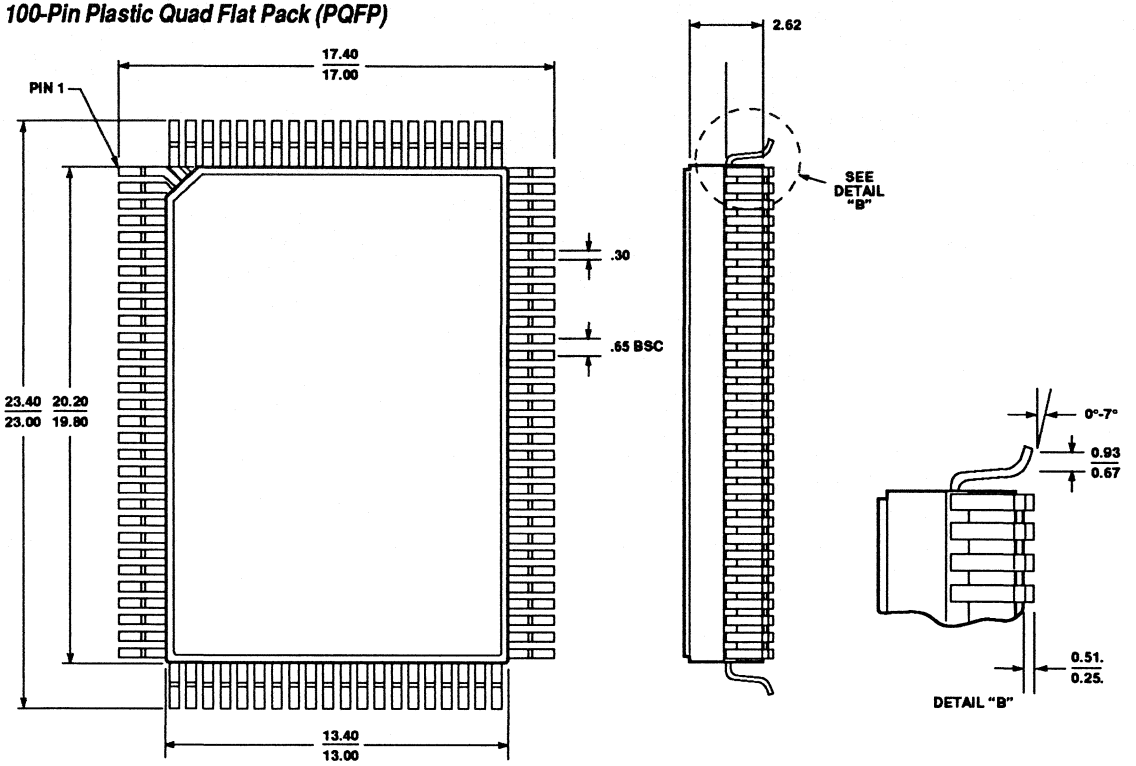
100-Pin Ceramic Pin-Grid Array (PGA)



100-Pin Ceramic Quad Flat Pack (WQFP)



100-Pin Plastic Quad Flat Pack (PQFP)



Thermal Resistance (°C/W)

October 1990, ver. 1

Data Sheet

Introduction

Table 1 gives thermal resistance data for Altera EPLDs. All thermal characteristics are measured using the Temperature Sensitive Parameter (TSP) test method described in MIL-STD-883C, Method 1012.1. Thermal resistance values were measured with the package soldered into a PC board (excluding 24-pin CerDIP and PDIP, which were socket-mounted), at 25 °C ambient temperature with no backplane or heat sink, and are accurate to ±5 °C/W.

Table 1. Thermal Resistance

Number of Pins	Package	θ_{JA} (° C/W)	θ_{JC} (° C/W)
20	CerDIP	62	17
	PDIP	65	14
	PLCC	103	27
	SOIC	90	25
24	CerDIP	64	8
	PDIP	64	11
	SOIC	92	25
28	CerDIP	52	24
	PDIP	62	40
	JLCC	72	16
	PLCC	85	27
	SOIC	90	25
40	CerDIP	40	7
	PDIP	46	19
44	JLCC	68	16
	PLCC	49	14
68	JLCC	47	7
	PLCC	41	15
	PGA	43	5
84	JLCC	24	6
	PLCC	38	15
	PGA	18	5
100	WQFP	30	8
	PQFP	48	13
	PGA	16	4

Note: The formula for determining θ_{JX} is $\theta_{JX} = (T_J - T_A)/PD$, where T_J = die junction temperature; T_A = ambient temperature; and PD = power being dissipated in the device causing a temperature rise at the die junction. T_J is determined by characterizing the relationship between the forward-biased voltage and temperature of the isolation diode between the power and ground pins of the IC.

Selecting Sockets for J-Lead Packages

October 1990, ver. 3

Application Brief 46

Introduction

EPLDs solve many of the problems designers face today. They offer low costs, low power, high reliability, and most importantly, high integration density. Altera offers windowed ceramic and plastic J-lead chip carrier (JLCC/PLCC) versions of many EPLDs to further reduce the "real estate" demands of a system. These small packages are generally intended for surface mounting. This application brief discusses the following topics:

- ❒ Types of sockets available for J-lead EPLDs
- ❒ Criteria for selecting burn-in or production sockets
- ❒ Carrier boards for use with wire-wrap panels and J-lead packages
- ❒ Results of Altera's evaluation of 68-pin production sockets for use with windowed ceramic J-lead EPLDs

Despite recent advances, the acceptance of surface-mounting technology has been slow, although considerable research and use have proved its feasibility. Most industrial applications still use traditional through-hole soldering. Surface-mount assembly places unique demands on the development and manufacturing processes: it requires different CAD symbols for PC board layout, different test and reliability procedures for buried vias within PC boards, and a different soldering process for production (vapor phase versus wave solder). Bonding EPLDs to a PC board also removes the possibility of convenient erasure and reprogramming, which are particularly important during development.

A popular compromise that preserves the advantages of J-lead packages without the complications of surface mounting is to socket the J-lead EPLD. Conventional mounting techniques can then be used, either by hole soldering to a PC board or by mounting in a socketed carrier board for wire wrap.

Mechanical Considerations

There are two distinct types of sockets: burn-in and production sockets. Burn-in sockets are zero-insertion-force sockets. Since they will not deform the device's leads, they are the preferred carrier for an EPLD during the prototyping phase of a design. Older-model burn-in sockets had the disadvantage of being significantly larger than production sockets; newer burn-in sockets are now available with dimensions similar to those of production sockets. Using a burn-in socket during prototyping is the best way to prolong the life of a windowed ceramic EPLD.

Once a design enters the production phase, cost becomes a major concern. Low-cost production sockets, designed to hold a device permanently and securely, are widely available. Obviously, they must exert a reasonable force on the leads to prevent the device from popping out. After several insertions, this force can deform the leads, and eventually the leads can short out or fail to make contact, making the EPLD unusable. Therefore, Altera strongly recommends using a burn-in socket during the design and development phases.

Production sockets must be chosen carefully. If the EPLD needs to be removed many times for reprogramming, low-insertion-force sockets that will not significantly deform the device pins for as many as ten insertions are preferable. For high-stress environments (e.g., G-forces, thermal shock, humidity), sockets with high insertion forces and optional retention clips are available. To further reduce the possibility of deforming device pins, most manufacturers of high-quality sockets include a stand-off inside the socket that prevents a device from being forced too far into a socket and becoming bent.

Socket Evaluation

Altera has tested several production sockets for use with 68-pin windowed ceramic J-lead EPLDs. Each socket underwent three tests:

1. The change in the gap between the corner pins of each device was measured before and after each of 10 insertions.

2. Each pin of the socket was wired in series and tested for open or short circuits lasting longer than 10 µs. This opens-and-shorts test was performed while the socket was attached to a vibration block. The amplitude of vibration was 3.0 mm peak-to-peak at a frequency that varied from 10 Hz to 55 Hz to 10 Hz, in 1-min. cycles for 2 hours. The ambient temperature was 70° C for this test. The vibration test was performed on all three axes at a temperature of 70° C.

3. The actual point of contact between the socket pin and the device lead was photographed to determine the direction of the forces between them and the amount of surface contact.

Of the seven sockets tested, only four passed; three sockets failed the opens-and-shorts test. Table 1 shows a ranked list of the sockets that passed. Ranking was determined primarily by each socket's ability to maintain the EPLD's pin integrity after multiple device insertions. For more information about these socket tests, call Altera Applications Engineering at 1 (800) 800-EPLD.

Table 1. Summary of 68-Pin Production Socket Analysis

Vendor and Part Number	Comments
Augat Inc. PCS-068A-1	Least pin deformation. Contact force has a downward component. No retainer clip option.
ITT/Cannon Corporation LCS-68-2	Low pin deformation. Contact force has a downward component. Has a retainer clip option.
3M/Textool Corporation 2-0068-06234-070-038-077	Moderate pin deformation. Contact force is lateral. Has a retainer clip option.
AMP, Inc. 821574-1	Moderate pin deformation. Contact force has a downward component. No retainer clip option.

Vendors may provide additional information about their products, such as material selection, prevention of solder ingress during wave soldering, or lead shape. Altera recommends qualifying sockets, just as with other components, before committing to a particular vendor.

Table 2 shows the contact distance for Altera EPLDs. These measurements should be used to select a socket (preferably with internal stand-offs) for use with Altera EPLDs.

Table 2. EPLD Contact Distances

EPLD *(1)*	Pin Count	Contact Distance (mils)	
		Minimum	Maximum
EPB2001J, EPM5192J	84	1180	1200
EPB2001L, EPM5192L	84	1185	1195
EP1810J, EP1830J, EPM5128J	68	985	995
EP1810L, EP1830L, EPM5128L	68	985	995
EP910J, EPM5064J	44	685	695
EP910L, EPM5064L	44	685	695
EP610J, EP630J, EP610AJ EPM5032J, EPS448J	28	485	495
EP610L, EP630L, EP610AL EPM5032L, EPB2002AL, EPS448L	28	485	495
EP330L, EPM5016L	20	385	395

Note:
(1) J = Ceramic J-lead chip carrier (JLCC); L = Plastic J-lead chip carrier (PLCC).

11

Packaging Options for Wire-Wrap Applications

Wire-wrap applications require a through-hole mount compatible with the J-lead package. The sockets specified do not typically mechanically conform to most wire-wrap panels. Wire-wrap cards have machine receptacles in rows with 100-mil spacing between receptacles and 300-mil spacing between rows.

Carrier boards provide an effective way to bridge the gap. Mounting a socket to a carrier board provides the convenience of wire wrap with only a small real estate penalty. Some carrier boards have signal routing with shorter paths or 45-degree bends to minimize signal reflection.

Manufacturers

Table 3 lists corporate offices of several socket and adapter manufacturers. Contact the appropriate vendor for additional information.

Table 3. Manufacturers

Company	Product	Telephone Number
AMP, Inc.	Production Sockets	(800) 522-6752
Augat, Inc.		(800) 999-9863
ITT/Cannon Corporation		(714) 757-8221
3M/Textool Corporation		(800) 225-5373
3M/Textool Corporation	Test and Burn-In Sockets	(800) 225-5373
AMP, Inc.		(800) 552-6752
Emulation Technology, Inc.		(408) 982-0660
Emulation Technology, Inc.	Carrier Boards and Wire Wrap Adapters	(408) 982-0660

Information in this application brief is based on information provided to Altera by various vendors, and is believed to be accurate. Altera assumes no liability for the use of third-party products mentioned in this publication.

Sales Offices, Distributors & Representatives

October 1990

Altera U.S. Sales Offices

CALIFORNIA
(CORPORATE HEADQUARTERS)
Altera Corporation
2610 Orchard Parkway
San Jose, CA 95134-2020
TEL: (408) 984-2800
FAX: (408) 248-7097

SOUTHERN CALIFORNIA
Altera Corporation
17100 Gillette Avenue
Irvine, CA 92714
TEL: (714) 474-9616
FAX: (714) 474-7355

GEORGIA
Altera Corporation
1080 Holcomb Bridge Road
Bldg. 100, Suite 300
Roswell, GA 30076
TEL: (404) 594-7621
FAX: (404) 998-9830

ILLINOIS
Altera Corporation
200 W. Higgins Road
Suite 322
Schaumburg, IL 60195
TEL: (708) 310-8522
FAX: (708) 310-0909

MASSACHUSETTS
Altera Corporation
945 Concord Street
Framingham, MA 01701
TEL: (508) 626-0181
FAX: (508) 879-0698

NEW JERSEY
Altera Corporation
981 U.S. Highway 22
Suite 2000
Bridgewater, NJ 08807
TEL: (908) 526-9400
FAX: (908) 526-5471

TEXAS
ALTERA CORPORATION
Signature Place
14785 Preston Road
Suite 550
Dallas, TX 75240
TEL: (214) 233-1491
FAX: (214) 233-1493

Altera International Sales Offices

UNITED STATES
(CORPORATE HEADQUARTERS)
Altera Corporation
2610 Orchard Parkway
San Jose, CA 95134-2020
USA
TEL: (408) 984-2800
TLX: 888496
FAX: (408) 248-7097

BELGIUM
(EUROPEAN HEADQUARTERS)
Altera Europe
25, Avenue Beaulieu
1160 Bruxelles
Belgium
TEL: (32) 2-660.20.77
TLX: (886) 27087901(AVVALB)
FAX: (32) 2-660 52 25

FRANCE
Altera France
72-78 Grande Rue
92310 Sèvres
France
TEL: (33) 1.45.34.3787
FAX: (33) 1.45.34.0109

GERMANY
Altera GmbH
Ismaninger Straße 21
8000 München 80
West Germany
TEL: (49) 89/413.00.6-14
TLX: (841) 5213250
FAX: (49) 89/470.62.84

Altera International Sales Offices *(continued)*

JAPAN
Altera Japan K.K.
Ichikawa Gakugeidai Building
2nd Floor
12-8 Takaban 3-chome
Meguro-ku, Tokyo 152
Japan
TEL: (03) 716-2241
FAX: (03) 716-7924

UNITED KINGDOM
Altera UK
21 Broadway
Maidenhead, Berkshire
England SL6 1JK
TEL: (44) 628-32516
TLX: (851) 94016389 (TIME G)
FAX: (44) 628-770892

North American Distributors

Alliance Electronics

Anthem

Future Electronics (Canada only)

Pioneer-Standard Electronics

Pioneer Technologies Group

Lex Electronics

Semad (Canada only)

Wyle Laboratories

U.S. Sales Representatives

ALABAMA
Montgomery Marketing, Inc.
1910 Sparkman Drive
Huntsville, AL 35816
(205) 830-0498

ARIZONA
Tusar
6016 E. Larkspur
Scottsdale, AZ 85254
(602) 998-3688

ARKANSAS
Technical Marketing, Inc.
3320 Wiley Post Road
Carrollton, TX 75006
(214) 387-3601

CALIFORNIA
Exis, Inc.
11223 Welty Lane
Auburn, CA 95603
(916) 885-3062

Exis, Inc.
2860 Zanker Road, Suite 108
San Jose, CA 95134
(408) 433-3947

QuadRep Southern, Inc.
28720 Roadside Drive, Suite 227
Agoura, CA 91301
(818) 597-0222

CALIFORNIA *(continued)*
QuadRep Southern, Inc.
4 Jenner Street, Suite 120
Irvine, CA 92718
(714) 727-4222

QuadRep Southern, Inc.
7585 Ronson Road, Suite 100
San Diego, CA 92111
(619) 560-8330

COLORADO
Lange Sales, Inc.
1500 W. Canal Court, Bldg. A, Suite 100
Littleton, CO 80120
(303) 795-3600

CONNECTICUT
Technology Sales, Inc.
237 Hall Avenue
Wallingford, CT 06492
(203) 269-8853

DELAWARE
BGR Associates
Evesham Commons
525 Route 73, Ste.100
Marlton, NJ 08053
(609) 983-1020

DISTRICT OF COLUMBIA
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
(301) 995-1900

FLORIDA
EIR, Inc.
1057 Maitland Center Commons
Maitland, FL 32751
(407) 660-9600

GEORGIA
Montgomery Marketing, Inc.
3000 Northwoods Pkwy., Suite 110
Norcross, GA 30071
(404) 447-6124

IDAHO
Lange Sales, Inc.
5440 Franklin St., Suite 110
Boise, ID 83705
(208) 345-8207

Westerberg & Associates, Inc.
12505 NE Bel-Red Road, Suite 112
Bellevue, WA 98005
(206) 453-8881

ILLINOIS
AEM, Inc.
11520 St. Charles Rock Road, Suite 131
Bridgeton, MO 63044
(314) 298-9900

Oasis Sales Corporation
1101 Tonne Road
Elk Grove Village, IL 60007
(708) 640-1850

INDIANA
Electro Reps, Inc.
7240 Shadeland Station, Suite 275
Indianapolis, IN 46256
(317) 842-7202

IOWA
AEM, Inc.
4001 Shady Oak Drive
Marion, IA 52302
(319) 377-1129

KANSAS
AEM, Inc.
8859 Long Street
Lenexa, KS 65215
(913) 888-0022

KENTUCKY
Electro Reps, Inc.
7240 Shadeland Station, Suite 275
Indianapolis, IN 46256
(317) 842-7202

LOUISIANA
Technical Marketing, Inc.
2901 Wilcrest Drive, Suite 139
Houston, TX 77042
(713) 783-4497

MAINE
Technology Sales, Inc.
332 Second Avenue
Waltham, MA 02154
(617) 890-5700

MARYLAND
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
(301) 995-1900

MASSACHUSETTS
Technology Sales, Inc.
332 Second Avenue
Waltham, MA 02154
(617) 890-5700

MICHIGAN
Rathsburg Associates, Inc.
34605 Twelve Mile Road
Farmington Hills, MI 48331
(313) 489-1500

MINNESOTA
Cahill, Schmitz & Cahill, Inc.
315 N. Pierce
St. Paul, MN 55104
(612) 646-7217

MISSISSIPPI
Montgomery Marketing, Inc.
3000 Northwoods Parkway, Suite 110
Norcross, GA 30071
(404) 447-6124

MISSOURI
AEM, Inc.
11520 St. Charles Rock Road, Suite 131
Bridgeton, MO 63044
(314) 298-9900

MONTANA
Lange Sales, Inc.
1500 W. Canal Court, Bldg. A, Suite 100
Littleton, CO 80120
(303) 795-3600

NEBRASKA
AEM, Inc.
4001 Shady Oak Drive
Marion, IA 52302
(319) 377-1129

NEVADA
Exis, Inc.
2860 Zanker Road, Suite 108
San Jose, CA 95134
(408) 433-3947

U.S. Sales Representatives
(continued)

NEVADA *(continued)*
Tusar
6016 E. Larkspur
Scottsdale, AZ 85254
(602) 998-3688

NEW HAMPSHIRE
Technology Sales, Inc.
332 Second Avenue
Waltham, MA 02154
(617) 890-5700

NEW JERSEY
BGR Associates
Evesham Commons
525 Route 73, Suite100
Marlton, NJ 08053
(609) 983-1020

ERA, Inc.
354 Veterans Memorial Highway
Commack, NY 11725
(516) 543-0510

NEW MEXICO
Nelco Electronix
3240 C Juan Tabo Blvd. NE
Albuquerque, NM 87111
(505) 293-1399

NEW YORK
ERA, Inc. (New York Metro)
354 Veterans Memorial Highway
Commack, NY 11725
(516) 543-0510

Technology Sales, Inc.
205 N. McKinley Avenue
Endicott, NY 13760
(607) 257-7070

Technology Sales, Inc.
470 Perinton Hills Office Park
Fairport, NY 14450
(716) 223-7500

Technology Sales, Inc.
145 Oakwood Lane
Ithaca, NY 14850
(607) 273-1188

NORTH CAROLINA
Montgomery Marketing, Inc.
P.O. Box 520
Cary, NC 27512
(919) 467-6319

Montgomery Marketing, Inc.
1200 Trinity Road
Raleigh, NC 27607
(919) 851-0010

NORTH DAKOTA
Cahill, Schmitz & Cahill, Inc.
315 N. Pierce
St. Paul, MN 55104
(612) 646-7217

OHIO
The Lyons Corporation
4812 Frederick Road, Suite 101
Dayton, OH 45414
(513) 278-0714

The Lyons Corporation
4615 W Streetsboro Road
Richfield, OH 44286
(216) 659-9224

The Lyons Corporation
248 N. State Street
Westerville, OH 43081
(614) 895-1447

OKLAHOMA
Technical Marketing, Inc.
3320 Wiley Post Road
Carrollton, TX 75006
(214) 387-3601

OREGON
Westerberg & Associates, Inc.
7165 SW Fir Loop
Portland, OR 97223
(503) 620-1931

PENNSYLVANIA
BGR Associates
Evesham Commons
525 Route 73, Suite100
Marlton, NJ 08053
(609) 983-1020

The Lyons Corporation
4812 Frederick Rd., Ste. 101
Dayton, OH 45414
(513) 278-0714

PUERTO RICO
Technology Sales, Inc.
Edificio Rali 219
San German, PR 00753
(809) 892-4745

RHODE ISLAND
Technology Sales, Inc.
332 Second Avenue
Waltham, MA 02154
(617) 890-5700

SOUTH CAROLINA
Montgomery Marketing, Inc.
1200 Trinity Road
Raleigh, NC 27607
(919) 851-0010

SOUTH DAKOTA
Cahill, Schmitz & Cahill, Inc.
315 N. Pierce
St. Paul, MN 55104
(612) 646-7217

TENNESSEE
Montgomery Marketing, Inc.
1910 Sparkman Drive
Huntsville, AL 35816
(205) 830-0498

TEXAS
Technical Marketing, Inc.
3320 Wiley Post Road
Carrollton, TX 75006
(214) 387-3601

Technical Marketing, Inc.
2901 Wilcrest Drive, Suite 139
Houston, TX 77042
(713) 783-4497

Technical Marketing, Inc.
1315 Sam Bass Circle, Suite B-3
Round Rock, TX 78681
(512) 244 2991

UTAH
Lange Sales, Inc.
1864 S. State, Suite 295
Salt Lake City, UT 84115
(801) 487-0843

VERMONT
Technology Sales, Inc.
332 Second Avenue
Waltham, MA 02154
(617) 890-5700

VIRGINIA
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
(301) 995-1900

WASHINGTON
Westerberg & Associates, Inc.
12505 NE Bel-Red Road, Suite 112
Bellevue, WA 98005
(206) 453-8881

WEST VIRGINIA
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
(301) 995-1900

WISCONSIN
Cahill, Schmitz & Cahill, Inc.
315 N. Pierce
St. Paul, MN 55104
(612) 646-7217

Oasis Sales Corporation
1305 N. Barker Road
Brookfield, WI 53005
(414) 782-6660

WYOMING
Lange Sales, Inc.
1500 W. Canal Court, Bldg. A, Suite 100
Littleton, CO 80120
(303) 795-3600

Canadian Sales Representatives

BRITISH COLUMBIA
Kaytronics
#102-4585 Canada Way
Burnaby, BC V5G 4L6
Canada
(604) 294-2000

ONTARIO
Kaytronics
331 Bowes Road, Unit 1
Concord, Ontario L4K 1J2
Canada
(416) 669-2262

ONTARIO *(continued)*
Kaytronics
300 March Road, Suite 603
Kanata, Ontario K2K 2E2
Canada
(613) 564-0080

QUEBEC
Kaytronics
5800 Thimens Boulevard
Ville St. Laurent, Quebec H4S 1S5
Canada
(514) 745-5800

International Distributors

ARGENTINA
YEL S.R.L.
Virrey Cevallos 143
Piso 8, Of. 41
1077 Buenos Aires
Argentina
TEL: (54) 1-40-1025/2525
(54) 1-45-7140/7163
(54) 1-46-2211
TLX: (390) 18605 (YEL AR)
FAX: (54) 1-45-2551

AUSTRALIA
Veltek
22 Harker St.
Burwood, Victoria 3125
Australia
TEL: (61) 3-808-7511
FAX: (61) 3-808-5473

11

International Distributors
(continued)

AUSTRIA
Hitronik
St.-Veit-Gasse 51
1130 Wien
Austria
TEL: (43) 222-824-199
TLX: (847) 134404 (HIT)
FAX: (43) 222-828-557-285

BELGIUM
D&D Electronics
VIIe Olympiadelaan 93
2020 Antwerpen
Belgium
TEL: (32) 3-827-7934
TLX: (846) 73121 (DDELEC BU)
FAX: (32) 3-828-7254

BRAZIL
Uniao Digital COM E REP
Rua Texas No. 622-Brooklin
São Paulo SP CEP 04551
Brazil
TEL: (55) 11 533-0967
(55) 11 611-1256
TLX: (391) 11-37230 (BRVC)
FAX: (55) 11 533-6780

DENMARK
E.V. Johanssen Elektronik A/S
Titangade 15
2200 Koebenhavn N
Denmark
TEL: (45) 31.83.90.22
TLX: (855) 16522 (EVICAS DK)
FAX: (45) 31.83.92.22

FINLAND
Yleiselektroniikka Oy
P.O. Box 73
Luomannotko 6
02201 Espoo
Finland
TEL: (358) 0-452-1622
TLX: (857) 123212 (YLEOY SF)
FAX: (358) 0-452-3337

FRANCE
Tekelec Airtronic SA
Cité des Bruyères
Rue Carle Vernet
92310 Sèvres
France
TEL: (33) 1.45.34.75.35
TLX: (842) 204552 (TKLEC A F)
FAX: (33) 1.45.07.21.91

GERMANY
Electronic 2000
Vertriebs-AG
Stahlgruberring 12
8000 München 82
West Germany
TEL: (49) 89/42001-0
TLX: (841) 522561 (ELEC D)
FAX: (49) 89/42001-209

HONG KONG
RTI Industries
A19, 10/F, Proficient Ind. Centre
6 Wang Kwun Road
Kowloon
Hong Kong
TEL: (852) 3-7957421
FAX: (852) 3-7957839

INDIA
Capricorn Systems International
No. 3306, First Floor
10th 'A' Main Road
33rd Cross, 4th Block
Jayanagar, Bangalore 560 011
India
TEL: (91) 812-640661
TLX: (953) 08458162 (SRIS IN)
FAX: (91) 812-643608

Capricorn Systems International
1340 Tully Road, Suite 308
San Jose, CA 95122
USA
TEL: (408) 294-2833
TLX: 714997729 (CAPCNUI)
FAX: (408) 294-0355

ISRAEL
Vectronics Ltd.
60 Medinat Hayehudim Street
P.O. Box 2024
Herzlia B 46120
Israel
TEL: (972) 52-556-070
TLX: (922) 342579 (VECO IL)
FAX: (972) 52-556-508

Active Technologies
147-16 181st Street
Jamaica, NY 11413
USA
TEL: (718) 244-0909
FAX: (718) 244-0912

ITALY
Inter-Rep S.p.A.
Via Orbetello 98
10148 Torino
Italy
TEL: (39) 11/29191
TLX: (843) 221422 (IR TO I)
FAX: (39) 11/2165915

JAPAN
Japan Macnics Corporation
Hakusan High-Tech Park
807 Hakusan-Cho, Midori-Ku
Yokohama 226
Japan
TEL: (81) 45-939-6140
FAX: (81) 45-939-6141

JAPAN *(continued)*
Paltek Corporation
3-8-18 Yoga
Setagaya-Ku
Tokyo 158
Japan
TEL: (81) 3-707-5455
TLX: (781) 02425205 (PALTEK J)
FAX: (81) 3-707-5338

KOREA
MJL Korea, Ltd.
Samwhan Camus Bldg.
17-3 Youido-Dong
Yeungdeungpo-Ku
Seoul 150-010
Korea
TEL: (82) 2-784-8000
TLX: (787) 28907 (MJL)
FAX: (82) 2-784-4644

MJL Corporation - USA
622 Rosedale Road
Princeton, NJ 08540
USA
TEL: (609) 683-1700
TLX: 843457 (MJL MORV)
FAX: (609) 683-7447

LATIN AMERICA
Intectra
2629 Terminal Boulevard
Mountain View, CA 94043
USA
TEL: (415) 967-8818
TLX: 345545 (INTECTRA MNTV)
FAX: (415) 967-8836

NETHERLANDS
Koning en Hartman
1 Energieweg
2627 AP Delft
Netherlands
TEL: (31) 15 609906
TLX: 38250 (KOHA NL)
FAX: (31) 15 619194

NORWAY
Eltron A/S
Aslakveien 20 F
0753 Oslo 7
Norway
TEL: (47) 2-500650
TLX: (856) 77144 (ELTRO N)
FAX: (47) 2-502777

SINGAPORE
Impact Sound PTE
65 Upper Paya Lebar Road 03-01
Guang Ming Ind. Bldg., 1953
Singapore
TEL: (65) 2914953, (65) 2817244
TLX: (786) 39142 (IMPACT RS)
FAX: (65) 2812786

SINGAPORE *(continued)*
QuadRep Marketing (S) Pte Ltd
5611 North Bridge Road
#02-10 Eng Cheong Tower
Singapore 0719
TEL: (65) 2949998
FAX: (65) 2911213

SPAIN
Selco
Paseo de la Habana, 190
28036 Madrid
Spain
TEL: (34) 1-326-4213
TLX: (831) 45458 (EPAR E)
FAX: (34) 1-259-2284

SWEDEN
Nordisk Elektronik AB
Torshamnsgatan 39
7th Floor
16493 Kista
Sweden
TEL: (46) 8.703.46.30
FAX: (46) 8.703.98.45

SWITZERLAND
Stolz AG
Täfernstraße 15
5405 Baden-Dättwil
Switzerland
TEL: (41) 56.84.90.00
TLX: (845) 825088 (SAG CH)
FAX: (41) 56.83.19.63

TAIWAN
Galaxy Far East Corp.
8F-6 390, Sec. 1
Fu Hsing S. Road
Taipei
Taiwan
TEL: (886) 2-7057266
TLX: (785) 26110 (GALAXYER)
FAX: (886) 2-7087901

UNITED KINGDOM
Ambar Cascom Ltd.
Rabans Close
Aylesbury, Bucks HP19 3RS
England
TEL: (44) 296 434141
TLX: (851) 837427 (AMBAR G)
FAX: (44) 296 29670

Thame Components Ltd.
Thame Park Road
Thame, Oxfordshire 0X9 3UQ
England
TEL: (44) 844 261188
TLX: (851) 837917 (MEMEC G)
FAX: (44) 844 261681

11

Notes: