ALTERA

FLEX 8000 Handbook

The ALTERA Advantage

May 1994

FLEX 8000 Handbook
May 1994 A-FLEXHB-94-01

Printed on Recycled Paper

About this Handbook

May 1994

This handbook 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. Design guidelines and tips on how to use Altera's FLEX 8000 devices are also provided. For complete information on device characteristics, refer to the current *FLEX 8000 Programmable Logic Device Family Data Sheet*. For information on software characteristics and specifications, refer to the current *MAX+PLUS II Programmable Logic Development System & Software Data Sheet*. For information on Classic, MAX 5000, and MAX 7000 devices, refer to the current Altera data book.

For immediate assistance on technical questions, call:

Altera Applications Hotline

(800) 800-EPLD

For information on product availability, pricing, and order status, contact your local Altera representative or distributor listed in *Sales Offices, Distributors & Representatives* in this handbook.

If you have questions that cannot be answered by the local representative or distributor, contact:

Altera Marketing

Tel: (408) 894-7000
Fax: (408) 248-6924

ALTERA

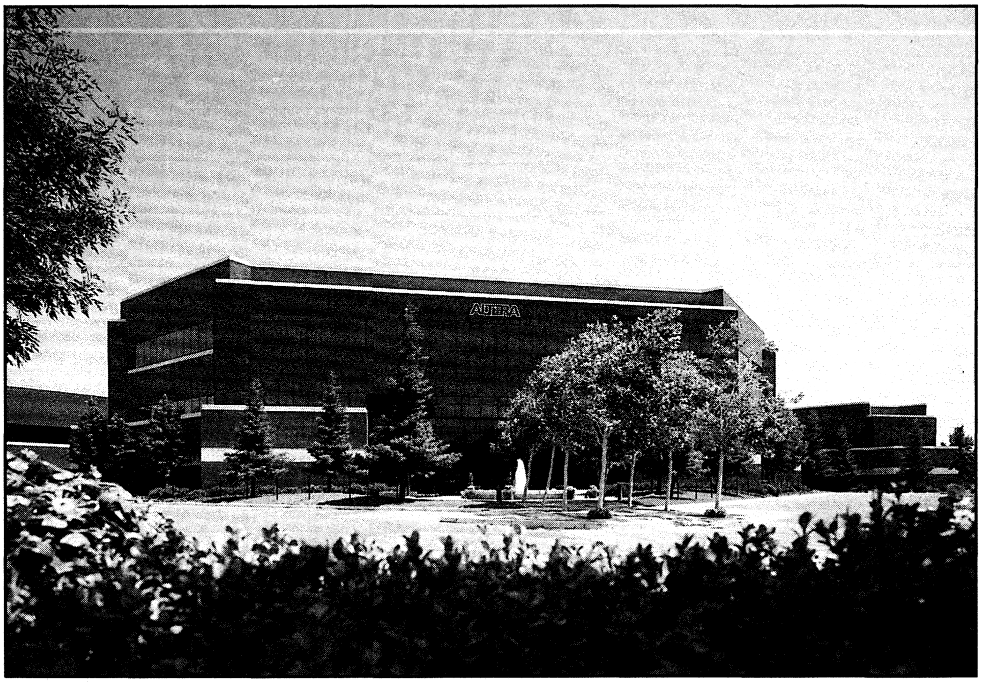
Contents

May 1994

Contents

May 1994

Section 1 Introduction


ALTERA

Introduction

May 1994, ver. 2

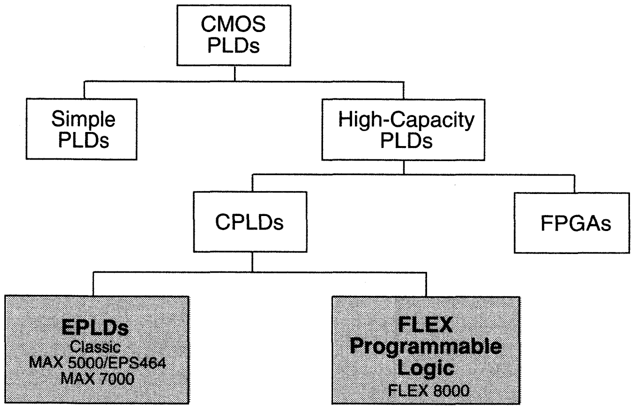
Programmable logic devices (PLDs) are digital, user-configurable integrated circuits (ICs) used to implement custom logic functions. PLDs can implement any Boolean expression or registered function using their built-in logic structures. In contrast, off-the-shelf logic ICs, such as TTL devices, provide a specific logic function and cannot be modified to meet individual circuit design requirements. PLDs were once viewed as an alternative to discrete logic and custom or semi-custom devices such as ASICs. In recent years, however, PLDs have become the preferred choice. As PLD costs have decreased through high-volume manufacturing and the use of aggressive process technologies, PLD manufacturers have been able to offer devices with higher integration, higher performance, and lower cost per function than most discrete and custom devices.

The PLD Market

Programmable logic encompasses all digital logic circuits configured by the end-user, including simple, low-density, 20-pin PAL/GAL devices, Field Programmable Gate Arrays (FPGAs), and Complex PLDs (CPLDs). PLDs are offered in different architectures and a variety of memory technologies for configuring the devices. Figure 1 shows the relative position of Altera general-purpose devices (FLEX 8000, MAX 7000, MAX 5000/EPS464, and Classic) in the CMOS programmable logic device market.

CPLDs and FPGAs have different interconnect structures. The segmented interconnect structure of FPGAs uses varying lengths of metal lines

Figure 1. Altera General-Purpose Logic Devices



connected by pass transistors to connect logic cells. In contrast, the continuous interconnect structure of CPLDs uses metal lines that are all of the same length to provide logic cell-to-logic cell connectivity. The continuous interconnect structure eliminates the timing variability associated with a segmented interconnect structure, and provides fast, fixed delay paths between logic cells. This structure makes it easier to implement a design, and thus shortens the development cycle.

Advantages of Altera PLDs

Designers generally develop a logic circuit with one of three distinctly different implementation options: discrete logic (TTL, CMOS, etc.), custom or semi-custom devices (ASICs), or PLDs. The best option is one that can meet the largest number of design requirements. Table 1 lists a number of important requirements and rates the three options according to their effectiveness in meeting these requirements.

Table 1. Device Options Rating

Requirement	PLD	Discrete Logic	Custom Device
Speed	●	❍	●
Density	●	❍	●
Cost	●	❍	● *(1)*
Development time	●	◗	❍
Prototyping & simulation time	●	❍	❍
Manufacturing time	●	◗	❍
Ease of use	●	◗	❍
Future modification	●	◗	❍
Inventory risk	●	●	❍
Development tool support	●	❍	●

Notes:
(1) Cost-effective only in high-volume production
● Very effective
◗ Adequate
❍ Poor

Altera PLDs not only offer the general benefits of PLD technology, but other advantages based on innovative architectures, aggressive technologies, and the MAX+PLUS II programmable logic development environment. These advantages include:

- ❑ Higher performance
- ❑ High-density logic integration
- ❑ Greater cost-effectiveness
- ❑ Shorter development cycles with MAX+PLUS II software

Higher Performance

Performance is a function of process and architecture. Altera devices are manufactured on state-of-the-art CMOS processes, which offer the shortest possible delays. In addition, the devices' continuous interconnect structures provide fast, consistent signal delays throughout the device.

High-Density Logic Integration

Designers often seek the highest possible logic integration for their designs, usually to reduce board space and cost. Also, existing designs often undergo secondary development cycles that aim to reduce cost by integrating more logic into fewer devices. In both cases, PLDs with high logic integration capability offer an excellent solution. Altera devices—which range in density from 300 to 50,000 usable gates—can easily integrate existing logic, whether it be a small or a large number of discrete logic devices, PLDs, FPGAs, or even custom devices. This high integration capability provides higher performance and reliability, as well as lower system cost.

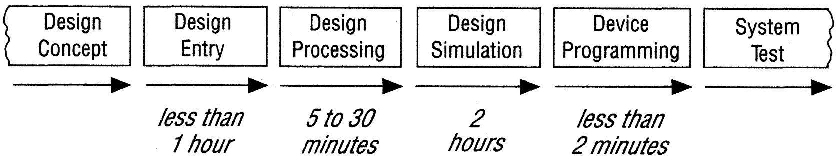
Greater Cost-Effectiveness

Altera continually strives to refine product development and manufacturing processes. The expertise accumulated over more than a decade of leadership has made both process technologies and the manufacturing flow highly efficient, and has enabled Altera to offer the most cost-effective, highest-performance programmable logic available.

Shorter Development Cycles with MAX+PLUS II Software

Time is the most precious resource for many design engineers. Large sums of money are wasted on projects that are not completed on schedule and therefore miss a window of opportunity. Consequently, the shorter the development cycle, the better. Altera's fast, intuitive, and easy-to-use MAX+PLUS II software can shorten the development cycle considerably. Design entry, processing, verification, and device programming together take only a few hours, potentially allowing several complete design iterations in one day. Figure 2 illustrates a typical PLD development cycle in the MAX+PLUS II development environment. Times shown are representative of a relatively sophisticated 10,000-gate logic design.

Figure 2. Development Cycle for Altera Devices



Altera Device Families

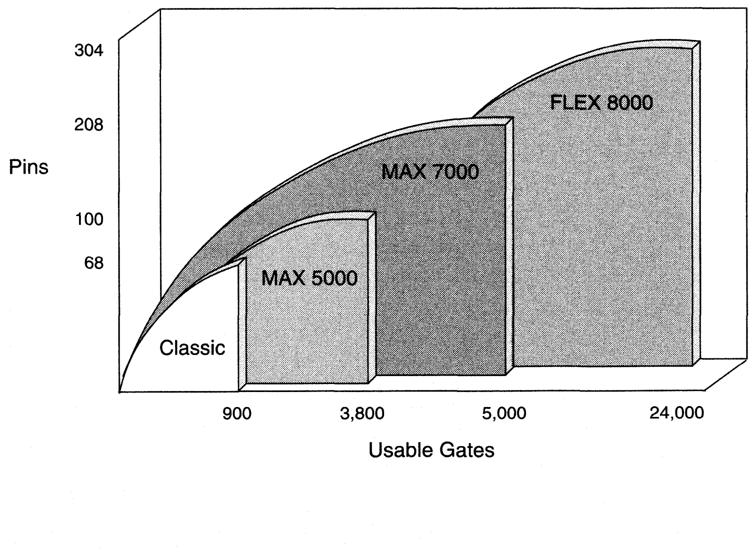
Altera offers four families of general-purpose PLDs: FLEX 8000, MAX 7000, MAX 5000/EPS464, and Classic. The FLEX 8000 family architecture uses look-up tables (LUTs) to implement logic functions, whereas the Multiple Array Matrix (MAX) and Classic families use a programmable-AND/fixed-OR product-term architecture. Each architecture offers distinct speed and utilization advantages for implementing a particular application. See Table 2.

Table 2. Altera Device Architecture & Technology

Device Family	Logic Cell Structure	Interconnect Structure	Technology
FLEX 8000	Look-Up Table	Continuous	SRAM
MAX 7000	Sum-of-Products	Continuous	EEPROM
MAX 5000/EPS464	Sum-of-Products	Continuous	EPROM
Classic	Sum-of-Products	Continuous	EPROM

The following descriptions summarize the key features and benefits of Altera's general-purpose, programmable logic device families. Figure 3 compares the pin count and density of each device family.

Figure 3. Pin Count & Density in Altera Device Families



FLEX 8000 Family

The FLEX 8000 family is ideal for applications that require a large number of registers and I/O pins. Devices in the family range in density from 2,500 to 50,000 usable gates, with 282 to 4,752 registers, and 78 to 360 user I/O pins. These features and the high-performance, predictable interconnect scheme make these devices as easy to use as product-term-based devices. In addition, the SRAM-based FLEX 8000 devices require low standby power and are in-circuit reconfigurable, making them ideal for such applications as PC add-on cards, battery-powered instruments, and multi-purpose telecommunication cards.

MAX 7000 Family

The MAX 7000 family is the fastest high-density programmable logic family in the industry. It ranges in density from 600 to 5,000 usable gates, with 32 to 256 macrocells, and 36 to 164 user I/O pins. These devices offer combinatorial propagation delays as low as 7.5 ns and 16-bit counter frequencies of 125 MHz. Moreover, they provide very fast input register setup times, multiple system Clocks, and a programmable speed/power control. The slew rate for I/O pins can be controlled, providing an extra level of switching noise control. The EEPROM-based MAX 7000 devices are non-volatile and electrically erasable.

MAX 5000/EPS464 Family

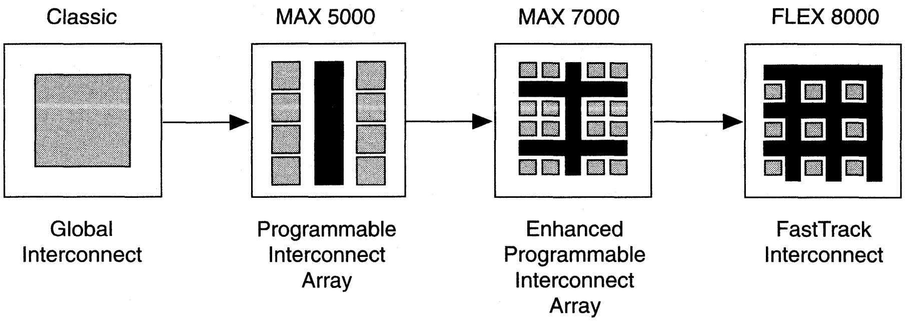
The MAX 5000/EPS464 family provides a comprehensive, cost-effective solution for designs that require a high level of combinatorial logic. These devices provide logic densities ranging from 300 to 3,800 usable gates and pin counts ranging from 20 to 100 pins. Because of the maturity of the devices and Altera's commitment to migrate existing families to newer, more aggressive technologies, MAX 5000/EPS464 devices offer excellent cost-per-macrocell values that compare favorably to ASICs and gate arrays for high-volume production. The EPROM-based MAX 5000/EPS464 devices are non-volatile and erasable.

Classic Family

The Classic family is Altera's original family of devices. It features densities up to 900 usable gates and pin counts up to 68 pins. Composed of single arrays of globally interconnected logic, the industry-standard Classic family offers a low-cost solution for low-density applications. This family offers a unique "zero-power" mode, which allows the devices to draw only microamps of current at standby, making them ideal for low-power applications. The Classic family is based on EPROM technology.

Figure 4 summarizes the architectures of Altera devices and illustrates how the interconnect structure has evolved to maintain high performance, even at the highest densities.

Figure 4. Altera Architecture Evolution



All Altera device families use CMOS process technology, which provides lower power dissipation and greater reliability than bipolar technology. As part of Altera's commitment to continual improvement, Altera transfers products to advanced processes as soon as these technologies become viable and can support reliable manufacturing. This transfer generally reduces manufacturing costs and provides performance enhancements, which translate into faster, cost-effective devices.

Solutions for High-Volume Production

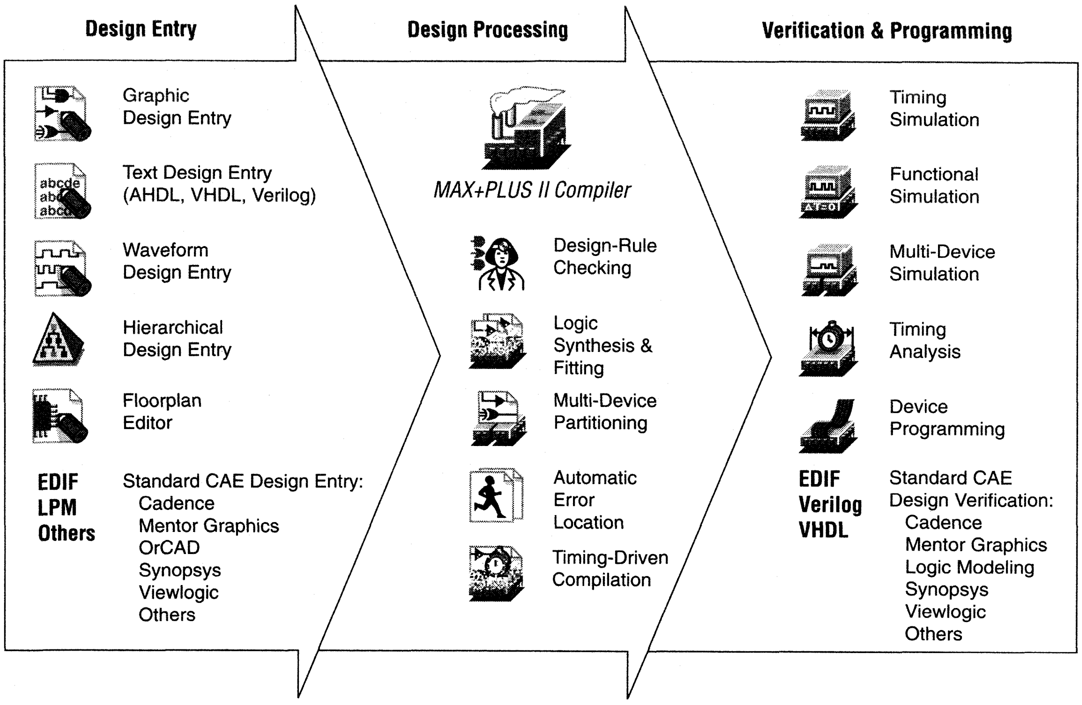
For applications that are targeted for high-volume production, Altera offers Mask-Programmed Logic Devices (MPLDs) as low-cost alternatives to high-density PLDs. MPLDs, which are masked versions of programmable logic devices, offer a unique turn-key approach that eliminates the engineering-intensive tasks required for custom and semi-custom devices. The quick turn-around for MPLD conversion guarantees fast time-to-market.

MAX+PLUS II Development Tools

Altera achieves maximum device performance and density not only with advanced processes and innovative logic architectures, but also through state-of-the-art design tools. The MAX+PLUS II programmable logic development software provides an architecture-independent design environment that supports designs for Altera's general-purpose PLD families, ensuring easy design entry, quick processing, and uncomplicated device programming. See Figure 5.

Using MAX+PLUS II, designers no longer need to master the complexities of device architectures. MAX+PLUS II translates their design—created with familiar design entry tools, such as schematic capture or a high-level behavioral language—into the format required by the target architecture.

Figure 5. MAX+PLUS II Design Environment



Since intimate architectural knowledge is built into Altera development tools, it is not necessary for designers to manually optimize their design, and can therefore complete their designs much more rapidly. With MAX+PLUS II, 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 complete design iterations in a single day.

Design Entry, Processing,Verification & Device Programming

MAX+PLUS II offers a full spectrum of logic design capabilities. Designers are free to combine text, graphic, and waveform design entry methods while creating hierarchical single- or multi-device designs. The MAX+PLUS II Compiler performs minimization and logic synthesis, fits the design into one or more devices, and generates programming data. Design verification with functional and timing simulation and delay prediction for speed-critical paths are available, as well as multi-device simulation across multiple device families. Altera and a number of programming hardware manufacturers offer hardware for programming the devices.

Access to Various Platforms & Other CAE Tools

Altera is committed to supporting the logic development environments that are most familiar to circuit designers. MAX+PLUS II interfaces to a wide variety of other CAE tools—provided by companies such as Cadence, Mentor Graphics, OrCAD, Synopsys, and Viewlogic—via EDIF, LPM, Verilog HDL, and VHDL. The MAX+PLUS II Compiler runs on PC and various workstation platforms, making MAX+PLUS II the industry's only platform-independent, architecture-independent programmable logic design environment. The ACCESS alliance, which Altera has formed with industry-leading CAE tool vendors, ensures smooth interfaces between Altera products and the products of the ACCESS partners, as well as timely support of Altera devices with these tools.

Conclusion

The advanced architectures and processing technologies used in Altera devices provide the greatest performance, highest density, and greatest flexibility available in the PLD market. Regardless of an application's requirements, Altera devices provide an efficient solution with high levels of integration, high I/O capabilities, and the fastest speeds available.

The sophisticated, highly integrated MAX+PLUS II development environment completes Altera's total solution. MAX+PLUS II gives designers the ability to take full advantage of all features offered in Altera devices. MAX+PLUS II can target a project to any device family, thus offering architecture-independent design capabilities to designers, regardless of their preferred design flow. Together, Altera devices and Altera development tools are the logical choice for all designs that require fast development cycles and cost-effective components.

# Technical Support
from Altera Applications

May 1994

Introduction

Altera's technical support team includes over 50 Applications Engineers dedicated to promptly answering customers' technical questions. Applications Engineers are located at Altera's headquarters in San Jose, California, and at locations around the world.

In addition, Altera Applications offers the following services:

- ❑ Training courses
- ❑ Electronic bulletin board service
- ❑ Applications publications
- ❑ Design evaluations
- ❑ Technical support hotline
- ❑ On-site support

Training Courses

(408) 894-7000

Altera provides a variety of training courses that help customers efficiently use Altera products. With these courses, customers can fine-tune their skills with Altera development tools or simply learn more about Altera products. Courses include device architectures, MAX+PLUS II demonstrations, and how-to sessions. All training courses can be tailored to fit customer needs.

Training courses are held at Altera in San Jose; in some cases, they can be held at customer sites. For more information, contact Altera Marketing at (408) 894-7000.

Electronic Bulletin Board Service

(408) 954-0104

Altera maintains a 24-hour electronic bulletin board service (BBS) for instant access to the latest Altera product information. On-line versions of Altera application notes and briefs, recent quarterly newsletters, and software utility programs are available from the BBS. The BBS can also be used to transfer design files to and from the Altera Applications department for technical support.

The telephone number for the BBS is (408) 954-0104. To connect to the BBS via modem, the following equipment and configuration is required:

- ❑ Up to 14,400 baud rate
- ❑ Bell Standard 212A or CCIT standard or compatible modem
- ❑ 8 data bits, 1 stop bit, no parity

The following file transfer protocols are supported:

- ❏ ASCII (Non-Binary)
- ❏ Xmodem (Checksum)
- ❏ Xmodem (CRC)
- ❏ 1K-XModem
- ❏ Ymodem (Batch U/L and D/L)
- ❏ Zmodem (Batch U/L and D/L)
- ❏ Kermit (MS-DOS Columbia Univ)

Applications Publications

Altera Applications produces technical application notes and briefs to help customers select and use programmable logic. All technical literature currently available from Altera is listed in the Applications quarterly customer newsletter, *News & Views*. *News & Views* also includes technical articles written by Altera Applications Engineers, a question and answer section that addresses many commonly asked questions, and the latest information on Altera products. All registered users of Altera products receive *News & Views* each quarter.

Design Evaluations

Altera Applications Engineers will evaluate customer designs and recommend the most efficient design methods and the appropriate Altera device(s) for the customer. Applications Engineers also provide estimates on device performance. For more information, customers can contact their local Altera sales office.

Technical Support Hotline

(800) 800-EPLD

From 7:30 a.m. to 5:00 p.m. Pacific Time, customers can talk directly to an Applications Engineer by calling (800) 800-EPLD. Questions are handled promptly and completely, ensuring that the design process keeps moving forward. Customers outside of the United States can contact their local Altera distributor or sales office, or send a fax to (408) 954-0348.

On-Site Support

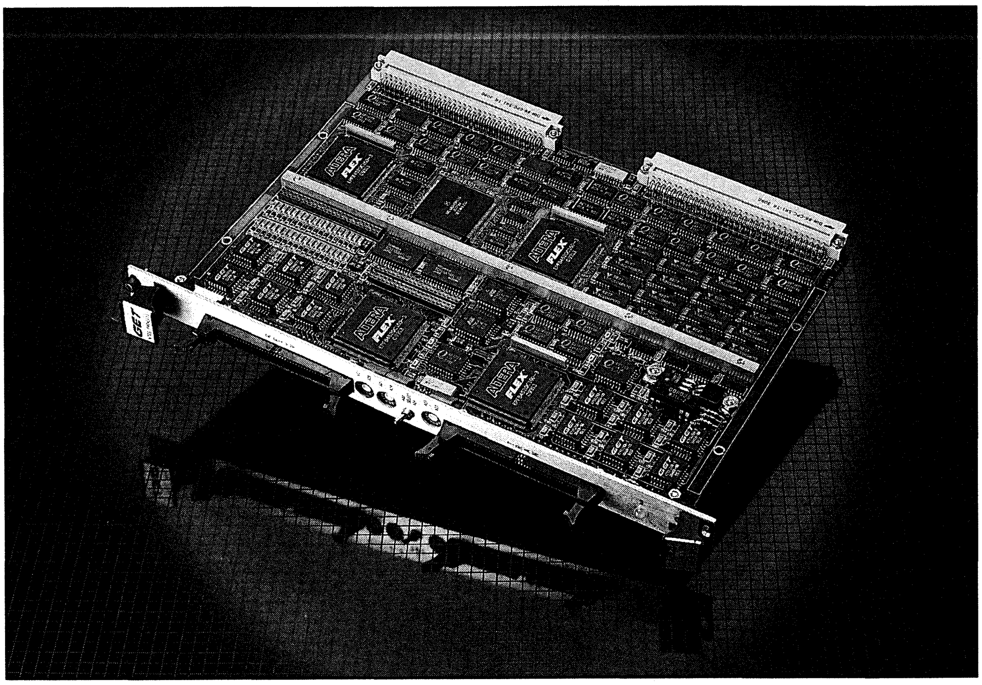
Field Applications Engineers will provide technical support at the customer site. They will evaluate customer designs, demonstrate MAX+PLUS II software, and provide on-site training. Customers can contact their local Field Applications Engineer by calling their local Altera sales office.

ALTERA

Contents

May 1994

Section 2 Technical Specifications



Altera FLEX 8000 devices are featured on GET Engineering Corporation's NTDS parallel interface adapters for VME bus systems. These systems allow users to interface parallel NTDS channels with a wide variety of VME-based computer systems.

FLEX 8000 Architecture

May 1994, ver. 1 — Application Note 40

Introduction

Altera's Flexible Logic Element MatriX (FLEX) programmable logic combines the high register counts of FPGAs with the fast, predictable interconnect of EPLDs, providing up to 16,000 usable gates, 1,500 flipflops, and 304 pins. The SRAM-based FLEX 8000 family features low standby power and in-circuit reconfigurability, making it ideal for applications such as PC add-on cards, battery-powered instruments, and multi-purpose telecommunication systems.

This application note describes the FLEX 8000 architecture in detail. For device specifications, refer to the current *FLEX 8000 Programmable Logic Device Family Data Sheet*. The following topics are covered in this application note:

❑ General description
❑ Functional description
❑ Logic element
❑ Logic Array Block
❑ FastTrack Interconnect
❑ Dedicated inputs
❑ I/O element

General Description

The fine-grained architecture and high register count characteristic of FPGAs are combined with the high speed and predictable interconnect delays of EPLDs to make the FLEX 8000 device family ideal for a wide range of applications. Logic is implemented with compact 4-input look-up tables (LUTs) and programmable registers. High performance is provided by a fast, continuous network of routing resources.

FLEX 8000 devices provide a large number of storage elements for applications such as digital signal processing, wide data-path manipulation, and data transformation. These devices are an excellent choice for bus interfaces, TTL integration, coprocessor functions, and high-speed controllers. The high-pin-count packages can integrate multiple 32-bit buses into a single device. Table 1 shows typical functions and performance for FLEX 8000 devices.

All FLEX 8000 device packages provide four dedicated inputs for synchronous control signals with large fan-outs. Each I/O pin has an associated register on the periphery of the device. As outputs, these registers provide fast Clock-to-output times; as inputs, they offer quick setup times.

Table 1. FLEX 8000 Performance

Application	Logic Elements Used	-2 Speed Grade	-3 Speed Grade
16-bit prescaled counter	24	133 MHz	115 MHz
16-bit loadable counter	16	71 MHz	45 MHz
16-bit up/down counter	16	71 MHz	45 MHz
24-bit accumulator	24	48 MHz	32 MHz
16-line-to-1-line multiplexer	10	14 ns	17 ns

The logic and interconnections in the FLEX 8000 architecture are configured with CMOS SRAM cells. FLEX 8000 devices are configured at system power-up, with data stored in a serial configuration EPROM device or provided by a system controller. Altera offers the EPC1213 and EPC1064 Configuration EPROMs, which configure FLEX 8000 devices via a serial data stream. Configuration data can also be stored in an industry-standard 32K × 8-bit or larger EPROM or downloaded from system RAM. After a FLEX 8000 device has been configured, it can be reconfigured in-circuit by resetting the device and loading new data. Because reconfiguration requires less than 100 ms, real-time changes can be made during system operation.

You can use Altera's MAX+PLUS II development system to create FLEX 8000 logic designs with any combination of graphic, text—including the Altera Hardware Description Language (AHDL), Verilog HDL, and VHDL—and waveform design entry. Full simulation, worst-case timing analysis, and functional testing are available for design verification. MAX+PLUS II also provides an EDIF netlist interface for additional design entry and simulation support with industry-standard CAE tools. In addition, MAX+PLUS II can export Verilog HDL and VHDL netlist files.

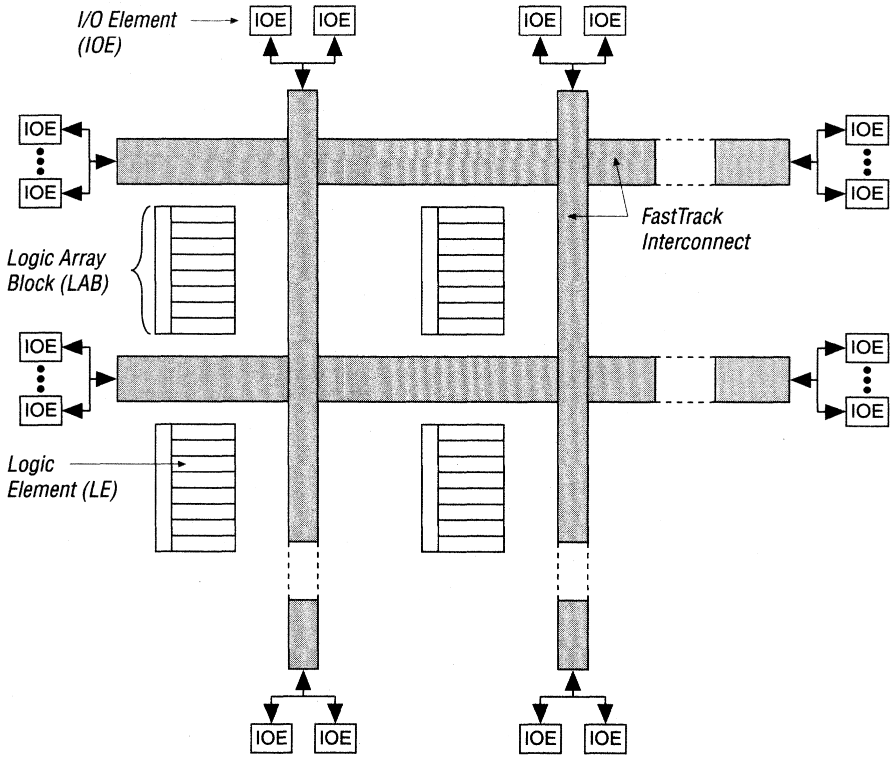
Functional Description

The FLEX 8000 architecture incorporates a large matrix of compact building blocks called logic elements (LEs). Each LE contains a 4-input LUT that provides combinatorial logic capability and a programmable register that offers sequential logic capability. The fine-grained structure of the LE provides highly efficient logic implementation.

LEs are grouped into sets of eight to create Logic Array Blocks (LABs). Each FLEX 8000 LAB is an independent structure with common inputs, interconnections, and control signals. The LAB architecture provides a coarse-grained structure for high device performance and easy routing.

Figure 1 shows a block diagram of the FLEX 8000 architecture. LABs are arranged into rows and columns. The I/O pins are supported by I/O elements (IOEs) located at the ends of rows and columns. Each IOE contains a bidirectional I/O buffer and a flipflop that can be used as either an input or output register.

Figure 1. FLEX 8000 Device Block Diagram

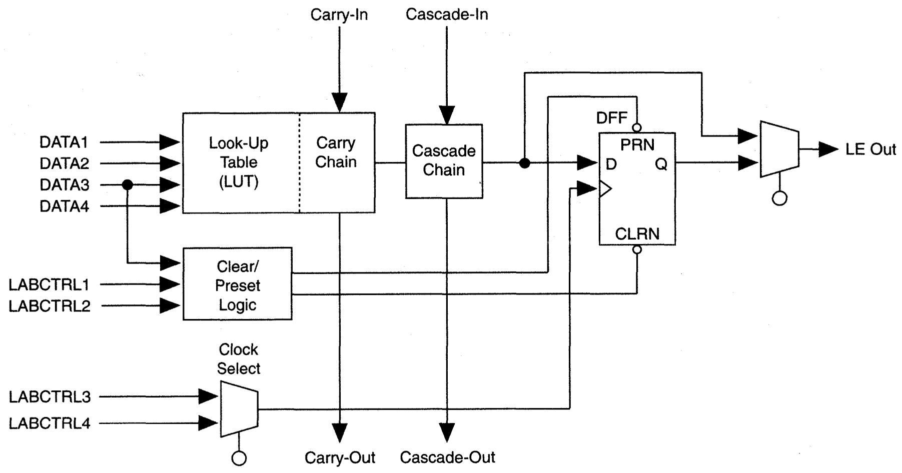


Signal interconnections within FLEX 8000 devices are provided by the FastTrack Interconnect, a series of fast, continuous channels that run the entire length and width of the device.

Logic Element

The logic element (LE) is the smallest unit of logic in the FLEX 8000 architecture, with a compact size that provides efficient logic implementation. Each LE contains a four-input LUT, a programmable flipflop, a carry chain, and a cascade chain. Figure 2 shows a block diagram of the LE.

Figure 2. FLEX 8000 Logic Element (LE)



The LUT is a function generator that can quickly compute any function of four variables. The programmable flipflop in the LE can be configured for D, T, JK, or SR operation. The Clock, Clear, and Preset control signals on the flipflop can be driven by dedicated input pins, general-purpose I/O pins, or any internal logic. For purely combinatorial functions, the flipflop is bypassed and the output of the LUT goes directly to the output of the LE.

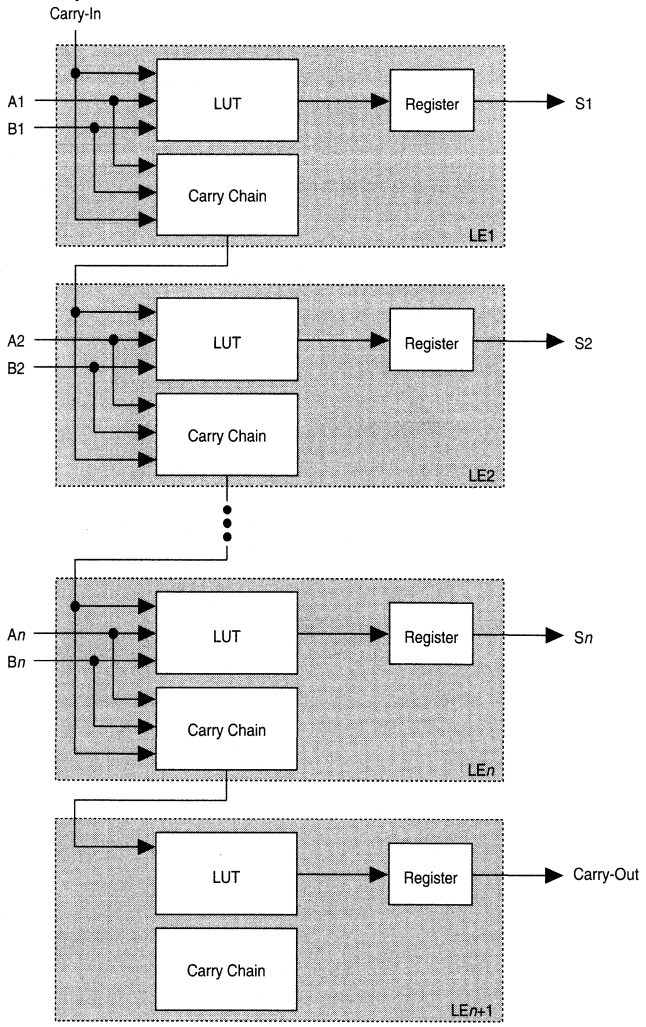
The FLEX 8000 architecture provides two dedicated high-speed paths, the carry and cascade chains, that connect adjacent LEs without using general-purpose interconnect paths. The carry chain supports high-speed counters and adders; the cascade chain implements wide-input functions with minimum delay. Cascade and carry chains connect all LEs in an LAB and all LABs in the same row. Since heavy use of carry and cascade chains can restrict the placement and routing of other logic, they are recommended only for use in speed-critical portions of a design.

Carry Chain

The carry chain provides a very fast (less than 1 ns) carry-forward function between LEs. The carry-in signal from a lower-order bit moves forward into the higher-order bit via the carry chain, and feeds into both the LUT and the next portion of the carry chain. This feature allows the FLEX 8000 architecture to implement high-speed counters and adders of arbitrary width. The MAX+PLUS II Compiler can create carry chains automatically during design processing; designers can also insert carry chain logic manually during design entry.

Figure 3 shows how an *n*-bit full adder can be implemented in *n*+1 LEs by using the carry chain. One portion of the LUT generates the sum of two bits using the input signals and the carry-in signal; the sum is routed to the output of the LE. The register is typically bypassed for simple adders, but can be used for an accumulator function. Another portion of the LUT generates the carry-out signal, which is routed directly to the carry-in signal of the next-higher-order bit. The final carry-out signal is routed to an LE, where it can be used as a general-purpose signal. In addition to mathematical functions, the carry chain logic supports very fast counters and comparators.

Figure 3. Carry Chain Operation

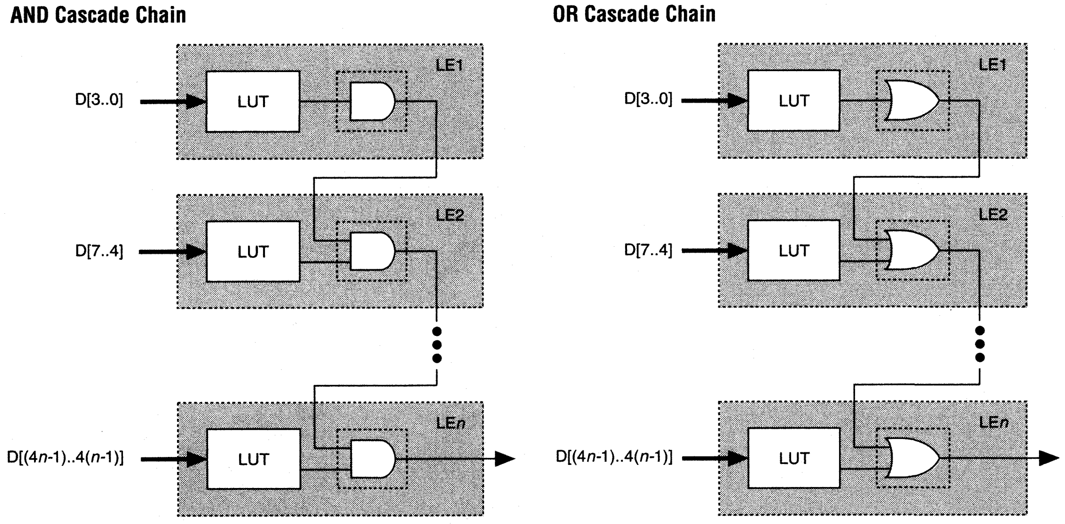


Cascade Chain

With the cascade chain, the FLEX 8000 architecture can implement functions that have a very wide fan-in. Adjacent LUTs can be used to compute portions of the function in parallel; the cascade chain serially connects the intermediate values. The cascade chain can use a logical AND or logical OR (via De Morgan's inversion) to connect the outputs of adjacent LEs. Each additional LE provides four more inputs to the effective width of a function, with a delay of approximately 1 ns per LE. The MAX+PLUS II Compiler can create cascade chains automatically during design processing; designers can also insert cascade chain logic manually during design entry.

Figure 4 shows how the cascade function can connect adjacent LEs to form functions with a wide fan-in. These examples show functions of 4*n* variables implemented with *n* LEs. The LE delay is approximately 6 ns; the cascade chain delay is 1 ns. With the cascade chain, 9 ns is needed to decode a 16-bit address.

Figure 4. Cascade Chain Operation

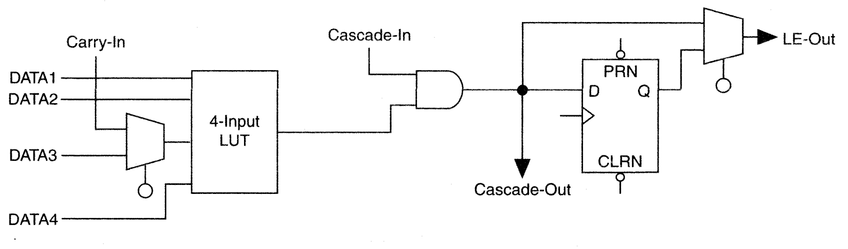


Logic Element Operating Modes

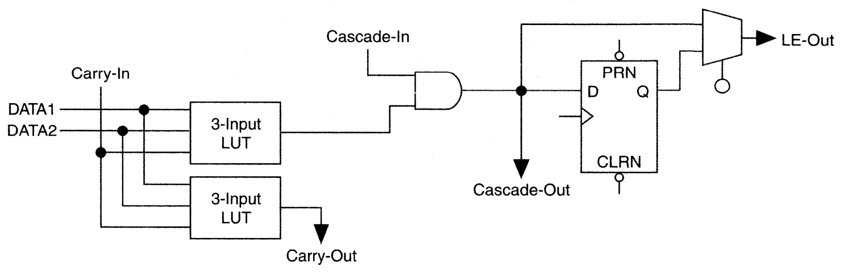
The FLEX 8000 logic element can operate in one of four modes, shown in Figure 5, each of which uses LE resources differently. In each mode, seven of the ten available inputs to the LE—the four data inputs from the LAB local interconnect, the feedback from the programmable register, and the carry-in and cascade-in from the previous LE—are directed to different destinations to implement the desired logic function. The three remaining inputs to the LE provide Clock, Clear, and Preset control for the register. MAX+PLUS II software automatically chooses the appropriate mode for each application. Design performance can be enhanced by designing for the operating mode that best supports the desired application.

Figure 5. FLEX 8000 Logic Element Operating Modes

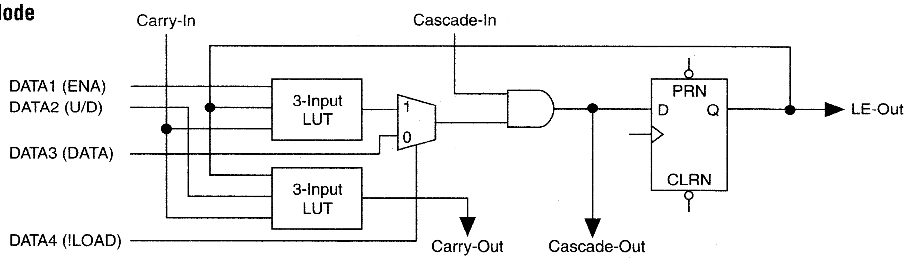
Normal Mode



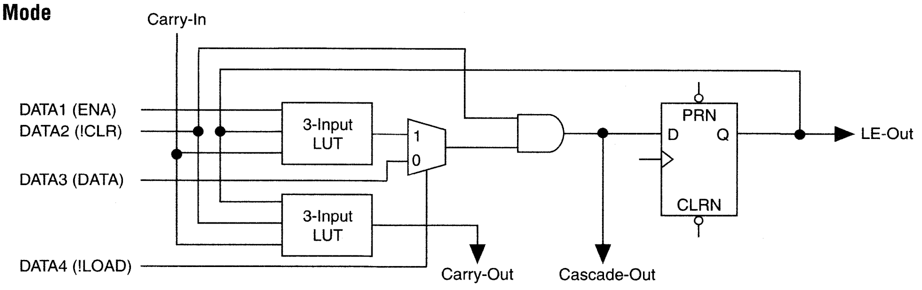
Arithmetic Mode



Up/Down Counter Mode



Clearable Counter Mode



Normal Mode

The Normal mode is suitable for general logic applications and wide decode functions that can take advantage of a cascade chain. In Normal mode, four data inputs from the LAB local interconnect and the carry-in are the inputs to a 4-input LUT. The MAX+PLUS II Compiler automatically selects the carry-in or the `DATA3` signal as an input that is physically controlled by a configurable SRAM bit. The LUT output can be combined with the cascade-in signal to form a cascade chain through the cascade-out signal. The LE Out signal—the data output of the LE—is either the combinatorial output of the LUT and cascade chain, or the `Q` output of the programmable register.

Arithmetic Mode

The Arithmetic mode offers two 3-input LUTs that are ideal for implementing adders, accumulators, and comparators. One LUT provides a 3-bit function; the other generates a carry bit. As shown in Figure 5, the first LUT uses the carry-in signal and two data inputs from the LAB local interconnect to generate a combinatorial or registered output. For example, in an adder, this output would be the sum of three bits: `A`, `B`, and carry-in. The second LUT uses the same three signals to generate a carry-out signal, thereby creating a carry chain. The Arithmetic mode also supports a cascade chain.

Up/Down Counter Mode

The Up/Down Counter mode offers counter enable, synchronous up/down control, and data loading options. These control signals are generated by the data inputs from the LAB local interconnect, the carry-in signal, and output feedback from the programmable register. Two 3-input LUTs are used: one generates the counter data, the other generates the fast carry bit. A 2-to-1 multiplexer provides synchronous loading. Data can also be loaded asynchronously with the Clear and Preset register control signals, without using the LUT resources.

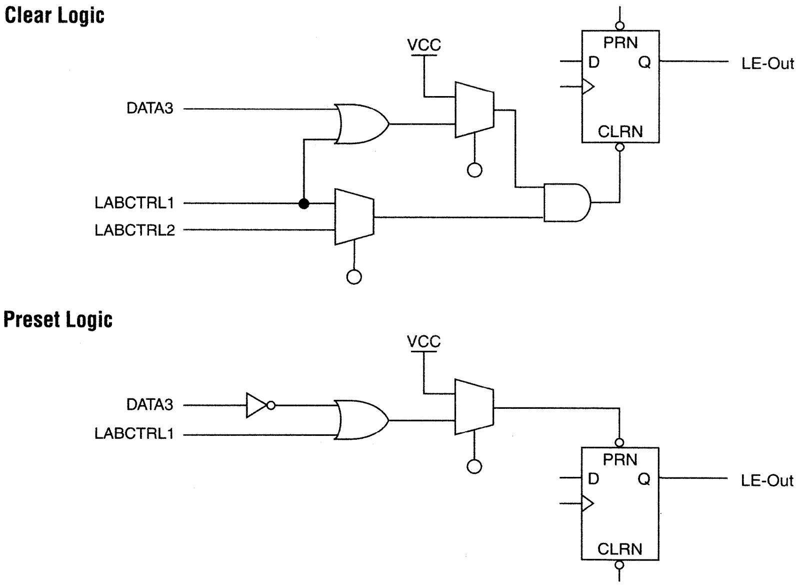
Clearable Counter Mode

The Clearable Counter mode is similar to the Up/Down Counter mode, but supports a synchronous Clear instead of the up/down control. The Clear function is substituted for the cascade-in signal in the Up/Down Counter mode. Two 3-input LUTs are used: one generates the counter data, the other generates the fast carry bit. Synchronous loading is provided by a 2-to-1 multiplexer, the output of which is ANDed with a synchronous Clear.

Clear/Preset Logic Control

Logic for the programmable register's Clear and Preset functions is controlled by the `DATA3`, `LABCTRL1`, and `LABCTRL2` inputs to the LE (see Figure 6). The Clear function is controlled by `DATA3`, `LABCTRL1`, and `LABCTRL2`; the Preset function is controlled by `DATA3` and `LABCTRL1`. The MAX+PLUS II Compiler automatically selects the best control signal implementation during compilation. Preset control can also be provided by using a Clear and inverting the output of the register. Inversion control is available for the inputs to both LEs and IOEs. Therefore, if a register is cleared by only one of the two `LABCTRL` signals, the `DATA3` input is not required and can be used for one of the LE operating modes.

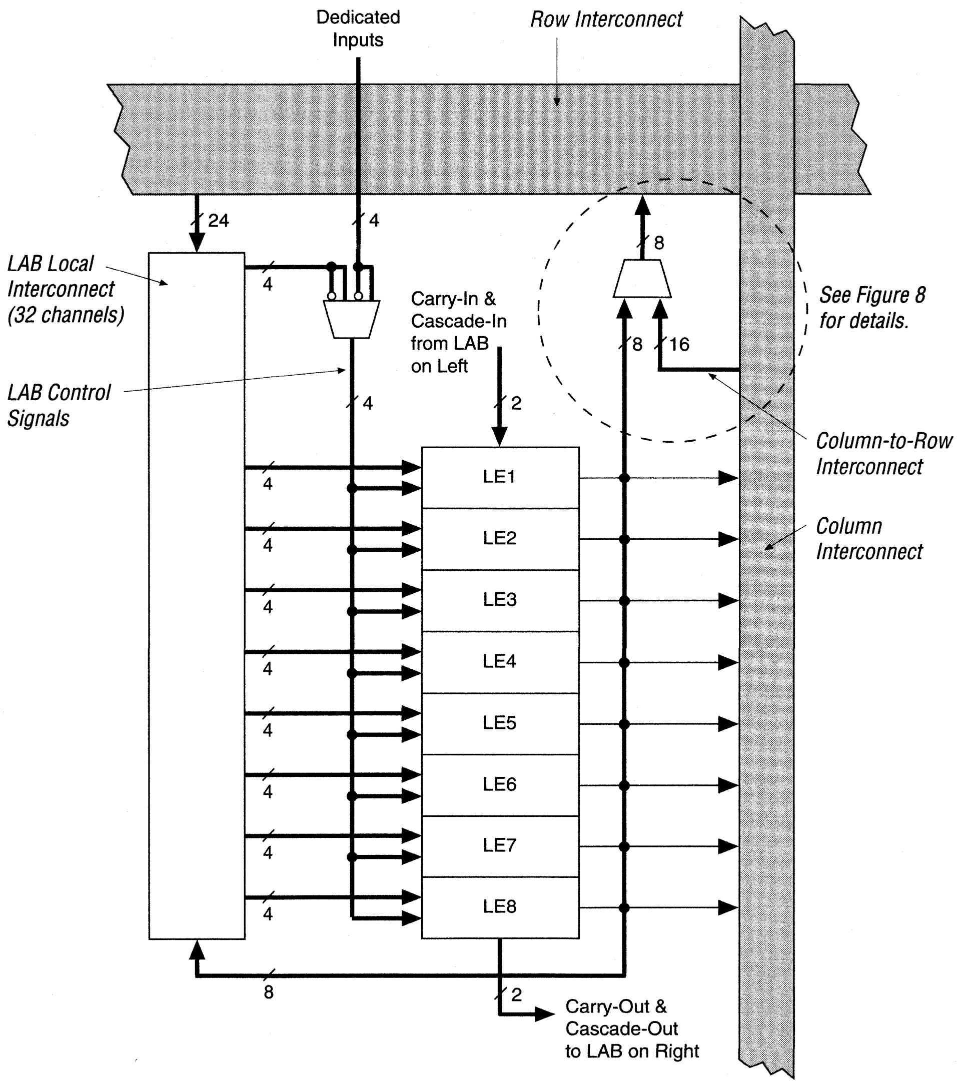
Figure 6. Logic Element Clear & Preset Logic



Logic Array Block

A Logic Array Block (LAB) consists of eight LEs, their associated carry and cascade chains, LAB control signals, and the LAB local interconnect. The LAB provides the coarse-grained structure of the FLEX 8000 architecture for efficient routing with high device utilization and high performance. Figure 7 shows a block diagram of the FLEX 8000 LAB.

Figure 7. Logic Array Block (LAB)


Dedicated Inputs
Row Interconnect
24
4
8
LAB Local Interconnect (32 channels)
4
Carry-In & Cascade-In from LAB on Left
See Figure 8 for details.
8
16
LAB Control Signals
4
2
Column-to-Row Interconnect
LE1
LE2
LE3
LE4
LE5
LE6
LE7
LE8
Column Interconnect
4
8
2
Carry-Out & Cascade-Out to LAB on Right

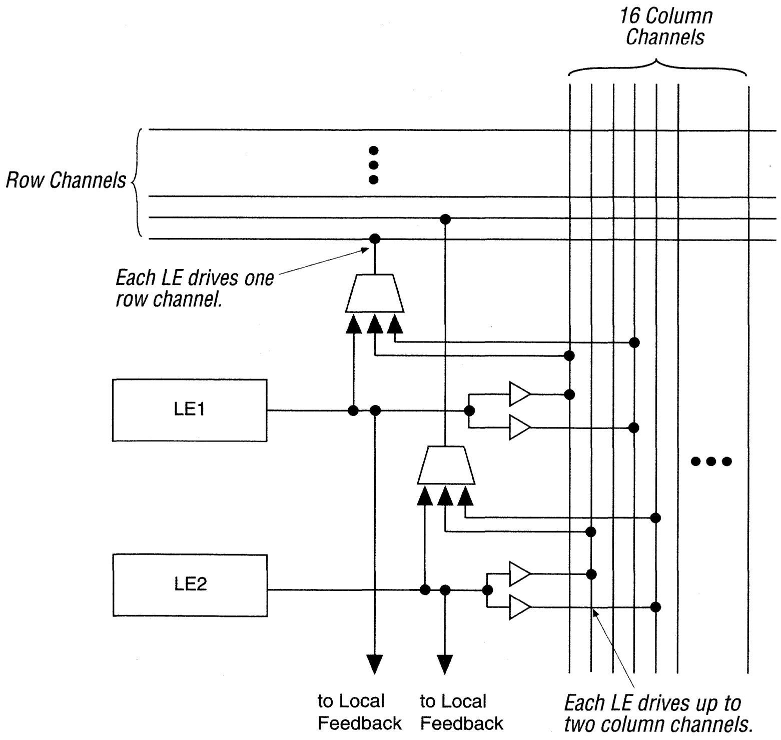
Each LAB provides four control signals that can be used in all eight LEs. Two of these signals can be used as Clocks, the other two for Clear/Preset control. The LAB control signals can be driven directly from a dedicated input pin, an I/O pin, or any internal signal via the LAB local interconnect. The dedicated inputs are typically used for global Clock, Clear, or Preset signals because they provide synchronous control with very low skew across the device. If logic is required on a control signal, it can be generated in one or more LEs in any LAB and driven into the local interconnect of the target LAB. Programmable inversion is available for all four LAB control signals.

FastTrack Interconnect

In the FLEX 8000 architecture, connections between LEs and device I/O pins are provided by the FastTrack Interconnect, a series of continuous horizontal and vertical routing channels that traverse the entire FLEX 8000 device. This device-wide routing structure provides predictable performance even in complex designs. In contrast, the segmented routing in FPGAs requires switch matrices to connect a variable number of routing paths, increasing the delays between logic resources and reducing performance.

The LABs within FLEX 8000 devices are arranged into a matrix of columns and rows. Each row of LABs has a dedicated row interconnect that routes signals both into and out of the LABs in the row. The row interconnect can then drive I/O pins or feed other LABs in the device. Figure 8 shows how an LE drives the row and column interconnect.

Figure 8. LAB Connections to Row and Column Interconnect


16 Column Channels
Row Channels
Each LE drives one row channel.
LE1
LE2
to Local Feedback
to Local Feedback
Each LE drives up to two column channels.

Each LE in an LAB can drive up to two separate column interconnect channels. Therefore, all 16 available column channels can be driven by the LAB. The column channels run vertically across the entire device, and LABs in different rows share access to them via partially populated multiplexers. The MAX+PLUS II Compiler chooses which LEs must be connected to a column channel. A row interconnect channel can be fed by the output of the LE or by two column channels. These three signals feed a multiplexer that connects to a specific row channel. Each LE is connected to one 3-to-1 multiplexer. In an LAB, the multiplexers provide all 16 column channels with access to the row channels.

Each column of LABs has a dedicated column interconnect that routes signals out of the LABs in the column. The column interconnect can then drive I/O pins or feed into the row interconnect to route the signals to other LABs in the device. A signal from the column interconnect, which can be either the output of an LE or an input from an I/O pin, must transfer to the row interconnect before it can enter an LAB. Table 3 summarizes the FastTrack Interconnect resources available in each FLEX 8000 device.

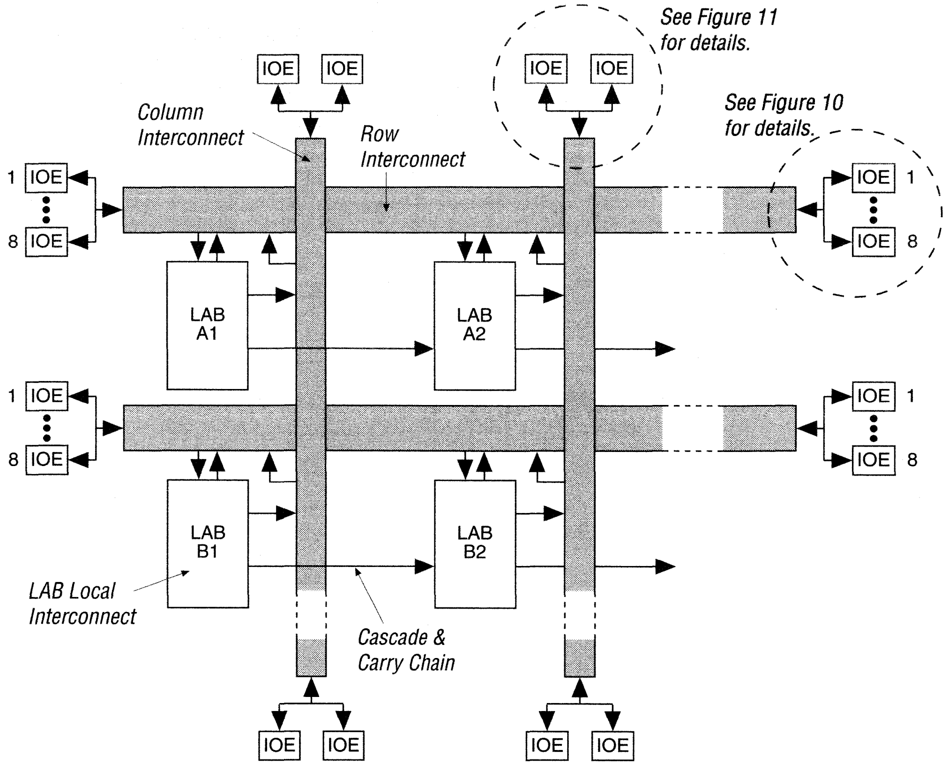
Table 2. FLEX 8000 FastTrack Interconnect Resources

Device	Rows	Channels per Row	Columns	Channels per Column
EPF8282, EPF8282V	2	168	13	16
EPF8452	2	168	21	16
EPF8636	3	168	21	16
EPF8820	4	168	21	16
EPF81188	6	168	21	16
EPF81500	6	216	27	16

Figure 9 shows the interconnection of four adjacent LABs, with row, column, and local interconnects, as well as the associated cascade and carry chains.

Figure 9. FLEX 8000 Device Interconnect Resources

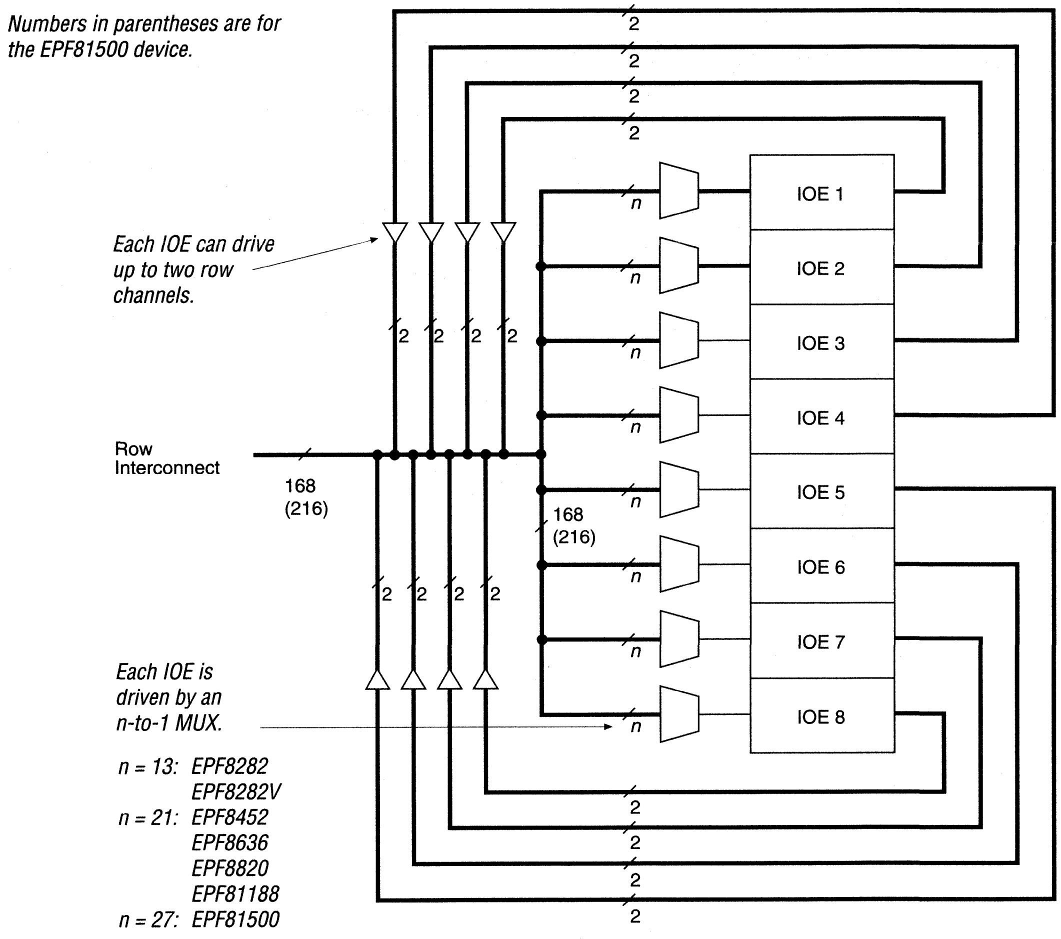
Each LAB is named on the basis of its physical row (A, B, C, etc.) and column (1, 2, 3, etc.) position within the device.



Row-to-IOE Connections

Figure 10 illustrates the connection between row interconnect channels and IOEs. An input signal from an IOE can drive two separate row channels. When an IOE is used as an output, the signal is driven by an *n*-to-1 multiplexer that selects the row channels. The size of the multiplexer varies with the number of columns in a device. The EPF81500 uses a 27-to-1 multiplexer; the EPF81188, EPF8820, EPF8636, and EPF8452 use a 21-to-1 multiplexer; and the EPF8282 and EPF8282V use a 13-to-1 multiplexer. Eight IOEs are connected to each side of the row channels.

Figure 10. FLEX 8000 Row-to-IOE Connection


Numbers in parentheses are for the EPF81500 device.
Each IOE can drive up to two row channels.
Row Interconnect
168 (216)
168 (216)
2
n
IOE 1
IOE 2
IOE 3
IOE 4
IOE 5
IOE 6
IOE 7
IOE 8
Each IOE is driven by an n-to-1 MUX.
n = 13: EPF8282 EPF8282V
n = 21: EPF8452 EPF8636 EPF8820 EPF81188
n = 27: EPF81500

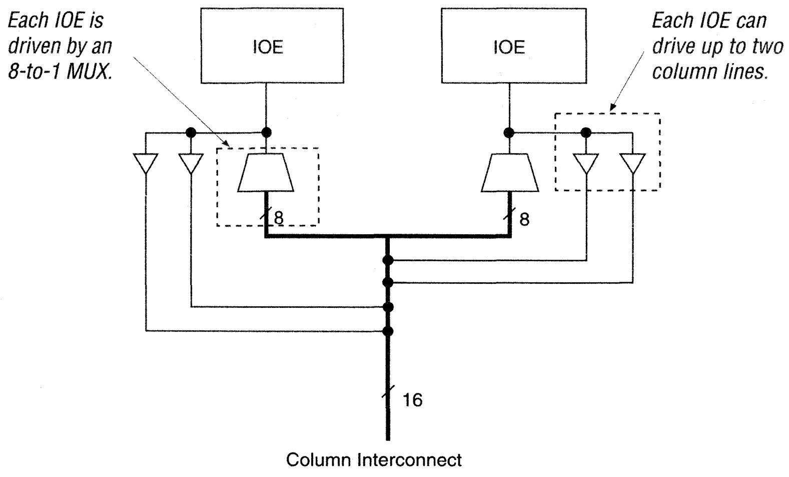
Column-to-IOE Connections

On the top and bottom of the column channels are two IOEs (see Figure 11). When an IOE is used as an input, it can drive up to 2 separate column channels. The output signal to an IOE can choose from 8 of the 16 column channels through an 8-to-1 multiplexer.

Dedicated Inputs

In addition to the general-purpose I/O pins, FLEX 8000 devices have four dedicated input pins. These dedicated inputs provide low-skew, device-wide signal distribution, and are typically used for global Clock, Clear, and Preset control signals. The signals from the dedicated inputs are available as control signals for all LABs and I/O elements in the device.

Figure 11. FLEX 8000 Column-to-IOE Connection


Each IOE is driven by an 8-to-1 MUX.
IOE
IOE
Each IOE can drive up to two column lines.
8
8
16
Column Interconnect

The dedicated inputs can also be used as general-purpose data inputs for nets with large fan-outs because they can feed the local interconnect of each LAB in the device.

I/O Element

Figure 12 shows the I/O element (IOE) block diagram. Signals enter the FLEX 8000 device from either the I/O pins that provide general-purpose input capability or the four dedicated inputs that are typically used for fast, global control signals. The IOEs are located at the ends of the row and column interconnect channels.

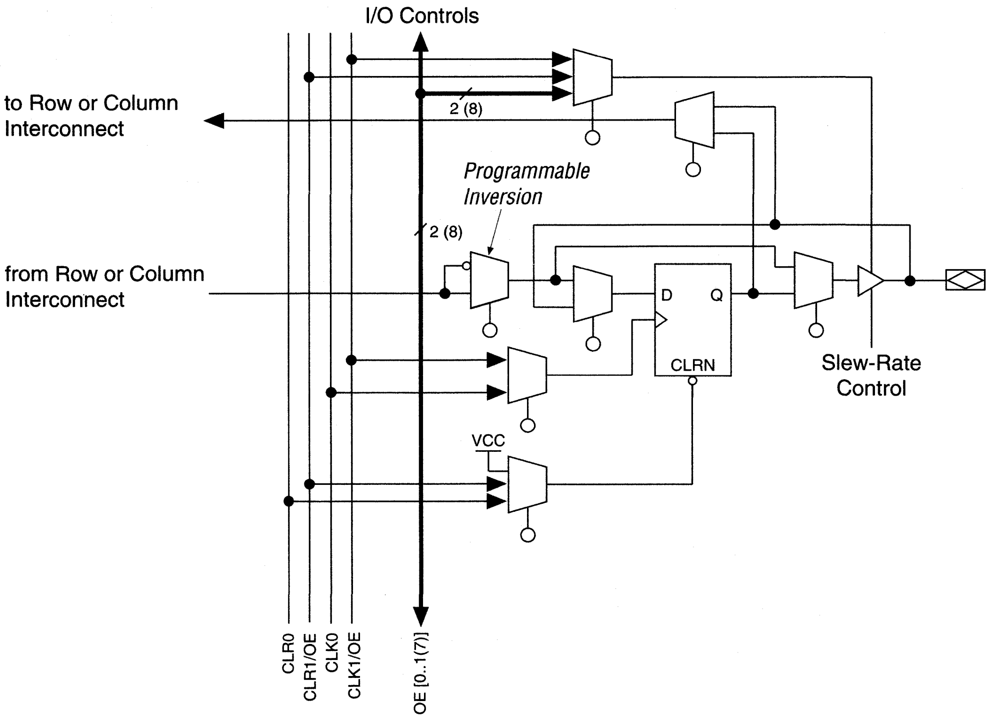
I/O pins can be used as input, output, or bidirectional pins. Each I/O pin has a register that can be used either as an input register for external data that requires fast setup times, or as an output register for data that requires fast Clock-to-output performance. The MAX+PLUS II Compiler uses the programmable inversion option to automatically invert signals from the row and column interconnect when appropriate.

The output buffer in each IOE has an adjustable output slew rate that can be configured for low-noise or high-speed performance. A faster slew rate provides a speed increase of up to 4 ns, but may introduce more noise into a system than a slow slew rate. The fast slew rate should be used for speed-critical outputs in systems that are adequately protected against noise. Designers can specify the slew rate on a pin-by-pin basis during design entry or assign a default slew rate to all pins on a global basis.

The Clock, Clear, and Output Enable controls for the IOEs are provided by a network of I/O control signals. These signals can be supplied by either

Figure 12. I/O Element (IOE)

Numbers in parentheses are for the EPF81500 device.



the dedicated input pins or internal logic. The IOE control-signal paths are designed to minimize the skew across the device. All control-signal sources are buffered onto high-speed drivers that drive the signals around the periphery of the device. This "peripheral bus" can be configured to provide up to four Output Enable signals (ten in the EPF81500), and up to two Clock or Clear signals. Figure 12 illustrates how two Output Enable signals are shared with one Clock (`CLK1`) and one Clear (`CLR1`) signal.

The signals for the peripheral bus can be generated by any of the 4 dedicated inputs or signals on the row interconnect channels, as shown in Figure 13. The number of row channels used correlates to the number of columns in the FLEX 8000 device. The EPF8282 and EPF8282V, for example, use 13 channels; the EPF8452, EPF8636, EPF8820, and EPF81188 use 21 channels; and the EPF81500 uses 27 channels. The first LE in each LAB is the source of the row channel signal. The 6 peripheral control signals (12 in the EPF81500) can be accessed by every I/O element. The MAX+PLUS II Compiler uses the programmable inversion option to automatically invert signals from the dedicatd inputs or row channels.

Figure 13. FLEX 8000 Peripheral Bus

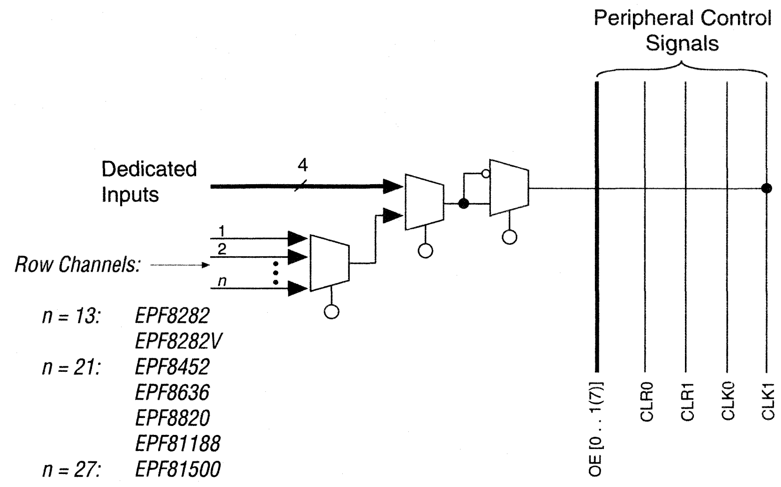


Table 4 lists the row source of the peripheral control signal for each FLEX 8000 device.

Table 3. Row Sources of Peripheral Control Signals

Peripheral Control Signal	EPF8282 EPF8282V	EPF8452	EPF8636	EPF8820	EPF81188	EPF81500
CLK0	Row A	Row A	Row A	Row A	Row E	Row E
CLK1	Row B	Row B	Row C	Row C	Row B	Row B
CLR0	Row A	Row A	Row B	Row B	Row F	Row F
CLR1	Row B	Row B	Row C	Row D	Row C	Row C
OE0	Row A	Row A	Row A	Row A	Row D	Row A
OE1	Row B	Row B	Row B	Row B	Row A	Row A
OE2	–	–	–	–	–	Row B
OE3	–	–	–	–	–	Row C
OE4	–	–	–	–	–	Row D
OE5	–	–	–	–	–	Row D
OE6	–	–	–	–	–	Row E
OE7	–	–	–	–	–	Row F

JTAG

The EPF8282, EPF8282V, EPF8636, EPF8820, and EPF81500 devices support the Joint Test Action Group (JTAG) boundary-scan testing. For detailed information on JTAG operation in these FLEX 8000 devices, refer to *Application Note 39 (JTAG Boundary-Scan Testing in FLEX 8000 Devices)* in this handbook.

Device Configuration

FLEX 8000 devices support a variety of configuration schemes. Refer to *Application Note 33 (Configuring FLEX 8000 Devices)* and *Application Note 38 (Configuring Multiple FLEX 8000 Devices)* in this handbook.

Conclusion

The architecture of FLEX 8000 devices allows you to achieve maximum optimization and performance. The different LE operating modes facilitate efficient use of LE resources. Carry and cascade chains provide high-speed data paths in performance-critical portions of the design. Input and output signals can be registered in the IOEs without wasting internal LE resources. Together, the features of the FLEX 8000 architecture provide a high-density logic solution for a wide variety of logic applications.

ALTERA

Configuring FLEX 8000 Devices

May 1994, ver. 3

Application Note 33

Introduction

The architecture of Altera's Flexible Logic Element MatriX (FLEX) devices supports several different configuration schemes for loading a design into a single FLEX 8000 device on the circuit board. This application note provides complete details on all aspects of configuring individual FLEX 8000 devices, including sample schematics and timing information.

This application note should be used together with the current *FLEX 8000 Programmable Logic Device Family* and *Configuration EPROMs for FLEX 8000 Devices* data sheets. For information on configuring multiple FLEX 8000 devices in a system, refer to *Application Note 38 (Configuring Multiple FLEX 8000 Devices)* in this handbook. If appropriate, illustrations in this application note show devices with generic "FLEX 8000" and "Configuration EPROM" labels to indicate that they are valid for all FLEX 8000 devices and Altera Configuration EPROMs. All timing parameters shown in figures and tables apply to all FLEX 8000 device speed grades.

The following topics are discussed:

FLEX 8000 Device Operating Modes

The FLEX 8000 architecture uses SRAM cells to store the configuration data for the device. These SRAM cells must be loaded each time the circuit powers up and begins operation. The process of physically loading the SRAM programming data into the FLEX 8000 device is called *configuration*. After configuration, the FLEX 8000 device resets its registers, enables its I/O pins, and begins operation as a logic device. This reset operation is called *initialization*. Together, the configuration and initialization processes are called *command mode*; normal in-circuit device operation is called *user mode*.

SRAM technology allows FLEX 8000 devices to be reconfigured in-circuit by loading new configuration data. Real-time reconfiguration can be performed by forcing the device into command mode with a dedicated device pin, loading different configuration data, reinitializing the device, and resuming user-mode operation. The entire process requires less than 100 ms, and can be used to dynamically reconfigure FLEX 8000 devices during system operation.

You can update existing systems that incorporate FLEX 8000 devices by installing new data in the system. Such in-field upgrades can be as simple as copying a new configuration file to a hard disk or inserting an EPROM programmed with new configuration data into the circuit.

Device configuration can occur either automatically at system power-up or under the control of external logic. Initialization can be controlled by the internal oscillator in the FLEX 8000 device or by an external Clock signal. Dedicated device configuration pins can be used to control when configuration and initialization begin. This range of command-mode control features provides excellent flexibility for designs implemented in FLEX 8000 devices.

Overview of Configuration Schemes

The configuration data for a FLEX 8000 device can be loaded with one of six configuration schemes, which you choose on the basis of the target application. Both active and passive schemes are available. In an active configuration scheme, the FLEX 8000 device guides the configuration operation, controlling external memory devices and the initialization process. The Clock source for all active configuration schemes is an internal oscillator in the FLEX 8000 device that typically operates in the range of 2 to 6 MHz. In a passive configuration scheme, an external controller guides the configuration of the FLEX 8000 device, which operates as a slave. Table 1 shows the source of data for each of the six configuration schemes.

Table 1. Configuration Schemes

Configuration Scheme	Acronym	Data Source
Active serial	AS	Altera Configuration EPROM
Active parallel up	APU	Parallel EPROM
Active parallel down	APD	Parallel EPROM
Passive serial	PS	Serial data path
Passive parallel synchronous	PPS	Intelligent host
Passive parallel asynchronous	PPA	Intelligent host

Each FLEX 8000 device has a different size requirement for its configuration data, based on the number of SRAM cells in the device. Table 2 shows the approximate size of data, expressed in both bits and Kbytes, necessary to configure each FLEX 8000 device. You can use this table to calculate the data space (i.e., data storage resources) required in a parallel or serial data source for a system that incorporates FLEX 8000 devices.

Table 2. FLEX 8000 Device Data-Size Spaces

Device	Data Size (bits)	Data Size (Kbytes)
EPF8282, EPF8282V	40,000	5
EPF8452	64,000	8
EPF8636	96,000	12
EPF8820	128,000	16
EPF81188	192,000	24
EPF81500	250,000	31

Active Configuration

In an active configuration scheme, the FLEX 8000 device controls the entire configuration process and generates the synchronization and control signals necessary to configure and initialize itself from an external memory. The active serial (AS) configuration scheme uses an Altera Configuration EPROM to store the configuration data. The active parallel up (APU) and active parallel down (APD) configuration schemes use a parallel-format memory such as a 32K × 8-bit EPROM as the data source.

Passive Configuration

In a passive configuration scheme, the FLEX 8000 device is incorporated into a system with an intelligent host that controls the configuration process. The intelligent host transparently selects a serial or parallel data source, and the data is presented to the FLEX 8000 device on a common data bus. In this type of system, the configuration data can be stored in a mass-storage medium, such as a hard disk. With passive configuration schemes, new configuration data is easily installed by supplying a new configuration file on a diskette or tape.

Choosing a Configuration Scheme

The best configuration scheme for a particular application depends on many factors, such as the presence of an intelligent host in the system, the need to reconfigure in real-time, and the need to periodically install new configuration data. Available board space is also a consideration for configuration schemes that use parallel or serial EPROMs to store configuration data.

The following guidelines can help you decide which configuration scheme is most appropriate for your application:

❑ For fast time-to-market, the easiest and quickest configuration schemes to implement are the three active configuration schemes: active serial (AS), active parallel up (APU), and active parallel down (APD). These configuration schemes require no external intelligence. The FLEX 8000 device is typically configured automatically at system power-up. If the FLEX 8000 device senses a power failure, it automatically triggers a reconfiguration cycle.

❑ For fast prototyping and development work, the passive serial (PS) configuration scheme, together with the FLEX Download Cable, provides the quickest means of iterative design analysis. The MAX+PLUS II Programmer can directly download configuration data to a FLEX 8000 device on the prototype circuit board.

❑ If a FLEX 8000 device is incorporated into a system with an intelligent host, you can use this host to control the configuration process in one of the passive configuration schemes: passive parallel asynchronous (PPA), passive parallel synchronous (PPS), or passive serial (PS). The configuration data can be stored in a mass-storage medium, such as a hard disk, thereby reducing the number of ICs required for the system. The FLEX 8000 device configuration can also be synchronized with any other system resources that must be initialized.

❑ In applications that require real-time device reconfiguration—such as data transformation filters, video formatters, and encryption/decryption circuits—the best choice is one of the passive configuration schemes. Reconfigurability allows you to reuse the logic resources within the FLEX 8000 device, instead of designing redundant or duplicate circuitry into your systems. Passive configuration schemes easily support the multiple sources of configuration data that may be required for real-time configuration. However, these schemes require more external circuitry. The FLEX 8000 device must rely on an intelligent host to retrieve and load new configuration data, and cannot perform any of the tasks required for reconfiguration.

❑ If field upgrades are anticipated, passive configuration schemes offer the ability to easily install new configuration data. New configuration files can be supplied to end users on diskette or tape. (In active schemes, a new EPROM must be inserted into the system.)

You can also use multiple configuration schemes during system operation. If you choose a single configuration scheme, you can simply hard-wire the three configuration scheme selection pins (`nSP`, `MSEL1`, and `MSEL0`) to their necessary levels (V_{CC} or GND). If you use multiple configuration schemes, you can drive these selection pins with some controlling logic or connect them to a port on an intelligent host. For example, you can configure a FLEX 8000 device with an AS configuration scheme to load its "start-up" configuration data, then dynamically change the configuration scheme selection bits to select a different configuration scheme, and provide a different configuration data source.

FLEX 8000 Device Configuration Schemes

The following sections describe each configuration scheme in detail:

- Active serial (AS) configuration
- Active parallel up (APU) and active parallel down (APD) configuration
- Passive parallel synchronous (PPS) configuration
- Passive parallel asynchronous (PPA) configuration
- Passive serial (PS) configuration

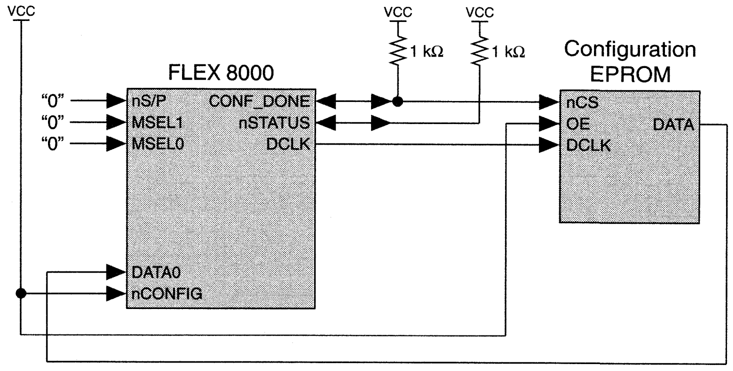
In-circuit reconfiguration, device configuration option bits, device configuration pins, and the source of data for each configuration scheme are described later in this application note.

Active Serial Configuration

The active serial (AS) configuration scheme uses an Altera-supplied serial Configuration EPROM (e.g., EPC1213) as a data source for FLEX 8000 devices. The Configuration EPROM presents its data to the FLEX 8000 device in a serial bit-stream. Figure 1 shows a typical circuit in which the FLEX 8000 device controls the configuration process and uses a Configuration EPROM as the data source.

The `nCONFIG` pin on the FLEX 8000 device in Figure 1 is connected to V_{CC}, so the device automatically configures itself at system power-up. The system can monitor the `nSTATUS` pin to ensure that configuration occurs

Figure 1. Active Serial Device Configuration



correctly. Immediately after power-up, the FLEX 8000 device pulls the nSTATUS pin low and releases it within 100 ms. Once released, the open-drain nSTATUS pin is pulled up to V_{CC} by an external 1.0-kΩ pull-up resistor. If an error occurs during configuration, the FLEX 8000 device pulls the nSTATUS pin low, indicating that configuration was unsuccessful.

The DCLK signal, which is driven by the FLEX 8000 device, clocks sequential data bits from the Configuration EPROM. While the SRAM data is being loaded, the FLEX 8000 device holds the open-drain CONF_DONE pin at GND, indicating that data is loading. A 24-bit program-length counter within the FLEX 8000 device stores the program length, i.e., the total number of configuration bits. Once the terminal count value for the configuration data (i.e., the last configuration data bit) has been reached, the FLEX 8000 device releases the CONF_DONE pin, which is subsequently pulled up to V_{CC} by an external 1.0-kΩ pull-up resistor. The resulting high input on the nCS pin causes the Configuration EPROM to tri-state its DATA output, electrically removing the Configuration EPROM from the circuit.

After it releases the CONF_DONE pin, the FLEX 8000 device uses it as an input for monitoring the configuration process. When the FLEX 8000 device senses a high logic level on CONF_DONE, it completes the initialization process and enters user mode. Figure 2 shows the timing associated with the AS configuration process and the order of transitions on the control signals.

Figure 2. Active Serial Configuration Timing Waveforms

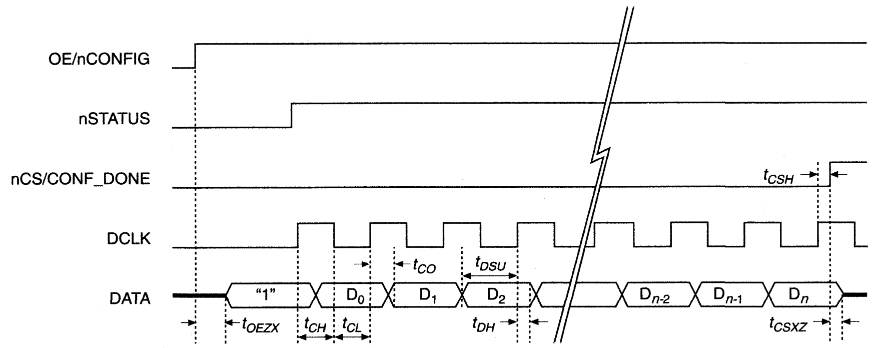


Table 3 provides values for the AS timing parameters.

Table 3. Active Serial Configuration Timing Parameters

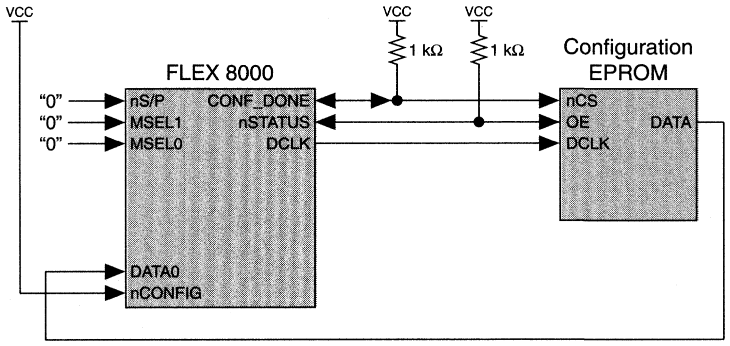
Symbol	Parameter	Min	Max	Unit
t_{OEZX}	OE high to DATA output enabled		50	ns
t_{CSZX}	nCS low to DATA output enabled		50	ns
t_{CSXZ}	nCS high to DATA output disabled		50	ns
t_{CH}	DCLK high time	80	250	ns
t_{CL}	DCLK low time	80	250	ns
t_{DSU}	Data setup time before rising edge on DCLK	50		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CO}	DCLK to DATA out		75	ns
t_{OEW}	OE low pulse width to guarantee counter reset	100		ns
t_{CSH}	nCS low hold time after DCLK rising edge	0		ns
f_{MAX}	DCLK frequency	2	6	MHz

In the circuit shown in Figure 1, the nCONFIG pin on the FLEX 8000 device is tied to the Output Enable (OE) input of the Configuration EPROM; both are tied to V_{CC}. A high logic level on the nCONFIG input automatically starts the configuration. The output of the Configuration EPROM is enabled by a high input on its OE pin. If an error occurs during circuit configuration, the FLEX 8000 device pulls and holds the nSTATUS pin low, indicating a configuration error. External circuitry is used to monitor the nSTATUS pin and take appropriate action if configuration fails. This circuitry must assert a high-low-high pulse on the nCONFIG pin to reconfigure the device after the error. The same circuitry can also be used to begin reconfiguring the FLEX 8000 device at any time after system power-up.

The FLEX 8000 device's built-in *Auto-Restart Configuration on Frame Error* option bit, which can be set with MAX+PLUS II software, allows the device to automatically reconfigure itself if it encounters an error during configuration. (For descriptions of all FLEX 8000 device option bits, refer to "Device Configuration Option Bits" later in this application note.) If this option bit is turned on, a configuration error causes the FLEX 8000 device to pull the nSTATUS pin low for 10 internal Clock cycles and then release it. This 1- to 3-µs pulse on the nSTATUS pin provides an external indication that reconfiguration is about to begin. It also can be used to reset the Configuration EPROM.

Figure 3 shows a circuit that uses the *Auto-Restart Configuration on Frame Error* option. The nSTATUS pin is connected to the OE input on the Configuration EPROM so that the error-reset pulse on nSTATUS resets the internal address counter on the Configuration EPROM and prepares it to reconfigure the FLEX 8000 device. The nCONFIG input is also available to initiate a reconfiguration cycle externally. Since the nSTATUS pin is pulled low and then released whenever configuration begins, it resets the Configuration EPROM before reconfiguration. If V_{CC} drops below the power-on reset (POR) threshold for the FLEX 8000 device during device operation, nSTATUS is pulsed and the Configuration EPROM is reset in the same way to provide automatic reconfiguration. Timing for the circuit in Figure 3 is identical to the timing shown in Figure 2 for the AS configuration scheme (the error-reset pulse on nSTATUS is not shown).

Figure 3. Active Serial Device Configuration with Automatic Reconfiguration on Error



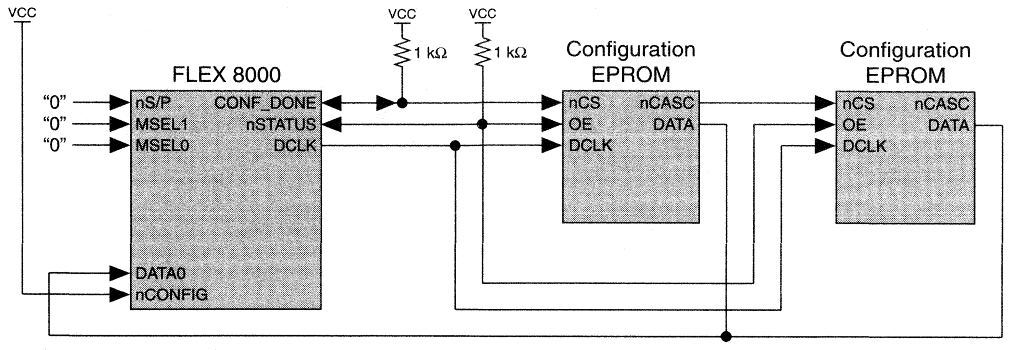
Altera Configuration EPROMs are designed for performance that is compatible with the setup and hold time requirements of FLEX 8000 devices. Refer to the current *Configuration EPROMs for FLEX 8000 Devices Data Sheet* for complete details on timing and circuitry. Details on device programming are given in "Programming a Configuration EPROM" later in this application note.

Active Serial Configuration for Multiple Configuration EPROMs

Multiple Configuration EPROMs can be serially connected to configure a FLEX 8000 device that requires more configuration data than a single Configuration EPROM can store. For example, the EPF81500 requires approximately 250 Kbits of configuration data, but an EPC1213 Configuration EPROM stores a maximum of 213 Kbits. Therefore, two Configuration EPROMs are needed to configure an EPF81500 device.

Figure 4 shows a typical circuit in which a FLEX 8000 device is configured by two Altera Configuration EPROMs. The FLEX 8000 device drives the DCLK signal out to both Configuration EPROMs during configuration, and receives configuration data on its DATA0 input. The first Configuration EPROM drives its nCASC output low and tri-states its DATA pin after clocking out all of its configuration data. The high-to-low transition on nCASC enables the nCS input on the second Configuration EPROM and

Figure 4. Active Serial Configuration of an EPF81500 Device with Automatic Reconfiguration on Error



activates the EPROM within one DCLK cycle. This handshaking is transparent to the FLEX 8000 device.

Once all configuration data has been clocked into the FLEX 8000 device, the device releases the CONF_DONE pin, which is subsequently pulled up to V_{CC} by an external 1.0-kΩ pull-up resistor. The resulting high input on the nCS input to the first Configuration EPROM drives its nCASC output high, which in turn drives the nCS input to the second Configuration EPROM high, electrically removing both Configuration EPROMs from the circuit.

In the circuit shown in Figure 4, the FLEX 8000 device's built-in *Auto-Restart Configuration on Frame Error* option bit allows the device to automatically reconfigure itself if it encounters an error during configuration. The nSTATUS pin is connected to the OE pins on the Configuration EPROMs. If the FLEX 8000 device detects a configuration error, it pulls the nSTATUS pin low for 10 internal Clock cycles and then releases it. This 1- to 3-µs pulse on the nSTATUS pin resets the Configuration EPROMs with a low pulse on the OE pins.

Active Parallel Up & Active Parallel Down Configuration

In the active parallel up (APU) and active parallel down (APD) configuration schemes, the FLEX 8000 device generates sequential addresses that drive the address inputs to an external PROM. The PROM then returns the appropriate byte of data on the data pins DATA[7..0]. Sequential addresses are generated until the FLEX 8000 device has been completely loaded. The CONF_DONE pin is then released and pulled high externally, indicating that configuration has been completed. The counting sequence can be ascending (00000H to 3FFFFH) for APU configuration or descending (3FFFFH to 00000H) for APD configuration.

Figure 5 shows a typical circuit with a FLEX 8000 device and a parallel EPROM for APU or APD configuration. In this circuit, the `nCONFIG` input to the FLEX 8000 device is connected to a system-wide, active-low Reset signal. The `nCONFIG` pin can be tied to V_{CC} (as shown in Figure 1) to start configuration automatically at system power-up; however, the system-wide Reset allows you to explicitly control the time at which configuration begins. The `nCONFIG` pin must be held low to meet the minimum low pulse width requirement for t_{CFG} (see Table 4 later in this application note).

Figure 5. Active Parallel Device Configuration with a 256-Kbyte EPROM

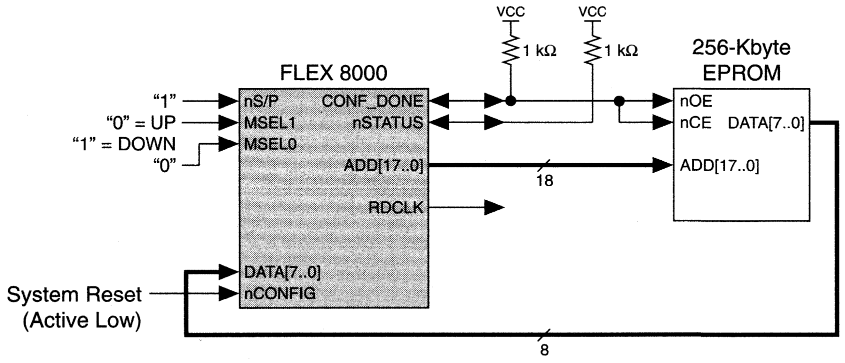
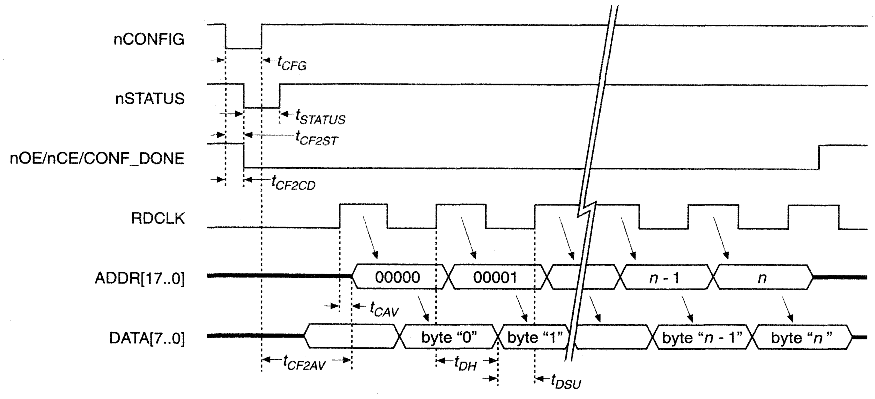


Figure 6 shows the timing associated with the circuit in Figure 5. The high-low-high pulse on the `nCONFIG` pin starts the configuration process. The `nSTATUS` pin is pulled low for up to 100 ms, and the `CONF_DONE` pin is pulled down to GND. Once the `CONF_DONE` pin is low, address generation begins. The low logic level on the `CONF_DONE` pin also enables the output of the EPROM. In an APU configuration scheme, the first address generated is 00000H; in an APD configuration scheme, it is 3FFFFH.

The configuration events in Figure 6 are based on the `RDCLK` signal rather than the `DCLK` signal. The `RDCLK` signal, a Clock signal that is generated by dividing the `DCLK` signal by eight, is used to frame the data bytes supplied by the parallel EPROM. In the APU and APD configuration schemes, the FLEX 8000 device generates the `DCLK` signal internally and uses it to serialize the incoming data words. On each pulse of the `RDCLK` signal, the FLEX 8000 device latches an 8-bit byte, and the following eight pulses on `DCLK` convert that 8-bit value into a serial data stream. The `RDCLK` signal is available as an output pin during configuration. (In user mode, the `RDCLK` pin is available as an I/O pin.) You can monitor this signal to ensure that the parallel EPROM observes the data setup and hold time requirements for the FLEX 8000 device.

Figure 6. Active Parallel Up Configuration Timing Waveforms

A rising edge on RDCLK increments the address counter ADDR[17..0], which is driven out to the parallel EPROM. The parallel EPROM then sends the addressed byte of configuration data to the FLEX 8000 device.



A new address is presented on the ADD[17..0] pins a short time (t_{CAV}) after a rising edge on RDCLK. Table 4 shows the timing parameters for the APU and APD configuration in Figure 5. Before the subsequent rising edge on RDCLK, the external parallel EPROM must present valid data soon enough to meet the t_{DSU} setup time for the data. This subsequent rising edge on RDCLK latches data, based on the address generated by the previous Clock cycle. EPROMs with access times faster than 500 ns should be used to guarantee the data setup time.

Table 4. Active Parallel Up & Down Configuration Timing Parameters

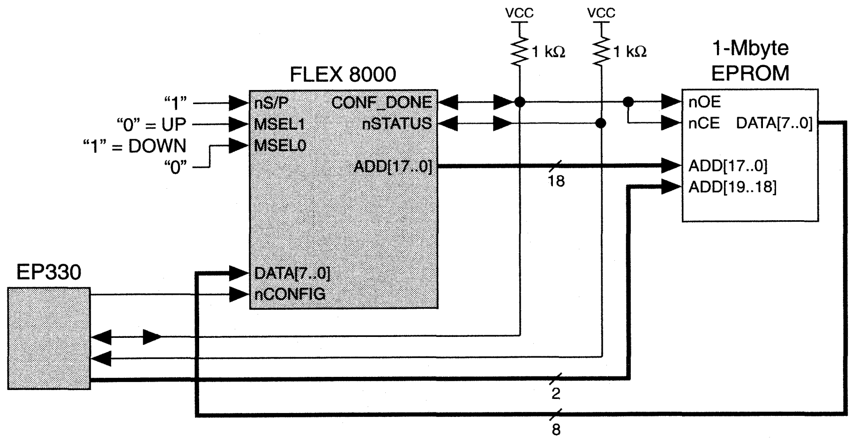
Symbol	Parameter	Min	Max	Unit
t_{CF2ST}	nCONFIG low to nSTATUS low		1	µs
t_{CFG}	nCONFIG low pulse width	2		µs
t_{STATUS}	nSTATUS low pulse width	2.5		µs
t_{CF2CD}	nCONFIG low to CONF_DONE low		1	µs
t_{CF2AV}	nCONFIG high to first valid address		3.5	µs
t_{CAV}	RDCLK rising edge to address valid		1	µs
t_{DH}	Data hold time after rising clock edge (RDCLK)	0		ns
t_{DSU}	Data setup time before rising clock edge (RDCLK)	50		ns

Once the terminal count value for the FLEX 8000 device configuration data is reached, the FLEX 8000 device releases the `CONF_DONE` pin. The `CONF_DONE` pin is pulled up to V_{CC} via the pull-up resistor, and the FLEX 8000 device disables the output on the EPROM.

All FLEX 8000 devices provide 18 address lines, which are sufficient to uniquely decode up to 256 Kbytes of data, much more than the largest FLEX 8000 device requires (see Table 2). Although the 18 address lines limit FLEX 8000 devices to addressing 256 Kbytes of data, you can use a larger EPROM device (e.g., 512 Kbytes, 1 Mbyte, 2 Mbytes, etc.) by masking in the necessary offset addresses. In larger EPROMs, the FLEX 8000 device configuration information is treated as a separate "page" in the EPROM, and can be placed on any convenient boundary. However, some additional logic is required to provide the offset address.

Figure 7 shows how you can use an Altera EP330 device as a decoder that asserts the necessary page-offset address onto the address bus during configuration. The EP330 allows the 18-bit address generated by the FLEX 8000 device to select one of four 256-Kbyte "pages" in the EPROM. The EP330 should monitor the `nSTATUS` and `CONF_DONE` signals to ensure that errors are handled correctly. The inputs to the EP330 must be system-level control signals that select the appropriate page in the EPROM to be loaded into the FLEX 8000 device, and control when the configuration actually occurs. Timing for the circuit in Figure 7 is identical to the timing shown in Figure 6.

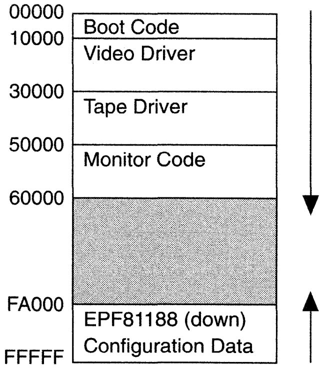
Figure 7. Active Parallel Device Configuration with Offset Address Generation Circuitry



The active parallel configuration schemes can generate addresses in either an ascending or descending order, depending on your system requirements. Counting up (APU configuration) is appropriate if the FLEX 8000 configuration data is stored at the beginning of an EPROM, or if the configuration data has been placed at some known offset in an EPROM larger than 256 Kbytes. Counting down (APD configuration) is appropriate if the low addresses are not available, e.g., if the CPU code must use the beginning of the EPROM or if the EPROM is also used to store other information that is expected to increase as an application evolves. The changing nature of the data size is characteristic of basic I/O system (BIOS) and boot PROMs.

Figure 8 shows an example of a BIOS EPROM memory map, in which the FLEX 8000 configuration data is placed at the bottom of the memory space in an APD configuration.

Figure 8. Typical BIOS EPROM Memory Map



Passive Parallel Synchronous Configuration

In a passive parallel synchronous (PPS) configuration scheme, the FLEX 8000 device is tied to an intelligent host. With PPS configuration, data can be driven directly onto a common data bus between the host and the FLEX 8000 device. The `DCLK`, `CONF_DONE`, `nCONFIG`, and `nSTATUS` signals are connected to a port on the host. Although you can drive the `DCLK` signal from the system Clock, you must have precise control of any interrupts that can influence the internal counting of the FLEX 8000 devices. This precise control is required because the FLEX 8000 device latches data on the rising edge of the `DCLK` signal, and the next eight falling edges of the `DCLK` signal serialize the latched data. New data is latched on every eighth rising edge of the `DCLK` signal until the FLEX 8000 device is completely configured.

Figure 9 illustrates PPS configuration of a FLEX 8000 device. In this circuit, the CPU generates a byte of configuration data and directs the FLEX 8000 device to latch and serialize the data by strobing a high pulse on the `DCLK` input. In Figure 9, no specific source is shown for the data bus `DATA[7..0]`, which is typically driven by a dedicated data latch. A microcontroller host usually has byte-wide ports that can be used for this data bus. If the host is a CPU or intelligent logic, a dedicated data register can be implemented with an octal latch. Depending on the capability of the host and the memory space implementation in the system, you can use an external memory instead to drive the data onto the system data bus. This type of external memory usage requires the memory to hold the data on the bus while the host executes the commands to direct the FLEX 8000 device to latch and serialize the data. However, not all processors can accommodate this type of operation.

Figure 9. Passive Parallel Synchronous Device Configuration

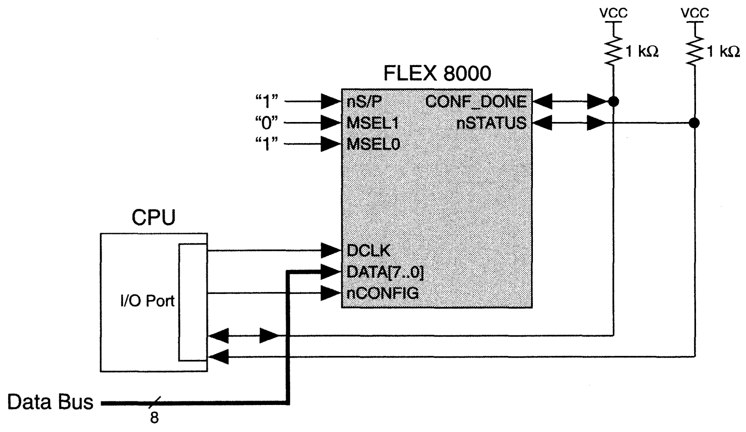


Figure 10 shows the timing for the PPS configuration scheme. The CPU generates Clock cycles and data; eight `DCLK` cycles are required to latch and serialize each 8-bit data word. A new data word must be present at the `DATA[7..0]` inputs upon every eighth `DCLK` cycle.

Figure 10. Passive Parallel Synchronous Configuration Timing Waveforms

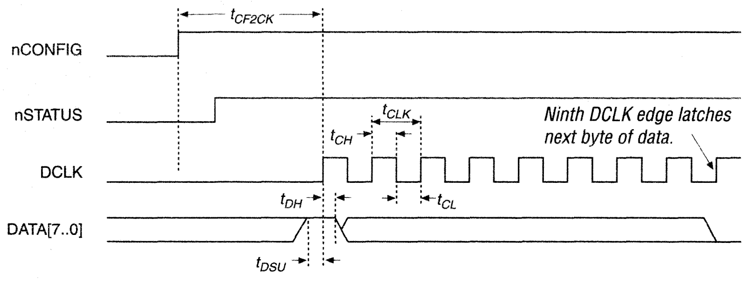


Table 5 shows the timing parameters associated with PPS configuration.

Table 5. Passive Parallel Synchronous Configuration Timing Parameters

Symbol	Parameter	Min	Max	Unit
t_{CF2CK}	`nCONFIG` high to first rising edge on `DCLK`	5		µs
t_{DSU}	Data setup time before rising edge on `DCLK`	50		ns
t_{DH}	Data hold time after rising edge on `DCLK`	0		ns
t_{CH}	`DCLK` clock high time	80		ns
t_{CL}	`DCLK` clock low time	80		ns
t_{CLK}	`DCLK` period	160		ns
f_{MAX}	`DCLK` maximum frequency		6	MHz

Passive Parallel Asynchronous Configuration

With the passive parallel asynchronous (PPA) configuration scheme, a FLEX 8000 device in a system can be configured in parallel with the rest of a board. The FLEX 8000 device accepts a parallel byte of input data, then serializes the data with its internal synchronization Clock. The device is selected with the `nCS` and `CS` chip select pins, so multiple devices can reside on the same data bus. The ability to select individual FLEX 8000 devices allows multiple devices to be configured in parallel by a single intelligent host.

This efficient handshaking allows an intelligent host to simultaneously configure multiple FLEX 8000 devices or other configurable portions of the system. Figure 11 illustrates PPA configuration of a FLEX 8000 device. A microcontroller is used as the intelligent host to ensure that sufficient dedicated I/O ports are available to drive all control signals and the data bus to the FLEX 8000 device. The chip select signals `CS` and `nCS` are both used to select the device. However, you can also tie `nCS` to GND and control chip selection with the `CS` pin only (or vice-versa), thus saving one bit in the I/O port.

Figure 11. Passive Parallel Asynchronous Device Configuration with Dedicated Ports

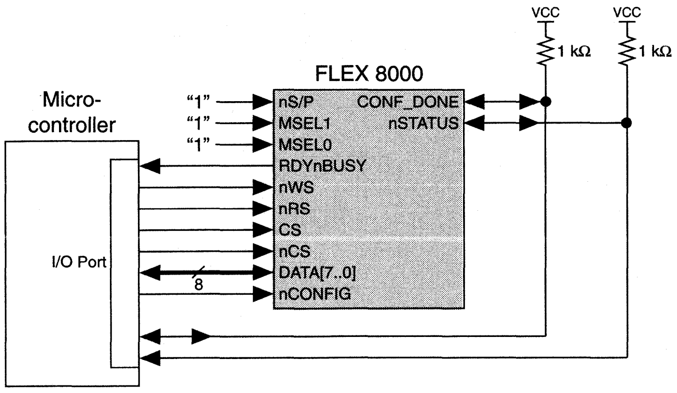
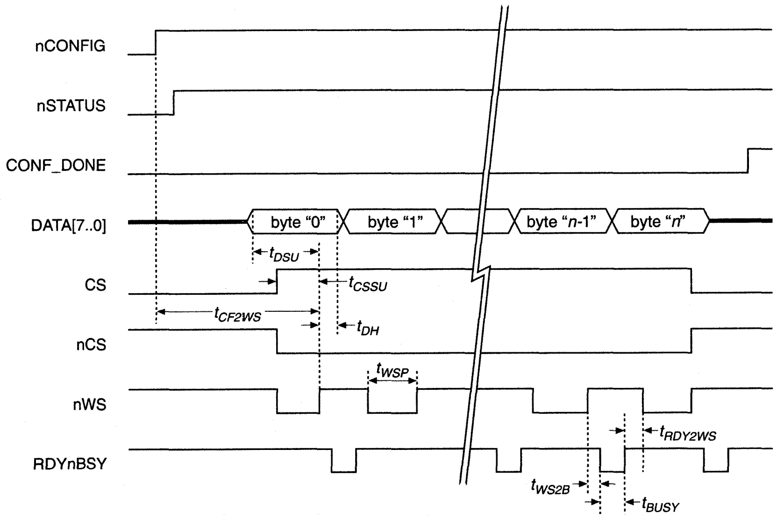


Figure 12 shows the timing for the PPA configuration scheme. The CPU presents an 8-bit data word to the FLEX 8000 device, and indicates that the word is valid by strobing a low pulse on the nWS input. The FLEX 8000 device senses the rising edge of the nWS signal, latches the data on the DATA[7..0] inputs, and uses its internal oscillator to serialize the 8-bit data word.

Figure 12. Passive Parallel Asynchronous Timing Waveforms



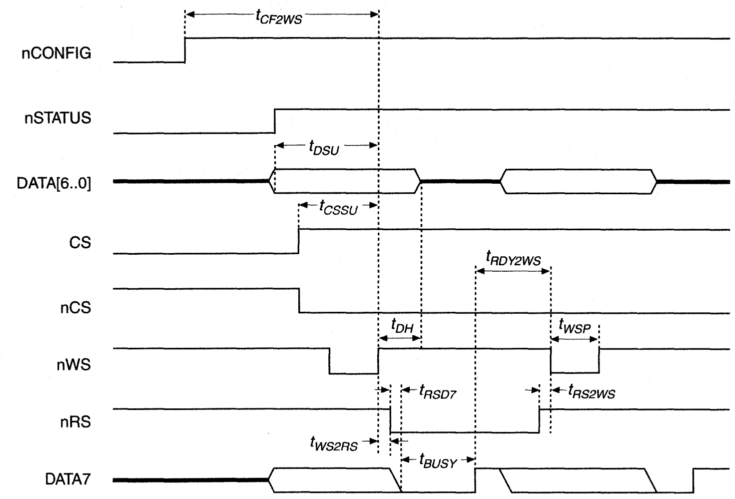
The CPU must poll the RDYnBUSY signal to establish when the FLEX 8000 device is ready to receive more data. RDYnBUSY falls immediately after the rising edge of the nWS signal that latches data, indicating that the device is busy. While the FLEX 8000 device processes the data byte, RDYnBUSY remains low. On the eighth falling edge of DCLK, RDYnBUSY returns to V_{CC}, indicating that another byte of data can be latched. Table 6 shows the timing parameters associated with PPA configuration.

Table 6. Passive Parallel Asynchronous Configuration Timing Parameters

Symbol	Parameter	Min	Max	Unit
t_{CF2WS}	nCONFIG high to first nWS rising edge	5		µs
t_{DSU}	Data setup time before rising edge on nWS	50		ns
t_{DH}	Data hold time after rising edge on nWS	0		ns
t_{CSSU}	Chip selected delay before rising edge on nWS	50		ns
t_{WSP}	nWS low pulse width	500		ns
t_{WS2B}	nWS rising edge to RDYnBSY low		50	ns
t_{BUSY}	RDYnBSY low pulse width		4	µs
t_{RDY2WS}	RDYnBSY rising edge to nWS falling edge	50		ns
t_{WS2RS}	nWS rising edge to nRS falling edge	500		ns
t_{RS2WS}	nRS rising edge to nWS falling edge	500		ns
t_{RSD7}	nRS falling edge to DATA7 valid with RDYnBSY signal		50	ns

As an alternative to polling the RDYnBUSY signal, the CPU can determine the status of the FLEX 8000 device by strobing a low pulse on the nRS input to the FLEX 8000 device. This strobe causes the FLEX 8000 device to present the RDYnBUSY status on the bidirectional pin DATA7 so that the CPU can determine device status from the data bus, instead of using an additional port on the CPU for the RDYnBUSY signal. This low pulse on nRS must occur only during the corresponding high pulse (inactive) on the nWS signal. The timing waveforms in Figure 13 show how the nRS pin can be used to poll the status of the device with the bidirectional pin DATA7 of the circuit shown in Figure 11. The timing parameters given in Table 6 also apply to Figure 13.

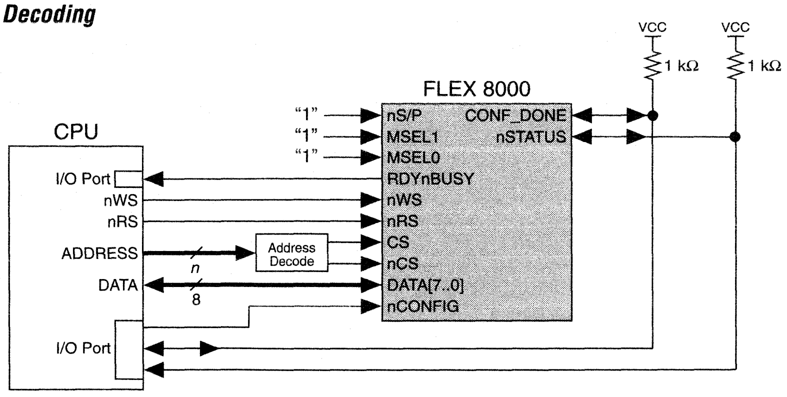
Figure 13. Passive Parallel Asynchronous Timing Waveforms Using nRS & DATA7



The circuit in Figure 11 takes advantage of the architecture of a microcontroller host. Figure 14 shows an alternative to this circuit, in which a CPU serves as the intelligent host and the FLEX 8000 device is treated more as a memory than as a port. The nWS and nRS inputs to the FLEX 8000 device are driven by the CPU's memory read/write control pins; the DATA[7..0] inputs to the FLEX 8000 device are driven directly by the system data bus. As in Figure 11, the nSTATUS and nCONFIG control signals must be driven by an intelligent I/O port, but the CS and nCS chip select signals are decoded from the address bus and not driven from an I/O port on the CPU. This address decoding scheme allows the CPU to write to the FLEX 8000 device as a memory. A small programmable logic device, such as the Altera EPM7032, is ideal for quickly decoding a wide address and selecting the FLEX 8000 device.

PPA configuration is useful when multiple FLEX 8000 devices are configured simultaneously. The CPU reads a byte of configuration data from the disk or from memory, and then writes it to the FLEX 8000 device. The CPU then polls the RDYnBUSY signal (or the DATA7 pin via the nRS input) to determine when another data byte can be written. Timing for this circuit is identical to the timing shown in Figure 13, although t_{CSSU}, the minimum chip select delay before the rising edge of nWS, must increase to account for the time required to decode the address.

Figure 14. Passive Parallel Asynchronous Device Configuration with Address Decoding



The configuration process is generally controlled with a precise order of steps, so the timing constraints are minimal. The following steps show the typical control sequence executed by the CPU:

1. Pull the `nCONFIG` pin to GND, hold it for 10 µs, then pull it up to V_{CC}.
2. Read the next byte of configuration data from an EPROM or a mass storage device such as a hard disk.
3. Generate the address of the FLEX 8000 device.
4. Perform a memory write cycle to the FLEX 8000 device address using the stored configuration data byte.
5. Poll the `RDYnBUSY` signal. When it goes high, transfer the next byte of configuration data by repeating steps 2 through 4.
6. Repeat steps 2 through 5 until the FLEX 8000 device pulls the `CONF_DONE` net high, which indicates that configuration is complete.

Passive Serial Configuration

The passive serial (PS) configuration scheme uses an external controller to configure the FLEX 8000 device with a serial bit-stream. The FLEX 8000 device is treated as a slave device with a 5-wire interface to the external controller. The external controller can be one of the following:

❏ The MAX+PLUS II Programmer, used together with the PL-MPU Master Programming Unit, an appropriate device adapter, and the FLEX Download Cable.

❏ An intelligent host such as a microcontroller or a CPU. This type of PS configuration is similar to the PPA and PPS configuration schemes, but uses a bit-wide serial data path instead of a byte-wide parallel data path.

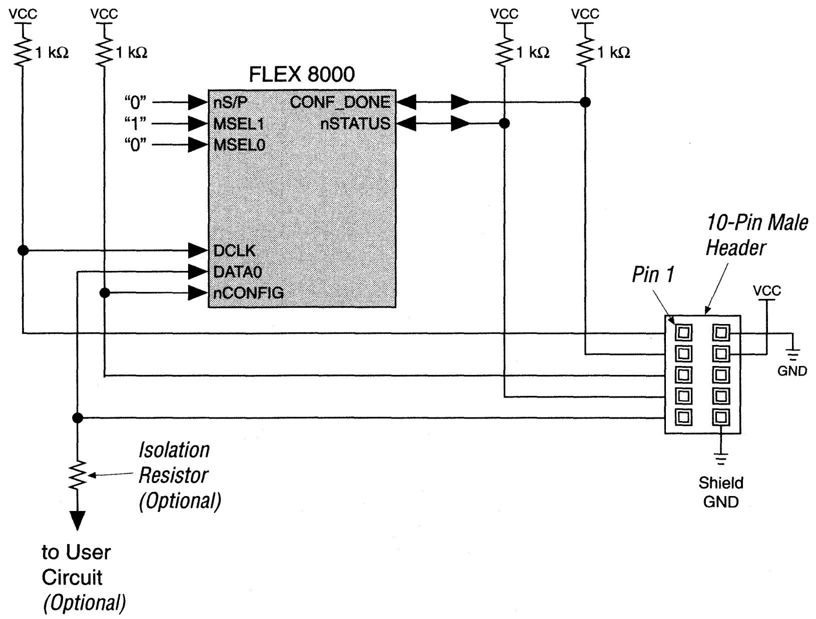
- ❑ The Altera BitBlaster, an RS-232-compatible serial download cable. For information on using the BitBlaster for configuring FLEX 8000 devices, refer to the current *BitBlaster Serial Download Cable Data Sheet*.

Passive Serial Configuration with the FLEX Download Cable

Passive serial configuration with the FLEX Download Cable uses the MAX+PLUS II Programmer and Altera programming hardware as the external controller. The Altera FLEX Download Cable can connect any Configuration EPROM programming adapter, which is installed on the PL-MPU Master Programming Unit, to a single target FLEX 8000 device in the prototype system. The FLEX Download Cable provides a 5-wire connection between the FLEX 8000 device and the programming adapter. Configuration data is taken from the SRAM Object File (**.sof**) generated automatically during project compilation and downloaded by the MAX+PLUS II Programmer. Once the device is configured, the programming hardware is tri-stated and electrically removed from the circuit. This type of PS configuration allows you to perform multiple design iterations rapidly.

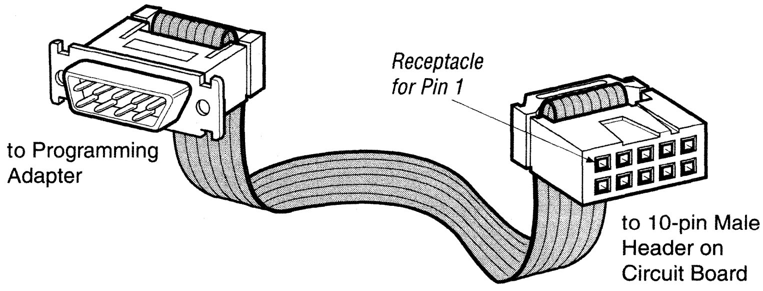
Figure 15 shows how the FLEX Download Cable interfaces to the target FLEX 8000 device. The 10-pin male header on the circuit board has two rows of five pins, spaced on 0.1-inch centers, that connect to the

Figure 15. Passive Serial Device Configuration with the FLEX Download Cable



configuration pins on the FLEX 8000 device. Standard 10-pin IDS-type male headers are readily available to provide the target board connections. A 10-pin female plug on one end of the FLEX Download Cable is connected to the 10-pin male header on the circuit board; the other end of the FLEX Download Cable is connected to a Configuration EPROM programming adapter. See Figure 16. Timing for PS configuration is identical to the timing for bit-wide PS configuration shown later in this application note.

Figure 16. FLEX Download Cable Signals & Positions



Header Pin Connections:

DCLK	CONF_DONE	nCONFIG	nSTATUS	DATA0
GND	VCC	N.C.	N.C.	GND

When a FLEX 8000 device is configured via the FLEX Download Cable, the DCLK, CONF_DONE, nCONFIG, DATA0, and nSTATUS pins on the cable are connected directly to the pins of the same names on the FLEX 8000 device. The VCC and GND pins must be tied to the system power planes. These VCC and GND pins supply power to the optical isolation circuitry in the programming adapter; they do not supply power to the target FLEX 8000 device. Refer to the current *FLEX 8000 Programmable Logic Device Family Data Sheet* for device pin numbers.

The DCLK, CONF_DONE, nCONFIG, and nSTATUS pins on the FLEX 8000 device are dedicated configuration pins. Since they are not available as user I/O pins, they do not require isolation from the rest of the circuit. However, a system must include pull-up resistors that pull these pins up to V_{CC}, as shown in Figure 15. These resistors allow you to remove the FLEX Download Cable after configuration is complete without introducing any noise from floating inputs.

The DATA0 pin is available as an I/O pin during user-mode operation, and may require isolation, depending on how it is used. During configuration, the DATA0 pin on the FLEX 8000 device acts as an input, and is driven by the programming hardware. If the DATA0 pin is an output pin during user mode, the signal that it drives does not need to be buffered. However, if the DATA0 pin is an input or bidirectional pin during user mode, contention may occur between the user-mode signal and the FLEX Download Cable during configuration.

If the signal that drives the DATA0 pin during user mode is tri-stated during configuration and initialization, no conflict occurs. However, if this signal is active during configuration, the DATA0 input pin must be isolated from the active source. You can isolate the DATA0 pin by inserting a tri-state buffer between the DATA0 pin and the rest of the network that it drives. This tri-state buffer must be controlled by external logic.

If you cannot use active isolation, placing a 550-Ω resistor between the user-mode signal and the DATA0 pin should provide adequate isolation. The FLEX Download Cable is driven by 12-mA drivers, which supply sufficient current to mask any signals that may be present at the other end of the resistor. Resistive isolation may not be suitable for very-high-speed circuits. Actual in-circuit performance should be evaluated in the laboratory to ensure that this isolation scheme does not affect other portions of the circuit.

The no connect (N.C.) pins shown in Figure 16 are reserved, and should not be tied to any data or power signals. The header should be placed as close as possible to the FLEX 8000 device.

For additional information on passive serial configuration with the FLEX Download Cable, refer to "Configuring a FLEX 8000 Device In-System with MAX+PLUS II & the FLEX Download Cable" later in this application note.

Bit-Wide Passive Serial Configuration

The passive serial configuration scheme provides a bit-wide passive interface for device configuration. No handshaking is provided in any PS configuration. Therefore, the FLEX 8000 device must be configured at 6 MHz or less. Figure 17 shows how a bit-wide PS configuration is implemented. Data bits are presented on the DATA0 input, with the least significant bit of each byte of data presented first. The DCLK is strobed with a high pulse to latch the data. This serial data loading continues until the CONF_DONE pin goes high, indicating that the device is fully configured. The data source can be any source that the host can address.

Figure 17. Bit-Wide Passive Serial Device Configuration

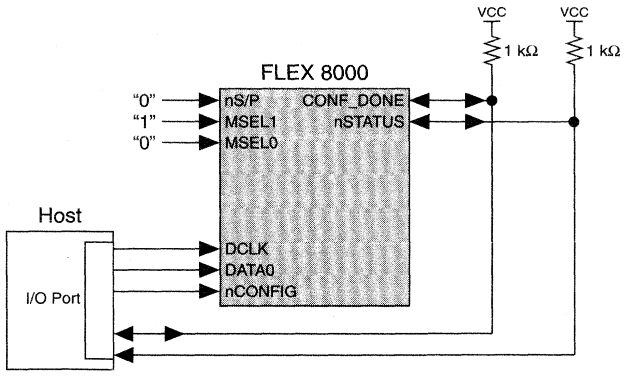


Figure 18 shows the timing for bit-wide PS configuration.

Figure 18. Bit -Wide Passive Serial Timing Waveforms

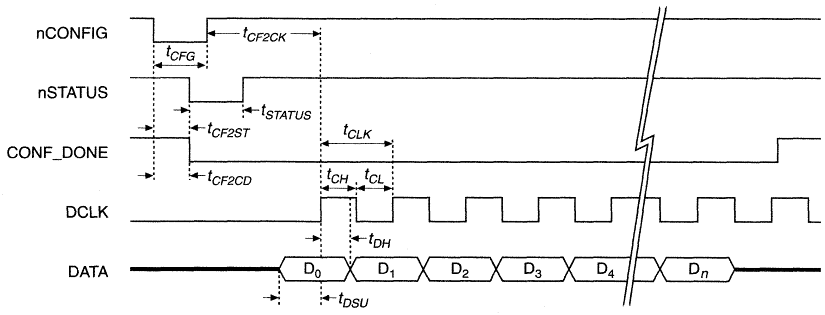


Table 7 gives the timing parameters for bit-wide PS configuration.

Table 7. Passive Serial Configuration Timing Parameters

Symbol	Parameter	Min	Max	Unit
t_{CF2CD}	nCONFIG low to CONF_DONE low		1	µs
t_{CF2ST}	nCONFIG low to nSTATUS low		1	µs
t_{CFG}	nCONFIG low pulse width	2		µs
t_{STATUS}	nSTATUS low pulse width	2.5		µs
t_{CF2CK}	nCONFIG high to first rising edge on DCLK	5		µs
t_{DSU}	Data setup time before rising edge on DCLK	50		ns
t_{DH}	Data hold time after rising edge on DCLK	0		ns
t_{CH}	DCLK high time	80		ns
t_{CL}	DCLK low time	80		ns
t_{CLK}	DCLK period	160		ns
f_{MAX}	DCLK maximum frequency		6	MHz

In-Circuit Reconfiguration

After a FLEX 8000 device has entered the user mode, you can choose to replace the configuration data pattern inside the device at any time. In this process, called *in-circuit reconfiguration*, new configuration data is selected using one of three methods, depending on the configuration scheme:

- ❏ In a passive configuration scheme, a different file can be downloaded from a mass-storage system.
- ❏ In the AS configuration scheme, multiple sets of configuration data can be stored in one or more serial Configuration EPROMs. Each set of data is used in succession.
- ❏ In the APU and APD configuration schemes, new configuration data is selected by externally multiplexing a different EPROM source onto the data path or by providing offset address generation circuitry to select a different page within the same EPROM.

Because the SRAM cells used to configure the functionality of the FLEX 8000 architecture are volatile, they can be reprogrammed without removing the FLEX 8000 device from the circuit board.

The nCONFIG input controls device reconfiguration. In the active configuration schemes shown in Figures 1, 3, and 4, the nCONFIG pin is tied to V_{CC} to force the FLEX 8000 device to automatically configure itself at system power-up. In the PPA and PPS configuration schemes, controlling logic is used on the nCONFIG input to determine when the configuration starts, as shown in Figures 9, 11, and 14. However, all configuration schemes allow you to connect the nCONFIG pin to a port on an intelligent host, which can be used to control the configuration process. If nCONFIG is held low, the configuration process can be delayed as necessary. For example, the nCONFIG pin can be held low during system initialization and then pulled high when it is appropriate to configure the FLEX 8000 device.

At any time during system operation, regardless of the current state of the FLEX 8000 device, the nCONFIG pin can be used to restart the configuration process. When nCONFIG is driven low and then high again, the device resets itself and prepares for configuration. In an active configuration scheme, the FLEX 8000 device immediately starts retrieving data from the external EPROM; in a passive configuration scheme, it prepares to receive the data from the intelligent host. An example of a reset pulse on nCONFIG in an APU configuration scheme is shown in Figure 6 earlier in this application note. This nCONFIG timing applies to all configuration schemes whenever the device is reconfigured.

All latched and registered data in the device is lost during reconfiguration, so any counter values or the current state of the device should be stored either in the intelligent host's storage system or in some external circuitry, such as an Altera EPLD. The entire reconfiguration process requires about 100 ms. The system resumes normal operation after the FLEX 8000 device releases the CONF_DONE pin, indicating that initialization is complete.

Configuration Control Features

Within a FLEX 8000 device, the configuration and initialization processes can be controlled with two types of built-in resources:

❏ Device configuration option bits
❏ Device configuration pins

This section provides detailed information on configuration option bits and pins. The usage of various options and pins is discussed in the descriptions of individual configuration schemes earlier in this application note. Some configuration pins and options can also be used together to provide additional configuration and initialization control.

Device Configuration Option Bits

FLEX 8000 devices have device configuration option bits that allow you to control device behavior during configuration. Table 8 describes all FLEX 8000 device option bits and their availability in different configuration schemes. You can set these options on a device-by-device basis in the MAX+PLUS II software with the **FLEX 8000 Individual Device Options** dialog box. You can also enter global default device option settings for an entire project with the **FLEX 8000 Device Options** dialog box.

Table 8. FLEX 8000 Device Configuration Option Bits (Part 1 of 2)

Device Option	Configuration Scheme	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
User-Supplied Start-Up Clock	All	After a FLEX 8000 device is configured, it must be initialized over the course of 10 Clock cycles. The user can choose the source of the Clock.	In the AS, APU, APD, and PPA configuration schemes, the internal FLEX 8000 device oscillator supplies the initialization Clock. In the PS and PPS configuration schemes, the internal oscillator is disabled, so external circuitry must provide the initialization Clock on the `DCLK` pin.	The user provides the Clock on the `CLKUSR` pin. This type of Clock can be used to fully synchronize initialization for multiple FLEX 8000 devices. The maximum user-supplied Clock frequency is 6 MHz, and the Clock should have a 50% duty cycle.
Auto-Restart Configuration on Frame Error	AS, APU, APD	If a data error occurs when a FLEX 8000 device is configured with an active configuration scheme, the user can choose how to restart the configuration.	The configuration process halts and the user must externally direct the device to restart the configuration process. If a configuration error occurs, the `nSTATUS` pin is driven and held low until the `nCONFIG` pin is externally pulled low and then high again.	Directs the device to automatically restart the configuration process. The `nSTATUS` pin is driven and held low for 10 Clock cycles and is then released. The `nSTATUS` pin subsequently pulls up to V_{CC}, indicating to any external circuitry that the reconfiguration process has started.
	AS		In an AS configuration scheme, the external `nCONFIG` reset pulse resets the Configuration EPROM if the `nCONFIG` pin on the FLEX 8000 device is tied to the Output Enable pin on the Configuration EPROM.	In an AS configuration scheme, the `nSTATUS` reset pulse automatically resets the Configuration EPROM if the `nSTATUS` pin on the FLEX 8000 device is tied to the Output Enable pin on the Configuration EPROM.
Release Clears Before Tri-States	All	During configuration, the I/O pins on the device are tri-stated by an Output Enable override. The user can choose the order in which the tri-states are released and the registered logic cells and peripheral registers are cleared during initialization.	Directs the device to release the Output Enable override on the tri-state buffer before releasing the Clear signal on registered logic cells and peripheral registers during initialization.	Directs the device to release the Clear signal on registered logic cells and peripheral registers before releasing the Output Enable override on the tri-state buffer during initialization.
Enable DCLK Output In User Mode	AS, APU, APD, PPA	FLEX 8000 devices drive the `DCLK` signal during configuration in all active configuration schemes and the PPA configuration scheme. The `DCLK` signal can range from 2 to 6 MHz in frequency. The user can choose whether to enable the `DCLK` signal during user mode. The duty cycle and frequency of the `DCLK` signal are not guaranteed.	Disables the `DCLK` pin when the device operates in user mode after device configuration and initialization have been completed.	Enables the `DCLK` pin when the device operates in user mode after device configuration and initialization have been completed.

Table 8. FLEX 8000 Device Configuration Option Bits (Part 2 of 2)

Device Option	Configuration Scheme	Option Usage	Default Configuration (Option Off)	Modified Configuration (Option On)
Disable Start-Up Time-Out	All	The CONF_DONE pin, a bidirectional open-drain pin, is held at GND by the FLEX 8000 device during configuration. Once configuration is complete, the CONF_DONE pin is released and the FLEX 8000 device treats the pin as an input pin. In most applications, the CONF_DONE pin is pulled up to V_{CC} via a 1.0-kΩ resistor. This low-to-high transition directs the FLEX 8000 device to begin initialization. The user can enable or disable the time-out error checking that determines whether CONF_DONE goes high within 10 Clock cycles.	If the CONF_DONE pin does not go high within 10 Clock cycles after being released by the device, the device drives the nSTATUS pin low at the end of the configuration cycle, indicating an error condition.	If the CONF_DONE pin does not go high within 10 Clock cycles after being released by the device, the device continues to wait for CONF_DONE to go high. To delay initialization, the CONF_DONE node can be held low externally after the FLEX 8000 device has released the CONF_DONE pin, if, for example, the user wishes to control the time required for the FLEX 8000 device to enter user mode.
Enable JTAG Support	All	Enables post-configuration JTAG boundary-scan testing support in FLEX 8000 devices that provide JTAG circuitry.	JTAG boundary-scan testing is not available. In the EPF8282, EPF8282V, EPF8636, and EPF8820 devices, the four JTAG pins (TDI, TDO, TMS, and TCLK) are available as user I/O pins. In EPF81500 devices, these four pins are disabled.	JTAG boundary-scan testing is available on the four JTAG pins (TDI, TDO, TMS, and TCLK) after device configuration has been completed.

Device Configuration Pins

FLEX 8000 devices include control pins that modify the sequence and timing of the configuration and initialization processes, and provide a variety of configuration options. Some configuration pins have the same effect regardless of the selected configuration scheme; others are specific to a particular configuration scheme. Table 9 summarizes the functionality of each configuration pin.

Table 9. Pin Functions (Part 1 of 2) *Note (1)*

Pin Name	User Mode	Configuration Scheme	Pin Type	Description
nSP	n/a	All	Input	Serial/Parallel selection input. A low input selects a serial configuration scheme; a high input selects a parallel configuration scheme.
MSEL1 MSEL0	n/a	All	Input	2-bit configuration scheme selection inputs that are used in conjunction with nSP to select the configuration scheme. The bit patterns of nSP:MSEL1:MSEL0 are associated with the following configuration schemes: 000 = AS 100 = APU 001 = Reserved 101 = PPS 010 = PS 110 = APD 011 = Reserved 111 = PPA
nSTATUS	n/a	All	Bidirectional Open Drain	Command mode status output. The FLEX 8000 device drives the nSTATUS pin low immediately after power-up, then releases it within 100 ms. The nSTATUS pin must be pulled up to V_{CC} with a 1.0-kΩ resistor. If an error occurs during configuration, nSTATUS is pulled low again by the FLEX 8000 device.
nCONFIG	n/a	All	Input	Configuration control input. A low input resets the FLEX 8000 device. A low-to-high transition starts a configuration cycle.
CONF_DONE	n/a	All	Bidirectional Open Drain	Status output. Driven low by the FLEX 8000 device during the configuration process.
			Input	Status input. A high input directs the device to execute the initialization process and enter user mode. The CONF_DONE net must be pulled up to V_{CC} with a 1.0-kΩ resistor. The CONF_DONE pin may be actively driven low by an external source to delay the FLEX 8000 device initialization process. This feature is useful when the configuration process will be completed some time before actual operation is necessary.
DCLK	*(2)* *(3)*	AS PPS, PS	Output Input	Clock source for external PROM devices. Clock input from external host.
nWS	I/O	PPA	Input	Write Strobe input. A low-to-high transition causes the FLEX 8000 device to latch a byte of data on the DATA[7..0] pins.
nRS	I/O	PPA	Input	Read Strobe input. A low input directs the FLEX 8000 device to place the RDYnBUSY signal on the DATA7 pin.
RDCLK	I/O	APD, APU	Output	Divide-by-8 of DCLK output. Used internally to serialize an 8-bit data stream in the byte-wide APU or APD configuration scheme.
nCS CS	I/O	PPA	Input	Chip Select inputs. A low input on nCS and a high input on CS selects a specific FLEX 8000 device for configuration. If only one of the chip selects is used, the other must be tied to its active level (e.g., nCS would be tied to GND).
RDYnBUSY	I/O	PPA	Output	Ready output. A high output indicates that the FLEX 8000 device is ready to accept another byte of data. A low output indicates that the device is not ready to receive data.
CLKUSR	I/O	All	Input	Optional user-supplied Clock input. Synchronizes the initialization process.
ADD17 to ADD0	I/O	APD, APU	Outputs	Address outputs. Driven by the FLEX 8000 device to uniquely address up to 256 Kbytes of external configuration memory devices.

Table 9. Pin Functions (Part 2 of 2)

DATA7 to DATA0	I/O	APD, APU, PPA, PPS	Inputs	Data inputs. Byte-wide configuration data is presented to the FLEX 8000 device on all 8 data pins.
DATA0		AS, PS	Input	Data input. Bit-wide configuration data is presented to the FLEX 8000 device on the DATA0 pin.
DATA7		PPA	Output	In the PPA configuration scheme, the DATA7 pin presents the RDYnBUSY signal after the device receives an nRS strobe. Using the DATA7 pin may be more convenient than using the RDYnBUSY output pin.
SDOUT	I/O	All	Output	Reserved configuration output. Drives out during command mode.

Notes:
(1) The maximum number of dual-purpose configuration pins that can be used as I/O pins in user mode varies in different configuration schemes:

AS:	3 pins	APU:	29 pins	APD:	29 pins
PS:	3 pins	PPS:	10 pins	PPA:	15 pins

(2) The internally generated DCLK signal that is used to configure FLEX 8000 devices with the AS, APU, APD, and PPA configuration schemes is available during user-mode operation if the *Enable DCLK Output in User Mode* configuration option bit is turned on. The DCLK signal can range from 2 to 6 MHz in frequency; the duty cycle and frequency are not guaranteed.
(3) An externally generated DCLK signal is used to configure FLEX 8000 devices with the PS and PPS configuration schemes. After configuration has finished, the external host can continue to drive the DCLK signal during user-mode operation.

Seven of the device pins are dedicated to the configuration process and cannot be used as I/O pins in user mode. Other configuration pins are dual-purpose pins that also can be used as I/O pins when the device operates in user mode. You can choose whether to use each dual-purpose pin as an I/O pin in user mode, as well as whether to force a dual-purpose pin to tri-state (i.e., drive a high-impedance logic level).

You can specify these settings for each pin on a device-by-device basis in MAX+PLUS II with the **FLEX 8000 Individual Device Options** dialog box. You can also enter global default pin settings for an entire project with the **FLEX 8000 Device Options** dialog box. Turning on the *Reserve* option for a specified pin in either dialog box prevents the pin from being used as an I/O pin during user mode; turning on the *Tri-State* option forces the pin to tri-state. A reserved pin should not be connected to any circuitry on the target board unless it is also tri-stated. Otherwise, the reserved pin will drive an unknown logic level that may cause logic contention with other signals on the board.

The nSTATUS, nCONFIG, CONF_DONE, and CLKUSR device configuration pins are available to monitor the configuration process and control how the device loads data, initializes, and enters user-mode operation. These pins can be used together with configuration option bits to provide additional configuration and initialization control.

nSTATUS Pin

The nSTATUS pin is an open-drain, bidirectional pin. When the FLEX 8000 device powers up, it pulls this pin low and then releases it within 100 ms. During configuration, the nSTATUS pin can be polled externally to verify that the FLEX 8000 device is being configured. If an error occurs during configuration, the nSTATUS pin is pulled and held low. In addition, if an external circuit pulls the nSTATUS pin low during either command-mode or user-mode operation, the FLEX 8000 device senses an error condition. After the pin is pulled low, configuration must be restarted.

Configuration is restarted with a high-low-high pulse on the nCONFIG pin. As an alternative, if the *Auto-Restart Configuration on Frame Error* option bit is turned on, the FLEX 8000 device can restart the configuration automatically when an error is detected. If this option bit is turned on, the nSTATUS pin is pulled low for a few microseconds and then released, indicating that the reconfiguration cycle has started. See Figure 3 earlier in this application note for an example of an AS configuration scheme that supports auto-reconfiguration.

If V_{CC} falls below an acceptable level during user-mode operation, the nSTATUS pin is pulled and held low, indicating an error condition. See "Configuration Reliability" later in this application note for more details.

nCONFIG Pin

The nCONFIG pin is a dedicated input that is used to start a configuration cycle. In most applications, the nCONFIG pin is tied to V_{CC}, directing the FLEX 8000 device either to immediately start configuration in an active configuration scheme, or to prepare immediately for configuration in a passive configuration scheme.

When the nCONFIG pin is held at GND, the FLEX 8000 device is reset and ready to start configuration. Configuration begins only after the pin is pulled up to V_{CC}. The nCONFIG pin can thus be held low to delay the configuration process and thus prevent data from loading until the desired time.

If an application requires a FLEX 8000 device to be reconfigured after system power-up, the nCONFIG pin must be tied to some external intelligent circuitry that monitors and controls that configuration process, as described in "In-Circuit Reconfiguration" earlier in this application note.

CONF_DONE Pin

The CONF_DONE pin is an open-drain, bidirectional pin that reflects the configuration status. When a FLEX 8000 device is ready to begin loading data, the CONF_DONE pin is pulled to GND and remains at GND while the data is loading, indicating that the FLEX 8000 device is being configured. After the last configuration data byte has been read, the CONF_DONE pin is released and pulled to V_{CC} by an external pull-up resistor, indicating that configuration is finished. The FLEX 8000 device interprets this low-to-high transition on the CONF_DONE signal as the command to initialize and enter the user mode.

If the CONF_DONE pin does not pull up to V_{CC} within ten Clock cycles of the final configuration data byte, the FLEX 8000 device detects an error condition, aborts the initialization process, and drives and holds the nSTATUS pin low. If the nSTATUS pin is low, it indicates either that an error has occurred in the application circuit, or that the configuration data-stream is corrupt.

The CONF_DONE pin can also be used to control the initialization process. You can disable error checking on the CONF_DONE net by turning on the device's *Disable Start-Up Time-Out* configuration option bit, so that the failure of CONF_DONE to pull to V_{CC} does not cause an error condition. The CONF_DONE network can then be driven by some external logic and held low until initialization is desired.

CLKUSR Pin

The CLKUSR pin can coordinate the initialization of multiple FLEX 8000 devices or synchronize the configuration of a FLEX 8000 device with other application logic in the system. In most applications, the FLEX 8000 device uses its internal oscillator (available externally as DCLK) to complete the initialization. After ten Clock cycles, the device enters user mode. You can turn on the *User-Supplied Start-Up Clock* configuration option bit and supply these ten Clock cycles on the CLKUSR pin to ensure that the device enters the user mode precisely when desired. Since the internal oscillators on all FLEX 8000 devices are not guaranteed to have the same frequency, you can use the CLKUSR pin to synchronize multiple FLEX 8000 devices in the same system.

MAX+PLUS II Configuration & Programming Support

The MAX+PLUS II software can generate four different types of configuration files for FLEX 8000 devices, as shown in Table 10. During project compilation, MAX+PLUS II automatically generates a POF and an SOF for each FLEX 8000 device. If necessary, you can generate a TTF or Hex File, as well as different POF(s), after compilation with the **Combine Programming Files** command (File menu) in the MAX+PLUS II Programmer or Compiler.

Table 10. FLEX 8000 Device Programming Files

File Type	Filename Extension	File Format	File Utilization
SRAM Object File	.sof	Binary	Downloaded directly into the FLEX 8000 device with the MAX+PLUS II Programmer using the FLEX 8000 Download Cable and Altera programming hardware.
Programmer Object File	.pof	Binary	Programmed into an Altera Configuration EPROM. The POF contains the configuration data, as well as the header, CRC, and pad bytes for configuring the FLEX 8000 device in an AS configuration scheme.
Hexadecimal (Intel-Format) File	.hex	ASCII text	Programmed into an industry-standard parallel EPROM. The Hex File contains the configuration data, as well as the header, CRC, and pad bytes for programming a parallel EPROM that configures a FLEX 8000 device in an APU or APD configuration scheme.
Tabular Text File	.ttf	ASCII text	A comma-separated version of the Hex File, used as source code in high-level programming languages. The TTF can be included in the source code for an intelligent host that configures the FLEX 8000 device in a PPA, PPS, or bit-wide PS configuration scheme. It can also be converted into an equivalent binary format that is directly loaded (LSB first) into the FLEX 8000 device.

Together, the MAX+PLUS II Programmer and Altera programming hardware provide the following capabilities:

❑ A POF can be programmed into an Altera Configuration EPROM for an AS configuration scheme.

❑ An SOF can be downloaded via the FLEX Download Cable for in-circuit PS configuration of a FLEX 8000 device.

For information on configuring FLEX 8000 devices with the BitBlaster, refer to the current *BitBlaster Serial Download Cable Data Sheet.*

Programming & Configuration Files

This section provides information on the characteristics of each type of configuration file. The process of creating different configuration files is described in "Combining & Converting Programming Files" later in this application note.

SRAM Object File

The SRAM Object File (**.sof**) is used during passive serial configuration when the data is downloaded directly into the FLEX 8000 device in-system with the MAX+PLUS II Programmer, the FLEX Download Cable, and Altera programming hardware. MAX+PLUS II automatically inserts the necessary header, formatting, and synchronization bits into the data stream when it downloads an SOF into a FLEX 8000 device. See "Configuring a FLEX 8000 Device In-System with MAX+PLUS II & the FLEX Download Cable" later in this application note for more information.

If configuration files are needed for other configuration schemes, MAX+PLUS II uses the data in SOF(s) to generate the appropriate POF(s), a TTF, or a Hex File.

Programmer Object File

The Programmer Object File (**.pof**) is used to program Altera Configuration EPROMs for an AS configuration scheme. MAX+PLUS II automatically generates a POF for every FLEX 8000 device in a project. In a multi-device project, each FLEX 8000 device has a dedicated Configuration EPROM. (Two POFs are generated and two Configuration EPROMs are required for an EPF81500 device.) MAX+PLUS II selects the appropriate Configuration EPROM to most efficiently store the data for each FLEX 8000 device.

Hexadecimal (Intel-Format) File

The Hexadecimal File (**.hex**) is an ASCII file in the Intel Hex format. This file contains the configuration and formatting data for an industry-standard byte-wide parallel EPROM that is used to configure a FLEX 8000 device in an APU or APD configuration scheme. The data in the Hex File is interpreted by the programming software when it is programmed into a parallel EPROM.

The usual base address for FLEX 8000 configuration data is the origin of the EPROM. In some applications, the origin of the EPROM is required by other system resources, so some offset is necessary. In an APU configuration scheme, the FLEX 8000 device generates ascending addresses starting at 00000H; in an APD configuration scheme, it generates descending addresses starting at 3FFFFH. The FLEX 8000 device provides these base addresses for the configuration data during configuration, but any needed offset address must be generated externally, as shown earlier in Figure 7. The APU scheme is appropriate if the FLEX 8000 configuration data can be stored at the beginning of an EPROM or at some known offset in an EPROM larger than 256 Kbytes. The APD scheme is appropriate if the FLEX 8000 configuration data is placed in an EPROM in which the low addresses are not available (as shown in Figure 8), or in an EPROM that also stores other information that is expected to increase as an application evolves.

Tabular Text File

The Tabular Text File (**.ttf**) is a tabular ASCII file that provides a comma-separated version of the configuration data for the PPA, PPS, and bit-wide PS configuration schemes. In some applications, the storage device that contains the FLEX 8000 configuration data is neither dedicated to nor connected directly to the FLEX 8000 device. For example, an EPROM can also contain executable code for a system (e.g., BIOS routines) and other data. The TTF allows you to include the FLEX 8000 configuration data as part of the source code for the intelligent host (using "include" or "source" commands). The host can access this data from an EPROM or a mass-storage device and load it into the FLEX 8000 device.

A TTF can be imported into nearly any Assembly Language or high-level language compiler. Consult the documentation for your compiler or assembler for information on including other source files.

If you do not include the TTF in the source code for an intelligent host, the file's comma-separated ASCII representation of the binary data must be converted into its equivalent 8-bit binary format (e.g., 85 would become 01010101) before it is loaded into the FLEX 8000 device. Data must be stored so that the least significant bit (LSB) of each byte of data is loaded first. The Altera Applications bulletin board service (BBS) provides the **ttf2rbf** conversion utility for this purpose. The converted binary image can be stored on a mass storage device. The intelligent host can then read data from the binary file and load it into the FLEX 8000 device. You can also use the intelligent host to perform real-time conversion during configuration. In the PPA and PPS configuration schemes, the FLEX 8000 device receives its information in parallel from the data bus, a data port on the CPU, or some other byte-wide channel. In the bit-wide PS configuration scheme, the data is shifted in serially.

Programming a Configuration EPROM

You can program Altera Configuration EPROMs with MAX+PLUS II, the PL-MPU Master Programming Unit, and the appropriate Configuration EPROM programming adapter. The PLMJ1213 adapter programs Configuration EPROMs in 8-pin plastic dual in-line packages (PDIP) and 20-pin plastic J-lead chip carrier (PLCC) packages; the PLMT1064 adapter programs Configuration EPROMs in 32-pin thin quad flat pack (TQFP) adapters.

To program an Altera Configuration EPROM:

1. Choose the **Programmer** command (MAX+PLUS II menu) to open the Programmer window.
2. By default, the Programmer loads the POF for the current project. If necessary, load a different POF with the **Select Programming File** command (File menu). The appropriate device for the current programming file is displayed in the Device field.
3. Insert a blank Configuration EPROM into the 8-pin DIP, 20-pin J-lead, or 32-pin QFP socket on the programming adapter. The socket for the FLEX 8000 device (if any) must be empty.
4. Choose the **Program** button.

After successful programming, you can place the Configuration EPROM on the target board to configure a FLEX 8000 device in the AS configuration scheme.

Configuring a FLEX 8000 Device In-System with MAX+PLUS II & the FLEX Download Cable

To configure a FLEX 8000 device with the FLEX Download Cable:

1. Connect the FLEX Download Cable to the 9-pin D-type connector on a Configuration EPROM programming adapter.
2. Connect the other end of the FLEX Download Cable to the 10-pin male header on the target board.
3. Start MAX+PLUS II and choose the **Programmer** command (MAX+PLUS II menu) to open the Programmer window.
4. Choose the **Select Programming File** command (File menu).
5. Select the desired SOF filename in the *Files* box or type a name in the *File Name* box. If you choose a programming file from another project, you are asked if you wish to change the current project name.
6. Choose **OK**.
7. Choose the **Program** button to configure the device.

After the device is configured and initialized, it enters user mode and operates as a logic device. The FLEX Download Cable is electrically removed from the circuit and does not influence circuit operation. You can also physically disconnect the FLEX Download Cable without disturbing the FLEX 8000 configuration data or device operation.

Combining & Converting Programming Files

MAX+PLUS II automatically generates a POF and an SOF for every FLEX 8000 device in a project, as described earlier in this application note. The POF can be programmed into an Altera Configuration EPROM used in an AS configuration scheme; by default, each FLEX 8000 device has one dedicated Configuration EPROM (two Configuration EPROMs are required for an EPF81500 device).

You may wish to combine and/or convert the automatically generated SOFs into a different format for the following purposes:

- ❑ To use a configuration scheme other than AS. You must convert an SOF into a Hex File or a TTF for programming a parallel EPROM, BIOS EPROM, or another data source.
- ❑ To combine multiple sets of configuration data to be used for in-circuit reconfiguration in any configuration scheme.

To convert an SOF into a Hex File or TTF:

1. Refer to Table 2 to calculate the required data space in a parallel or serial data source.
2. Choose the **Combine Programming Files** command (File menu) in the MAX+PLUS II Programmer or Compiler.
3. Select the desired SOF name in the *Files* box or type a name in the *File Name* box under *Input Files*. Choose the **Add** button to add it to the *Selected Files* box.
4. Specify information for the desired configuration scheme:

   – If the FLEX 8000 device will be configured with a parallel EPROM in the APU or APD configuration scheme, select *.hex (Single-Device)* in the *File Format* drop-down list box under *Output File*.

     In addition, if the FLEX 8000 configuration data will not start at the origin of the EPROM, specify the base address for the configuration data in the *Address* box under *Input Files*. Choose *Up* or *Down* under *Count* to specify whether the FLEX 8000 device should count up or down. The counting sequence can be either ascending (00000H to 3FFFFH) for APU configuration or descending (3FFFFH to 00000H) for APD configuration, as described in "Hexadecimal (Intel-Format) File" earlier in this application note.

   or:

   – If the FLEX 8000 device will be configured with a PPA, PPS, or bit-wide PS scheme, select *.ttf (Sequential)* in the *File Format* drop-down list box under *Output File*. The TTF can be incorporated as source code for a data structure in a high-level programming language. Otherwise, the TTF data must be converted into its equivalent 8-bit binary format before it is loaded into the FLEX 8000 device, as described in "Tabular Text File" earlier in this application note.

5. The default name for the output file is the current project name plus the extension **.hex** or **.ttf**. To give a different name to the file, type a name in the *File Name* box under *Output File*.
6. Choose **OK** to generate the Hex File or TTF. The file is placed in the current project directory.

You can use in-circuit reconfiguration to load multiple sets of configuration data into the FLEX 8000 device in a system. The following procedure describes how to combine SOFs for in-circuit reconfiguration of a FLEX 8000 device.

To combine SOFs for in-circuit reconfiguration with multiple sets of configuration data:

1. Refer to Table 2 to calculate the required data space in a parallel or serial data source.
2. Choose the **Combine Programming Files** command (File menu) in the MAX+PLUS II Programmer or Compiler.
3. Select the SOF with the first set of configuration data and choose the **Add** button to add it to the *Selected Files* box.
4. Repeat step 3 until all SOFs have been added to the *Selected Files* box.
5. Arrange the selected files in the order in which the different sets of configuration data will be used by selecting each SOF filename and choosing the **Up** or **Down** button under *Order*.
6. Specify information for the desired configuration scheme, select the output filename, and choose **OK**. The default name for the output file is the current project name plus the extension **.hex**, **.ttf**, or **.pof**. To give a different name to the file, type a name in the *File Name* box under *Output File*. If multiple POFs are generated, they are uniquely identified by a sequence number appended to the filename (e.g., the first is **device.pof**, the second is **device1.pof**, etc.). You can specify an output filename that has less than the maximum of eight characters to leave room for the numerical index; otherwise, the last character(s) are truncated to include it.

Configuration Reliability

The FLEX architecture has been designed to minimize the effects of power supply and data noise in a system, and to ensure that the configuration data is not corrupted during configuration or normal user-mode operation. A number of circuit design features are provided to ensure the highest possible level of reliability from this SRAM technology.

Cyclic redundancy check (CRC) circuitry is used to validate every data frame (i.e., sequence of data bits) as it is loaded into the FLEX 8000 device. If the CRC generated by the FLEX 8000 device does not match the data stored in the data stream, the configuration process is halted, and the `nSTATUS` pin is pulled and held low to indicate an error condition. This

CRC circuitry ensures that noisy systems will not cause errors that yield an incorrect or incomplete configuration.

The FLEX architecture also provides a very high level of reliability in low-voltage brown-out conditions. The SRAM cells require a certain V_{CC} level to maintain accurate data. Since this voltage threshold is significantly lower than that required to activate the power-on reset (POR) circuitry in the FLEX 8000 device, the FLEX 8000 device stops operating if the V_{CC} starts to fail, and indicates an operation error by pulling and holding the `nSTATUS` pin low. The device must then be reconfigured before it can resume operation as a logic device. In active configuration schemes, reconfiguration begins as soon as V_{CC} returns to an acceptable level if the `nCONFIG` pin is tied to V_{CC}. Otherwise, the host system must start the reconfiguration process.

These device features ensure that FLEX 8000 devices have the highest possible reliability in a wide variety of environments, and provide the same high level of system reliability that exists in other families of Altera programmable logic devices.

Configuring Multiple FLEX 8000 Devices

May 1994, ver. 2 Application Note 38

Introduction

The architecture of Altera's Flexible Logic Element MatriX (FLEX) devices supports several methods for configuring multiple FLEX 8000 devices in a single system. You can configure FLEX 8000 devices either individually or together in a parallel or serial fashion.

This application note describes how to create configuration circuits for multiple FLEX 8000 devices. It provides sample schematics, required configuration option bit and configuration pin settings, programming file information, and, where appropriate, timing information. The following topics are discussed:

- ❑ Choosing a configuration circuit
- ❑ Common features in multi-device configuration circuits
- ❑ Multi-Device Sequential Active Serial (MD-SAS) configuration
- ❑ Multi-Device Active Serial Bit-Slice (MD-ASB) configuration
- ❑ Multi-Device Passive Serial Bit-Slice (MD-PSB) configuration
- ❑ Multi-Device Passive Parallel Synchronous (MD-PPS) configuration
- ❑ Multi-Device Passive Parallel Asynchronous (MD-PPA) configuration
- ❑ Multi-Device Active Parallel Hybrid (MD-APH) configuration

This application note must be used together with *Application Note 33 (Configuring FLEX 8000 Devices)*, which provides detailed information on FLEX 8000 device operating modes, data-space sizes, in-circuit reconfiguration, configuration option bits, configuration pins, programming file generation, and single-device configuration. Refer also to the *FLEX 8000 Programmable Logic Device Family* and *Configuration EPROMs for FLEX 8000 Devices* data sheets for additional details on device architecture.

Choosing a Configuration Circuit

The best type of configuration for a particular system depends on a variety of factors, including the existing resources in the system, the number of devices to be configured, the desired configuration time, reconfiguration requirements, and the need to periodically load new configuration data. Table 1 summarizes the characteristics of the multi-device configuration circuits supported by the FLEX 8000 architecture.

Table 1. FLEX 8000 Configuration Schemes

Configuration Circuit	Intelligent Host Required	Auto-Reconfiguration Available	Concurrent Device Configuration	Simultaneous Device Initialization	Configuration Data Location	Max. Devices Configured	Programming File(s)
Multi-Device Sequential Active Serial (MD-SAS)	No	No	No	No	Configuration EPROM(s)	Unlimited	Programmer Object File (**.pof**)
Multi-Device Active Serial Bit-Slice (MD-ASB)	No	Yes	Yes	Yes	Parallel EPROM	8	Hexadecimal (Intel-format) File (**.hex**)
Multi-Device Passive Serial Bit-Slice (MD-PSB)	Yes	No	Yes	Yes	Data file(s)	8 per data file	Tabular Text File (**.ttf**)
Multi-Device Passive Parallel Synchronous (MD-PPS)	Yes	No	Yes	Yes	Data files	Unlimited *Note (1)*	Tabular Text File (**.ttf**)
Multi-Device Passive Parallel Asynchronous (MD-PPA)	Yes	No	Yes	Yes	Data files	Unlimited *Note (2)*	Tabular Text File (**.ttf**)
Multi-Device Active Parallel Hybrid (MD-APH)	No	No	No	No	Parallel EPROM	9	Hexadecimal (Intel-format) File (**.hex**)

Notes:
(1) One FLEX 8000 device can be configured for each unique DCLK signal generated by an intelligent host.
(2) One FLEX 8000 device can be configured for each uniquely decodable address.

This application note describes each type of configuration circuit in detail, the configuration scheme used for each device, the connections between devices, and how to generate the configuration data. The term *configuration scheme* refers to the bit pattern of the nSP, mSEL1, and mSEL0 selection bits—and the attendant behavior—of a single, specific FLEX 8000 device. In contrast, the term *configuration circuit* refers to a set of multiple FLEX 8000 devices, the configuration schemes used for each FLEX 8000 device, and the connections between the devices. In a multi-device system, each FLEX 8000 device in the configuration circuit can use a different configuration scheme.

Common Features in Multi-Device Configuration Circuits

Multi-device configuration circuits have several common characteristics. The following features have similar purposes in each configuration circuit:

- ❑ Configuration Clock frequency
- ❑ nCONFIG pin
- ❑ nSTATUS pin
- ❑ CONF_DONE pin

For more detailed information on these items, refer to *Application Note 33 (Configuring FLEX 8000 Devices).*

Configuration Clock Frequency

The Clock source for all active configuration schemes is an internal oscillator in the FLEX 8000 device, which typically operates in the range 2 MHz to 6 MHz. In all passive configuration schemes, an external controller guides the device configuration at a maximum frequency of 2 MHz.

nCONFIG Pin

In most configuration circuits, the nCONFIG input pin on a FLEX 8000 device is connected to V_{CC}. At system power-up, this connection directs the device to immediately start configuration (in an active configuration scheme) or to prepare for immediate configuration (in a passive configuration scheme).

If an application requires a delay in the FLEX 8000 device configuration, the nCONFIG pin must be tied to external logic. A high-to-low transition on nCONFIG resets the FLEX 8000 device, and a subsequent low-to-high transition starts the configuration process.

nSTATUS Pin

In most configuration circuits, the bidirectional nSTATUS pin on a FLEX 8000 device is connected to an intelligent host or to external support logic. If an error occurs during device configuration, this pin is pulled and held low.

CONF_DONE Pin

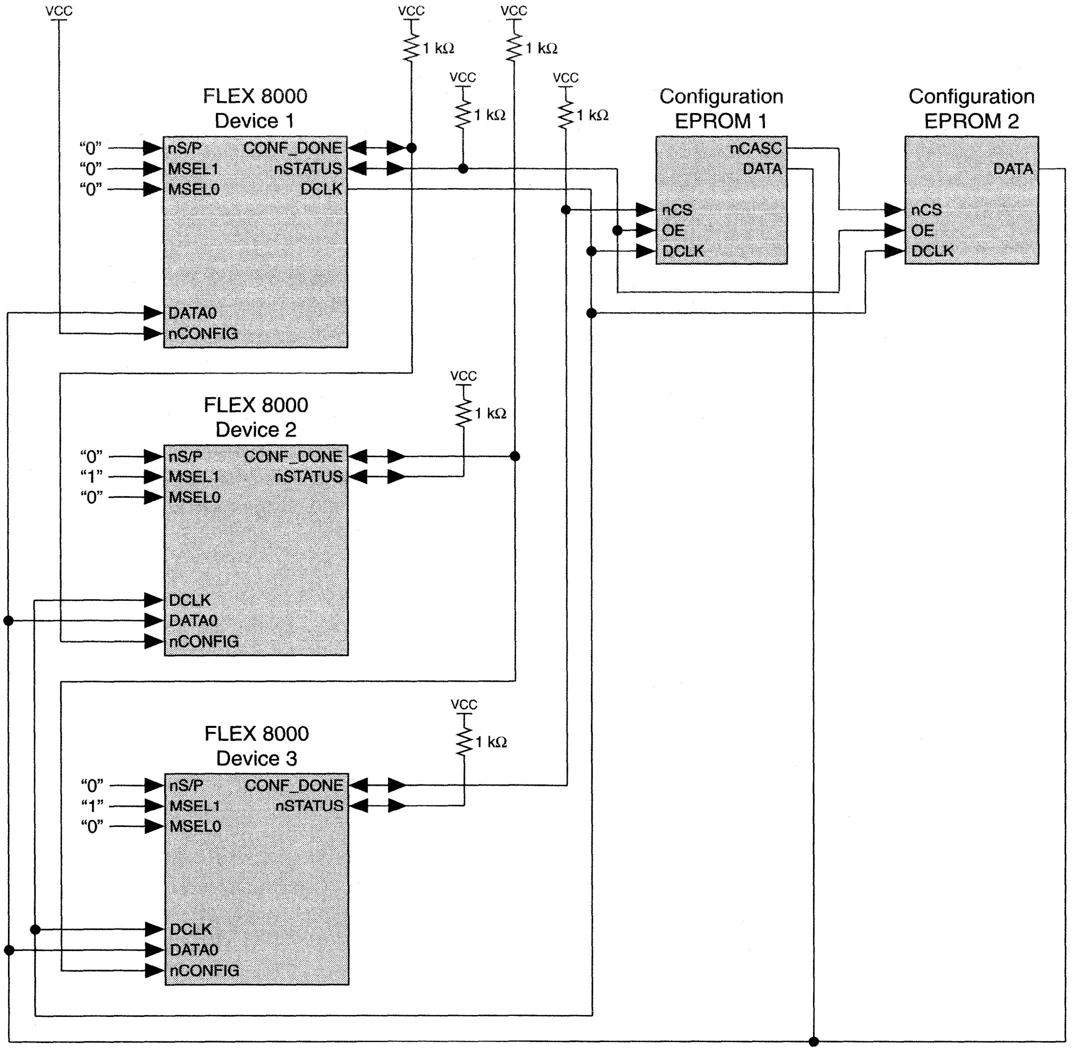
In most configuration circuits, the bidirectional CONF_DONE pins on each FLEX 8000 device are connected to the same net. The FLEX 8000 devices in the circuit hold the CONF_DONE net low until all devices are fully configured, thereby allowing devices of different sizes to be configured and initialized simultaneously. The CONF_DONE net is also connected to the DONE input of the external support logic or an intelligent host to indicate that configuration has been successful.

Multi-Device Sequential Active Serial (MD-SAS) Configuration

In an MD-SAS configuration circuit, the configuration data is stored in one or more Altera serial Configuration EPROMs. The first FLEX 8000 device controls the configuration by generating a DCLK signal that clocks data out from the Configuration EPROMs. The CONF_DONE pin on the first FLEX 8000 device is connected to the nCONFIG pin of the next FLEX 8000 device, and the connection is repeated through the entire configuration circuit. Once the first FLEX 8000 device is fully configured, its CONF_DONE pin is pulled up to V_{CC} via an external pull-up resistor. This low-to-high transition on the nCONFIG input to the next FLEX 8000 device directs it to begin configuration.

Figure 1 shows three FLEX 8000 devices and two Configuration EPROMs in an MD-SAS configuration circuit. By default, each FLEX 8000 device in a project has one dedicated Configuration EPROM. In this example, however, the configuration data for the three FLEX 8000 devices has been combined and programmed into two Configuration EPROMs. In some circuits, you may need more Configuration EPROMs than FLEX 8000 devices to store the configuration data (e.g., three EPC1213 Configuration EPROMs are required to configure two EPF81500 devices). When you combine the programming files for the Configuration EPROMs, the MAX+PLUS II software automatically calculates the minimum number of Configuration EPROMs needed to support a multi-device configuration circuit.

Figure 1. Multi-Device Sequential Active Serial (MD-SAS) Configuration Circuit



The nCS pin on the first Configuration EPROM must be connected to the CONF_DONE output of the last FLEX 8000 device in the circuit to ensure that all Configuration EPROMs are disabled after the last FLEX 8000 device is completely configured. In addition, if the configuration circuit includes more than six devices, the DCLK and DATA0 nets should have external active buffering to maintain the signal integrity. Table 2 summarizes the configuration parameters for MD-SAS configuration circuits.

Table 2. MD-SAS Configuration Parameters

Parameter	Description
Configuration scheme	First FLEX 8000 device: Active Serial (nSP:mSEL1:mSEL0 = 000) Subsequent FLEX 8000 device(s): Passive Serial (nSP:mSEL1:mSEL0 = 010)
Non-default device option & configuration pin settings	For the first FLEX 8000 device only, turn on the *Enable DCLK Output in User Mode* option in the **FLEX 8000 Individual Device Options** dialog box. DCLK is inactive in subsequent FLEX 8000 devices, which use the Passive Serial configuration scheme.
Device configuration/ programming file	Configuration data is stored in one or more POFs, depending on the number of Configuration EPROMs required to configure the FLEX 8000 devices. POFs are generated by combining the SRAM Object Files (**.sof**) from all FLEX 8000 devices in the serial order in which they are configured on the board. Select *.pof (Sequential)* in the *File Format* drop-down list box in the **Combine Programming Files** dialog box when generating POFs for MD-SAS configuration.
Reconfiguration on error	No automatic reconfiguration is available. The nSTATUS pin is connected to V_{CC} separately on each FLEX 8000 device. Each of the nSTATUS nets must be monitored for a high-to-low transition, which indicates that an error has occurred during configuration. The nCONFIG pin on the first FLEX 8000 device must be pulled low and then released to initiate a reconfiguration cycle.

Multi-Device Active Serial Bit-Slice (MD-ASB) Configuration

In an MD-ASB configuration circuit, the configuration data is stored in a parallel EPROM. The EPROM must have a maximum access time of 100 ns. Each bit in the EPROM data word (up to 8 bits wide) configures a different FLEX 8000 device. Data in the EPROM is presented as parallel streams of serial configuration data. A standard byte-wide EPROM can configure up to eight FLEX 8000 devices simultaneously, with each data pin in the EPROM data word connected to the DATA0 pin of the corresponding FLEX 8000 device in the configuration circuit.

Figure 2 shows an MD-ASB circuit in which two FLEX 8000 devices are configured with a parallel EPROM. A support PLD such as the EPM7032 device translates the DCLK signals generated by the first FLEX 8000 device into sequential addresses for the parallel EPROM. This support device must contain an 18-bit counter and other logic to translate nSTATUS into a global Reset signal.

Figure 2. Multi-Device Active Serial Bit-Slice (MD-ASB) Configuration Circuit

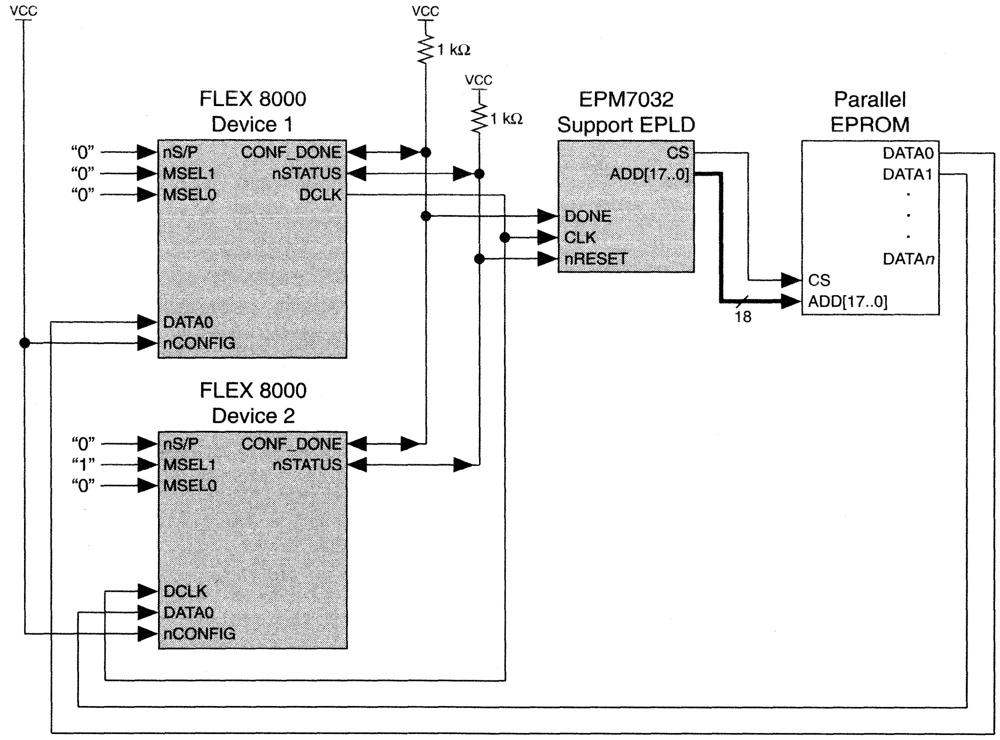


Figure 3 shows an Altera Hardware Description Language (AHDL) Text Design File (**.tdf**) that implements the features required in an EPM7032 support device.

Figure 3. AHDL Text Design File for EPM7032 Support Device (asbpld.tdf)

```
DESIGN IS asbpld
   DEVICE IS EPM7032LC44;

SUBDESIGN asbpld
(
   clk, done, nreset : INPUT;
   cs, add[17..0]    : OUTPUT;
)

VARIABLE
   count[17..0]      : DFF;
   atri[17..0]       : TRI;

BEGIN
   add[]        = atri[];
   atri[]       = count[];
   atri[].oe    = global(!done);
   cs           = !done;

   count[].clk  = global(clk);
   count[].clrn = global(nreset);
   count[].d    = count[].q + 1;

END;
```

Table 3 summarizes the configuration parameters for MD-ASB configuration circuits.

Table 3. MD-ASB Configuration Parameters

Parameter	Description
Configuration scheme	First FLEX 8000 device: Active Serial (nSP:mSEL1:mSEL0 = 000) Subsequent FLEX 8000 device(s): Passive Serial (nSP:mSEL1:mSEL0 = 010)
Non-default device option & configuration pin settings	For all FLEX 8000 devices, turn on the *Disable Start-Up Time-Out* option in the **FLEX 8000 Device Options** dialog box. For the first FLEX 8000 device, turn on the *Auto-Restart Configuration on Frame Error* option in the **FLEX 8000 Individual Device Options** dialog box.
Device configuration/ programming file	Configuration data is stored in a single Hex File, generated by combining the SOFs from all FLEX 8000 devices in the parallel order in which they are configured on the board. Select *.hex (Bit-Slice)* in the *File Format* drop-down list box in the **Combine Programming Files** dialog box when generating Hex Files for MD-ASB configuration. The first file listed in the **Combine Programming Files** dialog box corresponds to DATA0 on the EPROM, the second corresponds to DATA1, and so on.
Reconfiguration on error	The circuit in Figure 2 supports automatic reconfiguration on error. A FLEX 8000 device drives a high-low-high pulse on the nSTATUS signal whenever a configuration error (e.g., bad data) or an operation error (e.g., V_{CC} failure) occurs. This pulse resets the counter in the EPM7032 support PLD and restarts the configuration process.

Multi-Device Passive Serial Bit-Slice (MD-PSB) Configuration

In the MD-PSB configuration circuit, the configuration data is typically stored in a data file and presented to the FLEX 8000 devices by an intelligent host. The data in the configuration file incorporates parallel streams of serial configuration data. Each bit in the 8-bit-wide configuration file provides configuration data to the DATA0 pin of a separate FLEX 8000 device in the configuration circuit. After it has presented a data word on the data bus, the intelligent host sends a DCLK pulse to all FLEX 8000 devices, instructing them to latch the data.

Figure 4 shows two FLEX 8000 devices that are configured by an intelligent host in an MD-PSB configuration circuit.

Figure 4. Multi-Device Passive Serial Bit-Slice (MD-PSB) Configuration Circuit

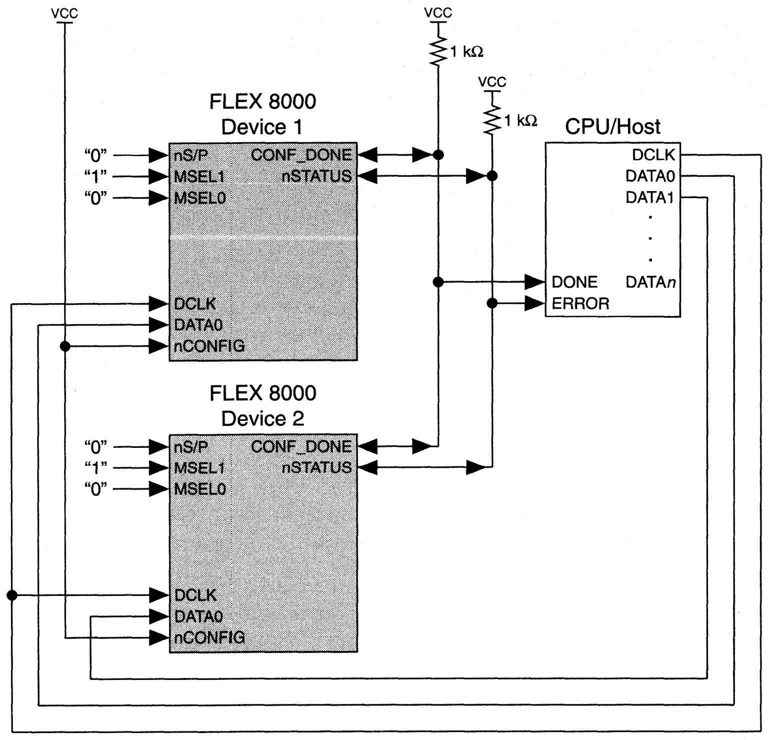


Figure 5 shows the sequence of control signals that the intelligent host must generate to correctly implement the circuit. A single configuration file can provide the data to simultaneously configure up to eight FLEX 8000 devices; multiple files can be used to extend an MD-PSB configuration circuit without limit.

Figure 5. Multi-Device Passive Serial Bit-Slice (MD-PSB) Configuration Waveforms

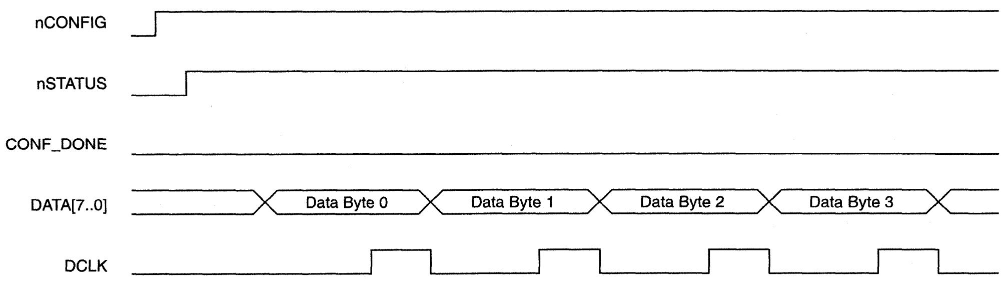


Table 4 summarizes the configuration parameters for MD-PSB configuration circuits.

Table 4. MD-PSB Configuration Parameters

Parameter	Description
Configuration scheme	First FLEX 8000 device: Passive Serial (`nSP:mSEL1:mSEL0 = 010`) Subsequent FLEX 8000 device(s): Passive Serial (`nSP:mSEL1:mSEL0 = 010`)
Non-default device option & configuration pin settings	If the FLEX 8000 device uses `DATA0` during user mode, you must turn off the *Reserve* option for `DATA0` for all FLEX 8000 devices in the **FLEX 8000 Device Options** dialog box. For all FLEX 8000 devices, turn on the *Disable Start-Up Time-Out* option in the **FLEX 8000 Device Options** dialog box.
Device configuration/ programming file	Configuration data is stored in a single TTF, generated by combining the SOFs from all FLEX 8000 devices in the *parallel* order in which they are configured on the board. The first file listed corresponds to the least significant bit (LSB) of the TTF. Select *.ttf (Bit-Slice)* from the *File Format* drop-down list box in the **Combine Programming Files** dialog box when generating TTFs for MS-PSB configuration. A TTF can contain up to eight parallel configuration bit-streams; the data in the file must be converted from ASCII to binary format before being presented to the FLEX 8000 devices during configuration. The Altera Applications BBS provides the **ttf2rbf** conversion utility for this purpose.
Reconfiguration on error	No automatic reconfiguration is available. The circuit shown in Figure 4 shows an input to the intelligent host called `ERROR`, which must be monitored for a high-to-low transition on the `nSTATUS` signal. This transition indicates an error during configuration or user-mode operation. The intelligent host must respond by pulling `nCONFIG` low to initiate a reconfiguration cycle, then releasing it.

Multi-Device Passive Parallel Synchronous (MD-PPS) Configuration

In an MD-PPS configuration circuit, the configuration data is typically stored in data files on a hard disk. An intelligent host presents the data to the FLEX 8000 devices in a parallel format on an 8-bit-wide data bus. Each FLEX 8000 device in the circuit can be configured sequentially, so that each successive device is completely configured before the next device starts configuration. Alternatively, the configuration can be interleaved, with each FLEX 8000 device receiving one data byte in rotation. Each FLEX 8000 device requires a separate `DCLK` control input from the intelligent host, and must be clocked eight times for each byte at a frequency up to 2 MHz.

Figure 6 shows an MD-PPS configuration circuit in which an intelligent host configures two FLEX 8000 devices. This configuration circuit can be extended to include one FLEX 8000 device for each unique `DCLK` signal generated by the intelligent host.

Figure 6. Multi-Device Passive Parallel Synchronous (MD-PPS) Configuration Circuit

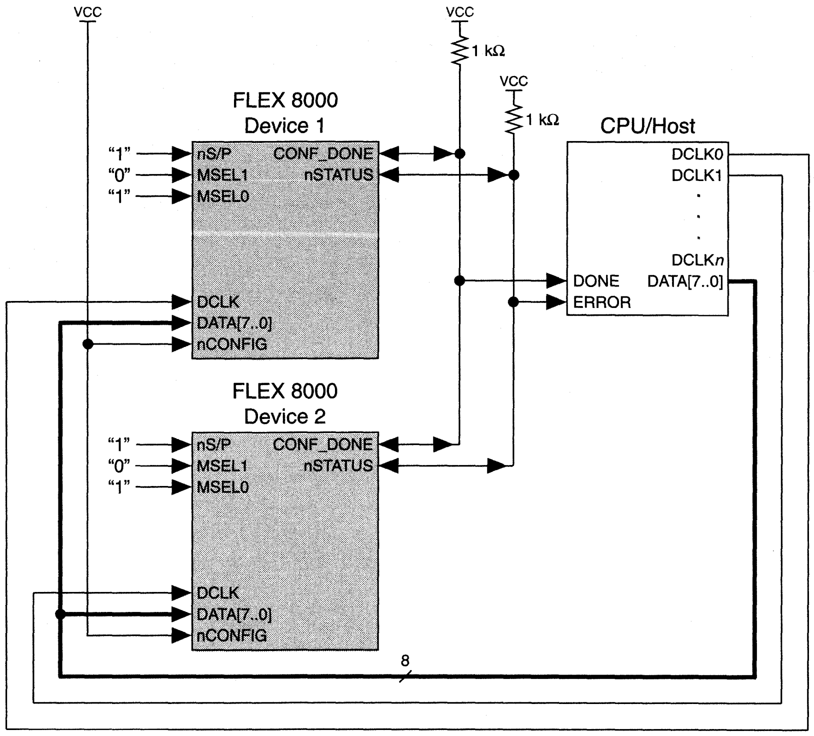
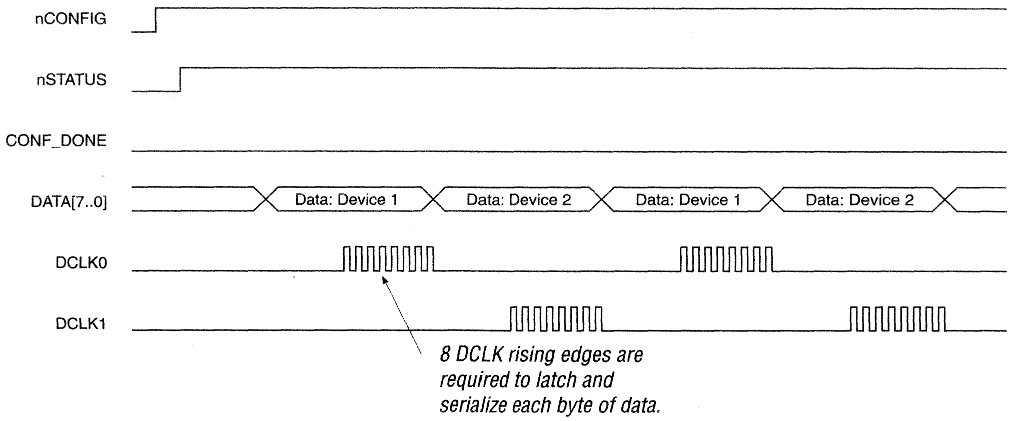


Figure 7 shows the sequence of control signals necessary for both interleaved and non-interleaved MD-PPS configuration.

Figure 7. Multi-Device Passive Parallel Synchronous (MD-PPS) Configuration Waveforms

Interleaved Configuration



Non-Interleaved Configuration

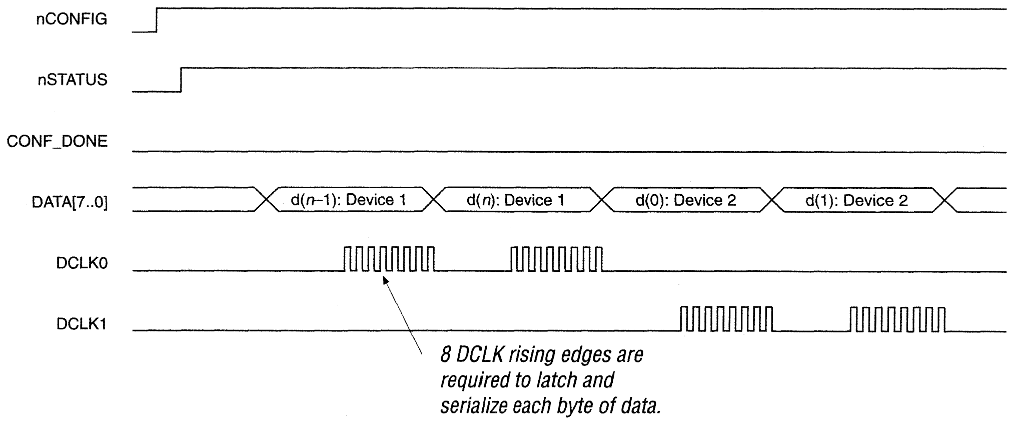


Table 5 summarizes the configuration parameters for MD-PPS configuration circuits.

Table 5. MD-PPS Configuration Parameters

Parameter	Description
Configuration scheme	First FLEX 8000 device: Passive Parallel Synchronous (`nSP:mSEL1:mSEL0 = 101`) Subsequent FLEX 8000 device(s): Passive Parallel Synchronous (`nSP:mSEL1:mSEL0 = 101`)
Non-default device option & configuration pin settings	If the FLEX 8000 device uses the data bus during user mode, you must turn off the *Reserve* option for `DATA[0..7]` for all FLEX 8000 devices in the **FLEX 8000 Device Options** dialog box. For all FLEX 8000 devices, turn on the *Disable Start-Up Time-Out* option in the **FLEX 8000 Device Options** dialog box.
Device configuration/ programming file	Configuration data is stored in a separate TTF for each device. Select *.ttf (Sequential)* from the *File Format* drop-down list box in the **Combine Programming Files** dialog box when generating TTFs for MD-PPS configuration. The TTF data must be converted from ASCII to binary format before being presented to the FLEX 8000 devices during configuration. The Altera Applications BBS provides the **ttf2rbf** conversion utility for this purpose.
Reconfiguration on error	No automatic reconfiguration is available. The circuit in Figure 6 shows an input to the intelligent host called `ERROR`, which must be monitored for a high-to-low transition on the `nSTATUS` signal. This transition indicates an error during configuration or user-mode operation. The intelligent host must respond by pulling `nCONFIG` low to initiate a reconfiguration cycle, then releasing it.

Multi-Device Passive Parallel Asynchronous (MD-PPA) Configuration

In the MD-PPA configuration circuit, the configuration data is typically stored in data files on hard disk. An intelligent host presents the data to the FLEX 8000 devices in a parallel format on an 8-bit-wide data bus. Each FLEX 8000 device in the circuit can be configured sequentially, so that each successive device is completely configured before the next device starts configuration. Alternatively, the configuration can be interleaved, with each FLEX 8000 device receiving one data byte in rotation. If the data bus is very fast, you may wish to use the interleaving method to take advantage of the FLEX 8000 device's 4-µs (250-kHz) minimum configuration time per byte. Otherwise, sequential configuration may be more appropriate.

Each FLEX 8000 device is uniquely addressed by a decoder PLD. When the intelligent host is ready to present a data byte to a FLEX 8000 device, the host generates the corresponding address and the decoder PLD selects the correct FLEX 8000 device using the `nCS` pin. The intelligent host then provides a high-low-high pulse on `nWS`, which directs the selected FLEX 8000 device to latch the data. A high-low-high pulse on `nRS` directs the addressed FLEX 8000 device to present the `RDYnBUSY` signal on the `DATA7` pin, which must be monitored to determine when the FLEX 8000 device is ready to receive another byte of data. The `DATA7` pin on the intelligent host must be tri-stated during the monitoring process.

Figure 8 shows two FLEX 8000 devices, an intelligent host, and a decoder PLD in an MD-PPA configuration circuit. This configuration circuit can be extended to include one FLEX 8000 device for each uniquely decodable address, with no upper limit to the number of devices.

Figure 8. Multi-Device Passive Parallel Asynchronous (MD-PPA) Configuration Circuit

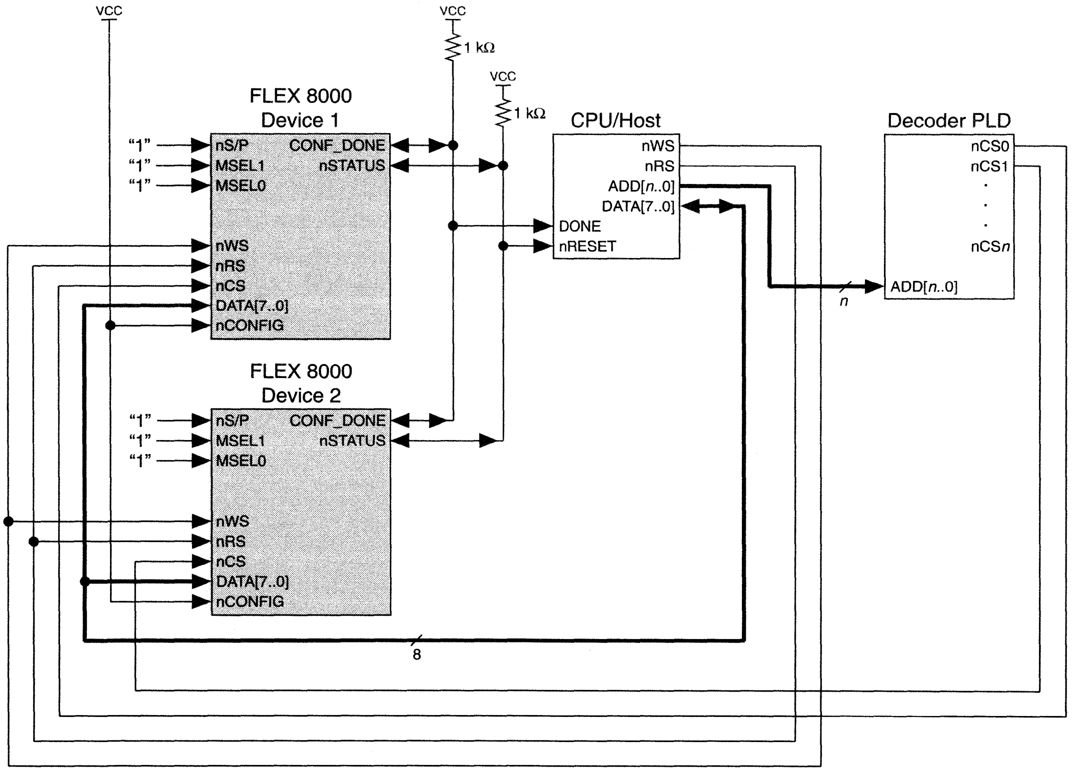


Figure 9 shows the sequence of control signals necessary for a non-interleaved MD-PPA configuration circuit that uses the `DATA7` pin for status-checking.

Figure 9. Multi-Device Passive Parallel Asynchronous (MD-PPA) Configuration Waveforms

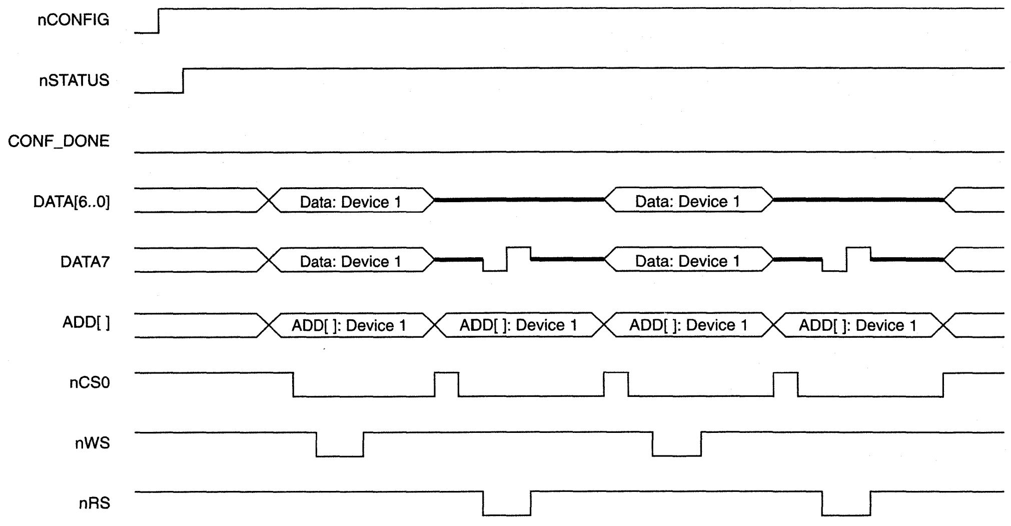


Table 6 summarizes the configuration parameters for MD-PPA configuration circuits.

Table 6. MD-PPA Configuration Parameters

Parameter	Description
Configuration scheme	First FLEX 8000 device: Passive Parallel Asynchronous (`nSP:mSEL1:mSEL0 = 111`) Subsequent FLEX 8000 device(s): Passive Parallel Asynchronous (`nSP:mSEL1:mSEL0 = 111`)
Non-default device option & configuration pin settings	If the FLEX 8000 device uses the data bus during user mode, you must turn off the *Reserve* option for `DATA[0..7]` for all FLEX 8000 devices in the **FLEX 8000 Device Options** dialog box. For all FLEX 8000 devices, turn on the *Disable Start-Up Time-Out* option in the **FLEX 8000 Device Options** dialog box.
Device configuration/ programming file	Configuration data is stored in a TTF for each device. Select *.ttf (Sequential)* from the *File Format* drop-down list box in the **Combine Programming Files** dialog box when generating TTFs for MD-PPA configuration. The TTF data must be converted from ASCII to binary format before being presented to the FLEX 8000 devices during configuration. The Altera Applications BBS provides the **ttf2rbf** conversion utility for this purpose.
Reconfiguration on error	No automatic reconfiguration is available. The circuit in Figure 8 shows an input to the intelligent host called `ERROR`, which must be monitored for a high-to-low transition on the `nSTATUS` signal. This transition indicates an error during configuration or user-mode operation. The intelligent host must respond by pulling `nCONFIG` low to initiate a reconfiguration cycle, then releasing it.

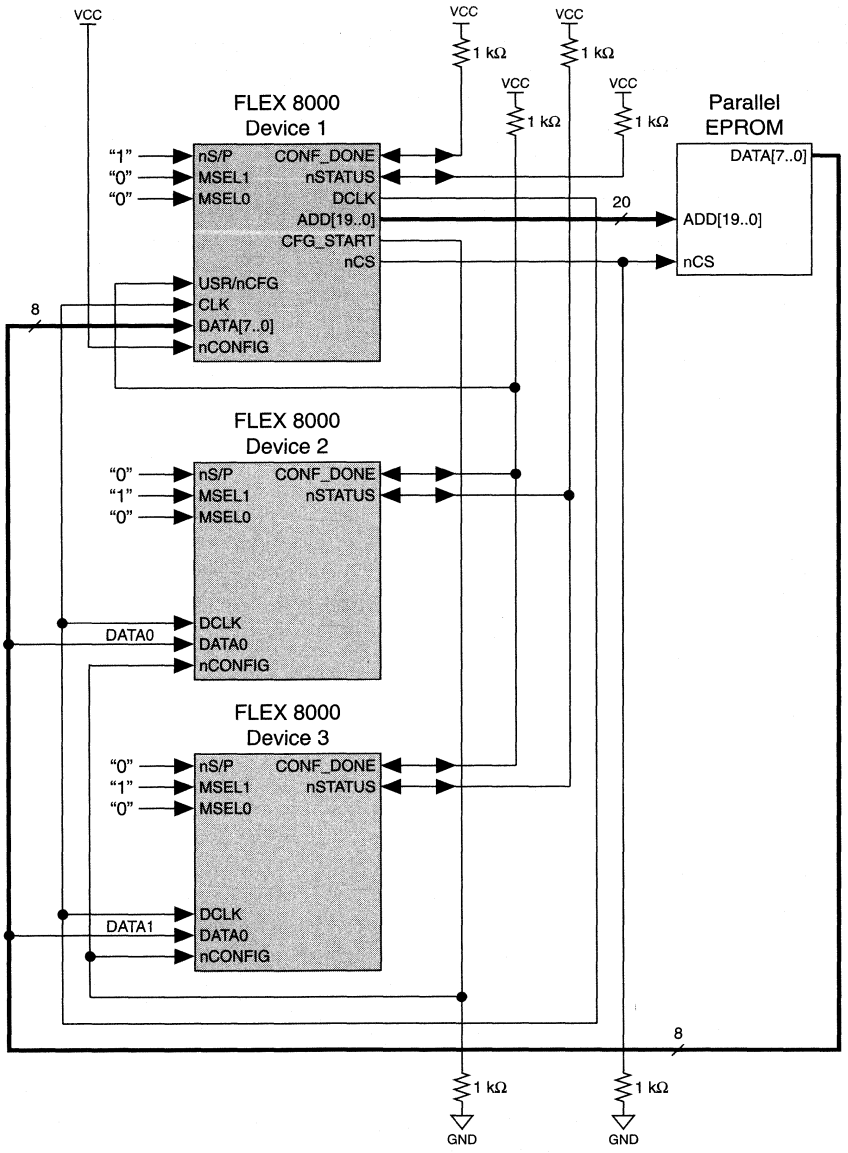
Multi-Device Active Parallel Hybrid (MD-APH) Configuration

In an MD-APH configuration circuit, two configuration data files are stored in a parallel EPROM. The EPROM must have a maximum access time of 100 ns. The first file is used to configure the first FLEX 8000 device in an active parallel up (APU) configuration scheme. The second file consists of serial bit-slice data that can configure up to eight additional FLEX 8000 devices in a passive serial (PS) configuration scheme.

The design file for the first (actively configured) FLEX 8000 device must contain a 20-bit counter and support logic for passively configuring additional FLEX 8000 devices. This logic emulates the address generation used in a single-device APU configuration. The bit-slice data is presented to the passively configured FLEX 8000 devices as parallel streams of serial configuration data. Each bit in the configuration data word (up to eight bits wide) configures a separate FLEX 8000 device. The MD-APH configuration support logic for the first FLEX 8000 device is available from the Altera Applications bulletin board service (BBS) in the self-extracting file **md_aph.exe**.

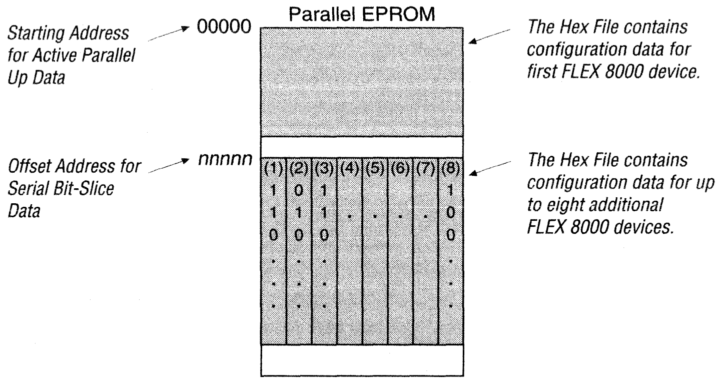
Figure 10 shows an MD-APH circuit in which a parallel EPROM configures three FLEX 8000 devices.

Figure 10. Multi-Device Active Parallel Hybrid (MD-APH) Configuration Circuit



A byte-wide 256-Kbyte EPROM can configure up to nine EPF81188 FLEX 8000 devices. The first 32 Kbytes store the APU data for the first FLEX 8000 device; the next 192 Kbytes contain the bit-slice configuration data for the passively configured devices. See Figure 11. EPROMs of

Figure 11. Multi-Device Active Parallel Hybrid (MD-APH) Data Storage


Parallel EPROM
Starting Address for Active Parallel Up Data
00000
Offset Address for Serial Bit-Slice Data
nnnnn
The Hex File contains configuration data for first FLEX 8000 device.
The Hex File contains configuration data for up to eight additional FLEX 8000 devices.
(1) (2) (3) (4) (5) (6) (7) (8)

different sizes can accommodate configuration data for FLEX 8000 devices of different sizes.

The configuration support logic in the first FLEX 8000 device drives the outputs ADD[19..0], CFG_START, and nCS; it uses the inputs CLK and USR/nCFG. The address pins consist of the dual-purpose configuration pins ADD[17..0] and two I/O pins ADD[19..18]. After all passively configured devices are fully configured and have entered user-mode operation, multiplexers in the support logic release the ADD[19..0] address pins on the first FLEX 8000 device for use as normal I/O pins. The CFG_STRT signal synchronizes device configuration by driving the low-high-low pulse on the nCONFIG inputs to the passively configured FLEX 8000 devices.

The CLK input to the first FLEX 8000 device is tied to its DCLK output so that the DCLK signal, which is not available internally, can drive the support logic. The USR/nCFG input on the first FLEX 8000 device is tied to the CONF_DONE net of all passively configured FLEX 8000 devices. Once these devices are fully configured and have released CONF_DONE, the high logic level on the USR/nCFG input to the first FLEX 8000 device turns off the address counter, releasing the ADD[19..0] address pins for use as I/O pins during user-mode operation. This high logic level also causes the first FLEX 8000 device to assert a high logic level on nCS, which disables the EPROM, releases the DATA[7..0] pins on the FLEX 8000 device, and latches CFG_STRT at V_{CC} to prevent erroneous reconfiguration.

The first FLEX 8000 device must enter user mode before the passively configured FLEX 8000 devices so that its support logic can direct their configuration. Therefore, the CONF_DONE signal on the first FLEX 8000 device is not tied to the CONF_DONE net of the other FLEX 8000 devices.

Table 7 summarizes the configuration parameters for MD-APH configuration circuits.

Table 7. MD-APH Configuration Parameters

Parameter	Description
Configuration scheme	First FLEX 8000 device: Active Parallel Up (`nSP:mSEL1:mSEL0 = 100`) Subsequent FLEX 8000 device(s): Passive Serial (`nSP:mSEL1:mSEL0 = 010`)
Non-default device option & configuration pin settings	For the first FLEX 8000 device only, turn on the *Enable DCLK Output in User Mode* option in the **FLEX 8000 Individual Device Options** dialog box. For all other FLEX 8000 devices in the circuit, turn on the *Disable Start-Up Time-Out* option in the **FLEX 8000 Individual Device Options** dialog box.
Device configuration/ programming file	Configuration data is stored in two Hex Files. One Hex File is used for the first FLEX 8000 device, with an offset address of `00000`. Select *.hex (Sequential)* in the *File Format* drop-down list box in the **Combine Programming Files** dialog box when generating the Hex File for the first FLEX 8000 device. The other Hex File contains bit-slice data, generated by combining the SOFs for all passively configured FLEX 8000 devices in the parallel order in which they are configured on the board. Select *.hex (Bit-Slice)* in the *File Format* drop-down list box in the **Combine Programming Files** dialog box when generating Hex Files for the passively configured devices. The first file in the *Selected Files* list in the **Combine Programming Files** dialog box corresponds to `DATA0` on the parallel EPROM (i.e., the first bit of the bit-slice data for the passively configured devices). You must enter a starting address value for this file in the *Address* box that is after the end of the Hex File for the actively configured (first) FLEX 8000 device.
Reconfiguration on error	No automatic reconfiguration is available. The `nSTATUS` pin on the first FLEX 8000 device is connected to V_{CC} separately from the `nSTATUS` pins of the passively configured FLEX 8000 devices. Each `nSTATUS` net must be monitored for a high-to-low transition, which indicates that an error has occurred during configuration. The `nCONFIG` pin on the first FLEX 8000 device must be pulled low and then released to initiate a reconfiguration cycle.

ALTERA

JTAG Boundary-Scan Testing in FLEX 8000 Devices

May 1994, ver. 2 | Application Note 39

Introduction

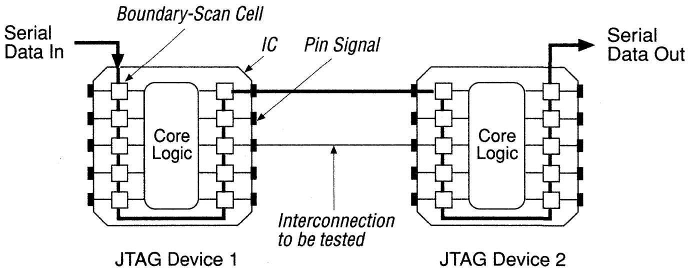
As devices become more complex, the need for thorough testing becomes increasingly important. Advances in surface-mount packaging and circuit board manufacturing have resulted in smaller circuit boards, making traditional test methods—e.g., external test probes and "bed-of-nails" test fixtures—harder to implement. As a result, cost savings from board reductions may be offset by increases in the cost of traditional testing methods.

In the 1980s, the Joint Test Action Group (JTAG) developed the IEEE 1149.1-1990 specification for boundary-scan testing. The Boundary-Scan Test (BST) architecture offers the capability to efficiently test components on circuit boards with tight lead spacing. The EPF81500, EPF8820, EPF8636, EPF8282, and EPF8282V FLEX 8000 devices comply with the IEEE 1149.1-1990 specification by providing BST capability for input pins, output pins, and dedicated configuration pins.

You can use the BST architecture to test pin connections without using physical test probes and to capture functional data while a device is operating normally. Boundary-scan cells in a device can force logic onto pin signals or capture data from pin or core logic signals. Forced test data is serially shifted into the boundary-scan cells and captured data is serially shifted out and externally compared to expected results. Figure 1 illustrates the concept of boundary-scan testing.

Figure 1. JTAG Boundary-Scan Testing

The JTAG methodology provides a serial scan path that can capture the contents of the core logic or test the pin connections between JTAG-compliant devices.



This application note describes how to use the JTAG BST circuitry provided in FLEX 8000 devices. It discusses the following topics:

- ❏ FLEX 8000 JTAG BST architecture
- ❏ JTAG boundary-scan register
- ❏ JTAG BST operation control
- ❏ Enabling JTAG BST circuitry
- ❏ Guidelines for JTAG boundary-scan testing
- ❏ Boundary-Scan Description Language (BSDL) support
- ❏ References
- ❏ JTAG boundary-scan order

FLEX 8000 JTAG BST Architecture

When a device is operating in JTAG BST mode, four I/O pins are used as the `TDI`, `TDO`, `TMS`, and `TCLK` JTAG pins. A dedicated active-low Reset pin, `nTRST`, can asynchronously initialize or reset the JTAG BST circuitry.

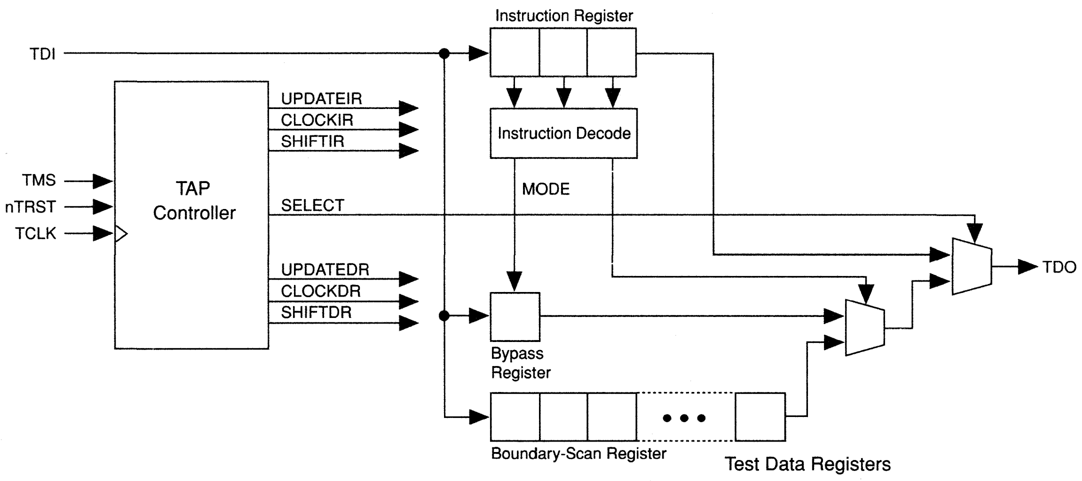
When a FLEX 8000 device is not operating in JTAG BST mode, `nTRST` should be driven low to keep the JTAG circuitry initialized. The `TDI`, `TDO`, and `TCLK` pins should also be held low, and `TMS` should be driven high. In the EPF81500 device, these five pins are dedicated for JTAG BST testing only. In JTAG devices other than the EPF81500, you can use `TDI`, `TDO`, `TMS`, and `TCLK` as normal I/O pins by turning off the JTAG option with MAX+PLUS II (see "Enabling JTAG BST Circuitry" later in this application note). Table 1 summarizes the functions of each of the JTAG BST pins.

Table 1. JTAG Pin Descriptions

Pin	Name	Description
`TDI`	Test data input	Serial input pin for instructions and test data. Data is shifted in on the rising edge of `TCLK`.
`TDO`	Test data output	Serial data output pin for instructions and test data. Data is shifted in on the falling edge of `TCLK`. The signal is tri-stated if data is not being shifted out of the device.
`TMS`	Test mode select	Serial input pin to select the JTAG instruction mode. Data is shifted in on the rising edge of `TCLK`. `TMS` should be driven high during user-mode operation.
`TCLK`	Test Clock input	Clock pin to shift the serial data and instructions in and out of the `TDI` and `TDO` pins, respectively. `TCLK` is also used to shift serial instruction data into the `TMS` pin.
`nTRST`	Test Reset input	Active-low input to asynchronously initialize or reset the boundary-scan circuit.

JTAG boundary-scan testing is controlled by a Test Access Port (TAP) Controller, which is described in "JTAG BST Operation Control" later in this application note. The TAP Controller drives three registers: a 3-bit instruction register that directs the flow of scan test data, a 1-bit bypass data register, and a large boundary-scan data register located on the periphery of the FLEX 8000 device. The boundary-scan registers for the EPF8282 and EPF8282V contain 273 bits; for the EPF8636, 917 bits; for the EPF8820, 465 bits; and for the EPF81500, 645 bits. Figure 2 shows the JTAG register control functions. The `TMS`, `nTRST`, and `TCLK` pins operate the TAP Controller, and the `TDI` and `TDO` pins provide the serial scan path for the test data. The `TDI` pin also provides data to the instruction register, which then generates control logic for the data registers.

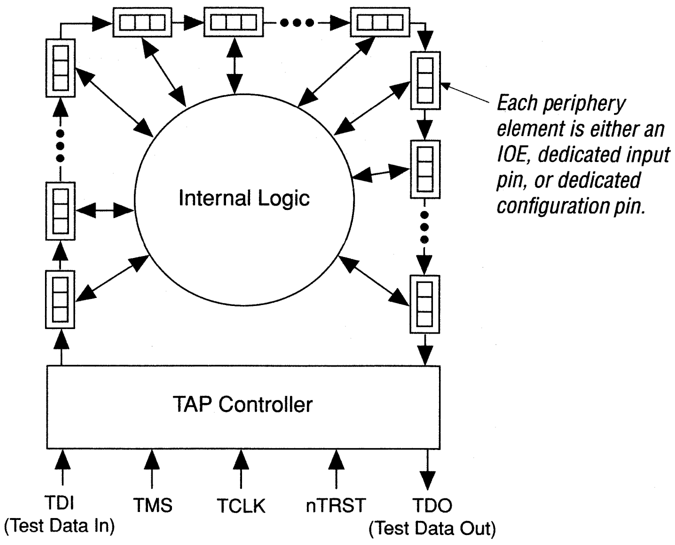
Figure 2. JTAG Register Control



JTAG Boundary-Scan Register

The boundary-scan register is a large serial-shift register that uses the `TDI` pin as an input and the `TDO` pin as an output. Figure 3 shows how test data is serially shifted around the periphery of the FLEX 8000 device. The boundary-scan register consists of 3-bit periphery elements that are either I/O elements (IOEs), dedicated inputs, or dedicated configuration pins. You can use the boundary-scan register to test external pin connections or to capture internal data while the device operates.

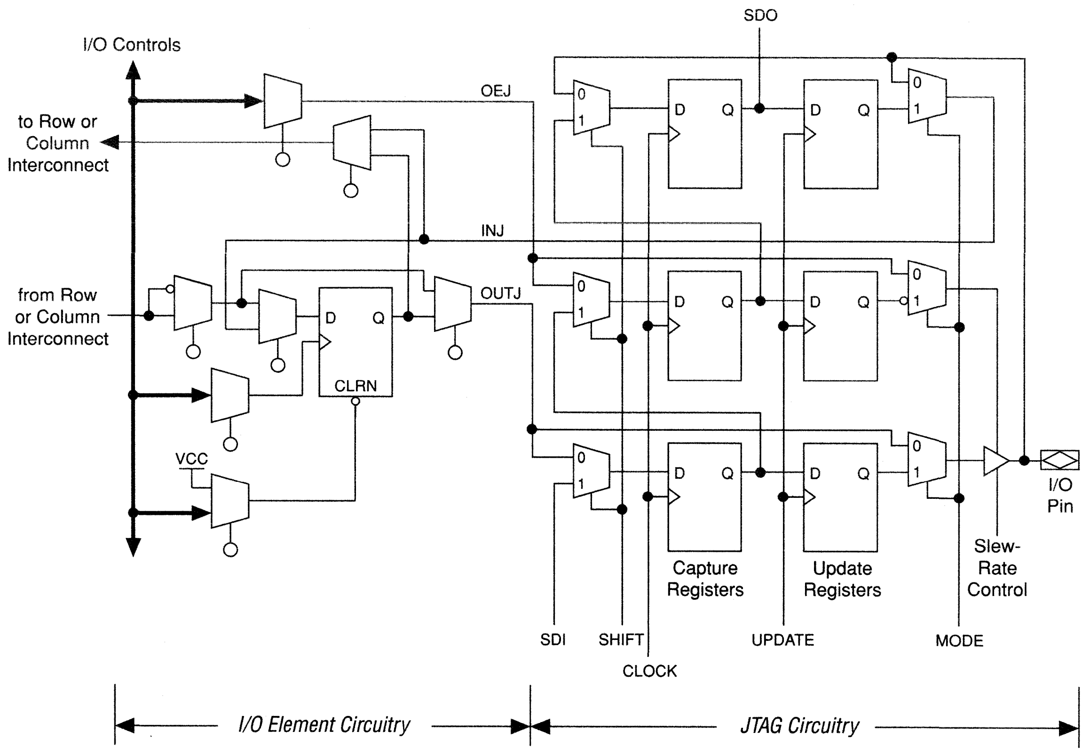
Figure 3. Boundary-Scan Register


Internal Logic
TAP Controller
TDI
(Test Data In)
TMS
TCLK
nTRST
TDO
(Test Data Out)
Each periphery element is either an IOE, dedicated input pin, or dedicated configuration pin.

I/O Elements with JTAG BST Registers

Figure 4 shows the IOE of FLEX 8000 devices with JTAG BST registers. The 3-bit boundary-scan cell consists of a set of capture registers and a set of update registers in each IOE. The capture registers connect to internal device data via the `OUTJ`, `OEJ`, and I/O pin signals, while the update registers connect to external data through the tri-state data input, tri-state control, and `INJ` signals. The control signals for the JTAG BST registers (e.g., `SHIFT`, `CLOCK`, and `UPDATE`) are generated internally by the TAP Controller; the `MODE` signal is generated by an instruction registers decode. The data signal path for the boundary-scan register runs from the Serial Data In (`SDI`) signal to the Serial Data Out (`SDO`) signal. The scan register begins at the `TDI` pin and ends at the `TDO` pin of the device.

Figure 4. I/O Element with JTAG Architecture



Dedicated Input & Configuration Pins with JTAG BST Registers

The boundary-scan register also includes dedicated input pins and dedicated configuration pins. Since these pins have special functions, some bits of the boundary-scan register are internally connected to VCC or GND, or used only for device configuration; these bits are thus forced to a static high (1) or low (0) state, or are used internally for configuration.

Figure 5 shows the JTAG BST registers for the dedicated input pins. The register normally associated with an output signal in an IOE, OUTJ, is tied to GND, and the tri-state control OEJ is connected to VCC. The signal data from the dedicated input is the only register that contains test data. The data shifts out of SDO in the order D, 1, and 0, where the variable D is the data associated with the dedicated input. Since only the D bit has valid data, a scan test pattern must either ignore or expect the 1 and 0 that follow the D bit.

Figure 5. Boundary-Scan Registers on Dedicated Input Pins

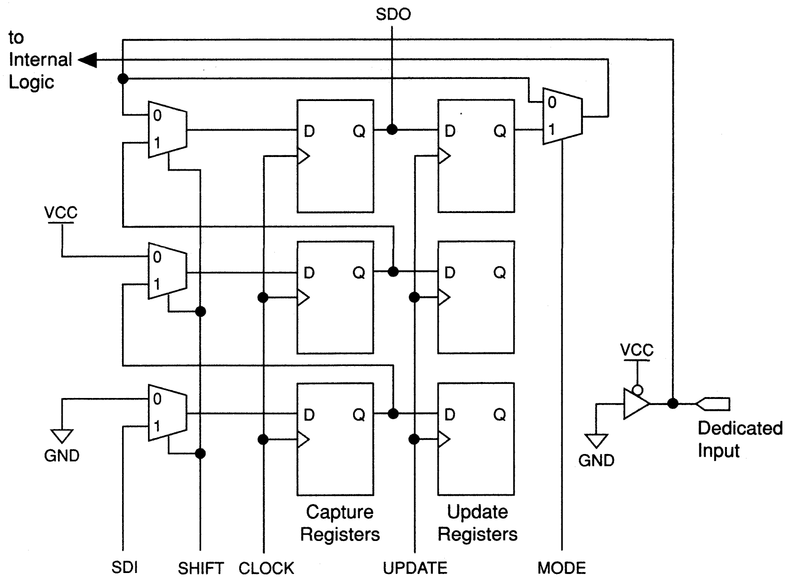


Figure 6 shows the peripheral elements associated with the dedicated configuration pins (i.e., nCONFIG, MSEL0, MSEL1, nSP, CONF_DONE, nSTATUS, and DCLK). These pins are used only during device configuration, but the capture register associated with the I/O pin can be used for

Figure 6. Boundary-Scan Registers on Dedicated Configuration Pins (Part 1 of 2)

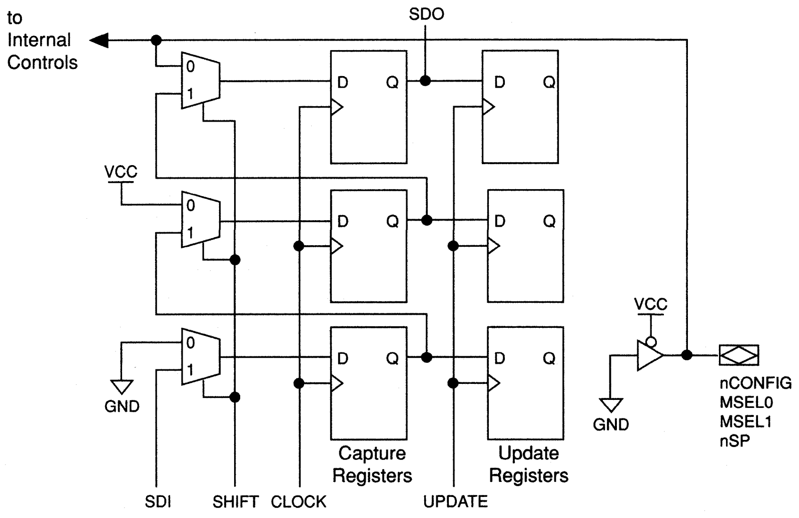
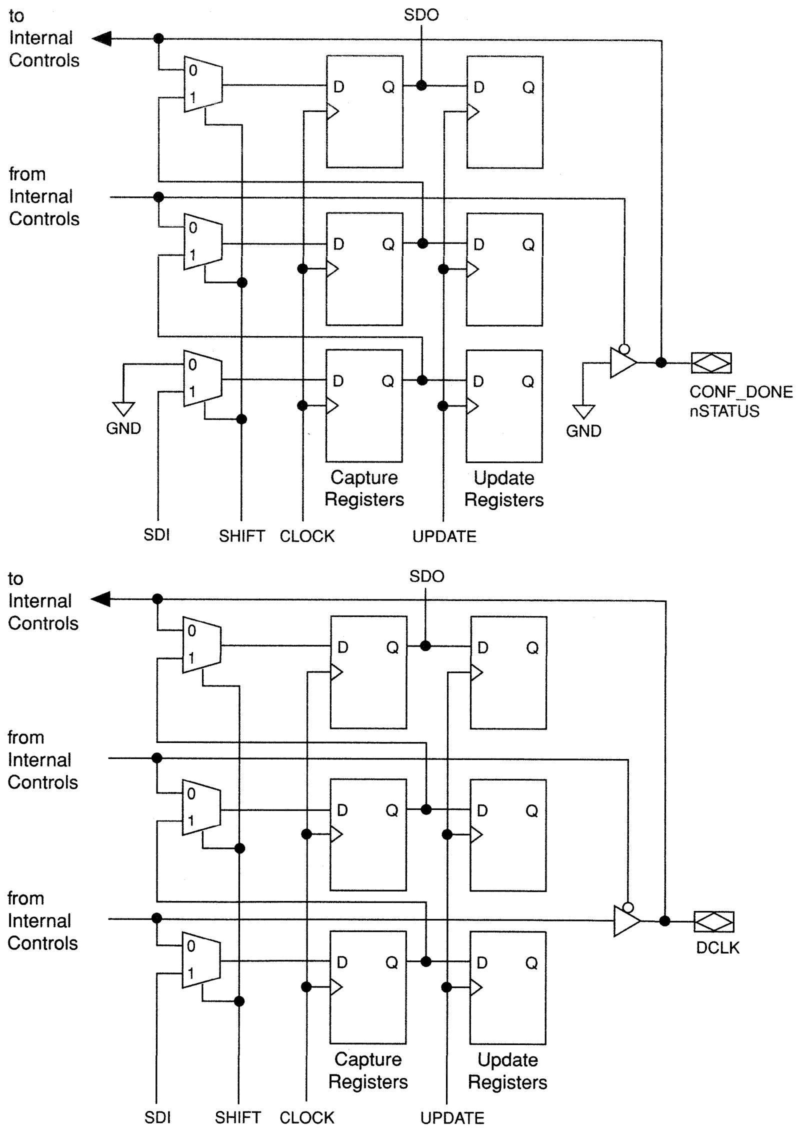


Figure 6. Boundary-Scan Registers on Dedicated Configuration Pins (Part 2 of 2)



external pin connectivity tests. The I/O pin can receive data but is not able to force data onto external connections. The data values associated with the other two capture registers should be ignored.

Tables 4 through 6 at the end of this application note list the boundary-scan order for the EPF8282 and EPF8282V, EPF8820, and EPF81500 devices.

JTAG BST Operation Control

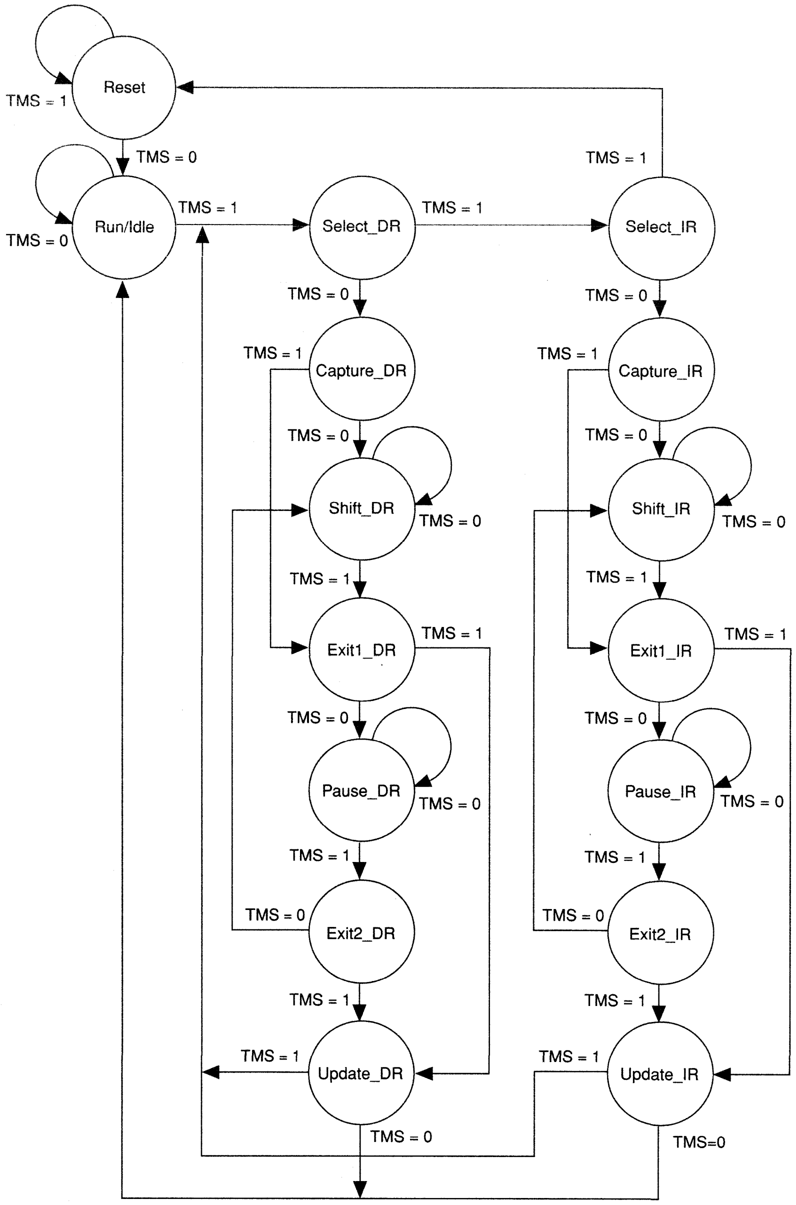
FLEX 8000 devices implement the SAMPLE/PRELOAD, EXTEST, and BYPASS JTAG BST instruction modes. A 3-bit instruction code clocked in through the `TDI` pin determines the mode. Table 2 summarizes the three instruction modes, which are described in detail later in this application note.

Table 2. JTAG Boundary-Scan Instruction Modes

Mode	Code	Description
SAMPLE/ PRELOAD	`101`	Allows a snapshot of the signals at the device pins to be captured and examined while the device is operating normally.
EXTEST	`000`	Allows the external circuitry and board-level interconnections to be tested by forcing a test pattern at the output pins and capturing test results at the input pins.
BYPASS	`111`	Enables the 1-bit bypass register between the `TDI` and `TDO` pins, which allows the BST data to pass through the selected device synchronously to adjacent devices during normal device operation.

The TAP Controller, a 16-state state machine clocked on the rising edge of `TCLK`, uses the `TMS` pin to control JTAG operation in the device. Figure 7 shows the flow of the TAP Controller.

Figure 7. JTAG TAP Controller State Machine

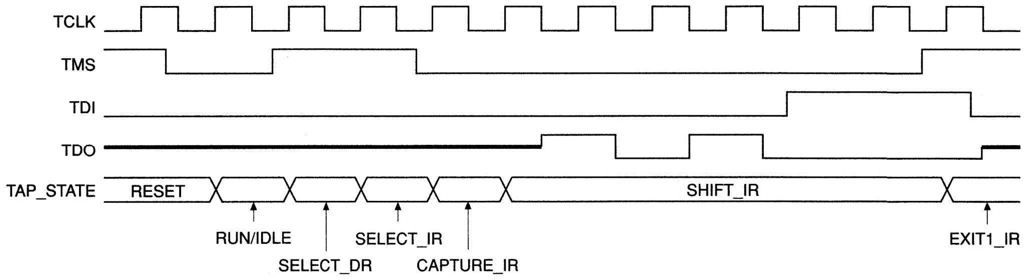


At device power-up, the TAP Controller is in the RESET state. It remains in RESET as long as TMS is held high when TCLK is clocked. During JTAG operation, the RESET state is entered if TMS stays high for at least five TCLK

Clock cycles. You can also return the TAP Controller to the RESET state by holding the nTRST pin low.

To start JTAG operation, you must select an instruction mode by advancing the TAP Controller to the SHIFT_IR (shift instruction register) state and then clocking the appropriate instruction code on the TDI pin. The waveform diagram in Figure 8 represents the entry of the instruction code into the instruction register. It shows the values of TCLK, TMS, TDI, and TDO and the states of the TAP Controller. From the RESET state, TMS is clocked with the pattern 01100 to advance the TAP Controller to SHIFT_IR.

Figure 8. Selecting the Instruction Mode



To ensure proper JTAG operation, the initial state of the instruction register has the code 101. When the SHIFT_IR state is activated, TDO is no longer tri-stated, and the code 101 shifts out on three consecutive falling TCLK edges. TDO continues to shift out the contents of the instruction register as long as the SHIFT_IR state is active. The TAP Controller remains in the SHIFT_IR state as long as TMS remains low.

During the SHIFT_IR state, an instruction code is entered by clocking data on the TDI pin on the rising edge of TCLK. The third bit of the code must be clocked at the same time that the next state, EXIT1_IR, is activated; EXIT1_IR is entered by clocking a high logic level on TMS. Once in the EXIT1_IR state, TDO becomes tri-stated again. It remains tri-stated except in the SHIFT_IR and SHIFT_DR (shift data register) states. After an instruction code is correctly entered, the TAP Controller is advanced to perform the serial shifting of test data in one of three modes—SAMPLE/PRELOAD, EXTEST, or BYPASS—which are described below.

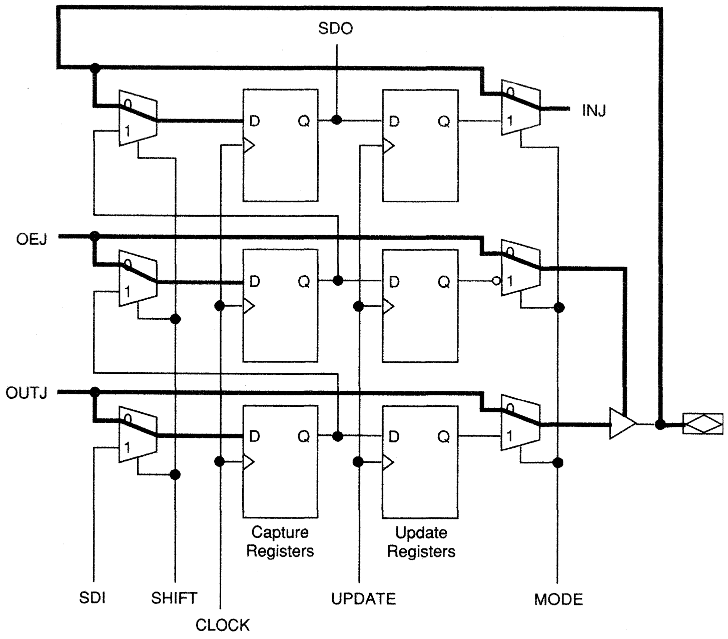
SAMPLE/PRELOAD Instruction Mode

The SAMPLE/PRELOAD instruction mode allows you to take a snapshot of device data without interrupting normal device operation. Figure 9 shows the capture, shift, and update phases of the SAMPLE/PRELOAD mode.

Figure 9. JTAG BST SAMPLE/PRELOAD Mode

Capture Phase

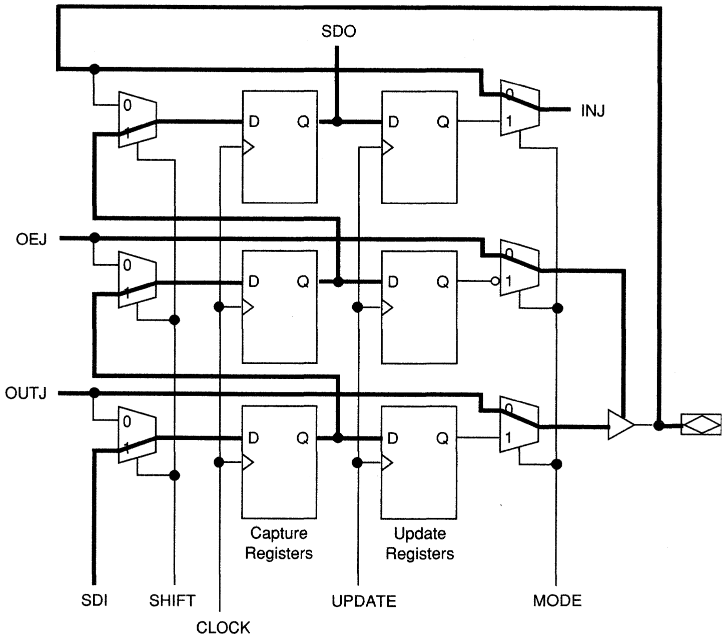
In the capture phase, the signals at the pin, OEJ and OUTJ, are loaded into the capture registers. The register CLOCK signal is supplied by the TAP Controller's CLOCKDR output. The data retained in these registers consists of signals from normal device operation.



Shift & Update Phases

In the shift phase, the previously captured signals at the pin, OEJ and OUTJ, are shifted out of the boundary-scan register via the TDO pin using CLOCK. As data is shifted out, the patterns for the next test can be shifted in via the TDI pin.

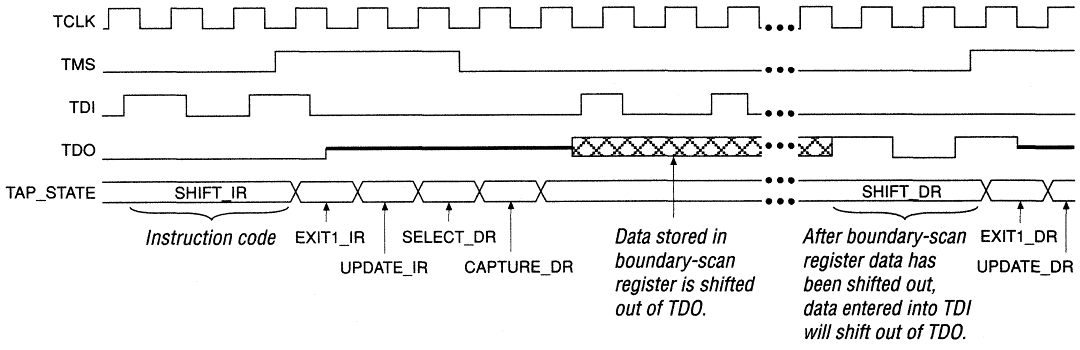
In the update phase, data is transferred from the capture registers to the update registers using the UPDATE Clock. The data stored in the update registers can be used for the EXTEST instruction.



During the capture phase, the multiplexers that precede the capture registers select the active device data signals; this data is then clocked into the capture registers. The multiplexers that follow the update registers also select active device data to prevent functional interruptions to the device. During the shift phase, the boundary-scan shift register is formed by clocking data through capture registers around the device periphery and then out of the TDO pin. New test data can simultaneously be shifted into TDI and replace the contents of the capture registers. During the update phase, data in the capture registers is transferred to the update registers. (This data can then be used in the EXTEST instruction mode described below.)

Figure 10 shows the SAMPLE/PRELOAD waveforms. The code 101 is shifted in through the TDI pin. The TAP Controller advances to the CAPTURE_DR state and then to the SHIFT_DR state, where it remains if TMS is held low. The data shifted out of the TDO pin consists of the data that was present in the capture registers after the capture phase. New test data shifted into the TDI pin appears at the TDO pin after being clocked through the entire boundary-scan register. Figure 10 shows that the 101 code at TDI does not appear at the TDO pin until after the capture register data is shifted out. If TMS is held high on two consecutive TCLK Clock cycles, the TAP Controller advances to the UPDATE_DR state for the update phase.

Figure 10. SAMPLE/PRELOAD Shift Data Register Waveforms


TCLK
TMS
TDI
TDO
TAP_STATE
SHIFT_IR
SHIFT_DR
Instruction code
EXIT1_IR
UPDATE_IR
SELECT_DR
CAPTURE_DR
Data stored in boundary-scan register is shifted out of TDO.
After boundary-scan register data has been shifted out, data entered into TDI will shift out of TDO.
EXIT1_DR
UPDATE_DR

EXTEST Instruction Mode

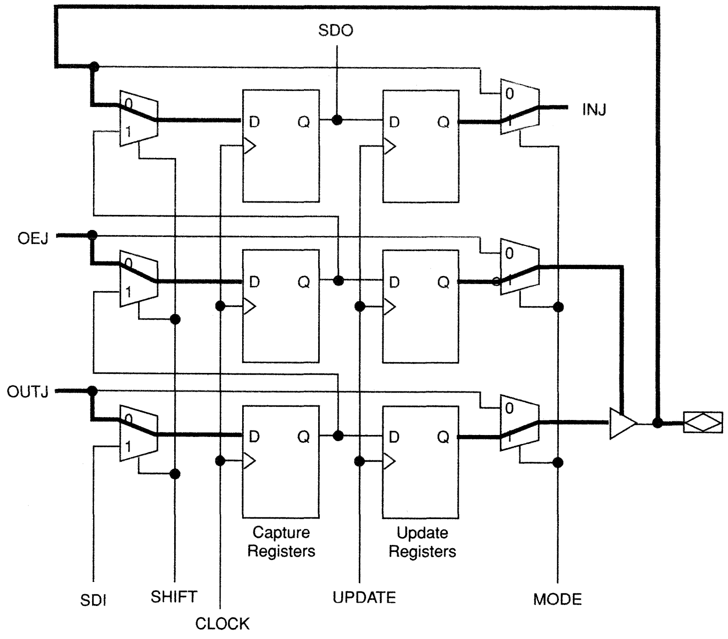
The EXTEST instruction mode is used primarily to check external pin connections between devices. Unlike the SAMPLE/PRELOAD mode, EXTEST allows test data to be forced onto the pin signals. By forcing known high and low logic levels on output pins, open and short circuits can be checked on input pins of any devices in the test scan chain. Figure 11 shows the capture, shift, and update phases of the EXTEST mode.

Figure 11. JTAG BST EXTEST Mode

Capture Phase

In the capture phase, the signals at the pin, OEJ and OUTJ, are loaded into the capture registers. The register CLOCK signal is supplied by the TAP Controller's CLOCKDR output. Previously retained data in the update registers drives the IOE input, INJ, and allows the I/O pin to tri-state or drive a signal out.

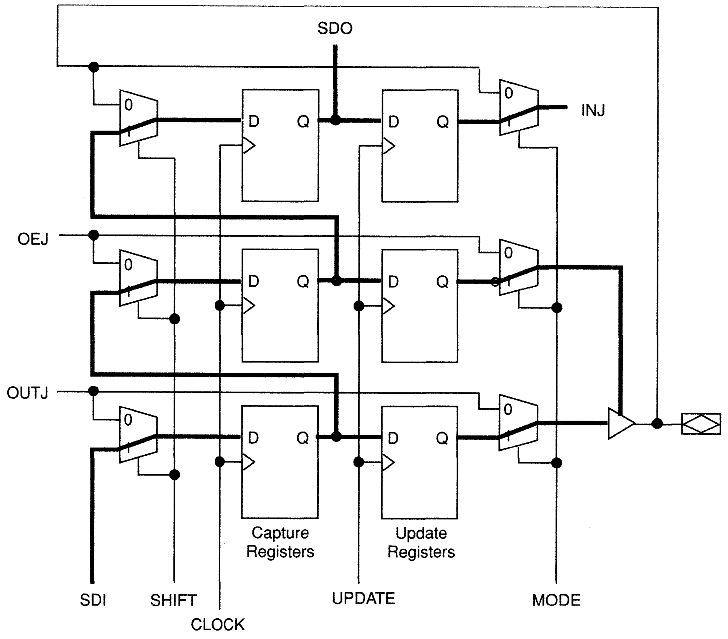
A "1" in the OEJ update register tri-states the output buffer.



Shift & Update Phases

In the shift phase, the previously captured signals at the pin, OEJ and OUTJ, are shifted out of the boundary-scan register via the TDO pin using CLOCK. As data is shifted out, the patterns for the next test can be shifted in via the TDI pin.

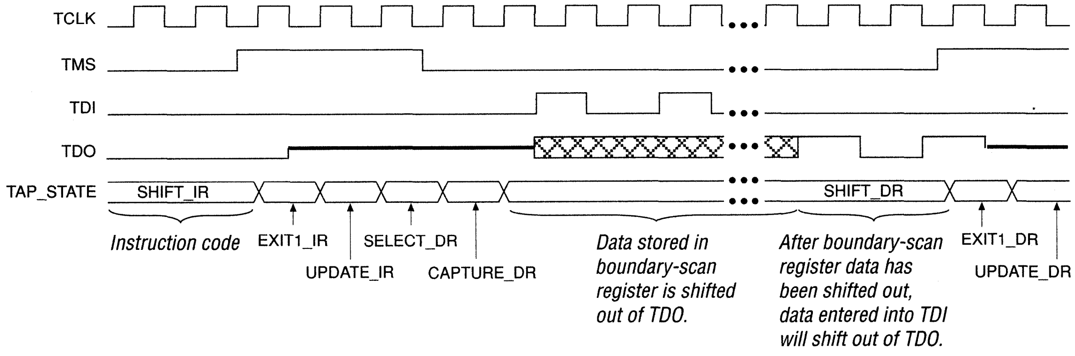
In the update phase, data is transferred from the capture registers to the update registers using the UPDATE Clock. The update registers then drive the IOE input, INJ, and allow the I/O pin to tri-state or drive a signal out.



EXTEST differs from SAMPLE/PRELOAD in the selection of data. EXTEST chooses data from the update registers as the source of the INJ, output, and Output Enable signals. Once the EXTEST instruction code is entered, the multiplexers select the update register data; thus, data stored in these registers from a previous EXTEST or SAMPLE/PRELOAD test cycle can be forced onto the pin signals. In the capture phase, the results of this test data are stored in the capture registers and then shifted out of TDO during the shift phase. New test data can then be stored in the update registers during the update phase.

The waveform diagram in Figure 12 resembles the SAMPLE/PRELOAD waveform diagram, except that EXTEST uses the instruction code 000. The data shifted out of TDO consists of the data that was present in the capture registers after the capture phase. New test data shifted into the TDI pin appears at the TDO pin after being clocked through the entire boundary-scan register.

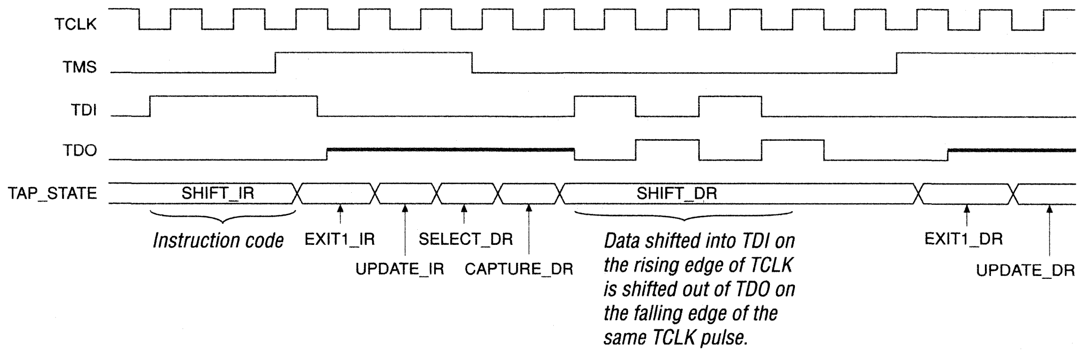
Figure 12. EXTEST Shift Data Register Waveforms


TCLK
TMS
TDI
TDO
TAP_STATE
SHIFT_IR
SHIFT_DR
Instruction code
EXIT1_IR
UPDATE_IR
SELECT_DR
CAPTURE_DR
Data stored in boundary-scan register is shifted out of TDO.
After boundary-scan register data has been shifted out, data entered into TDI will shift out of TDO.
EXIT1_DR
UPDATE_DR

BYPASS Instruction Mode

If the SAMPLE/PRELOAD or EXTEST instruction modes are not selected, the TAP Controller automatically enters the BYPASS mode. Otherwise, BYPASS is activated with the instruction code 111. The waveforms in Figure 13 show how scan data passes quickly through a device once the TAP Controller is in the SHIFT_DR state. In this state, data signals are clocked into the bypass register from TDI on the rising edge of TCLK and out of TDO on the falling edge of the same Clock pulse.

Figure 13. BYPASS Shift Data Register Waveforms


TCLK
TMS
TDI
TDO
TAP_STATE
SHIFT_IR
SHIFT_DR
Instruction code
EXIT1_IR
UPDATE_IR
SELECT_DR
CAPTURE_DR
Data shifted into TDI on the rising edge of TCLK is shifted out of TDO on the falling edge of the same TCLK pulse.
EXIT1_DR
UPDATE_DR

Enabling JTAG BST Circuitry

You can use JTAG boundary-scan testing either before or after a FLEX 8000 device is configured. The JTAG BST circuitry is enabled automatically prior to device configuration. If the dedicated configuration pin `nCONFIG` is held low, device configuration is delayed and you can perform JTAG boundary-scan testing. If the *Enable JTAG Support* option is turned on with MAX+PLUS II before compilation, you can also perform JTAG testing once configuration is finished.

You can enable JTAG support on a device-by-device basis with the **FLEX 8000 Individual Device Options** dialog box. You can also enable JTAG support on all devices in a project with the **FLEX 8000 Device Options** dialog box.

When the *Enable JTAG Support* option is turned on, `TDI`, `TDO`, `TMS`, and `TCLK` become dedicated JTAG pins after device configuration. If JTAG support is disabled, these pins function as standard I/O pins, except in the EPF81500, which has dedicated JTAG pins.

Go to the current *FLEX 8000 Programmable Logic Device Family Data Sheet* or MAX+PLUS II Help for JTAG pin numbers. For detailed information on configuring FLEX 8000 devices, refer to *Application Note 33 (Configuring FLEX 8000 Devices)* and *Application Note 38 (Configuring Multiple FLEX 8000 Devices)* in this handbook.

Guidelines for JTAG Boundary-Scan Testing

Use the following guidelines when performing boundary-scan testing with JTAG devices:

❑ If the `101` code does not shift out of the instruction register via the `TDO` pin during the first three Clock cycles of the `SHIFT_IR` state, then the proper TAP Controller state has not been reached. To solve this problem, try the following:

- Verify that the TAP Controller has reached the `SHIFT_IR` state correctly. To advance the TAP Controller to the `SHIFT_IR` state, return to the `RESET` state and clock the code `01100` on the `TMS` pin.
- Check the connections to the `VCC`, `GND`, JTAG, and dedicated configuration pins on the device.
- If the device is in user mode, make sure that the *Enable JTAG Support* option is turned on.

❑ You should perform a SAMPLE/PRELOAD test cycle prior to the first EXTEST test cycle to ensure that known data is present at the device pins when the EXTEST mode is entered. If the `OEJ` update register contains a `0`, then the IOE will drive out the data in the `OUTJ` update register. The IOE state must be known and correct to avoid contention with other devices in the system.

❑ The bypass and boundary-scan registers shift simultaneously when the TAP Controller is in the `SHIFT_DR` state. Using the BYPASS mode will shift test data out of the capture registers. Therefore, do not execute a BYPASS shift cycle before an EXTEST test cycle that requires preloaded test data.

If problems persist, call Altera Applications at (800) 800-EPLD.

Boundary-Scan Description Language (BSDL) Support

You can describe the testability features of a JTAG-compliant FLEX 8000 device using the Boundary-Scan Description Language (BSDL). BSDL is a subset of VHDL. You can use BSDL with test software development systems for test generation, analysis, and failure diagnosis.

Table 3 lists the BSDL files for JTAG-compliant FLEX 8000 devices. These self-extracting files are available from the Altera bulletin board service (BBS) at (408) 954-0104.

Table 3. BSDL Support

Device	Package	Filename
EPF8282 and EPF8282V	100-pin TQFP 84-pin PLCC	epf8282t.bsd epf8282l.bsd
EPF8636	208-pin RQFP 192-pin PGA 84-pin PLCC	Under development
EPF8820	225-pin BGA 208-pin RQFP 192-pin PGA	epf8820b.bsd epf8820r.bsd epf8820g.bsd
EPF81500	304-pin RQFP 280-pin PGA	epf8150r.bsd epf8150g.bsd

Conclusion

The JTAG BST circuitry available in FLEX 8000 devices provides a cost-effective and efficient way to test systems with tight lead spacing. Circuit boards with FLEX 8000 and other JTAG-compliant devices can use the EXTEST, SAMPLE/PRELOAD, and BYPASS modes to create serial patterns that internally test the pin connections between devices and check device operation.

References

Bleeker, H., P. van den Eijnden, and F. de Jong. ***Boundary-Scan Test: A Practical Approach***. Eindhoven, The Netherlands: Kluwer Academic Publishers, 1993.

Institute of Electrical and Electronic Engineers, Inc. ***IEEE Standard Test Access Port and Boundary-Scan Architecture*** (IEEE Std 1149.1-1990). New York: Institute of Electrical and Electronic Engineers, Inc., 1990.

Maunder, C. M., and R. E. Tulloss. ***The Test Access Port and Boundary-Scan Architecture***. Los Alamitos: IEEE Computer Society Press, 1990.

JTAG Boundary-Scan Order

Table 4 shows the boundary-scan order for the EPF8282 and EPF8282V JTAG-compliant devices. These devices are available in 100-pin thin quad flat pack (TQFP) and 84-pin plastic J-lead chip carrier (PLCC) packages.

Table 4. EPF8282 & EPF8282V JTAG Boundary-Scan Order

Scan Order	100-Pin TQFP	84-Pin PLCC	Pin Function	Scan Order	100-Pin TQFP	84-Pin PLCC	Pin Function	Scan Order	100-Pin TQFP	84-Pin PLCC	Pin Function
TDI	54	55	–	31	88	2	I/O	62	24	32	nSTATUS
1	55	56	I/O	32	88 *(1)*	2 *(1)*	I/O	63	25	33	nCONFIG
2	57	57	I/O	33	89	3	I/O	64	26	34	I/O
3	58	58	I/O	34	90	3 *(1)*	I/O	65	27 *(1)*	35	I/O
4	59	60	I/O	35	91	4	I/O	66	27	35 *(1)*	I/O
5	60	61	I/O	36	91 *(1)*	4 *(1)*	I/O	67	28	36	I/O
6	61	62	I/O	37	92	6	I/O	68	29	37	I/O
7	62	63	I/O	38	93	6 *(1)*	I/O	69	31 *(1)*	39	I/O
8	64	64	I/O	39	95	7	I/O	70	31	39 *(1)*	I/O
9	65	65	I/O	40	96	7 *(1)*	I/O	71	32	40	I/O
10	66	66	I/O	41	97	8	I/O	72	33	41	I/O
11	67	67	I/O	42	98	8 *(1)*	I/O	73	34 *(1)*	42	I/O
12	68	69	I/O	43	99	9	I/O	74	34	42 *(1)*	I/O
13	69	70	I/O	44	100	10	DCLK	75	35	43	I/O
14	71	71	I/O	45	1	11	CONF_DONE	76	36	43 *(1)*	I/O
15	73	73	INPUT	46	3	12	INPUT	77	38 *(1)*	44	I/O
16	74	74	MSEL0	47	4	13	I/O	78	38	44 *(1)*	I/O
17	75	75	nSP	48	5	14	I/O	79	39	45	I/O
18	76	76	I/O	49	7	15	I/O	80	40	45 *(1)*	I/O
19	77	77	I/O	50	8	16	I/O	81	41 *(1)*	46	I/O
20	77 *(1)*	77 *(1)*	I/O	51	9	18	I/O	82	41	46 *(1)*	I/O
21	78	78	I/O	52	10	19	I/O	83	42	48	I/O
22	79	79	I/O	53	12	21	I/O	84	43	48 *(1)*	I/O
23	81	81	I/O	54	14	22	I/O	85	45	49	I/O
24	81 *(1)*	81 *(1)*	I/O	55	15	23	I/O	86	46	49 *(1)*	I/O
25	82	82	I/O	56	16	24	I/O	87	47	50	I/O
26	83	83	I/O	57	17	25	I/O	88	48	50 *(1)*	I/O
27	84	84	I/O	58	19	28	I/O	89	49	51	I/O
28	84 *(1)*	84 *(1)*	I/O	59	21	29	I/O	90	51	53	MSEL1
29	85	1	I/O	60	22	30	I/O	91	53	54	INPUT
30	86	1 *(1)*	I/O	61	23	31	INPUT	TDO	18	27	–

Note:
(1) Double-bonded pin. IOE is not connected to internal logic and IOE output is tri-stated; JTAG data should be ignored.

Table 5 shows the boundary-scan order for the EPF8820 JTAG-compliant device. This device is available in 208-pin power quad flat pack (RQFP), 192-pin pin-grid array (PGA), and 225-pin ball-grid array (BGA) packages.

Table 5. EPF8820 JTAG Boundary-Scan Order (Part 1 of 3)

Scan Order	208-Pin RQFP	192-Pin PGA	225-Pin BGA	Pin Function	Scan Order	208-Pin RQFP	192-Pin PGA	225-Pin BGA	Pin Function
TDI	20	R11	F15	–	32	185	K16	C8	I/O
1	19	U12	F14	I/O	33	184	J17	B8	I/O
2	18	T12	F13	I/O	34	183	J15	A8	I/O
3	17	U13	E15	INPUT	35	181	J14	D8	I/O
4	14	R12	E13	I/O	36	180	J16	E8	I/O
5	13	U14	D15	I/O	37	179	H16	B7	I/O
6	12	T13	F11	I/O	38	178	H17	A7	I/O
7	11	U15	D14	I/O	39	177	H15	C7	I/O
8	10	R13	E12	I/O	40	176	G17	D7	I/O
9	9	U16	C15	I/O	41	175	G16	E7	I/O
10	8	T14	D13	I/O	42	174	F17	A6	I/O
11	7	T16	C14	I/O	43	172	E17	A5	I/O
12	4	T15	B14	MSEL0	44	171	G15	B5	I/O
13	207	R15	A15	nSP	45	170	D17	D6	I/O
14	205	P15	B13	I/O	46	169	F16	C5	I/O
15	204	U17	E11	I/O	47	168	C17	A4	I/O
16	203	P16	C12	I/O	48	167	F15	E6	I/O
17	202	R16	A13	I/O	49	166	B17	B4	I/O
18	201	N15	B12	I/O	50	165	E16	D5	I/O
19	199	N16	D11	I/O	51	163	E15	A3	I/O
20	198	T17	A12	I/O	52	162	C16	C4	I/O
21	197	M15	C11	I/O	53	161	D16	B3	I/O
22	196	R17	B11	I/O	54	160	A17	F6	I/O
23	195	M16	E10	I/O	55	158	C15	B2	DCLK
24	194	P17	A11	I/O	56	153	B15	A1	CONF_DONE
25	193	L15	D10	I/O	57	151	C14	C2	I/O
26	192	N17	C10	I/O	58	150	B16	E5	I/O
27	190	M17	D9	I/O	59	149	B14	D3	I/O
28	189	L16	C9	I/O	60	148	A16	C1	I/O
29	188	L17	B9	I/O	61	147	C13	D2	I/O
30	187	K15	A9	I/O	62	146	A15	E4	I/O
31	186	K17	E9	I/O	63	145	B13	D1	I/O

Table 5. EPF8820 JTAG Boundary-Scan Order (Part 2 of 3)

Scan Order	208-Pin RQFP	192-Pin PGA	225-Pin BGA	Pin Function	Scan Order	208-Pin RQFP	192-Pin PGA	225-Pin BGA	Pin Function
64	144	A14	E3	I/O	96	98	C2	R3	I/O
65	143	C12	E2	I/O	97	97	E2	P4	I/O
66	140	A13	F4	INPUT	98	95	F3	M5	I/O
67	139	B12	F3	I/O	99	94	B1	R4	I/O
68	138	A12	F2	I/O	100	93	F2	N5	I/O
69	137	C11	F1	I/O	101	92	C1	P5	I/O
70	136	A11	G4	I/O	102	91	G3	L6	I/O
71	135	B11	G3	I/O	103	90	D1	R5	I/O
72	134	A10	G2	I/O	104	89	G2	M6	I/O
73	133	B10	G1	I/O	105	88	E1	N6	I/O
74	132	A9	G5	I/O	106	86	F1	M7	I/O
75	128	B8	J1	I/O	107	85	H3	N7	I/O
76	127	A8	J3	I/O	108	84	G1	P7	I/O
77	126	B7	J4	I/O	109	83	H2	R7	I/O
78	125	A7	J5	I/O	110	82	H1	L7	I/O
79	124	C7	K1	I/O	111	81	J3	N8	I/O
80	123	A6	K2	I/O	112	80	J2	P8	I/O
81	122	B6	K3	I/O	113	79	J4	R8	I/O
82	121	A5	L1	INPUT	114	77	K2	M8	I/O
83	118	C6	L3	I/O	115	76	J1	L8	I/O
84	117	A4	M1	I/O	116	75	K3	P9	I/O
85	116	B5	K5	I/O	117	74	K1	R9	I/O
86	115	A3	M2	I/O	118	73	L2	N9	I/O
87	114	C5	L4	I/O	119	72	L1	M9	I/O
88	113	A2	N1	I/O	120	71	L3	L9	I/O
89	112	B4	M3	I/O	121	70	M1	R10	I/O
90	111	B2	N2	I/O	122	68	N1	R11	I/O
91	108	B3	P2	nSTATUS	123	67	M2	P11	I/O
92	103	C3	R1	nCONFIG	124	66	P1	M10	I/O
93	101	D2	P3	I/O	125	65	M3	N11	I/O
94	100	A1	L5	I/O	126	64	R1	R12	I/O
95	99	E3	N4	I/O	127	63	N2	L10	I/O

Table 5. EPF8820 JTAG Boundary-Scan Order (Part 3 of 3)

Scan Order	208-Pin RQFP	192-Pin PGA	225-Pin BGA	Pin Function	Scan Order	208-Pin RQFP	192-Pin PGA	225-Pin BGA	Pin Function
128	62	T1	P12	I/O	143	39	R6	L14	I/O
129	61	N3	M11	I/O	144	36	U5	K12	INPUT
130	59	P2	R13	I/O	145	35	T6	K13	I/O
131	58	R2	N12	I/O	146	34	U6	K14	I/O
132	57	P3	P13	I/O	147	33	R7	K15	I/O
133	56	U1	K10	I/O	148	31	T7	J13	I/O
134	49	T3	R15	MSEL1	149	29	T8	J15	I/O
135	47	R4	N14	I/O	150	28	T9	J11	I/O
136	46	T2	L11	I/O	151	25	U9	G14	I/O
137	45	T4	M13	I/O	152	24	T10	G15	I/O
138	44	U2	N15	I/O	153	23	U10	G13	I/O
139	43	R5	M14	I/O	154	22	T11	G12	I/O
140	42	U3	L12	I/O	155	21	U11	G11	I/O
141	41	T5	M15	I/O	TDO	129	B9	J2	–
142	40	U4	L13	I/O					

Table 6 shows the boundary-scan order for the EPF81500 JTAG-compliant device. This device is available in 304-pin power quad flat pack (RQFP) and 280-pin PGA packages.

Table 6. EPF81500 JTAG Boundary-Scan Order (Part 1 of 2)

Scan Order	304-Pin RQFP	280-Pin PGA	Pin Function	Scan Order	304-Pin RQFP	280-Pin PGA	Pin Function	Scan Order	304-Pin RQFP	280-Pin PGA	Pin Function
TDI	80	B1	–	36	20	N2	I/O	72	279	V8	I/O
1	76	C1	I/O	37	19	P1	I/O	73	278	W7	I/O
2	75	D2	I/O	38	18	N3	I/O	74	277	T9	I/O
3	74	D1	I/O	39	17	R1	I/O	75	276	W8	I/O
4	73	E3	I/O	40	16	N4	I/O	76	275	U9	I/O
5	72	E1	I/O	41	15	T1	I/O	77	274	V9	I/O
6	71	E2	I/O	42	14	P2	I/O	78	272	W9	I/O
7	70	F2	I/O	43	13	T2	I/O	79	271	U10	I/O
8	69	F4	I/O	44	12	P3	INPUT	80	270	W10	I/O
9	64	F1	INPUT	45	8	U1	I/O	81	269	U11	I/O
10	63	F3	I/O	46	7	R2	I/O	82	268	V10	I/O
11	62	G2	I/O	47	6	U2	I/O	83	267	T10	I/O
12	61	G4	I/O	48	5	R3	I/O	84	264	W11	I/O
13	60	G1	I/O	49	4	V1	I/O	85	263	V13	I/O
14	59	G3	I/O	50	3	T3	I/O	86	261	U12	I/O
15	58	H2	I/O	51	2	V2	I/O	87	260	V11	I/O
16	57	H4	I/O	52	1	U3	I/O	88	259	T11	I/O
17	56	H1	I/O	53	304	W1	nSP	89	258	W12	I/O
18	51	H3	MSEL1	54	299	U4	I/O	90	257	U13	I/O
19	50	J2	I/O	55	298	W2	I/O	91	256	V12	I/O
20	49	J4	I/O	56	297	T6	I/O	92	255	T12	I/O
21	48	J1	I/O	57	296	V3	I/O	93	254	W13	I/O
22	47	J3	I/O	58	295	U5	I/O	94	252	W14	I/O
23	46	K1	I/O	59	294	W3	I/O	95	251	V14	I/O
24	45	K3	I/O	60	293	U6	I/O	96	250	W15	I/O
25	44	K2	I/O	61	292	V4	I/O	97	249	U14	I/O
26	43	K4	I/O	62	290	W4	I/O	98	248	W16	I/O
27	34	L1	I/O	63	289	T7	I/O	99	247	V15	I/O
28	33	L3	I/O	64	288	V5	I/O	100	246	W17	I/O
29	32	L2	I/O	65	287	U7	I/O	101	245	U15	I/O
30	31	L4	I/O	66	286	W5	I/O	102	243	V16	I/O
31	30	M1	I/O	67	285	T8	I/O	103	242	W18	I/O
32	29	M3	I/O	68	284	V6	I/O	104	241	U16	I/O
33	28	M2	I/O	69	283	V7	I/O	105	240	V18	I/O
34	27	M4	I/O	70	281	U8	I/O	106	239	V17	I/O
35	26	N1	MSEL0	71	280	W6	I/O	107	238	W19	I/O

Table 6. EPF81500 JTAG Boundary-Scan Order (Part 2 of 2)

Scan Order	304-Pin RQFP	280-Pin PGA	Pin Function
108	230	U18	DCLK
109	229	V19	I/O
110	228	T17	I/O
111	227	U19	I/O
112	226	T18	I/O
113	225	T19	I/O
114	224	R17	I/O
115	223	R19	I/O
116	222	R18	I/O
117	217	P19	INPUT
118	216	P17	I/O
119	215	N18	I/O
120	214	P18	I/O
121	213	N19	I/O
122	212	N16	I/O
123	211	M18	I/O
124	210	N17	I/O
125	209	M19	I/O
126	204	M16	CONF_DONE
127	203	L18	I/O
128	202	M17	I/O
129	201	L19	I/O
130	200	L16	I/O
131	199	K19	I/O
132	198	L17	I/O
133	197	K18	I/O
134	196	K17	I/O
135	186	J19	I/O
136	185	K16	I/O
137	184	J18	I/O
138	183	J17	I/O
139	182	H19	I/O
140	181	J16	I/O
141	180	H18	I/O
142	179	H17	I/O
143	178	G19	nSTATUS
144	172	H16	I/O
145	171	G18	I/O
146	170	G17	I/O
147	169	F19	I/O
148	168	G16	I/O
149	167	F18	I/O
150	166	F17	I/O
151	165	E19	I/O
152	164	F16	INPUT
153	160	E18	I/O
154	159	E17	I/O
155	158	D19	I/O
156	157	D18	I/O
157	156	C19	I/O
158	155	D17	I/O
159	154	B19	I/O
160	153	C18	I/O
161	152	B18	nCONFIG
162	143	A17	I/O
163	142	C16	I/O
164	141	A16	I/O
165	140	B16	I/O
166	139	A15	I/O
167	138	C15	I/O
168	136	B15	I/O
169	135	B14	I/O
170	134	C14	I/O
171	133	A14	I/O
172	132	D13	I/O
173	131	B13	I/O
174	130	D12	I/O
175	129	A13	I/O
176	127	B12	I/O
177	126	C13	I/O
178	125	A12	I/O
179	124	C12	I/O
180	123	B11	I/O
181	122	D11	I/O
182	121	A11	I/O
183	120	C11	I/O
184	118	C10	I/O
185	117	A10	I/O
186	114	D10	I/O
187	113	B10	I/O
188	112	B9	I/O
189	111	A9	I/O
190	110	B8	I/O
191	109	A8	I/O
192	107	A7	I/O
193	106	C9	I/O
194	105	A6	I/O
195	104	D9	I/O
196	103	A5	I/O
197	102	C8	I/O
198	101	B6	I/O
199	100	B7	I/O
200	98	C7	I/O
201	97	A4	I/O
202	96	D8	I/O
203	95	B5	I/O
204	94	D7	I/O
205	93	A3	I/O
206	92	C6	I/O
207	91	B4	I/O
208	89	A2	I/O
209	88	C5	I/O
210	87	B3	I/O
211	86	C4	I/O
212	85	A1	I/O
213	84	D6	I/O
214	83	B2	I/O
215	82	C3	I/O
TDO	149	C17	–

Notes:

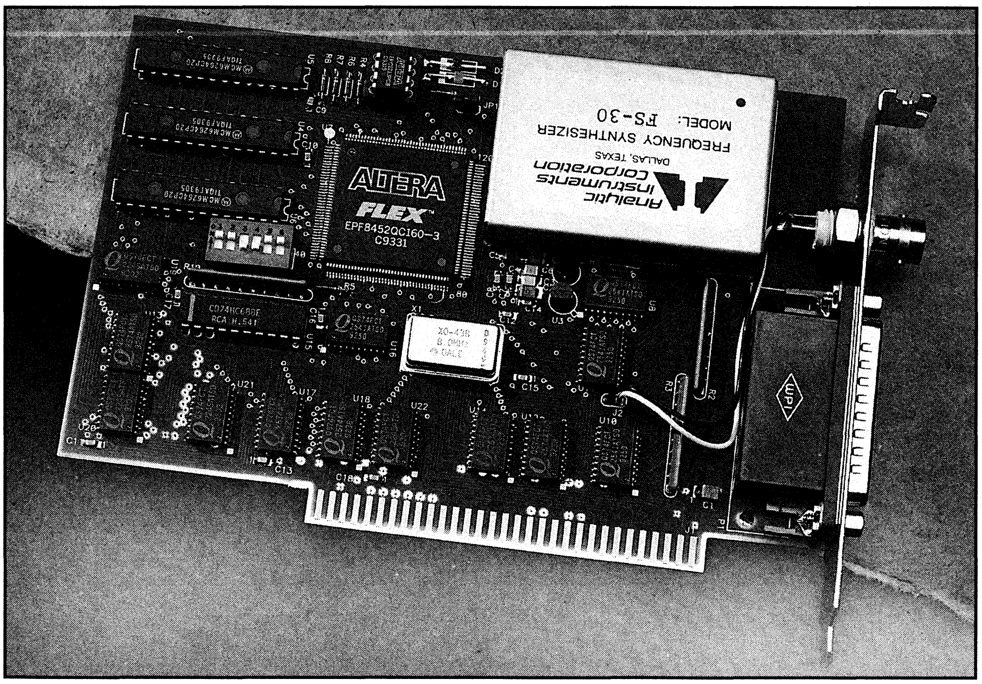
ALTERA

Contents

May 1994

Section 3

Design Tips



Physical Science Lab designs instrumentation packages for rockets, balloons, and spacecraft. PSL uses the EPF8452 in their Pulse Code Modulation (PCM) Simulator Board. The PCM Simulator Board simulates the data stream output of a flight telemetry encoder and is used to test ground station hardware.

Designing with FLEX 8000 Devices

May 1994, ver. 2 Application Note 36

Introduction

Historically, programmable logic devices have fallen into two broad categories: Erasable Programmable Logic Devices (EPLDs) and Field-Programmable Gate Arrays (FPGAs). Widespread use of both EPLDs and FPGAs has revealed the strengths of each type of device. Altera's FLEX 8000 architecture combines the strengths of both EPLDs and FPGAs. This application note describes some of the basic characteristics of the FLEX 8000 architecture and offers several design guidelines that can help you use this architecture effectively.

Regardless of your level of familiarity with programmable logic devices, the information in this application note can help you use FLEX 8000 devices to their fullest potential. For detailed descriptions of the FLEX 8000 architecture and device configuration, refer to *Application Note 40* (*FLEX 8000 Architecture*), *Application Note 33* (*Configuring FLEX 8000 Devices*), and *Application Note 38* (*Configuring Multiple FLEX 8000 Devices*) in this handbook.

Designing for Speed & Density

Engineers have learned to expect different levels of performance and density from EPLDs and FPGAs. In both EPLD and FPGA architectures, trade-offs are made to optimize designs either for speed or for density. With the FLEX 8000 architecture, you can control speed/density trade-offs to suit the needs of your application. In addition, you can use Altera's MAX+PLUS II software to automatically optimize all or part of a circuit for speed or density. You can also structure designs to take advantage of the physical characteristics of the FLEX 8000 architecture.

Automatic Design Optimization

Altera's MAX+PLUS II software automatically optimizes a design for the architecture of its target device family. Therefore, you can change the target device family for a design without entering any special design modifications. You can also combine one or more existing EPLD designs and re-target them for one or more FLEX 8000 devices.

When you compile a design for the FLEX 8000 device family, the MAX+PLUS II Compiler automatically uses macrofunctions that are optimized for the FLEX 8000 architecture and performs FLEX 8000 family-specific logic synthesis and optimization. You can use logic synthesis styles and logic options that are tailored to the FLEX 8000 architecture to further control design optimization. The automatic design optimization in

MAX+PLUS II allows you to re-target designs to other device families quickly and with minimal effort.

Designing for FLEX 8000 Architecture

In addition to using automatic design optimization, you can take advantage of specific architectural features in FLEX 8000 devices to create circuits that run at higher speeds or use fewer device resources. MAX+PLUS II design entry methods provide the detailed control necessary to achieve the maximum possible speed and density in a FLEX 8000 device.

Basic Architectural Features

The FLEX 8000 architecture is based on logic elements (LEs) containing 4-input look-up tables (LUTs), registers, and several other features that are especially important in creating logic designs:

- ❑ Register control functions—Clock, Clear, and Preset signals that control a programmable flipflop.
- ❑ FastTrack Interconnect—A series of fast, continuous paths that run the entire length and width of the device and provide signal interconnections between different Logic Array Block (LABs) and between LABs and pins.
- ❑ Carry and cascade chains—High-speed data paths that connect adjacent LEs without using other interconnect resources.

The FLEX 8000 LUT can implement any function of four variables. When you compile a FLEX 8000 design, the MAX+PLUS II Compiler automatically divides functions of more than four variables into multiple 4-input functions.

In contrast, Altera's Classic, MAX 5000/EPS464, and MAX 7000 device families use an AND-OR array as the fundamental building block for combinatorial logic, with eight, five, and three product terms per macrocell, respectively. When you compile a design for any of these EPLD families, the Compiler divides functions requiring more than the available number of product terms into multiple macrocells.

Design Entry Methods

Altera provides and supports design entry methods that offer a full spectrum of low- to high-level control over actual design implementation. If your primary goal is a fast design cycle, you can describe a design with high-level constructs in a hardware description language (HDL) such as Verilog HDL, VHDL, or the Altera Hardware Description Language (AHDL). Although high-level constructs help simplify the design entry process, they can limit your control over the physical device implementation because the synthesized logic depends on the synthesis algorithms of the software that processes the HDL.

If you wish to obtain the maximum performance and density in FLEX 8000 devices, you can describe designs with primitive gates and registers (i.e., a "gate-level" design) using HDLs or schematics. MAX+PLUS II also provides family-specific macrofunctions that have been optimized for the FLEX 8000 architecture. Gate-level designs and designs that use FLEX 8000 family-specific macrofunctions provide the greatest control over the physical implementation in a device. Although designing at the gate level may slow the design process, it typically yields the highest speeds and lowest area costs.

Regardless of the design entry method you choose, you can assign logic options in MAX+PLUS II to guide logic synthesis on individual logic functions. You can also apply logic synthesis styles, which are combinations of logic option settings saved under a single name. These logic options and logic synthesis styles can be set to optimize a design for a particular device family. For example, specifying a setting of "Auto" for the Carry Chain and Cascade Chain logic options instructs the Compiler to automatically implement FLEX 8000 carry and cascade chains, which are useful for optimizing designs for high speed or minimum area.

These logic options are also set to Auto in the Altera-provided Fast logic synthesis style in MAX+PLUS II. In contrast, specifying a setting of "Ignore" for the Carry Chain and Cascade Chain logic options directs the Compiler to ignore any user-specified carry and cascade logic. These logic options are set to Ignore in the Altera-provided Normal logic synthesis style. Since the LEs in a carry or cascade chain must be adjacent to each other, long carry or cascade chains can limit fitting flexibility and may reduce the routability of a design. Therefore, you may wish to use different logic option settings in different portions of your design. For more information on logic options and logic synthesis styles, refer to MAX+PLUS II Help; for more information on carry and cascade chains, refer to *Application Note 40* (*FLEX 8000 Architecture*) in this handbook.

General FLEX 8000 Design Guidelines

The following design guidelines will help you use the FLEX 8000 architecture as efficiently as possible. Following these guidelines will yield maximum speed, reliability, and device resource utilization, and minimize fitting problems.

Reserve Resources in the Device for Future Expansion

The design process generally includes many modification cycles for logic changes or additional logic. Altera recommends that you leave 20% of the device's logic cells and I/O pins unused to accommodate future design modifications.

Allow the Compiler to Select Pin & Logic Cell Assignments

Although you can use FLEX 8000 device resources extremely efficiently, poorly or arbitrarily selected resource assignments can prevent a design from fitting. During compilation, MAX+PLUS II arranges and permutes logic cell and I/O pin locations to use the partially populated multiplexers in the FastTrack Interconnect as efficiently as possible. Pin and/or logic cell assignments, however, can limit the MAX+PLUS II Compiler's ability to arrange signals efficiently, thus reducing the probability of a successful fit. Therefore, Altera recommends that you allow the Compiler to choose all pin and logic cell locations automatically. You should also simulate a design as thoroughly as possible before you lay out your printed circuit board or back-annotate the Compiler's pin assignments.

Balance Ripple-Carry & Carry Look-Ahead Usage

Each FLEX 8000 LE contains high-speed carry and cascade generation logic. The dedicated carry chain in the FLEX 8000 architecture can propagate a ripple-carry for short- and medium-length counters and adders with minimum delay and maximum efficiency. Long carry chains, however, restrict the Compiler's ability to fit a design because the LEs in the chain must be contiguous.

You can design counters using either a ripple-carry or a carry look-ahead. In contrast to ripple-carry counters, logic cells used in carry look-ahead counters can be non-adjacent. When the Compiler processes a carry look-ahead counter, it can arrange and permute the LEs to map the design into the device more efficiently.

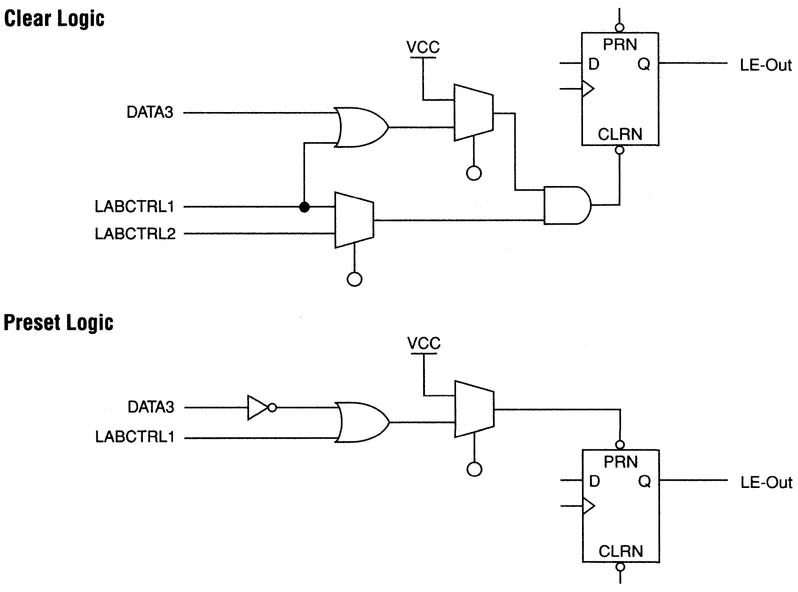
☞ Altera does not recommend using ripple-clocked counters, i.e., counters in which the output of one flipflop clocks another flipflop.

Altera recommends that you use carry chains only in the portions of a design that require maximum performance. You can choose between using ripple-carry and carry look-ahead counters on a case-by-case basis. In some cases, you may wish to trade the speed and silicon efficiency of a ripple-carry implementation for the increased routability and logic cell usage of a carry look-ahead implementation. For more information on counters, refer to *Application Brief 121* (*Designing Counters in FLEX 8000 Devices*) in this handbook.

Use Global Clock & Clear Signals

In FLEX 8000 devices, a programmable flipflop is used to support sequential functions. Sequential logic circuits are most reliable if they are fully synchronous, i.e., if every register in the design is clocked by the same global Clock signal and reset by the same global Clear signal. The FLEX 8000 architecture is optimized for this type of highly reliable, fully synchronous design. Four dedicated high-speed, low-skew global signals are available throughout each device, independent of the FastTrack Interconnect resources. Using these global signals for Clock and Clear functions will ensure a more reliable design and a much more efficient fit. Figure 1 shows the register control signals in FLEX 8000 devices.

Figure 1. Clear & Preset Logic



The Preset and Clear functions of the register can be functions of `LABCTRL1`, `LABCTRL2`, and `DATA3`. This structure is especially useful for sequential functions that require an asynchronous Clear with loading capability.

The asynchronous load (with or without a Clear input signal) and asynchronous Preset modes can be implemented within a single FLEX 8000 LE. Figure 2 shows an asynchronous load with a Clear input signal. Since the Clear signal has priority over the load signal, it does not need to feed the Preset circuitry.

Figure 2. Asynchronous Load with a Clear Input Signal

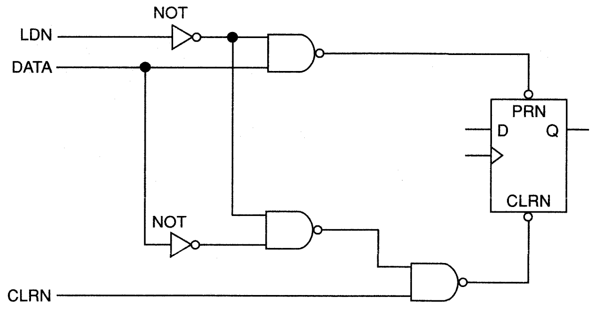


Figure 3 shows an asynchronous load without a Clear input signal.

Figure 3. Asynchronous Load without a Clear Input Signal

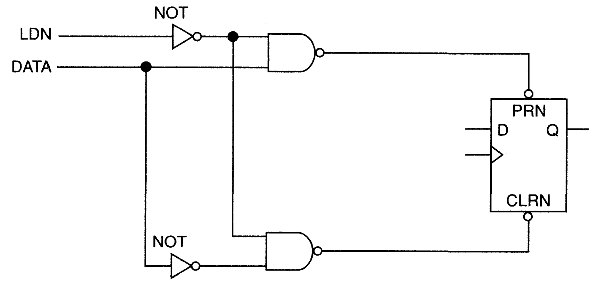
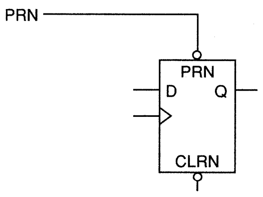


Figure 4 shows an asynchronous Preset signal. Asynchronous Preset signals are actually implemented as asynchronous loads in FLEX 8000 devices. The FLEX 8000 device loads a "1" into the register to implement a Preset. MAX+PLUS II uses the Clear input to the register for simple Preset signals, thus preserving the data input for use in the LUT while providing correct Preset functionality.

Figure 4. Asynchronous Preset



Use One-Hot State Bit Encoding

One-hot state bit encoding increases both the system speed and routability of a design. This type of encoding uses one register per state and allows only one state bit to be active at any time. Although one-hot encoding increases the number of registers, it also reduces the average fan-in to the state bits. This reduced fan-in minimizes the number of LEs required to implement the state decoding logic and yields a design that runs faster and uses less interconnect.

MAX+PLUS II automatically uses one-hot encoding when compiling state machines written in AHDL, VHDL, or Verilog HDL and targeted for FLEX 8000 devices. Altera also recommends using the options provided by other industry-standard CAE tools, such as the Mentor Graphics Autologic and Synopsys Design Compiler tools, to synthesize state machines described in VHDL or Verilog HDL with one-hot state bit encoding. For more information, refer to *Application Brief 131* (*State Machine Encoding*) in this handbook.

Use Pipelining for Complex Combinatorial Logic

Maintaining the system Clock speed at or above a certain frequency is often a major goal in a circuit design. For example, if you have a fully synchronous system that must run at 25 MHz, the longest delay path from the output of any register to the input(s) of the register(s) it feeds must be less than 40 ns. Maintaining system Clock speed can be difficult if some of the delay paths through the more complex logic are long. In these cases, Altera recommends pipelining complex blocks of combinatorial logic by inserting flipflops between combinatorial logic. Although pipelining may increase device resource usage, it lowers the propagation delay between registers and allows you to maintain high system Clock speeds.

The benefits of pipelining can be demonstrated with a 4-bit pipelined adder that adds two 4-bit numbers. This adder is based on two 2-bit adders that have outputs that are registered using D flipflops. Figure 5 shows one of the 2-bit pipelined adders. The function 2ADD is the 2-bit adder that feeds both sum bits (SUM1 and SUM2) and a carry bit (COUT) to the D flipflops in 4REG.

Figure 5. 2-Bit Pipelined Adder (2REGADD)

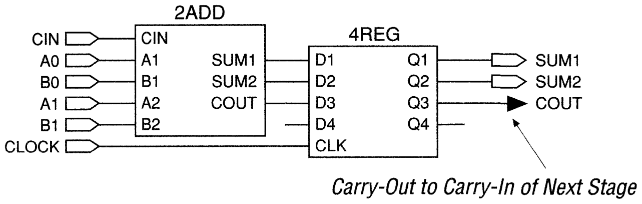
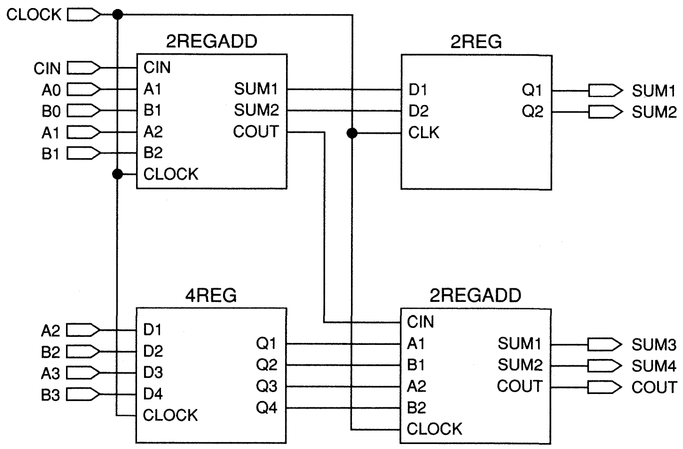


Figure 6 shows two 2-bit adders that are combined to form a 4-bit pipelined adder. The most significant bits (MSBs) of the 4-bit adder (A3, B3, A2, and B2) require the carry from the least significant bits (LSBs) for their sum. However, the MSB data inputs to the adder and the carry-in arrive at different times, due to the time it takes to generate the carry. Pipelining this design ensures that the MSBs are presented to the inputs of the adder at the same time as the carry-out signal from the previous stage. In Figure 6, the two sets of LSBs (A0, B0, A1, and B1) are added on the first Clock cycle, while bits A2, B2, A3, and B3 are added on the next Clock cycle. The outputs of 2REGADD are registered.

Figure 6. 4-Bit Pipelined Adder



Pipelining is most effective with register-intensive devices such as FLEX 8000 devices. While it can be used in product-term-based architectures such as those of the MAX 5000 and MAX 7000 devices, it may be less effective than in the FLEX 8000 architecture. Since each MAX 5000 and MAX 7000 logic cell has higher fan-in than a 4-input LUT, complex functions that require several FLEX 8000 LUTs may need only a single MAX 5000 or MAX 7000 logic cell.

Fitting Techniques

Occasionally, a design may require more interconnect resources than are available in the device. When a design does not fit, the MAX+PLUS II Compiler issues one or more error messages; up-to-date information on these error messages is available from on-line help. In many cases, it allows you to change compilation settings and pin and logic cell assignments or insert logic cells to adjust the fit during the compilation.

If the project does not fit after you have followed all of the design guidelines provided in this application note, you can use several techniques to help the Compiler fit the design:

- ❑ If you are willing to discard your pin assignments, you can allow the Compiler to automatically ignore all assignments, the minimum number of assignments possible, or specific individual assignments.

- ❑ If you wish to maintain your pin assignments, Altera recommends trying each of the following techniques, in order:

  1. Direct the Compiler to automatically insert logic cells between the design logic and the pins. Inserting logic cells gives the Compiler more fitting flexibility by separating device inputs and outputs from the design logic.
  2. Delete any logic cell assignments or allow the Compiler to ignore them.
  3. Allow the Compiler to ignore explicitly entered carry and cascade chain logic on a case-by-case basis or throughout the design.
  4. Break long carry chains by inserting logic cells into the chain.
  5. Redesign functions with long carry chains (e.g., adders and counters) with techniques such as carry look-ahead.
  6. Place input and bidirectional pins on column interconnects when possible.
  7. If an input pin has a high degree of fan-out, break the fan-out down by inserting `LCELL` primitives between the pin and some of its destinations.

Refer to MAX+PLUS II Help for additional information on entering pin, logic cell, and clique assignments; implementing carry and cascade logic; and adjusting the fit during compilation.

Conclusion

Altera has combined the strengths of both EPLDs and FPGAs into the FLEX 8000 architecture. Altera's MAX+PLUS II software allows you to quickly enter new designs or re-target existing designs for FLEX 8000 devices with design compilation that is automatically optimized for the FLEX 8000 architecture. In addition, MAX+PLUS II design entry methods offer detailed control over physical device implementation so that you can use your knowledge of the FLEX 8000 architecture to achieve the maximum speed and density for your designs.

High-Speed Board Designs

May 1994, ver. 1

Application Brief 119

Summary

To create a successful high-speed printed circuit board (PCB), you must integrate the device(s), board(s), and other elements into a coherent design. Altera devices typically provide 1- to 3-ns edge rates, which contribute to noise generation, signal reflection, cross-talk, and ground bounce. Therefore, your design must filter and evenly distribute power to all devices to reduce noise, terminate signal and transmission lines to diminish signal reflection, minimize cross-talk between parallel traces, and reduce the effects of ground bounce.

Power Filtering & Distribution

You can dramatically reduce system noise by providing clean, evenly distributed power to all boards and devices that is as close as possible to V_{CC}.

Filtering Noise

To diminish the low-frequency (< 1 kHz) noise caused by the power supply, you must filter the noise on the power lines at the point at which the power connects to the PCB, as well as at each device. Altera recommends placing a 100-µF electrolytic capacitor immediately adjacent to the location where the power supply lines enter the PCB. If you use a voltage regulator, place the capacitor immediately after the final stage that provides the V_{CC} signal to the device(s). Capacitors not only filter low-frequency noise from the power supply, but also supply extra current when many outputs switch simultaneously in a circuit.

The components on the PCB add high-frequency noise to the power plane. To filter high-frequency noise at the device, Altera recommends placing 0.02-µF or 0.2-µF decoupling capacitors as close as possible to each V_{CC} and GND pair. See *Operating Requirements for Altera Devices* in the current Altera ***Data Book*** for more information on bypass capacitors.

Distributing Power

Power distribution also has an impact on system noise. Power may be distributed throughout the board with either a power bus network or power planes.

A power bus network consists of two or more wide, metal traces that carry the V_{CC} and GND to the devices. Usually used on two-layer boards, power buses provide an inexpensive method of supplying power. The trace widths, which should be as wide as possible, are limited by the density of the board. Power buses have significant DC resistance; the last component on the bus may receive V_{CC} power that is degraded by as much as 0.5 V. Consequently, Altera recommends using power buses only for applications that do not require equal distribution of V_{CC}.

As an alternative, Altera recommends using power planes to distribute power. Power planes are used on multi-layer PCBs and consist of two or more metal layers that carry V_{CC} and GND to the devices. Because the power plane covers the full area of the PCB, its DC resistance is very low. The power plane maintains V_{CC} and distributes it equally to all devices. The power plane also provides near-infinite current-sink capability, noise protection, and shielding for the logic signals on the board.

Signal & Transmission Line Termination

Having established the PCB power network, you must consider the layout of the devices and traces. Fast edge rates contribute to noise, cross-talk, and ground bounce to varying degrees, depending on the PCB construction material.

Each PCB substrate has a different relative dielectric constant (E_r) that measures the effect of an insulator on the capacitance of a pair of conductors as compared to the capacitance of the conductor pair in a vacuum. The type of substrate used determines the length at which the signal traces must be handled as transmission lines. Table 1 lists E_r values for various dielectric materials.

Table 1. Relative Dielectric Constants

Material	E_r
Air	1.0
PTFE/Glass	2.2
Rogers RO 2800	2.9
CE/Goreply	3.0
BT/Goreply	3.3
CE/Glass	3.7
Silicon Dioxide	3.9
BT/Glass	4.0
Polymide/Glass	4.1
FR-4/Glass	4.1
Glass Cloth	6.0
Alumina	9.0

As shown in the following equation, the relative dielectric constant (E_r) of the material determines the velocity (V_P) at which signals may flow. The constant (C) equals 3×10^8 m/s or 30 cm/ns.

$$V_P = \frac{C}{\sqrt{E_r}}$$

The signal trace must be treated as a transmission line when the two-way propagation delay (PD) of the line exceeds the signal edge rate ($\mathbf{t_R}$). The propagation delay for MAX 7000 devices is the input to output ($\mathbf{t_{PD}}$), while the propagation delay for FLEX 8000 devices is either the transfer rate from I/O pin to I/O pin via row, LE, and column (t_1) or the transfer rate from I/O pin to I/O pin via row, LE, and row (t_2). See the following equation:

$$2 \times PD > \mathbf{t_R}$$

As shown in the following equation, the propagation delay (PD) is the length (l) of the line divided by the velocity (V_P):

$$PD = \frac{l}{V_P}$$

Solving for length (l) using the equation below, you can calculate the length at which the line must be treated as a transmission line:

$$l > \frac{\mathbf{t_R} \times C}{2\sqrt{E_r}}$$

As shown in Table 1, a PCB with glass cloth substrate has an E_r of 6. Table 2 lists the maximum line lengths for MAX 5000, MAX 7000, and FLEX 8000 devices using a glass cloth substrate under a 35-pF load.

Table 2. Maximum Line Lengths for Glass Cloth Substrate

Device Family	t_R (ns)	l (cm)	l (inches)
MAX 5000	2.7	16.52	6.50
MAX 7000	0.9	5.51	2.17
FLEX 8000	1.1	6.73	2.65

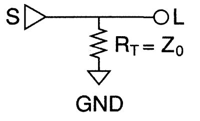
The impedance of the source (Z_S) must equal the impedance of the trace (Z_0) and the load (Z_L). Mismatched impedances cause signals to reflect back and forth and up and down the line, which causes ringing at the load.

The load impedance is typically much higher than the line impedance, which is higher than the source impedance. On an unmatched transmission line, a signal reflects 100% at the load and approximately 80% at the source, bouncing back and forth until it dies out. To reduce signal reflection, you can match the impedance either at the load (Z_L) or at the source (Z_S) to the line impedance (Z_0) by adding an impedance in parallel with the load to reduce its input impedance.

Termination Schemes

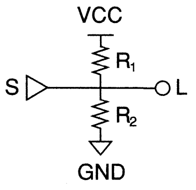
Parallel termination diminishes the first reflection by matching the load impedance to the line impedance. Of the four parallel termination circuits described, Altera recommends using either the Thevenin or resistor and capacitor (series-RC) scheme. For the matching to be effective, you must terminate each load, since any impedance mismatch will result in a signal reflection. As an alternative to parallel termination, you can use series termination, which matches the impedance at the signal source.

Simple Parallel Termination



In a simple parallel termination scheme, the terminating resistor (R_T) is equal to the line impedance. The current drain of this parallel termination scheme is excessive for the high-output state; the current drain may be as high as 48 mA for a 50-Ω termination. Since Altera devices are guaranteed to retain a 4-mA current, they cannot reliably support this termination scheme.

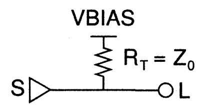
Thevenin Parallel Termination



$R_1 \parallel R_2 = Z_0$

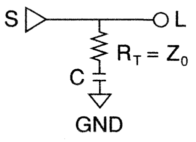
An alternative parallel termination scheme uses a Thevenin voltage divider. The terminating resistor is split between R_1 and R_2, which, when combined, equal the line impedance. Although this scheme reduces the current draw from the source device, it adds current draw from the power supply because the resistors are tied between V_{CC} and GND.

Active Parallel Termination



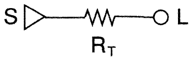
In an active parallel termination scheme, the terminating resistor ($R_T = Z_0$) is tied to a bias voltage (V_{BIAS}). The bias voltage is selected so that the output drivers are capable of drawing current from the high- and low-level signals. However, this scheme requires a separate voltage source that can sink and retain currents to match the output transfer rates.

Series-RC Parallel Termination



In a parallel termination scheme, a resistor and capacitor (series-RC) network is used as the terminating impedance. The terminating resistor (R_T) is equal to Z_0; the capacitor must be greater than 100 pF. The capacitor blocks low-frequency signals while passing high-frequency signals. Therefore, the DC loading effect of R_T does not impact the driver.

Series Termination



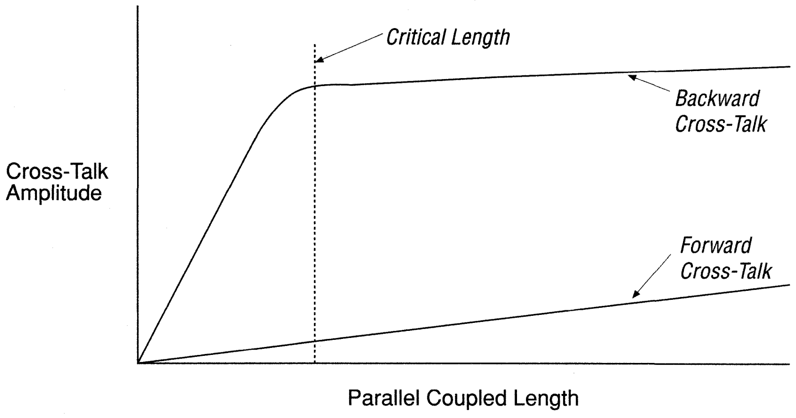
A series termination scheme matches the impedance at the signal source instead of matching the impedance at each load. Because the output impedance of Altera devices is low, you must add a series impedance to match the signal source to the line impedance.

On an unmatched line, the source eventually reduces the reflections; adding the series termination helps attenuate secondary reflections. The source impedance varies from 10 Ω to 18 Ω, and the line impedance varies depending on the distribution of the load. Therefore, you cannot choose a signal resistor value that applies to all conditions. Altera recommends using a 33-Ω series resistor to cover most impedances. This method requires only a single component at the source rather than multiple components at each load, but delays the signal path as it increases the RC time constant.

Cross-Talk

Cross-talk is the unwanted coupling of signals between parallel traces. Two types of cross-talk exist: forward (capacitive) and backward (inductive). Figure 1 illustrates the effect of each type of cross-talk as a function of the parallel length.

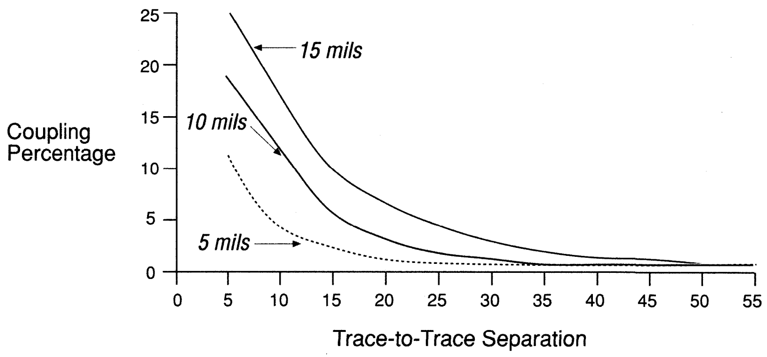
Figure 1. Cross-Talk as a Function of Parallel Length



Backward cross-talk, which has a more dramatic effect than forward cross-talk, occurs when the magnetic field from one trace induces a signal in a neighboring trace. In logic systems, the current flow through a trace is significant when the signals are switching or non-static. The magnetic fields created by switching currents induce the coupling transients.

You can dramatically reduce cross-talk by limiting the trace height to 10 mils above the GND plane. Figure 2 shows the effect of trace height on trace-to-trace coupling.

Figure 2. Effect of Trace Height on Trace-to-Trace Coupling



Ground Bounce

As digital devices become faster, their output switching times decrease. Faster switching times cause higher transient currents in outputs as they discharge load capacitances. These higher currents, which are generated when multiple outputs of a device switch simultaneously from a logic high to a logic low, can cause a board-level phenomenon known as ground bounce.

Many factors affect the magnitude of ground bounce. Therefore, no standard test method allows you to predict its magnitude for all possible board environments. You can only test the device under a given set of conditions to determine the relative contributions of each condition and of the device itself. Load capacitance, socket inductance, and the number of switching outputs are the predominant factors that influence the magnitude of ground bounce in programmable logic devices.

Design Recommendations

Altera recommends that you take the following steps to reduce the magnitude of ground bounce:

- ❑ Limit load capacitance by buffering loads with an external device such as the 74244 IC bus driver or by reducing the number of devices that drive the bus.
- ❑ Eliminate sockets whenever possible.
- ❑ Reduce the number of outputs that can switch simultaneously and/or distribute them evenly throughout the device.
- ❑ Move switching outputs close to a package ground pin.
- ❑ Eliminate pull-up resistors, or use pull-down resistors.
- ❑ Use multi-layer boards that provide separate V_{CC} and GND planes.
- ❑ Add 10- to 30-Ω resistors in series to each of the switching outputs to limit the current flow into each of the outputs.
- ❑ Turn on the Slow Slew Rate logic option for FLEX 8000 and MAX 7000E designs.

Go to "Slow Slew Rate" using **Search for Help on** (Help menu) in MAX+PLUS II for more information on this option.

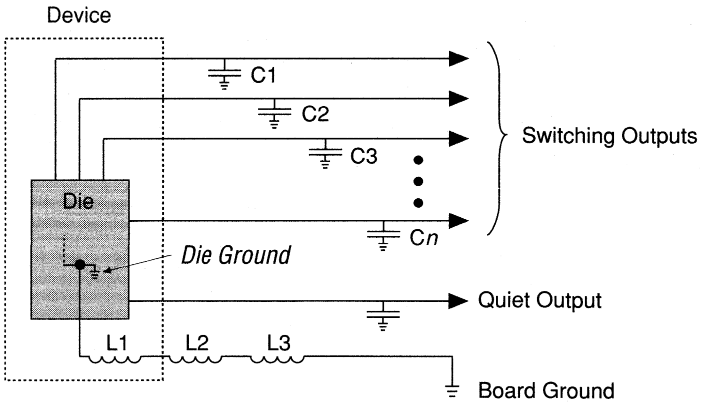
These design practices, many of which are described in detail in this application brief, should help you create effective high-speed logic designs that operate over a wide range of board conditions.

Analyzing Ground Bounce

Figure 3 shows a simple model for analyzing ground bounce. The external components driven by the device appear to that device as capacitive loads (`C1` to `C`*n*). These capacitive loads store a charge that is determined by the following equation:

charge (Q) = [voltage (V) × capacitance (C)]

Figure 3. Ground Bounce Model



Thus, the charge increases as the voltage and/or load capacitance increases.

The environment and ground path of a device have intrinsic inductances (shown in Figure 3 as L1, L2, and L3). L1 is the inductance of the bond wire from the device's die to its package pin, and of the pin itself. L2 is the inductance of the connection mechanism between the device's ground pin and the PCB. This inductance is greatest when the device is connected to the board through a socket. L3 is the inductance of the PCB trace between the device and the board location where other devices in the system reference ground.

Ground bounce occurs when multiple outputs switch from high to low. The transition causes the charge stored in the load capacitances to flow into the device. The sudden rush of current (di/dt) exits the device through the inductances (L) to board ground, generating a voltage (V) determined by the equation $V = L \times (di/dt)$. This voltage difference between board ground and device ground causes the relative ground level for low, or quiet, outputs to temporarily rise, or bounce. Although the rush of current is brief, the magnitude of the bounce can be large enough to trigger other devices on the board.

In synchronous designs, ground bounce is less often a problem because synchronous outputs have enough time to settle before the next clock edge. Also, synchronous circuits are not as likely to be falsely triggered by a voltage spike on a quiet output.

The magnitude of ground bounce is affected differently by capacitive loading on the switching outputs and quiet outputs.

Switching Outputs

When the capacitive loading on the switching outputs increases, the amount of charge available for instantaneous switching increases, which in turn increases the magnitude of ground bounce. Depending on the device, ground bounce increases with capacitive loading until the loading is approximately 200 pF per device output. At this point, the device output buffers reach their maximum current-carrying capacity and inductive factors become dominant.

One method of reducing the capacitive load and hence ground bounce is to connect the device's switching outputs to a bus driver IC. The outputs of the bus driver IC drive the heavy capacitive loads, reducing the loading on the device, thus minimizing ground bounce for the device's quiet outputs.

Some bus applications use pull-up resistors to create a default high value for the bus. These resistors cause the load capacitances to charge up to the maximum voltage. Consequently, the driving device produces a higher level of ground bounce. Therefore, you should eliminate pull-up resistors in applications in which ground bounce is a concern, or design a bus logic that uses pull-down resistors instead.

The number of switching outputs also affects ground bounce. As the number of switching outputs increases, the total charge stored also increases. The total charge is equal to the sum of the stored charges for each switching output. Therefore, the amount of current that must sink to ground increases as the number of switching outputs increases. Ground bounce can increase by as much as 40 mV to 50 mV for each additional output that is switching.

To counteract these effects, Altera devices provide multiple V_{CC} and GND pin pairs. You can reduce ground bounce by moving switching outputs close to a package GND pin, and by distributing simultaneously switching outputs throughout the device.

To further reduce ground bounce, limit the number of outputs that can switch simultaneously in your design to eight or fewer. For functions such as counters, you can use Gray coding as an alternative to standard sequential binary coding, since only one bit switches at a time.

In extreme cases, adding resistors (10 to 30 Ω is usually adequate) in series to each of the switching outputs in a high-speed logic device can limit the current flow into each of the outputs, and thus reduce ground bounce to an acceptable level.

Quiet Outputs

An increase in capacitive loading on quiet outputs acts as a low-pass filter and tends to dampen ground bounce. Capacitive loading on a quiet output can reduce the magnitude of ground bounce by as much as 200 to 300 mV. However, an increase in capacitive loading on a quiet output can increase the noise seen on other quiet outputs.

Minimizing Lead Inductance

Socket usage and board trace length are two elements of L2, the inductance of the connection mechanism between the device's ground pin and the PCB shown in Figure 3. Sockets can cause ground bounce voltage to increase by as much as 100%. You can often dramatically reduce the magnitude of ground bounce on the board by eliminating sockets.

The length of the board trace has a much smaller effect on ground bounce than sockets. For PCBs with a ground plane, the voltage drop across the inductance (L3) of the PCB trace between the device and the board location where other devices in the system reference ground is negligible, because L3 is significantly less than L2. The inductance in a 3-inch trace increases ground bounce for a quiet output by approximately 100 mV. Nevertheless, trace length should be kept to a minimum. As traces become longer, transmission line effects may cause other noise problems.

You can also reduce ground bounce due to PCB trace inductance by using multi-layer boards that provide separate V_{CC} and GND planes. Wire-wrapping the V_{CC} and GND supplies usually increases the amount of ground bounce. To reduce unwanted inductance, you should use low-inductance bypass capacitors between the V_{CC} supply pins and the board GND plane, as close to the package supply pins as possible. A standard decoupling capacitor (0.02 to 0.2 mF) used in parallel with a high-frequency decoupling capacitor (470 pF is a standard value) gives the best results.

References

Advanced Micro Devices, Inc. *High-Speed-Board Design Techniques.* Sunnyvale: Advanced Micro Devices, Inc., 1992.

Knack, Kella. *Debunking High-Speed PCB Design Myths.* ASIC & EDA, Los Altos: James C. Uhl, July 1993.

Contents

May 1994

Section 4 Counter Applications

Designing Counters in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 121

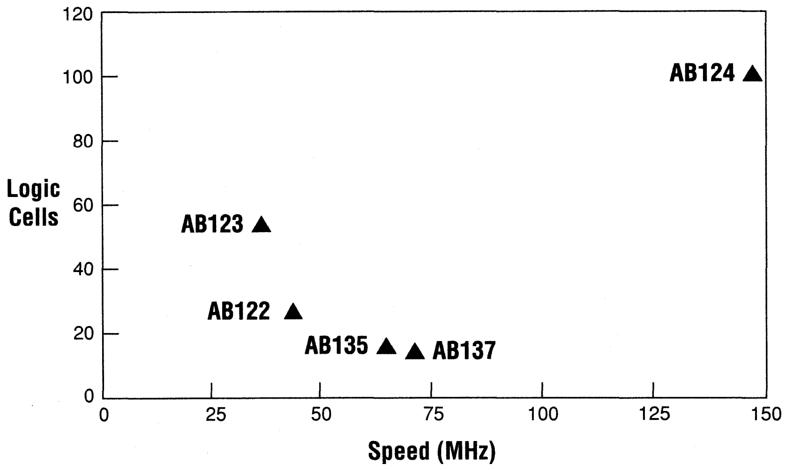
Introduction

FLEX 8000 devices feature look-up table (LUT) architecture and logic elements (LEs) that allow you to design counters optimized for speed, area, or routability. FLEX 8000 LEs offer dedicated carry chain and cascade chain logic that efficiently implements fast counters. This application brief summarizes some of the design techniques Altera recommends to take advantage of these architectural features.

Design Trade-Offs

The best counter to use for a particular FLEX 8000 design depends on the target application. Most programmable logic designs must make trade-offs between speed, area, and routability. The application briefs in this section describe different design techniques that you can use to optimize counters for your own application requirements. Figure 1 summarizes the advantages of the different counter design techniques.

Figure 1. Design Techniques for Counters



AB137	*Ripple-Carry Counters in FLEX 8000 Devices*
AB135	*Ripple-Carry Gray Code Counters in FLEX 8000 Devices*
AB122	*Carry Look-Ahead Counters in FLEX 8000 Devices*
AB123	*Pipelined Carry Look-Ahead Counters in FLEX 8000 Devices*
AB124	*Prescaled Counters in FLEX 8000 Devices*

If high-speed performance is your primary goal, you can create ripple-carry counters that take advantage of the dedicated carry chain, which offers a very fast (less than 1 ns) carry-forward function between contiguous LEs and contiguous Logic Array Blocks (LABs). Ripple-carry counters also use the fewest LEs per counter bit of all types of counters. Ripple-carry Gray code counters offer high-speed performance and reduce switching noise effects. However, the ripple-carry design approach is recommended primarily for counters containing no more than 16 bits, because carry chains must be placed in contiguous LEs (and, if longer than 8 bits, in contiguous LABs). Longer carry chains may reduce the routing resources available for implementing other logic. Prescalar counters use more LEs per counter bit than ripple-carry counters, but offer another path to high-speed performance by taking advantage of the 1-ns local interconnect delay within FLEX 8000 LABs. The prescalar design approach is most suitable for counters containing up to 16 bits.

If easy fitting and/or routability is your primary goal, carry look-ahead counters offer routing flexibility and good speed performance. By using additional LE resources, carry look-ahead counters can also be pipelined to increase their operating frequency. The carry look-ahead (and pipelined carry look-ahead) design approach is well suited for counters containing 16 or more bits. With forethought and creativity, you can implement counters efficiently in FLEX 8000 devices, regardless of the constraints imposed by any particular application.

☞ For detailed information on LE architecture and carry and cascade chain logic, refer to *Application Note 40 (FLEX 8000 Architecture)* and *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook.

Ripple-Carry Counters
in FLEX 8000 Devices

May 1994, ver. 1 Application Brief 137

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_137.exe

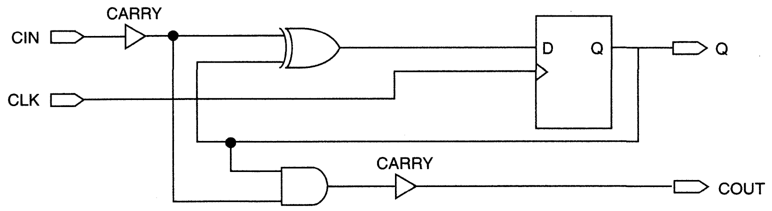
Ripple-carry counters take advantage of the dedicated carry chain feature available in FLEX 8000 devices. The carry chain in FLEX 8000 devices is a fast (less than 1 ns) carry-forward function path between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. Ripple-carry counters provide high-speed carry generation and use logic element (LE) resources efficiently. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		**Design Results:**		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	8	125
Product Term	✓ Speed	16 Bits	16	71
	✓ Area	32 Bits	32	42

Single-Bit Ripple-Carry Counters

You can create ripple-carry counters by using the FLEX 8000 LE Up/Down Counter mode, which makes them easier to create than other types of counters. The Up/Down Counter mode implements counter logic with both the sum and carry generation logic in a single LE, which is the most efficient implementation of counter logic. For more information about the Up/Down Counter mode, see *Application Note 40 (FLEX 8000 Architecture Details)* in this handbook. Figure 1 shows a single bit of a ripple-carry counter that is implemented within one LE.

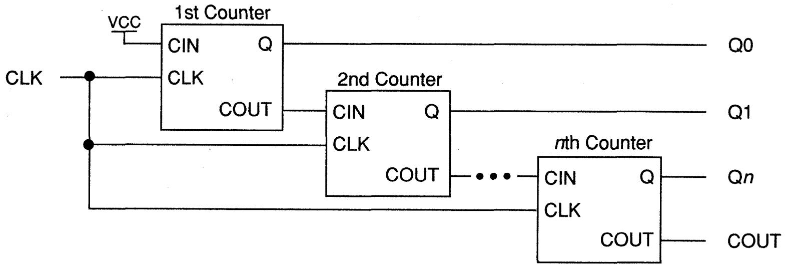
Figure 1. Single-Bit Ripple-Carry Counter



N-Bit Ripple-Carry Counters

You can implement *n*-bit ripple-carry counters of any desired width by copying the single-bit design shown in Figure 1 and connecting the copies to form longer chains. See Figure 2.

Figure 2. N-Bit Ripple-Carry Counter



Although ripple-carry counters offer the best performance per LE, you should use ripple-carry counters only for widths up to 16 bits. Since carry chains must be placed in contiguous LEs and Logic Array Blocks (LABs), longer carry chains may reduce the routing resources available for implementing other logic. However, you can improve the routability of large ripple-carry counters by inserting `LCELL` buffers to break the carry chain or by using more advanced counter designs, such as the pipelined carry look-ahead counters described in *Application Brief 123 (Pipelined Carry Look-Ahead Counters in FLEX 8000 Devices)* in this handbook.

Ripple-Carry Gray Code Counters in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 135

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_135.exe

Ripple-carry Gray code counters are counters in which only one bit switches at a time. Ripple-carry Gray code counters take advantage of the dedicated carry chain feature available in FLEX 8000 devices. A carry chain is a fast (less than 1 ns) carry-forward function path between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. Ripple-carry Gray code counters provide high-speed carry generation and use LE resources efficiently. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		**Design Results:**		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	9	103
Product Term	✓ Speed	16 Bits	17	63
	✓ Area	32 Bits	33	35

Ripple-Carry Gray Code Counters vs. Binary Counters

Ripple-carry Gray code counters have less system-level switching noise and, consequently, less ground plane noise than binary counters have, because multiple bits may switch simultaneously in binary counters. In addition, the output of a ripple-carry Gray code counter can be asynchronously sampled at any time, with a maximum sampling error of one. In contrast, if a binary counter is sampled at or near the Clock edge, some of the flipflops may have transitioned before others, which can cause errors with the asynchronous sample.

Ripple-Carry Gray Code Counters

You can implement a ripple-carry Gray code counter with T flipflops (`TFF` primitives), and include an additional `TFF`, called the "dummy" bit, with its `T` input tied high. The following equation calculates Q_n`.T`, where `Q` is the counter inputs and `T` is the input to the register:

```
(Qn-1..0, dummy) = B"100...001"
```

Table 1 shows the detailed counter bit pattern for a 4-bit ripple-carry Gray code counter. For example, `Q2` switches when `Q1=1`, `Q0=0`, and `dummy=1`. `Q3` switches when `Q3` or `Q2` is high, `Q1` and `Q0` are low, and `dummy` is high.

Table 1. 4-Bit Ripple-Carry Gray Code Counter Bit Pattern

Count Value	Bit Pattern for Q[3..0]	Dummy Bit
0	0000	0
1	0001	1
2	0011	0
3	0010	1
4	0110	0
5	0111	1
6	0101	0
7	0100	1
8	1100	0
9	1101	1
10	1111	0
11	1110	1
12	1010	0
13	1011	1
14	1001	0
15	1000	1

As shown in Table 1, the least significant bit (LSB) toggles when the `dummy` register is low. The highest possible value for a ripple-carry Gray code counter occurs when the most significant bit (MSB) is `1` and all others are `0`. The MSB must toggle when (Q_n `$` Q_{n-1}) and ((Q_{n-1}`..0, dummy) = B"000...001")`, which causes the MSB to switch to zero, rolling the counter over from the maximum value to zero.

You can adapt the Up/Down Counter mode available in FLEX 8000 LEs for use with ripple-carry Gray code counters. For more information about the Up/Down Counter mode, see *Application Note 40 (FLEX 8000 Architecture Details)* in this handbook. In a binary counter, the carry chain is used to propagate the AND of all preceding bits. However, in a Gray code counter, Q_n toggles when `Q[(`n`-2)..0] = 0` and the `dummy` bit is high. The carry chain propagates the signal.

Instead of propagating the signal when all the LSBs are high, the carry signal goes high when the `dummy` bit is high and all other bits are low. For example, the carry-out of the fifth logic cell is as follows: `!Q4 & !Q3 & !Q2 & !Q1 & !Q0 & dummy`.

In the Up/Down Counter mode, the output of the register is fed directly into the look-up table (LUT), emulating a `TFF`. The Up/Down Counter mode supports a synchronous load, a synchronous Clear, or an Output

Enable signal. To use a synchronous Clear signal, you must connect the load inputs to GND. To use an Output Enable signal, you must feed the counter outputs back to the load inputs. You can implement an asynchronous load without increasing LE usage. See *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook for more information.

Figure 1 shows the general Altera Hardware Description Language (AHDL) implementation of a ripple-carry Gray code counter.

Figure 1. AHDL Implementation of a Ripple-Carry Gray Code Counter

```
count0.d        = ldn &(!dummy $ count0) # (!ldn & data0);
countcarry0     = dummy;

count1.d        = ldn & ((count0 & countcarry0) $ count1)# (!ldn & data1);
countcarry1     = countcarry0 & !count0;

count2.d        = ldn & ((count1 & countcarry1) $ count2)# (!ldn & data2);
countcarry2     = countcarry1 & !count1;
```

The counter bits are implemented as D flipflops (DFF primitives), which use the D flipflop structure of the registers in the FLEX 8000 device. When the carry signals that feed the register and the previous bit are both high, each register is toggled (XORed with itself). The carry chain for each LE is the carry signal feeding that LE, ANDed with the inverse of the previous bit. Hence, the carry chain computes the dummy bit ANDed with the inverse of all counter bits.

Designs that use the FLEX 8000 carry chain run relatively fast and use the fewest number of LEs ($n + 1$). However, since carry chains must be placed in contiguous LEs and LABs, longer carry chains may reduce the routing resources available for implementing other logic.

For ripple-carry counters that are not speed-critical, it is not necessary to use the carry chain. However, ignoring the carry chain increases the design's LE usage and decreases its speed.

Notes:

Carry Look-Ahead Counters in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 122

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_122.exe

Counters designed with carry look-ahead techniques offer routing flexibility and medium speed performance. Carry look-ahead counters can take advantage of the FLEX 8000 dedicated carry chain feature, but do not require it. The dedicated carry chain feature increases speed and provides efficient logic element (LE) utilization. However, since carry chains must be placed in contiguous LEs and Logic Array Blocks (LABs), longer carry chains may reduce the routing resources available for implementing other logic. For this reason, Altera recommends that large counter designs use carry look-ahead techniques that bypass the dedicated carry chain feature. This application brief describes a medium-performance, loadable up/down counter that does not use the dedicated carry chain feature. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

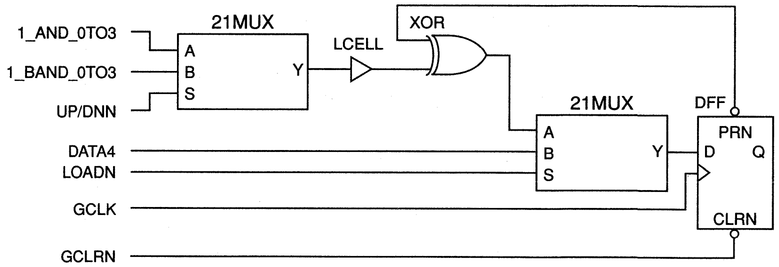
Design Goals:		**Design Results:**		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	✓ Routability	8 Bits	19	34
Product Term	Speed	16 Bits	43	25
	Area	32 Bits	–	–

Carry Look-Ahead Counters

The technique used to construct the carry look-ahead counter for the FLEX 8000 device architecture is similar to the technique used for the product-term architectures of Altera's MAX 5000 and MAX 7000 device families. In both architectures, a T flipflop (TFF) is used for each bit of the counter. The input to the TFF is a wide-input AND gate that ANDs together all of the least significant bits (LSBs) in the counter. Each bit toggles when all of the LSBs is high. If up/down functionality is required, a second AND gate is required for each bit. The second AND gate ANDs the complement of each of the LSBs. The up/down signal multiplexes the two AND gates into the TFF.

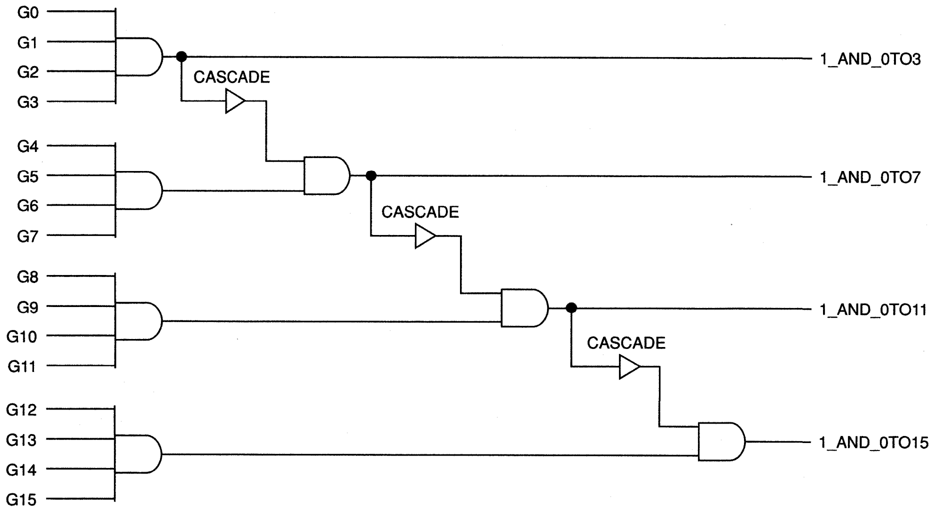
In the FLEX 8000 device architecture, a TFF is emulated using a D flipflop (DFF primitive) in conjunction with an XOR gate. The XOR gate and the load and data signals for each counter bit fit efficiently into a single LE. Figure 1 shows a single bit of a carry look-ahead counter for a FLEX 8000 device. In this sample counter bit, load data (DATA4) has been multiplexed into the DFF to implement a synchronous load.

Figure 1. Single-Bit Carry Look-Ahead Counter with Synchronous Load



The wide-input AND gates in this type of carry look-ahead counter are created using the dedicated cascade chain feature in the FLEX 8000 architecture. For more information on cascade chains, refer to *Application Note 40 (FLEX 8000 Architecture)* in this handbook. Figure 2 shows a 16-input AND gate with a cascade chain that illustrates how the wide-input terms (`1_AND_0to3`, `1_AND_0to7`, `1_AND_0to11`, and `1_AND_0to15`) in Figure 1 are generated. The 16-input AND gate decodes the LSBs of the counter and toggles the counter bit when all of the LSBs are high.

Figure 2. 16-Input AND Gate with Cascade Chain



ALTERA

Pipelined Carry Look-Ahead Counters in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 123

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_123.exe

You can create carry look-ahead counters that are pipelined to increase operating frequency. Pipelining consists of inserting flipflops between combinatorial logic, which decreases register-to-register delays and increases operating frequency. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		***Design Results:***		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	✓ Routability	8 Bits	14	80.0
Product Term	✓ Speed	16 Bits	36	53.5
	Area	32 Bits	–	–

Pipelined Carry Look-Ahead Counters

Table 1 shows the counter sequence for a 4-bit pipelined carry look-ahead counter, which illustrates the basis of the carry look-ahead technique. For more information on carry look-ahead counters, see *Application Brief 122 (Carry Look-Ahead Counters in FLEX 8000 Devices)* in this handbook. For more information on pipelining, see *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook.

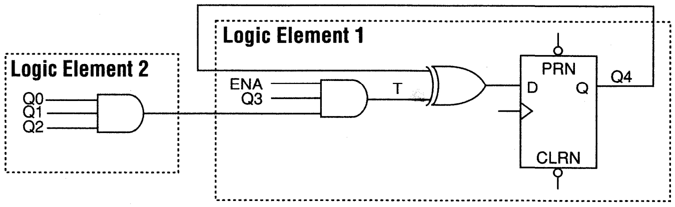
Table 1. Count Sequence for a 4-Bit Pipelined Carry Look-Ahead Counter

Q[3..0]	Carry Look-Ahead	Bit
0000	0	0
0001	0	0
0010	0	0
0011	0	0
0100	0	0
0101	0	0
0110	1	0
0111	0	1
1000	0	0

Each bit in the counter toggles only when all the least significant bits (LSBs) are high. If you use a T flipflop (`TFF` primitive) for each bit, the logic used to decode the `T` input is a wide-input AND gate with all of the LSBs as inputs. Since the logic element (LE) in a FLEX 8000 device is based upon a 4-input look-up table (LUT), the sixth bit would require an AND gate with more inputs than a single LE can accommodate. If the counter has an Enable input, the fourth bit requires more than four inputs.

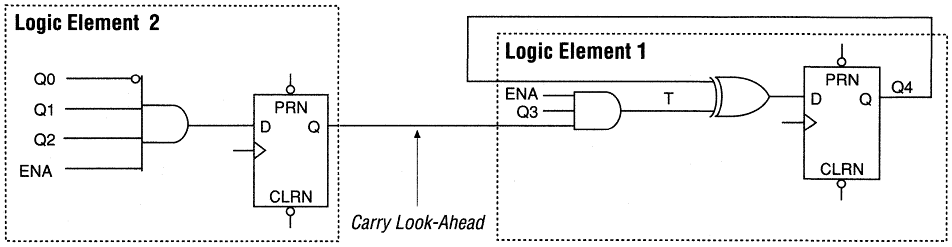
To reduce the number of inputs, you can split the wide AND gates in the counter among several LEs, as shown in Figure 1. However, splitting the AND gates among several LEs increases the combinatorial register-to-register delay and reduces the maximum operating frequency.

Figure 1. AND Gates Split Among Logic Elements



You can increase the operating frequency by pipelining the carry look-ahead counter, which decreases the register-to-register delay. Figure 2 shows a pipelined version of the carry look-ahead counter shown in Figure 1. In a binary count sequence, the fifth bit, `Q4`, toggles only when `Q[3..0] = 1111`. When the ANDing of `Q[3..0]` is true, the `T` node goes high, causing the register to toggle. In Figure 2, the pipelining causes `Q1` and `Q0` to be delayed by one Clock cycle. To account for this latency, the carry look-ahead register ANDs `Q1` with `Q0` one count cycle early, i.e., when `Q[3..0] = 1110`.

Figure 2. Sample Pipelined Carry Look-Ahead Counter



ALTERA

Prescaled Counters
in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 124

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_124.exe

Prescaled counters are counters that are specifically designed for high-frequency counting. Prescaled counters achieve their high performance because only the least significant bits (LSBs) transition at higher frequencies, while the most significant bits (MSBs) have more time to prepare for their transitions. FLEX 8000 devices can implement prescaled counters at frequencies of up to 142.9 MHz for counters of up to 16 bits. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		**Design Results:**		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	37	142.9
Product Term	✓ Speed	16 Bits	101	142.9
	Area	32 Bits	–	–

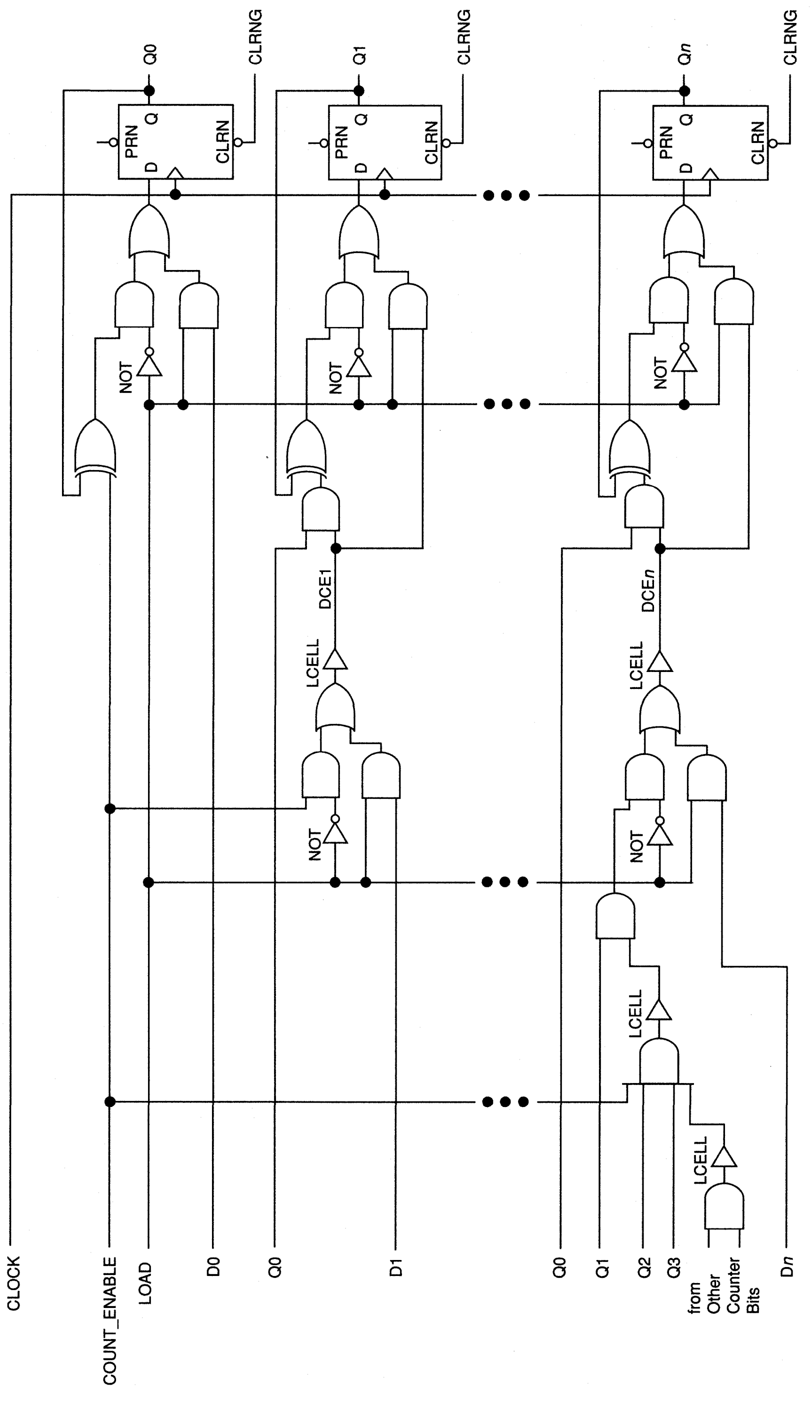
Prescaled Counters

Figure 1 shows a single bit of a prescaled counter implemented in a FLEX 8000 device. When binary counters are clocked, each counter bit is driven low if all of its LSBs are high. However, since all but the LSB will have been high in the previous Clock cycle, the performance-critical path is from the LSBs (Q0) to all other register bits. In fact, the MSBs will have been high for several Clock cycles.

To achieve the maximum possible counting frequencies, the loading of prescaled counters occurs outside of the speed-critical path. With this implementation, synchronous loading of a prescaled counter requires multiple Clock cycles. Loading functions are typically not required but are frequently used to divide very-high-speed incoming Clocks.

To achieve the highest performance, the two LSBs (Q0 and Q1) are repeated in each Logic Array Block (LAB) of this counter. By replicating these bits, the only interconnect delay in the performance-critical path is the local LAB interconnect of 1 ns. Thus, the speed-critical path for each bit requires only 7 ns (or 142.9 MHz). When fast Clock speed is the most important consideration, you can use additional logic resources to ensure the maximum possible performance.

Figure 1. Single-Bit Prescaled Counter



Contents

Section 5 Arithmetic Applications

Designing Adders, Accumulators & Subtractors in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 125

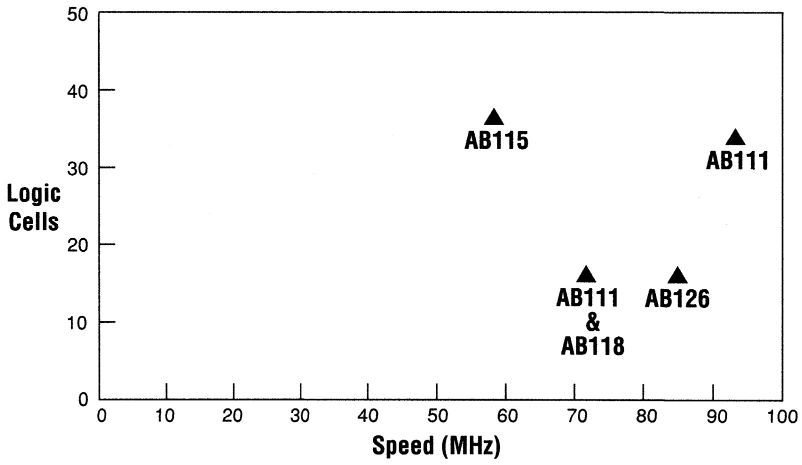
Introduction

FLEX 8000 devices feature look-up table (LUT) architecture and logic elements (LEs) that allow you to design register-intensive functions such as adders, accumulators, and subtractors. FLEX 8000 LEs also offer dedicated carry chain logic that efficiently implements fast adders, accumulators, and subtractors.

Design Trade-Offs

The best adder, accumulator, or subtractor for a particular FLEX 8000 design depends on the target application. Most programmable logic designs must make trade-offs between speed, area, and routability. The application briefs in this section describe different design techniques that you can use to optimize adders, accumulators, and subtractors for your own application requirements. Figure 1 summarizes the advantages of the different adder, accumulator, and subtractor design techniques.

Figure 1. Design Techniques for Adders, Accumulators, & Subtractors



AB118	*Ripple-Carry Adders in FLEX 8000 Devices*
AB115	*Carry-Select Adders in FLEX 8000 Devices*
AB111	*Accumulators in FLEX 8000 Devices*
AB126	*Subtractors in FLEX 8000 Devices*

If speed is your primary goal, you can create ripple-carry adders or adder-type functions that take advantage of the dedicated carry chain in FLEX 8000 LEs. The carry chain offers a very fast (less than 1 ns) carry-forward function between contiguous LEs within a Logic Array Block (LAB) and between contiguous LABs. Ripple-carry adders also use fewer LEs per adder bit than other types of adders. However, the ripple-carry design approach is best suited for adders with no more than 16 bits because carry chains must be placed in contiguous LEs and, if longer than 8 bits, in contiguous LABs. Longer carry chains may reduce the routing resources available for implementing other logic. Pipelined ripple-carry accumulators use more LEs per bit than non-pipelined ripple-carry accumulators, but offer enhanced speed performance and improved routability. The pipelined design approach is suitable for accumulators containing more than 8 bits.

If easy fitting and/or routability is your primary goal, you can create carry-select adders that use short carry chains and parallel computation. Carry-select adders use more LEs per adder bit than ripple-carry adders, but provide maximum flexibility during the design fitting process. Additionally, in adder designs that contain more than 8 bits, the parallel computation used in carry-select adders offers faster performance than ripple-carry adders because it reduces carry-generation delays. The carry-select design approach is suitable for adders containing more than 8 bits.

With forethought and creativity, you can implement functions efficiently in FLEX 8000 devices, regardless of the constraints imposed by any particular application.

☞ For detailed information on LE architecture and carry chain logic, refer to *Application Note 40 (FLEX 8000 Architecture)* and *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook.

ALTERA

Ripple-Carry Adders in FLEX 8000 Devices

May 1994, ver. 2 Application Brief 118

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_118.exe

Ripple-carry adders that use the carry chain feature in a FLEX 8000 device are the easiest adders to implement in the FLEX 8000 architecture. A carry chain is a fast (less than 1 ns) carry-forward function path between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. Ripple-carry adders also offer the best performance per LE. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		Design Results:		
Architecture	Optimization	Width	Logic Cells	Speed (MHz)
✓ Look-Up Table	Routability	8 Bits	8	125
Product Term	✓ Speed	16 Bits	16	71
	✓ Area	32 Bits	32	42

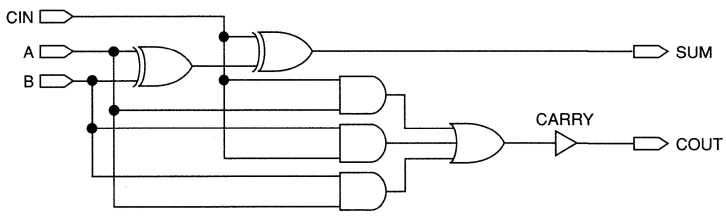
Ripple-Carry Adder

Ripple-carry adders are implemented using the FLEX 8000 Arithmetic mode. In this mode, both the sum and carry generation logic for each bit of the adder are implemented within a single LE. The Arithmetic mode offers two 3-input look-up tables (LUTs) that are ideal for implementing adders and accumulators. One LUT provides a 3-bit function (SUM); the other generates a carry-out (COUT). For more information on LE operating modes, see *Application Note 40 (FLEX 8000 Architecture)* in this handbook.

Single-Bit Adders

A single-bit adder has three inputs: two data inputs (A and B) and a carry-in (CIN) signal; it generates a sum and a carry-out. Figure 1 shows the implementation of a single bit of a full adder in a single LE.

Figure 1. Adder Bit Implemented in One LE



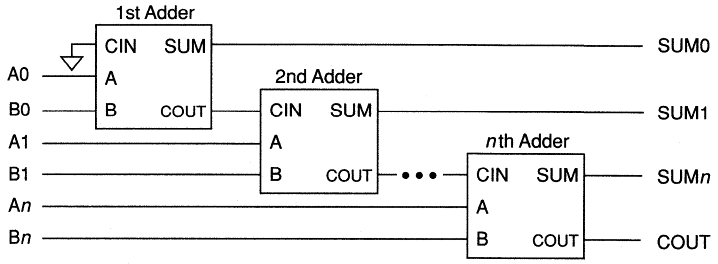
The following Altera Hardware Design Language (AHDL) equations describe a single-bit full adder:

```
sum  = a $ (b $ cin);
cout = carry((a & b) # (a & cin) # (b & cin));
```

Implementing N-Bit Ripple-Carry Adders

You can implement *n*-bit ripple-carry adders of any desired width by interconnecting full adder bit-slices to form longer chains. High-speed adders of arbitrary width can be implemented in one LE per sum-bit. Figure 2 shows how an *n*-bit combination adder is built from the single-bit adder shown in Figure 1.

Figure 2. N-Bit Combination Adder



Although ripple-carry adders offer the best performance per LE, they are recommended only for widths of up to 16 bits. Since carry chains must be placed in contiguous LEs and LABs, longer carry chains may reduce the routing resources available for implementing other logic. However, you can improve the routability of large ripple-carry adders by pipelining the design or by breaking the carry chain with an `LCELL` buffer.

ALTERA

Carry-Select Adders
in FLEX 8000 Devices

May 1994, ver. 2 Application Brief 115

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_115.exe

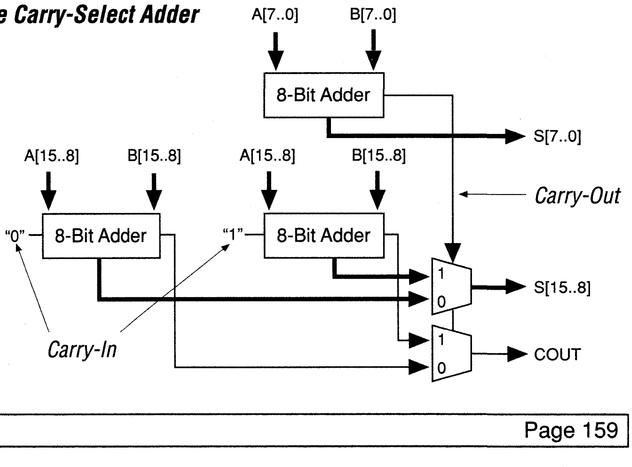
Carry-select adders in FLEX 8000 devices use short carry chains and parallel computation to produce fast designs that are very easy for the MAX+PLUS II Compiler to route. The carry chain feature in a FLEX 8000 device is a fast (less than 1 ns) carry-forward function path between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. When a FLEX 8000 design requires a larger adder, carry-select adders provide better routability and performance than ripple-carry adders. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		*Design Results:*		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	✓ Routability	8 Bits	–	–
Product Term	✓ Speed	16 Bits	37	59.0
	Area	32 Bits	69	36.8

Carry-Select Adders

A carry-select adder consists of three separate adders. Using a 16-bit adder as an example, one adder computes the least significant byte; two adders compute the most significant byte. These two adders consist of one adder with a carry-in of `1` and another with a carry-in of `0`. The carry-outs of the least significant byte and most significant byte are computed simultaneously, then a 2-to-1 multiplexer selects the correct result, which has been computed previously. Figure 1 shows a block diagram for a 16-bit carry-select adder.

Figure 1. Simple Carry-Select Adder



Carry-Select Adders vs. Ripple-Carry Adders

In ripple-carry adders, all carry chains must be placed in contiguous LEs and LABs. Therefore, long carry chains may reduce the routing resources available for other logic. Since carry-select adders use carry chains that are cut in half, implementing a carry-select adder instead of a ripple-carry adder in a FLEX 8000 device can improve routability, even though carry-select adders use more LE resources. See *Application Brief 118* (*Ripple-Carry Adders in FLEX 8000 Devices)* in this handbook for more information.

Another advantage of the carry-select adder is its speed. For adders containing more than 16 bits, the carry-select adder computes the LSB and the MSB in parallel, yielding a result more quickly than in ripple-carry adders. For smaller adders (up to 8 bits), implementing a carry-select adder does not offer much improvement over a ripple-carry adder. However, in larger adders, the routability and speed improvement of the carry-select adder may justify the increase in LE usage.

In 32-bit adders, you can divide the length of the carry chain into four, rather than two, equal parts, making each carry chain only 8 bits long. This technique yields further improvements in performance, routability, and speed.

Accumulators in FLEX 8000 Devices

May 1994, ver. 2 | Application Brief 111

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_111.exe

Small accumulators (i.e., 4- and 8-bit accumulators) can use the carry chain feature in FLEX 8000 devices to produce fast, efficient circuits. The carry chain is a fast (less than 1 ns) carry-forward function path between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. A single LAB can implement an 8-bit accumulator. However, since carry chains must be placed in contiguous LEs and LABs, 16- and 32-bit carry chains may reduce the routing resources available for implementing other logic. This application brief discusses both ripple-carry and pipelined accumulators. Design techniques described in this application brief can be used to create design files optimized for the following characteristics (numbers in parentheses are for pipelined accumulators):

Design Goals:		***Design Results:***		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	8	125
Product Term	✓ Speed	16 Bits	16 (33)	71 (93)
	✓ Area	32 Bits	32 (65)	42 (45)

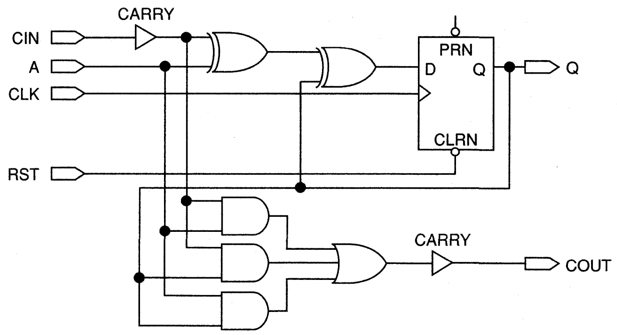
Single-Bit Accumulators

The basic building block of an accumulator is an adder. The adder's sum is registered and fed back as one of the inputs into the adder. A single-bit adder has three inputs: two data inputs and a carry-in. The adder outputs a sum and a carry-out. The following Altera Hardware Design Language (AHDL) equations show how the sum and carry-out are calculated:

```
sum = a $ b $ cin;
cout = (a & b) # (a & cin) # (b & cin);
```

Figure 1 shows a single-bit accumulator. The carry chain feature of the FLEX 8000 architecture can implement the single-bit adder's sum and carry-generation logic within a single LE.

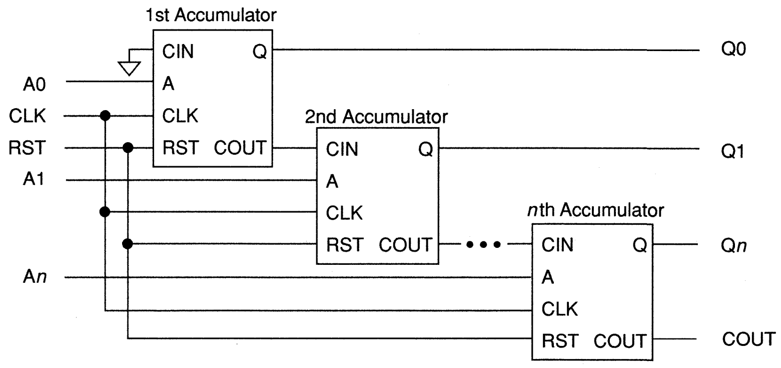
Figure 1. Single-Bit Accumulator



N-Bit Accumulators

You can design *n*-bit accumulators by cascading the carry-out of the (*n*-1)th stage to the carry-in of the *n*th stage to extend the single-bit accumulator shown in Figure 1. For example, to design a two-bit accumulator, you can simply connect the carry-out of the first stage to the carry-in of the second stage. Figure 2 shows an *n*-bit accumulator.

Figure 2. N-Bit Accumulator

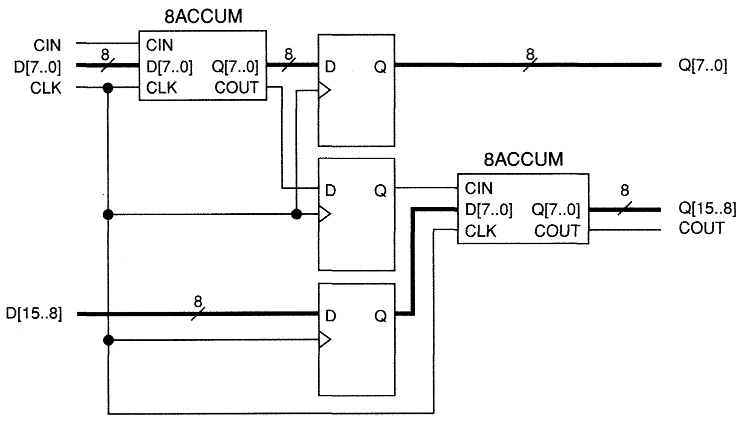


Pipelined 16- and 32-Bit Accumulators

You can create fast and compact accumulators using the FLEX 8000 carry chain feature. However, 16- and 32-bit carry chains may reduce the routing resources available for implementing other logic because carry chains must be placed in contiguous LEs and LABs.

You can use pipelining to increase the performance and routability of accumulator designs. Pipelining consists of inserting flipflops within combinatorial logic. The carry-out signal feeds a register, which in turn feeds the carry-in of the next bit. Pipelining decreases the combinatorial delay between registers, but also delays the carry signal by one Clock cycle. For example, two 8-bit accumulators are needed to build a 16-bit pipelined accumulator. The carry-out of the least significant 8-bit accumulator is registered and then fed to the carry-in of the most significant 8-bit accumulator. However, registering only the lower-order bits delays the carry by one Clock cycle, yielding two unsynchronized 8-bit accumulators. To synchronize the higher and lower order bits, the data inputs to the higher-order bits must also be registered to delay them by one Clock cycle. To complete the synchronization, the accumulator's lower-order sum bits must also be registered. See *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook for more information on pipelining. Figure 3 shows a 16-bit pipelined accumulator. Although this pipelined accumulator uses twice as many LEs as a non-pipelined version, its performance is increased by 33%.

Figure 3. 16-Bit Pipelined Accumulator



Other Accumulator Design Techniques

You can also insert `LCELL` buffers in the carry chain at 8- or 16-bit intervals to break the chain into shorter segments. Inserting `LCELL`s can improve routing, but also decreases performance.

Notes:

ALTERA

Subtractors
in FLEX 8000 Devices

May 1994, ver. 1 — Application Brief 126

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_126.exe

Subtractors are easy to design because their logic is similar to adder logic. The carry chain feature in FLEX 8000 devices can implement fast subtractors that use logic element (LE) resources efficiently. You can also use subtractors to construct dividers. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		*Design Results:*		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	8	154
Product Term	✓ Speed	16 Bits	16	84
	✓ Area	32 Bits	32	45

Subtractors

A full *n*-bit subtractor is similar to a full adder. You can easily design subtractors in MAX+PLUS II by starting from basic principles. You can also modify the FLEX 8000-optimized version of the existing full adder macrofunction, `8FADD`, which is available in the MAX+PLUS II Macrofunction Library.

Table 1 shows the truth table for a 2-bit subtractor. The inputs are `A`, `B`, and `BORROW_IN`; the outputs are `DIFF` and `BORROW_OUT`. The subtractor performs the function `DIFF = A - B`.

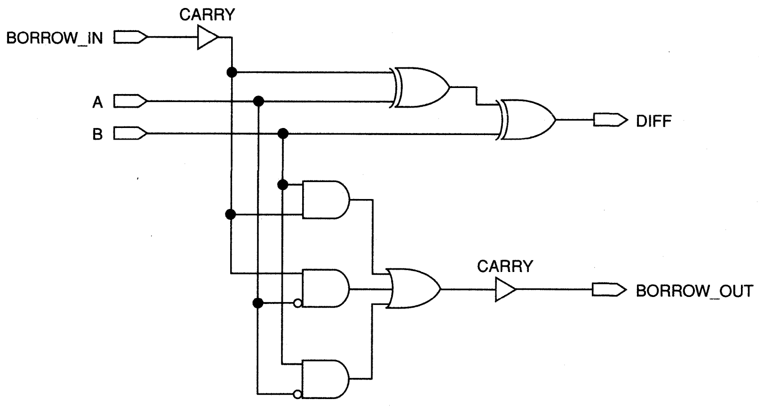
Table 1. 2-Bit Subtractor Truth Table

A	B	BORROW_IN	DIFF	BORROW_OUT
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

N-Bit Subtractors

Figure 1 shows a 2-bit subtractor that is optimized for FLEX 8000 devices.

Figure 1. 2-Bit Subtractor Optimized for the FLEX 8000 Architecture



You can construct an *n*-bit subtractor by connecting *n* 2-bit subtractors in a chain, with the BORROW_OUT of one stage feeding the BORROW_IN of the next stage. When BORROW_IN = 0, the subtractor functions as follows:

0 - 0 = 0 with BORROW_OUT = 0
1 - 0 = 1 with BORROW_OUT = 0
1 - 1 = 0 with BORROW_OUT = 0
0 - 1 = 1 with BORROW_OUT = 1 feeding the next stage

The following equation shows a sample subtraction of 0 - 1 with BORROW_IN = 0:

$$\begin{array}{r} 10_2 \\ -1_2 \\ \hline 1_2 \end{array}$$

When BORROW_IN = 1, the preceding stage has borrowed a 1 from the current stage. In this case, the subtractor functions as follows (see Table 1):

1 - 0 = 0 with a BORROW_OUT = 0
1 - 1 = 1 with a BORROW_OUT = 1 feeding the next stage

The following equation shows a sample subtraction of 1 - 1 with a BORROW_IN = 1:

$$\begin{array}{rl} 11 & \longleftarrow \texttt{BORROW\_IN}\ \textit{bits are} \\ 110_2 & \quad\ \textit{fed to the next stage} \\ -\ 11_2 & \\ \hline 11_2 & \end{array}$$

The `BORROW_OUT = 1` from the first stage eliminates the second bit in the `110` input, resulting in the subtraction shown in the equation $10_2 - 1_2 = 1_2$ for the second stage, with a `BORROW_OUT` bit passed to the third stage.

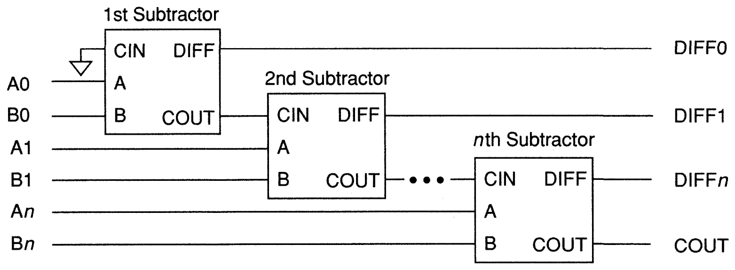
The remaining subtractions follow a similar pattern, yielding the following equations:

```
DIFF = A $ (B $ BORROW_IN)
BORROW_OUT = !A & B
           # !A & BORROW_IN
           #  B & BORROW_IN
```

The equations for a subtractor resemble those of a full adder. The only difference is that in a subtractor, the `A` input in the `BORROW_OUT` equation is inverted. For this reason, subtractors are very easy to implement once you have created a full adder. The 2-bit subtractor shown in Figure 1 is optimized for the FLEX 8000 architecture with `CARRY` primitives. These `CARRY` primitives implement high-speed carry chains, yielding fast `BORROW_IN` to `BORROW_OUT` times.

Adders are used frequently in multiplier applications. During the multiplication process, the multiplicand is added to itself *n* times, where *n* is the value of the multiplier. Conversely, subtractors are used to construct dividers, in which the divisor is subtracted from the dividend as many times as possible. You can create structures analogous to carry-select adders for subtractors, thereby creating borrow-select subtractors. Figure 2 shows an *n*-bit subtractor.

Figure 2. N-Bit Subtractor



Notes:

Contents

May 1994

Section 6 Multiplier Applications

Designing Multipliers in FLEX 8000 Devices

May 1994, ver. 1 Application Brief 127

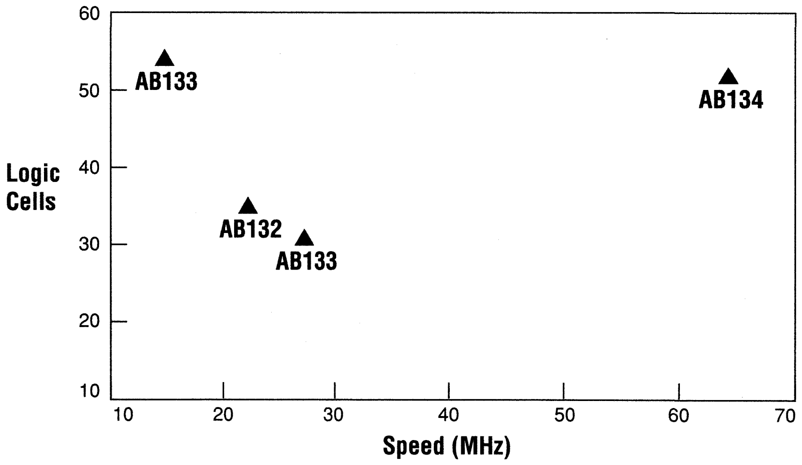
Introduction

FLEX 8000 devices feature look-up table (LUT) architecture and logic elements (LEs) that allow you to design register-intensive functions such as multipliers. FLEX 8000 LEs also offer dedicated carry chain logic that efficiently implements fast multiplier functions by optimizing the adders used within them. This application brief introduces some of the design techniques that Altera recommends to take advantage of these architectural features.

Design Trade-Offs

Most programmable logic designs must make trade-offs between speed, area, and routability. The application briefs in this section describe different design techniques that you can use to optimize multipliers for your own application requirements. Figure 1 summarizes the advantages of the different multiplier design techniques.

Figure 1. Design Techniques for Multipliers



AB132	*Ripple-Carry Multipliers in FLEX 8000 Devices*
AB133	*Combinatorial Multipliers Using Booth's Algorithm in FLEX 8000 Devices*
AB134	*Pipelined Multipliers in FLEX 8000 Devices*

If speed is your primary goal, you can create pipelined multipliers. These multipliers provide optimal performance and are easily routable, but they use more LE resources per multiplier bit than other types of multipliers. For high-speed, non-pipelined multipliers that use minimum area, you can create L-Booth algorithm multipliers (a variation of Booth's algorithm multipliers). Ripple-carry multipliers also offer good speed performance and efficient area usage. Both L-Booth algorithm and ripple-carry multipliers take advantage of the dedicated carry chain feature in FLEX 8000 LEs, which offers a fast (less than 1 ns) carry-forward function path between contiguous LEs within a Logic Array Block (LAB) and between contiguous LABs. However, the carry chain design approach is recommended primarily for multipliers with short carry chains since carry chains must be placed in contiguous LEs and LABs. Longer carry chains (e.g., carry chains using more than eight LEs) may reduce the routing resources available for implementing other logic.

If easy routability is your primary goal, pipelined multipliers offer maximum routability. You can also create more area-efficient Booth's algorithm multipliers that use short carry chains and parallel computation. Booth's algorithm multipliers use more LEs per multiplier bit than L-Booth algorithm and ripple-carry multipliers, but they offer the ability to process both positive and negative numbers. You can also modify a basic ripple-carry multiplier design to create a faster, more routable 4-bit × 4-bit multiplier in which the last multiplier bit is generated independently of the carry chain.

With planning and creativity, you can implement multipliers efficiently in FLEX 8000 devices regardless of the constraints imposed by any particular application.

☞ For detailed information on LE architecture and carry chain logic, refer to *Application Note 40 (FLEX 8000 Architecture)* and *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook.

Multiplier Theory

Figure 2 illustrates a basic 4-bit × 4-bit multiplier. The two 4-bit operands are `X[3..0]` and `Y[3..0]`. Multiplying the first bit of the multiplier, `Y0`, by each consecutive bit of the multiplicand yields the first partial product. Multiplying the second bit of the multiplier, `Y1`, by each consecutive bit of the multiplicand generates the second partial product. The second partial product is shifted to the left with respect to the first partial product. The generation of partial products continues until all of the multiplier terms are exhausted. The final product is obtained by adding the partial products vertically with respect to columns and observing carries to the next column.

Figure 2. 4-Bit × 4-Bit Multiplier Process

Multiplicand →						X3	X2	X1	X0
Multiplier →					×	Y3	Y2	Y1	Y0
Partial Product, Row A →						Y0X3	Y0X2	Y0X1	Y0X0
Partial Product, Row B →					Y1X3	Y1X2	Y1X1	Y1X0	0
Partial Product, Row C →				Y2X3	Y2X2	Y2X1	Y2X0	0	0
Partial Product, Row D →		+	Y3X3	Y3X2	Y3X1	Y3X0	0	0	0
Final Product →	S7		S6	S5	S4	S3	S2	S1	S0

AND gates can be used to perform a bit-by-bit multiplication that simplifies the process of generating the partial products. Bit-by-bit multiplication generates all the partial products independently and in parallel at very high speed. The additions, however, are not independent since each stage requires the carry from the previous column. Since some columns can produce up to eight carries, the carries can make addition quite slow. Most methods used to accelerate multiplication are simply techniques for quickly adding the partial products.

Altera's FLEX 8000 architecture includes an Arithmetic mode designed specifically to perform fast additions. This mode uses the sum and carry functions that are available within each LE to solve 2-bit (A + B) additions efficiently. In addition, the FLEX 8000 LAB provides a dedicated path for the carry signal to feed the next stage of the adder. For more information on the Arithmetic mode, refer to *Application Note 40 (FLEX 8000 Architecture)* in this handbook.

Notes:

Ripple-Carry Multipliers in FLEX 8000 Devices

May 1994, ver. 1 — Application Brief 132

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_132.exe

The FLEX 8000 device architecture provides dedicated carry and cascade chain resources to efficiently implement arithmetic functions such as ripple-carry multipliers. The dedicated carry chain feature in FLEX 8000 LEs offers a fast (less than 1 ns) carry-forward function between contiguous LEs with a Logic Array Block (LAB) and between contiguous LABs. By supporting very fast additions, the FLEX 8000 carry chain, in particular, speeds up multiplier designs.

This application brief describes a ripple-carry multiplier and a modified ripple-carry multiplier that are optimized for the FLEX 8000 architecture. Multiplier designs can either use the FLEX 8000 carry chain exclusively or incorporate additional logic to speed up the design. Design techniques described in this application brief can be used to create design files optimized for the following characteristics (numbers in parentheses are for the modified ripple-carry multiplier):

Design Goals:		*Design Results:*		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	4 × 4 Bits	35 (35)	22 (21.5)
Product Term	✓ Speed	16 × 16 Bits	–	–
	Area	32 × 32 Bits	–	–

Ripple-Carry Multipliers

Ripple-carry multipliers use the FLEX 8000 carry chain to perform very fast addition of the partial products. Separate logic elements (LEs) generate each of the partial products and ripple-carry adders generate the sums of the partial products. Figure 1 shows the architecture of a 4-bit × 4-bit ripple-carry multiplier. This multiplier is fast because it contains only three levels (or stages) of delay:

1. Generating partial products
2. Adding the partial products to form `A[5..0]` and `B[5..0]`
3. Adding `A[5..0]` and `B[5..0]` to compute the final product

Figure 1. 4-Bit × 4-Bit Ripple-Carry Multiplier Architecture

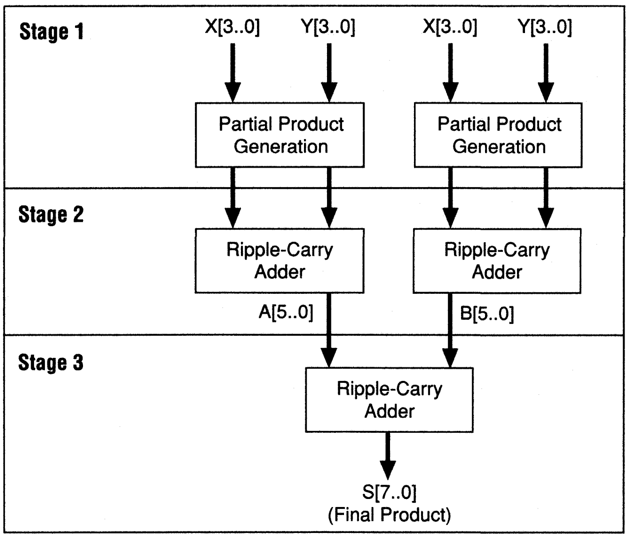
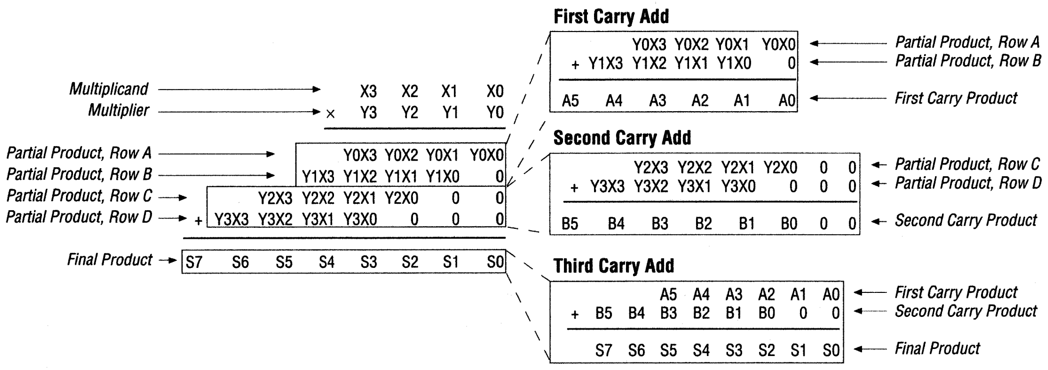


Figure 2 shows a generic multiplication broken into the three stages of the multiplier. In the first stage, the partial products are generated. Each partial product is derived by ANDing the appropriate multiplicand and multiplier terms (e.g., for the first term, `Y0X0` = `Y0` AND `X0`). In the second stage, ripple-carry adders generate the sums of the partial products to form the first carry product and the second carry product. In the third stage, the first and second carry products are added with a ripple-carry adder to form the final product. For more information on ripple-carry adders, refer to *Application Brief 118 (Ripple-Carry Adders in FLEX 8000 Devices)* in this handbook.

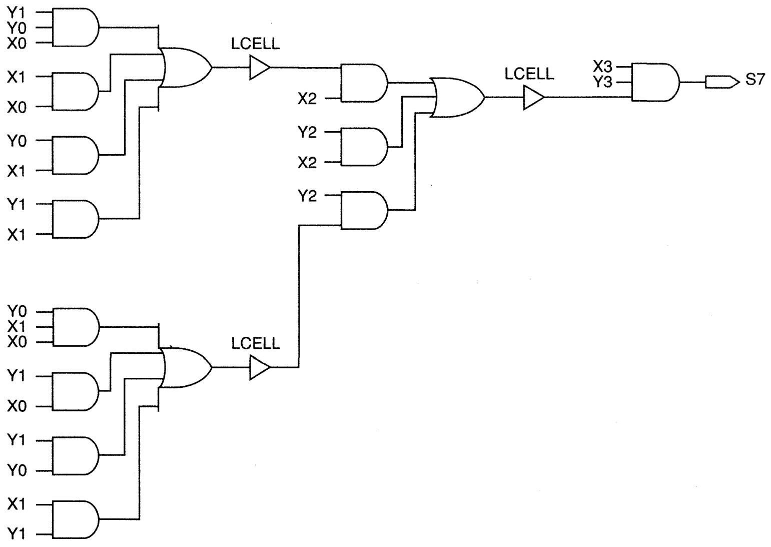
Figure 2. 4-Bit × 4-Bit Ripple-Carry Multiplier Process



Modified Ripple-Carry Multipliers

The slowest path in the ripple-carry multiplier described in Figure 1 is the carry propagation required to generate S7. However, in a 4-bit × 4-bit multiplier, you can generate the seventh bit independently of the carry quickly and easily by using the logic shown in Figure 3. (You can use this logic to generate the seventh bit only with 4-bit × 4-bit multipliers.) A modified ripple-carry multiplier that contains this logic is otherwise identical to the ripple-carry multiplier shown in Figure 1. The regular ripple-carry multiplier generates all other products.

Figure 3. Logic Schematic for S7 Bit



Notes:

Combinatorial Multipliers Using Booth's Algorithm in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 133

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_133.exe

Booth's algorithm decreases the time required to compute partial products and, consequently, increases the speed of combinatorial multipliers in FLEX 8000 Devices. This application brief describes combinatorial multipliers that use Booth's algorithm and a modified "L-Booth" algorithm. Design techniques described in this application brief can be used to create design files optimized for the following characteristics (numbers in parentheses are for the L-Booth multiplier):

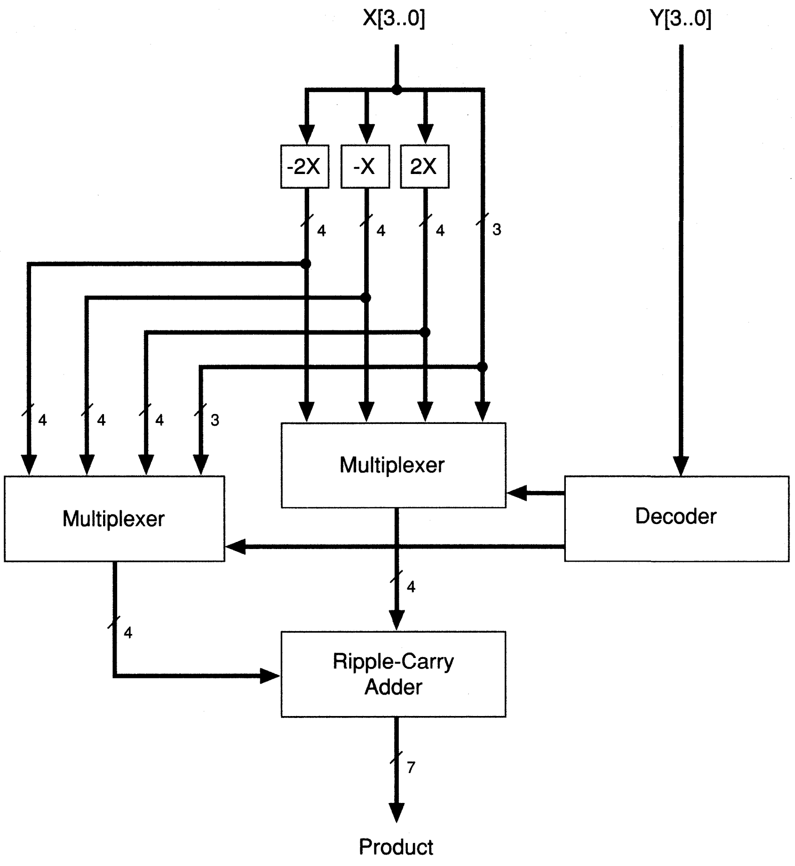
Design Goals:		Design Results:		
Architecture	Optimization	Width	Logic Cells	Speed (MHz)
✓ Look-Up Table	Routability	4 × 4 Bits	54 (31)	13.3 (27.1)
Product Term	✓ Speed	16 × 16 Bits	–	–
	Area	32 × 32 Bits	–	–

Booth's Algorithm Multipliers

Booth's algorithm expands upon general multiplier theory by reducing the number of partial products by half. This reduction in partial products yields a significant speed increase, especially in 4-bit × 4-bit multipliers. The 4-bit × 4-bit Booth's algorithm operates on two's complement numbers. It multiplies any two numbers between –8 and 7, and returns a two's complement result.

Figure 1 shows a block diagram for a 4-bit × 4-bit multiplier implemented using Booth's algorithm. The multiplier is partitioned into groups of three bits in the decoder portion of the design. In this decoder, zeroes are padded to the right of each bit to act as placeholders. Booth's algorithm then computes all the possible partial products by performing five operations on the multiplicand: `0(X[3..0])`, `1(X[3..0])`, `2(X[3..0])`, `-2(X[3..0])`, and `-1(X[3..0])`. Simultaneously, Booth's algorithm is applied to each group of three bits to determine which one of the five possible partial products is correct. These tasks are performed in parallel for speed. A simple combinatorial shift to the left produces the `2X` term. With the FLEX 8000 carry chain, Booth's algorithm uses the two's complement method to generate the `-2X` and `-X` terms. The decoder portion of the design controls the multiplexer and selects the correct partial products or zero. The selected partial products are added with a ripple-carry adder to form the final product.

Figure 1. Block Diagram for a 4-Bit × 4-Bit Multiplier Implemented Using Booth's Algorithm

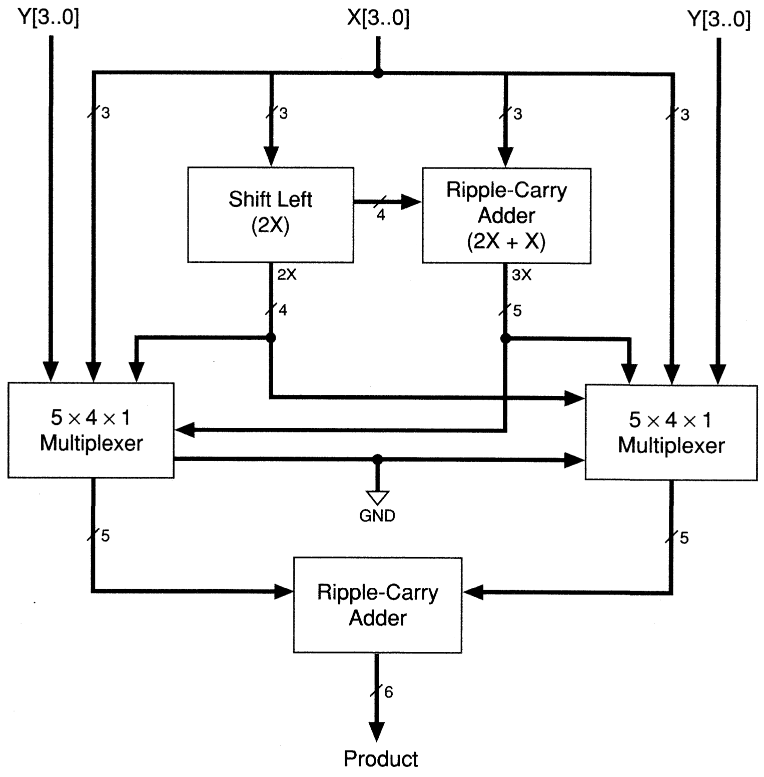


For any multiplier larger than 4 bits × 4 bits, the total number of partial products exceeds two. In such cases, the addition must be cascaded through two or more layers of ripple-carry adders. For more information on ripple-carry adders, refer to *Application Brief 118 (Ripple-Carry Adders in FLEX 8000 Devices)* in this handbook.

L-Booth Algorithm Multipliers

The L-Booth algorithm multiplier is a variation of a Booth's algorithm multiplier that works only for positive multiplicands. Figure 2 shows a block diagram of a 4-bit × 4-bit multiplier implemented using the L-Booth algorithm. It decodes the multiplicand into two sets, each of which is two bits wide. The first set, `Y[1..0]`, determines whether the addition of the first two partial products would have been zero, `X`, `2X`, or `3X`. The second set, `Y[3..2]`, determines whether the addition of the last two products would have been zero, `X`, `2X`, or `3X`. The L-Booth algorithm multiplier generates `2X` and `3X` as it decodes the multiplicand. It performs a simple combinatorial shift to the left to produce the `2X` term and, using the FLEX 8000 carry chain, it adds `X` and `2X` to form `3X`. These two steps represent the first level of the multiplier. The second level consists of multiplexers controlled by the decoded multiplicand. These multiplexers select the correct values of the two decoded partial products. The two selected partial products are then added with a ripple-carry adder to generate the final product.

Figure 2. Block Diagram for a 4-Bit × 4-Bit Multiplier Implemented Using the L-Booth Algorithm



Notes:

Pipelined Multipliers in FLEX 8000 Devices

May 1994, ver. 1 | Application Brief 134

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_134.exe

FLEX 8000 devices provide a register-intensive architecture that allows you to take advantage of pipelining techniques. Pipelining your design by interspersing registers within the combinatorial logic can significantly increase the system Clock rate while improving the overall routability of the design. Because the registers synchronize the data to a common system Clock, only register-to-register delays limit the Clock period.

This application brief describes a pipelined multiplier. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		Design Results:		
Architecture	Optimization	Width	Logic Cells	Speed (MHz)
✓ Look-Up Table	Routability	4 × 4 Bits	52	63.7
Product Term	✓ Speed	16 × 16 Bits	–	–
	Area	32 × 32 Bits	–	–

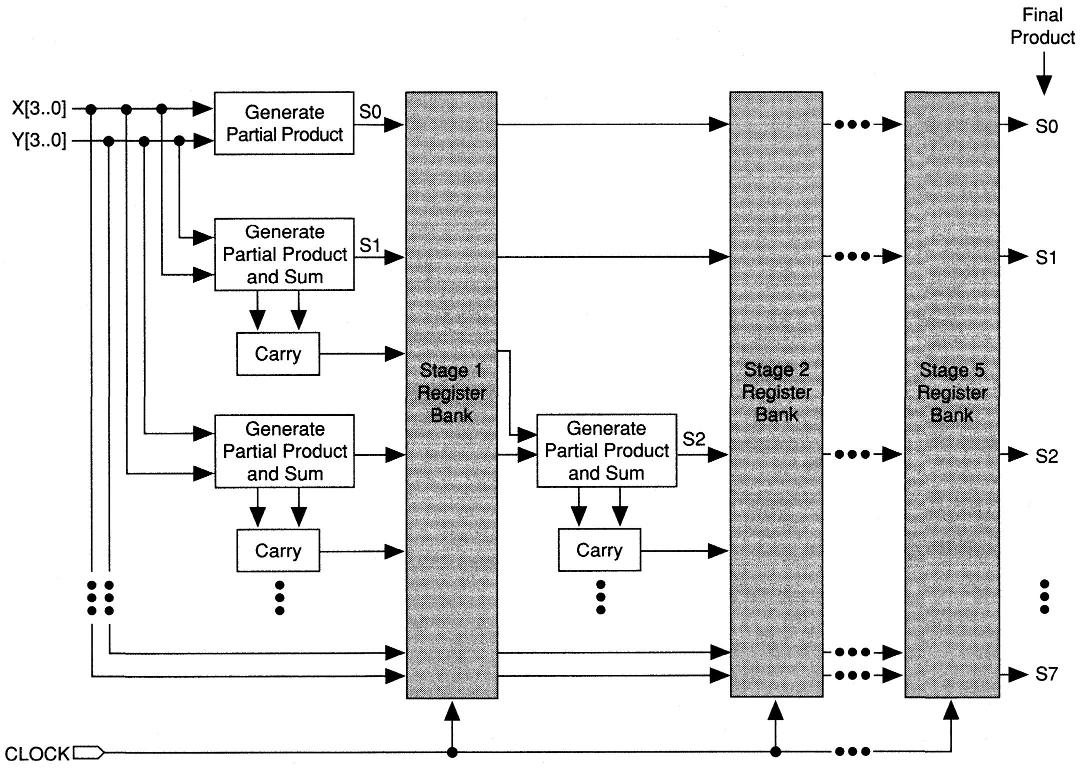
Pipelined Multipliers

Pipelining allows you to increase the performance of your multiplier design by distributing intermediate computations across latency stages. This distribution decreases the complexity and increases the speed of the intermediate computations. In other words, you can increase the speed of your design by adding latency stages. The fastest possible speed is limited by the delay of a logic element (LE), t_{LE}, plus the longest routing delay between any two LEs. For more information on pipelining, refer to *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook.

Pipelining a design requires only a limited amount of additional area because it uses registers that each LE already includes at its output. The only added area is the overhead of synchronizing the sums. All other logic uses the existing LE registers. Since pipelining can increase the design speed without significantly increasing area, it is recommended for many multiplier designs.

Figure 1 shows a block diagram of a pipelined combinatorial multiplier. This multiplier is designed with the maximum number of latency stages and, consequently, the fastest performance. The multiplication is computed by generating the partial products and adding them together in the 4-input look-up tables (LUTs) provided in each FLEX 8000 LE. Separate LUTs generate carries so that they can be routed to registers. (This design does not use the dedicated carry chain in FLEX 8000 LEs because the carry chains can only feed contiguous LEs and cannot feed registers.) S0 and S1 are the only two sums that can be generated by one stage. S2, for example, requires two stages. A register bank maintains the timing between the stages. The final products are generated in the same way. Five stages are required to generate the last bit, S7.

Figure 1. Block Diagram for a Pipelined Combinatorial Multiplier



Contents

May 1994

Section 7 State Machine Applications

State Machine Encoding

May 1994, ver. 1 Application Brief 131

Summary

Each state of a state machine can be represented with a unique pattern of high (1) and low (0) register output signals, a process called "encoding." The two primary encoding methods are binary and one-hot encoding. This application brief describes both methods and discusses how to select the encoding scheme that best suits your design, so that you can ensure efficient performance and resource usage.

Binary Encoding

The relationship between the number of state bits (B) and number of states (S) in a binary-encoded state machine is represented by the following equation:

$$B = \log_2 (S)$$

With this formula, you can easily determine the minimum number of state bits required for a binary-encoded state machine. For example, to implement a 4-state state machine with a binary encoding scheme, you can use two flipflops (i.e., state bits) to uniquely define the four states as follows:

```
state1 = "00"
state2 = "01"
state3 = "10"
state4 = "11"
```

This example implements the state machine with a left-to-right sequential binary encoding. You can also use a Gray code binary encoding scheme, in which only one state bit changes at a time:

```
state1 = "00"
state2 = "01"
state3 = "11"
state4 = "10"
```

Gray code binary encoding is especially useful when the outputs of the state bits are used asynchronously. For example, if a state machine switches from `"01"` to `"10"`—as it does in sequential binary encoding—and the registers do not switch outputs at exactly the same time, temporary outputs of either `"11"` or `"00"` can exist. This type of fluctuation can cause unpredictable results throughout your circuit.

One-Hot Encoding

A one-hot encoding scheme uses one register for each state—e.g., four registers for a 4-state state machine—with only one state bit at a high logic level at one time. You can implement a 4-state state machine with a one-hot encoding scheme as follows:

```
state1 = "0001"
state2 = "0010"
state3 = "0100"
state4 = "1000"
```

Selecting an Encoding Method

You should choose an encoding method based on the complexity of your state machine, the target device family, and requirements for recovering from illegal states.

Complex State Machines

Binary encoding uses fewer registers than one-hot encoding. Thus, binary encoding requires only seven registers to implement a 100-state state machine, whereas one-hot encoding needs 100 registers. On the other hand, although one-hot encoding requires more registers, the logic is generally less complex. In a binary-encoded state machine, the logic that controls the transitions from state to state depends on all seven state bits as well as the inputs to the state machine. This type of logic typically requires high-fan-in functions to the inputs of the state bits. In a one-hot-encoded state machine, however, the inputs to the state bits are often simply the functions of other state bits.

Device Architecture

Different architectures favor certain types of encoding. The MAX+PLUS II Compiler automatically selects the most appropriate encoding method for the targeted device family, unless you specify a particular scheme in one of your design files. For example, since the Altera FLEX 8000 device family is register-intensive, state machines targeted for these devices are best implemented with a one-hot encoding scheme. Since a one-hot-encoded state machine reduces the complexity of the logic feeding the state bits, one-hot encoding can increase the performance of your state machine design for FLEX 8000 devices.

The MAX 5000 and MAX 7000 device families are best suited to a binary state machine encoding scheme. Both of these device families can efficiently implement complex combinatorial logic with shared and parallel expander product terms. Thus, devices in these device families can accommodate complex combinatorial logic functions without wasting resources or sacrificing performance.

Recovery from Illegal States

When choosing an encoding method, you must consider the number of potential illegal states your state machine can enter. Your design can end up in an illegal state if you violate the setup or hold times of the state-bit registers and have not defined all possible states. MAX+PLUS II design entry methods allow you to define illegal states and to specify how your state machine should recover from them.

For example, a 14-state state machine implemented with a binary encoding scheme requires four state bits, for a total of 16 possible states. In this case, the state machine has only two possible illegal states. One-hot-encoded state machines, however, generally have more potential illegal states. A 14-state one-hot encoded state machine requires 14 state bits. The number of illegal states for a one-hot encoded state machine is determined by the equation $(2^n) - n$, where n equals the number of states in the state machine. Therefore, a 14-state state machine with one-hot encoding has 16,370 possible illegal states. However, as long as your design does not violate the setup or hold time of the state bit registers, your state machine should not enter an illegal state.

Notes:

ALTERA

Contents

May 1994

Section 8 Miscellaneous Applications

Multiplexers in FLEX 8000 Devices

May 1994, ver. 1 Application Brief 128

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_128.exe

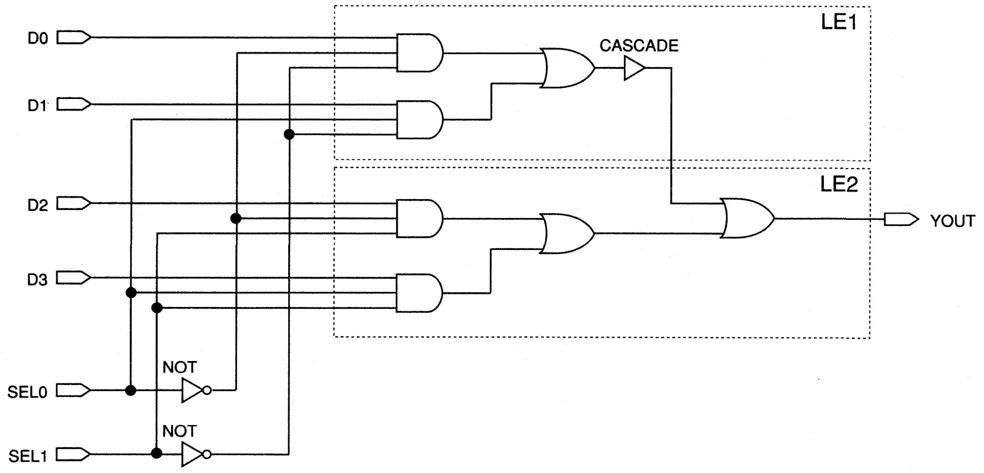
You can implement wide-fan-in combinational logic structures, such as multiplexers, using the cascade chain feature in FLEX 8000 devices. Cascade chains are dedicated, high-speed paths between adjacent logic elements (LEs) in the FLEX 8000 architecture, and can be used to improve both design performance and resource utilization. In wide multiplexers (8-to-1 and wider), pipelining can be used to further optimize the design. Design techniques described in this application brief can be used to create design files optimized for the following characteristics (numbers in parentheses are for the pipelined multiplexer):

Design Goals:		*Design Results:*		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	5	32.4
Product Term	✓ Speed	16 Bits	10	32.4 (48.8)
	✓ Area	32 Bits	21	22.4 (42.4)

Multiplexers with Cascade Logic

Each LE in the FLEX 8000 architecture contains a 4-input look-up table (LUT) that can compute any function of 4 variables. Figure 1 shows a 4-to-1 multiplexer in which the results of the LUTs of two LEs are ANDed using a cascade chain. DeMorgan's inversion is used to convert the AND to an OR.

Figure 1. 4-to-1 Multiplexer



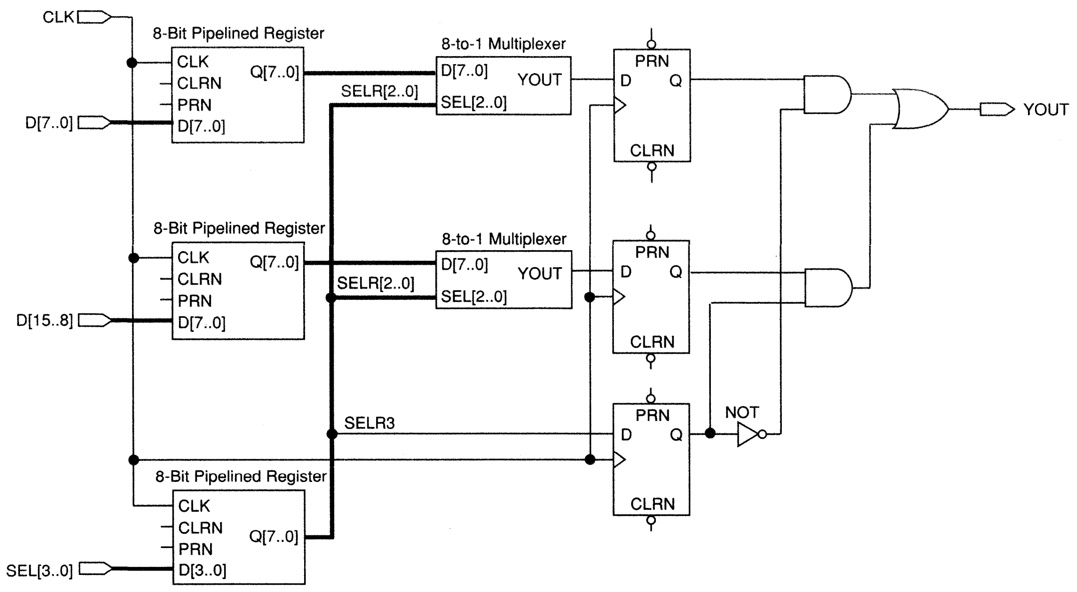
Using cascade chain logic also yields more compact designs. Since the maximum length of the cascade chain in a 4-to-1 or 8-to-1 multiplexer does not exceed two, the impact on routability is minimal. For more information about cascade chains, refer to *Application Note 36 (Designing with FLEX 8000 Devices)* and *Application Note 40 (FLEX 8000 Architecture)* in this handbook.

Pipelined Multiplexers

Pipelining is recommended for multiplexers wider than 8-to-1. Pipelining consists of inserting flipflops between combinatorial logic, which decreases register-to-register delays and increases operating frequency. For more information about pipelining, refer to *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook.

Figure 2 shows a 16-to-1 multiplexer, which uses three 8-bit pipelined registers and two 8-to-1 multiplexers. This multiplexer has a latency of two Clock cycles. The 4-to-1 multiplexer shown in Figure 1 can be used as a building block to construct the two 8-to-1 multiplexers in Figure 2.

Figure 2. 16-to-1 Multiplexer with Two Pipeline Stages



Notes:

ALTERA

Equality Comparators
in FLEX 8000 Devices

May 1994, ver. 2 Application Brief 113

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_113.exe

An equality comparator determines whether two bus signals are equal in value. You can optimize the area and performance of an equality comparator in a FLEX 8000 device by using FLEX cascade chains. Cascade chains are fast (less than 1 ns) carry-forward function paths between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. For larger buses, the cascade chain must be split and merged in an additional LE. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		**Design Results:**		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	4	46.5
Product Term	✓ Speed	16 Bits	8	35.1
	✓ Area	32 Bits	17	31.7

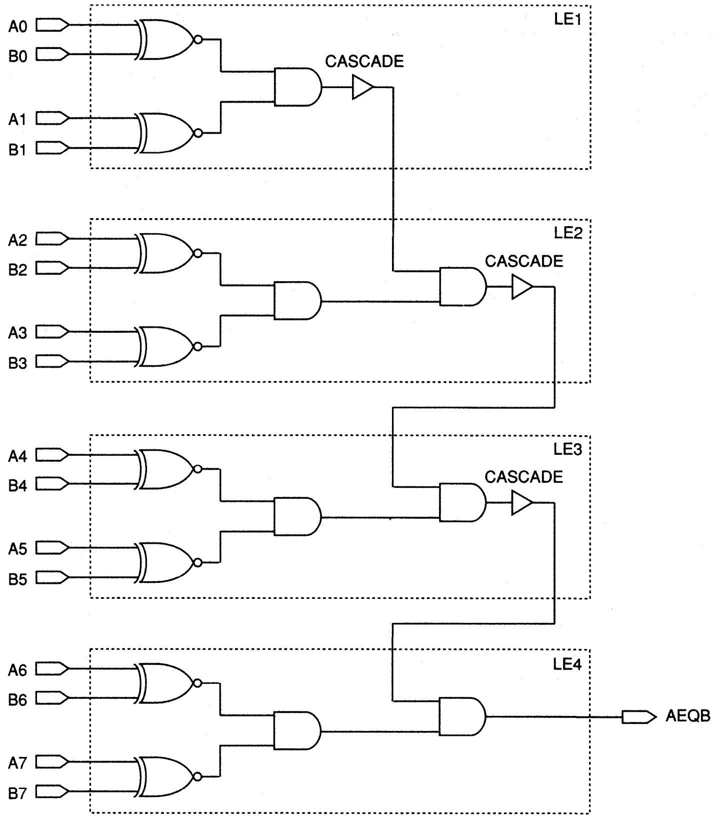
8-Bit Equality Comparisons

Equality comparators can be described as wide logical AND functions, as shown in the following Altera Hardware Description Language (AHDL) equation:

```
aeqb = (a[n..0] == b[n..0]);
```

In this equation, a0 equals b0, a1 equals b1, a2 equals b2, etc., through a*n* equals b*n*, where *n* represents the most significant bit (MSB) of the bus. Figure 1 shows an equality comparator implemented with a cascade chain. Four 2-bit comparators are cascaded together to form an 8-bit comparator that yields the output aeqb. Because each LE can accept up to four signals to its look-up table (LUT), two comparisons can be performed in each LE. In Figure 1, for example, a0 is compared to b0, a1 is compared b1, and the results of these two comparisons are ANDed. With the cascade chain, this sum is then ANDed together with the sums of the three subsequent LEs to generate the equality comparator output aeqb. Cascade chains can be used to improve both design performance and resource utilization. For more information about cascade chains, refer to *Application Note 36 (Designing for FLEX 8000 Devices)* and *Application Note 40 (FLEX 8000 Architecture)* in this handbook.

Figure 1. 8-Bit Equality Comparison with a Cascade Chain

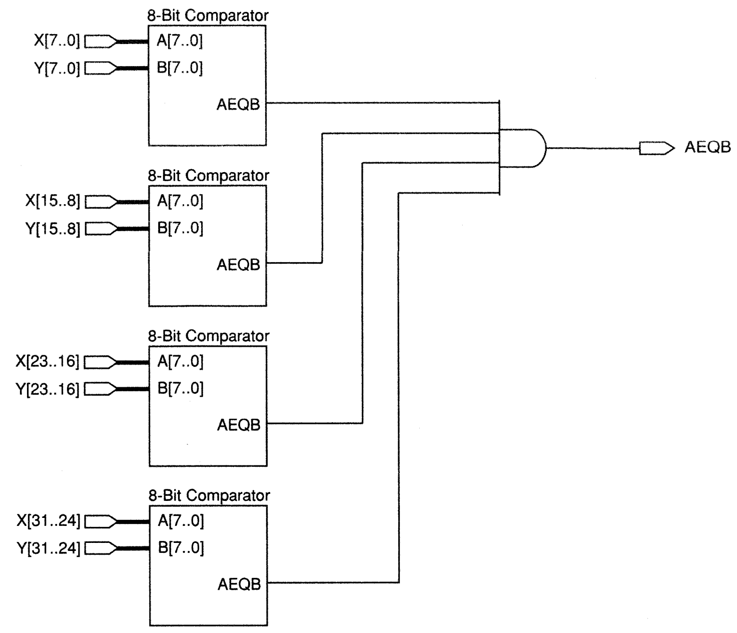


Equality Comparisons for Larger Buses

For buses larger than 8 bits, the cascade chain must be split to avoid exceeding the maximum number of inputs into an LAB. For example, a 16-bit equality comparator that uses 4 unique signals in each of the 8 LEs of an LAB would require 32 inputs. However, the LAB local interconnect can only receive 24 signals from the FLEX 8000 row interconnect. Therefore, the cascade chain must be split between two LABs to ensure that the number of inputs to each LAB is less than 24.

You can create a 32-bit comparator using 4 of the 8-bit comparators shown in Figure 1. Each 8-bit comparator requires 4 sets of 4 adjacent LEs. The output of these LEs can then be merged in an additional LE to generate the `aeqb` output. See Figure 2.

Figure 2. 32-Bit Equality Comparison



Notes:

Full Comparators
in FLEX 8000 Devices

May 1994, ver. 2 | Application Brief 114

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_114.exe

A full comparator is used to compare the values of two bus signals. In FLEX 8000 devices, you can use cascade chains and carry chains to optimize the area and performance of a full comparator. Carry and cascade chains are fast (less than 1 ns) carry-forward function paths between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. Design techniques described in this application brief can be used to create design files optimized for the following characteristics:

Design Goals:		***Design Results:***		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	10	30
Product Term	✓ Speed	16 Bits	18	24
	✓ Area	32 Bits	–	–

8 Miscellaneous Applications

Using Full Comparators

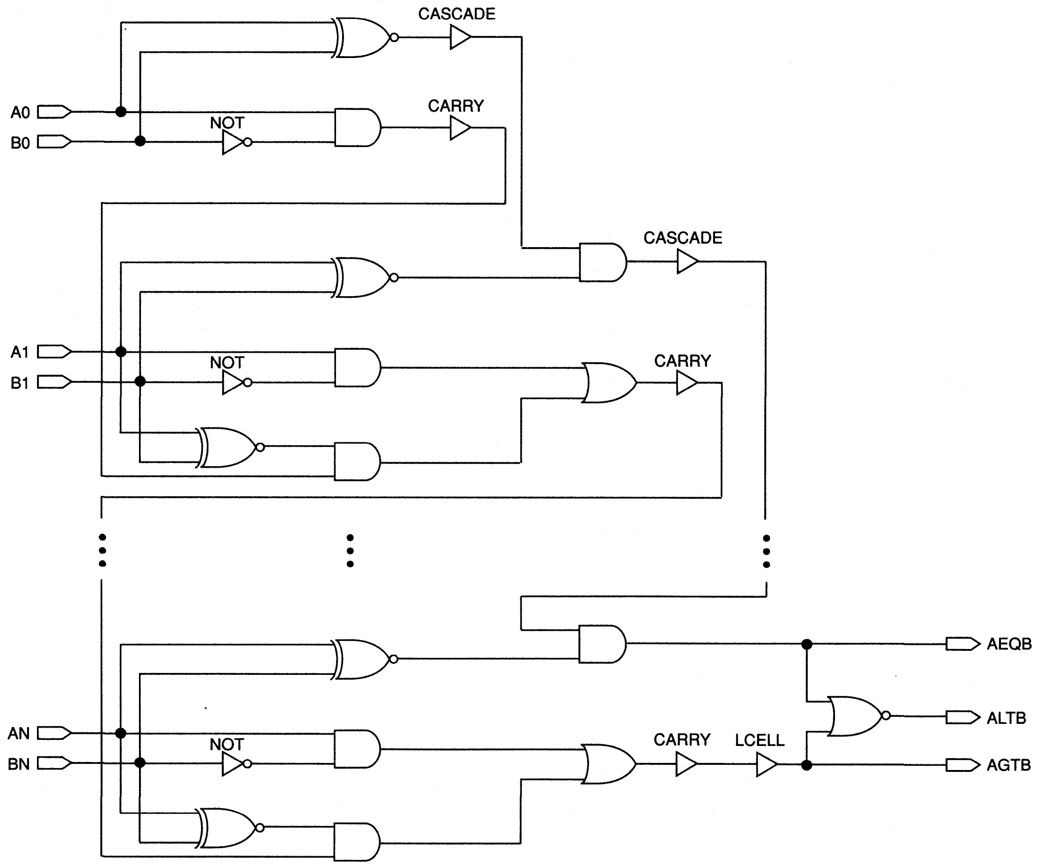
For the two buses `a[`*n*`..0]` and `b[`*n*`..0]`, where *n* represents the most significant bit (MSB) of the bus, a full comparator yields three results: `a = b` (`aeqb`), `a < b` (`altb`), and `a > b` (`agtb`). An *n*-bit comparison where `a[`*n*`..0] < b[`*n*`..0]` can be expressed as follows:

```
altb =  a[n] < b[n]
     #  (a[n-1] < b[n-1] & a[n] = b[n])
     #  (a[n-2] < b[n-2] & a[n..n-1] = b[n..n-1])
                 .
                 .
                 .
     # (a[0] < b[0] & a[n..1] = b[n..1])
```

In these equations, a cascade chain and a carry chain are used simultaneously to create two of the three comparator outputs. This type of comparator is implemented with the Arithmetic mode in FLEX 8000 LEs.

Figure 1 illustrates how each comparator output is generated. The AEQB signal is generated using a cascade chain. The AGTB signal is generated from a carry chain that shares an LE with the cascade chain. The ALTB signal uses an additional LE to generate a true signal when both AEQB and AGTB are false. Of the three outputs, AGTB has the smallest delay because it is generated using the high-speed carry chain. However, if ALTB is the speed-critical path, you can use the carry chain to generate ALTB instead.

Figure 1. Full Comparator Implemented with Cascade Chain & Carry Chain Logic



For more information about using carry and cascade chains, refer to *Application Note 36 (Designing for FLEX 8000 Devices)* in this handbook. For more information about the FLEX 8000 Arithmetic mode, refer to *Application Note 40 (FLEX 8000 Architecture)* in this handbook.

ALTERA

Barrel Shifters
in FLEX 8000 Devices

May 1994, ver. 2 | Application Brief 117

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_117.exe

Barrel shifters are circuits that perform a binary shift on a given input. Unlike a conventional shifter, which shifts data bits one position to the right or left, a barrel shifter can shift data to any position. Data bits shifted out at one end of a barrel shifter re-appear at the other end. This application brief describes two barrel shifter designs that are optimized for the FLEX 8000 architecture: one is optimized for area, the other is optimized for speed. Design techniques described in this application brief can be used to create design files optimized for the following characteristics (numbers in parentheses are for the speed-optimized barrel shifter):

Design Goals:		*Design Results:*		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	24 (38)	32.2 (98)
Product Term	✓ Speed	16 Bits	–	–
	✓ Area	32 Bits	–	–

Area-Optimized Barrel Shifter

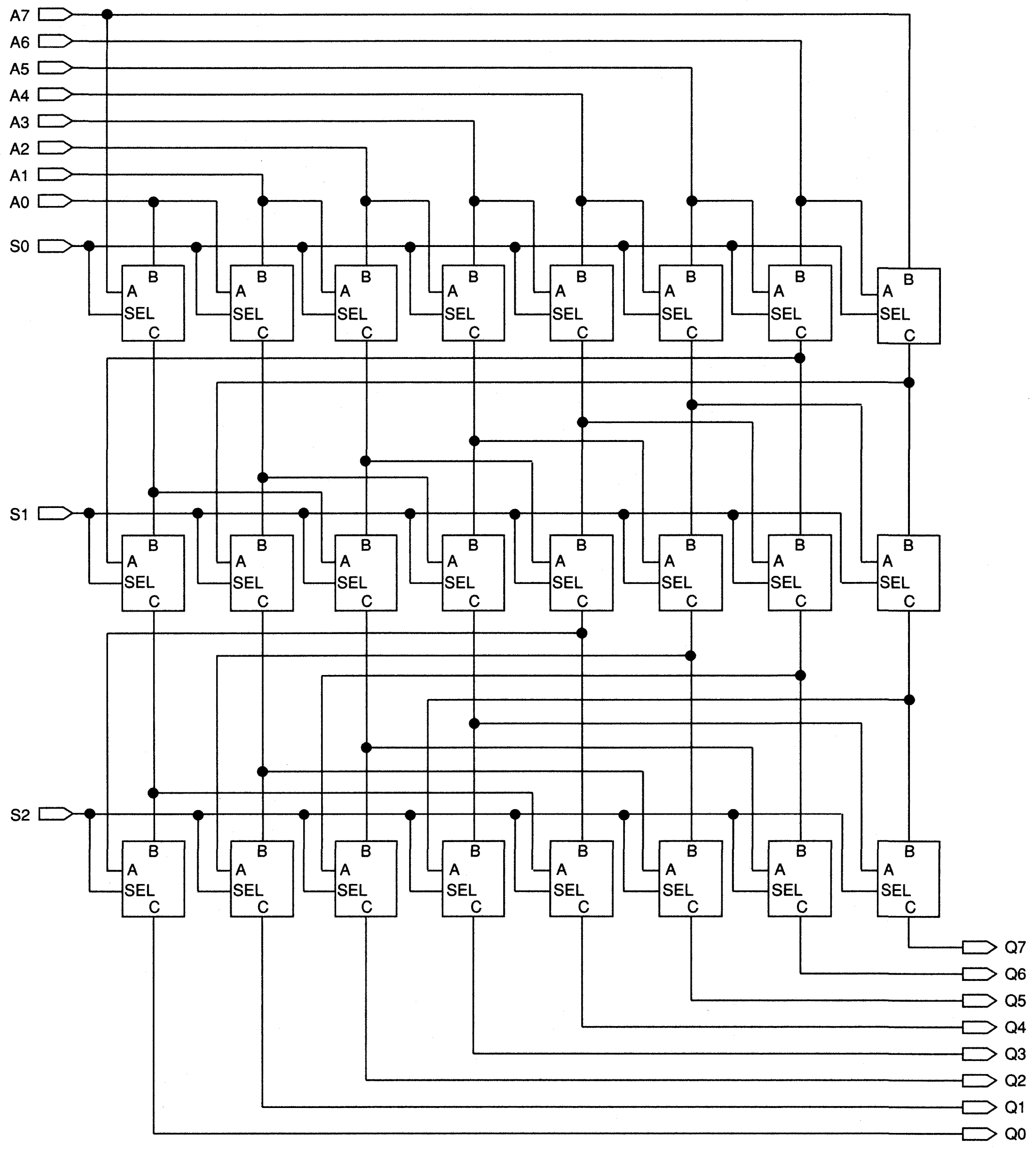
Table 1 shows the truth table for an 8-bit barrel shifter that is optimized for area. This design has 8 data bits, `a[7..0]`, and 3 shift control bits, `s[s..0]`. In this shifter, any shift greater than 7 can be performed by letting `s[2..0] =` *n* mod 7, where *n* is the number of shifts.

Table 1. Truth Table for 8-Bit Barrel Shifter

S2	S1	S0	A7..A0	Q7	Q6..Q1	Q0	Number of Shifts
0	0	0	A7..A0	Q7	Q6..Q1	Q0	0
0	0	1	A7..A0	Q0	Q7..Q2	Q1	1
0	1	0	A7..A0	Q1	Q0..Q3	Q2	2
0	1	1	A7..A0	Q2	Q1..Q4	Q3	3
1	0	0	A7..A0	Q3	Q2..Q5	Q4	4
1	0	1	A7..A0	Q4	Q3..Q6	Q5	5
1	1	0	A7..A0	Q5	Q4..Q7	Q6	6
1	1	1	A7..A0	Q6	Q5..Q0	Q7	7

A right barrel shift of 6 positions on the byte `B'10001100'` can be described as follows: if `S2 = 1, S1 = 1, S0 = 0`, and `A[7..0] = 10001100`, the output of the circuit is `Q[7..0] = 00110010`. In this example, the two leftmost bits of the input byte have been shifted to the two rightmost positions. See Figure 1.

Figure 1. 8-Bit Barrel Shifter



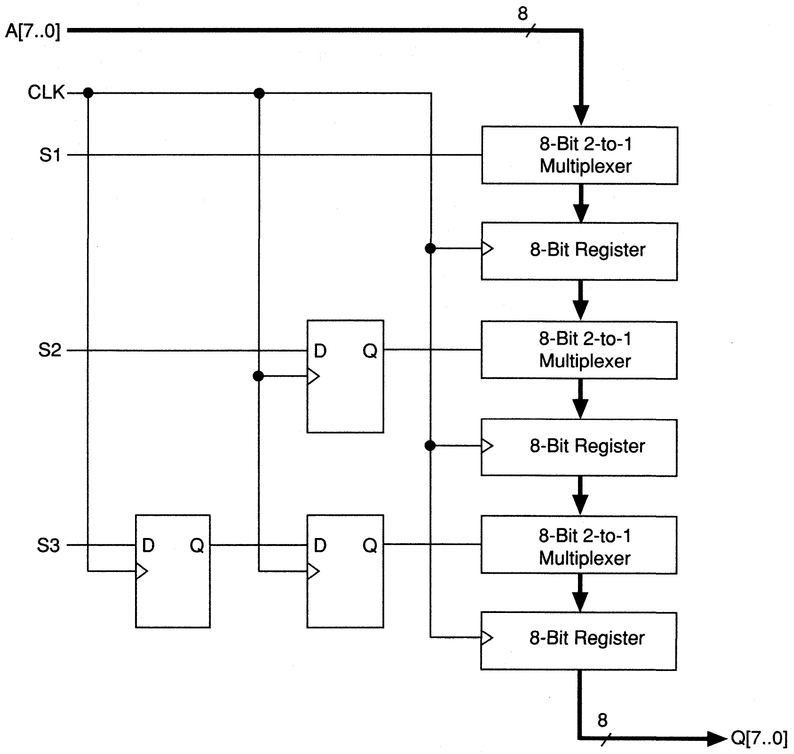
The barrel shifter design in Figure 1 consists of eight 2-to-1 multiplexers arranged into 3 rows with 8 multiplexers per row. Each row of multiplexers has a common select line where `S0` feeds the first row, `S1` the second, and `S2` the third. A "1" in the first row (`S0 = 1`) represents a shift of one bit to the right. When `S1 = 1`, the data is shifted by 2 bits; when `S2 = 1`, the data is shifted by 4 bits. When any of the shift control bits are zero, data is passed through the row of multiplexers to the next row. For example, when `S0 = 1`, the first row of multiplexers shifts one space to the right; when `S1 = 0`, the second row of multiplexers passes straight down without shifting; when `S2 = 1`, the third row of multiplexers shifts four spaces to the right. With these inputs, the total shift in Figure 1 is five positions.

Speed-Optimized Barrel Shifter

The speed-optimized barrel shifter design builds upon the first design and uses pipelining for synchronizaton. In pipelining, registers are inserted between combinatorial logic, decreasing register-to-register delays and increasing operating frequency. The second design is partitioned into stages with a register inserted between each intermediate stage. In this case, an intermediate stage corresponds to a shift performed by a single row.

Figure 2 shows an 8-bit pipelined barrel shifter created by inserting registers before each row of multiplexers. `A[7..0]` is registered before feeding the first 2-to-1 multiplexer. When pipelining a design, you must insert the pipelining registers so that the correct inputs correspond to the correct data byte at each stage. In Figure 2, `S2` requires three pipeline registers, `S1` requires two, and `S0` requires one for its row of multiplexers. With pipelining, this design runs at 98 MHz. For more information about pipelining, refer to *Application Note 36 (Designing with FLEX 8000 Devices)* in this handbook.

Figure 2. 8-Bit Barrel Shifter with Pipelining



ALTERA

Parity Generators in FLEX 8000 Devices

May 1994, ver. 1 Application Brief 130

Summary

Files using the techniques described in this application brief are available from the Altera BBS at (408) 954-0104 in the following self-extracting file:

ab_130.exe

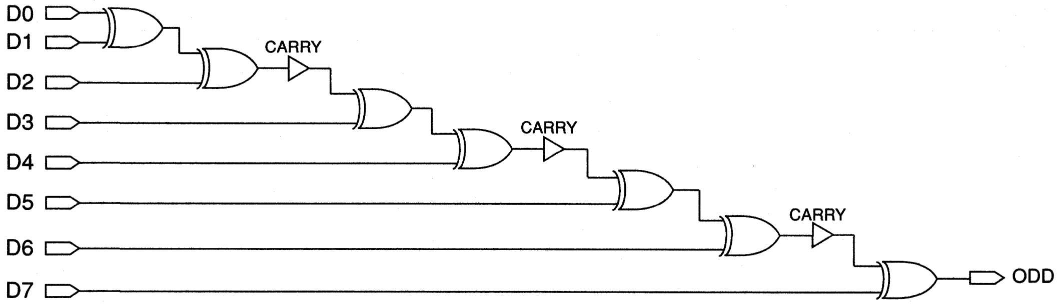
Parity generators are used to detect errors in data transmission. The data sender generates a parity signal by counting the number of ones in a data word and reporting whether that number is even or odd. The data receiver checks the parity signal to verify that the data was received without errors. In FLEX 8000 devices, you can implement high-speed parity generators using the carry chain feature, a fast (less than 1 ns) carry-forward function path between contiguous logic elements (LEs) within a Logic Array Block (LAB) and between adjacent LABs. You can also implement area-optimized parity generators that do not use the dedicated carry chain. Design techniques described in this application brief can be used to create design files optimized for the following characteristics (numbers in parentheses are for the parity generator without carry chains):

Design Goals:		***Design Results:***		
Architecture	**Optimization**	**Width**	**Logic Cells**	**Speed (MHz)**
✓ Look-Up Table	Routability	8 Bits	4	90.0 (66.7)
Product Term	✓ Speed	16 Bits	8	71.9 (66.7)
	✓ Area	32 Bits	17	51.3 (46.3)

Parity Generators with Carry Chains

Figure 1 shows a very-high-speed parity generator that uses carry chain logic. This parity generator accumulates the values of each bit in an 8-bit data word using a carry chain, and reports that the result is an odd number. This design can be extended to create parity generators for wider data words. Carry chain logic provides fast performance. However, since carry chains must be placed in contiguous LEs and LABs, long carry chains may reduce the routing resources available for implementing other logic. For more information about carry chains, refer to *Application Note 36 (Designing with FLEX 8000 Devices*) and *Application Note 40 (FLEX 8000 Architecture)* in this handbook.

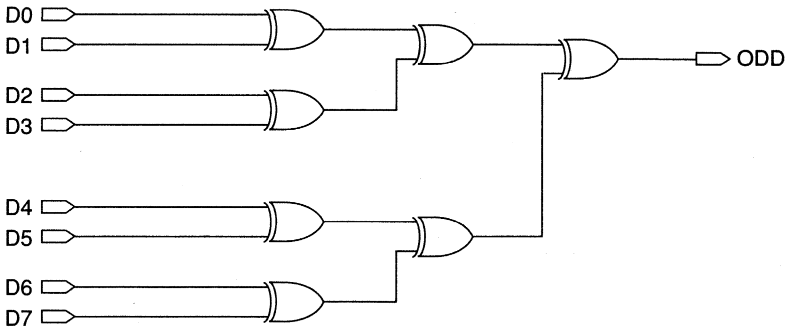
Figure 1. High-Speed Parity Generator Implemented with Carry Chain Logic



Parity Generators without Carry Chains

Figure 2 shows an 8-bit parity generator that does not use carry chain logic. This parity generator adds the values of each bit in an 8-bit data word using XOR gates. This design runs more slowly than the design shown in Figure 1, but also uses less area and leaves more routing resources available for implementing other logic. If you wish to implement even parity generation, you can invert the ODD output signal.

Figure 2. Parity Generator without Carry Chain Logic



Contents

May 1994

Section 9 General Information

9

General Information

ALTERA

Sales Offices, Distributors & Representatives

May 1994

Altera North American Regional Offices

NORTHERN CALIFORNIA
Altera Corporation
2610 Orchard Parkway
San Jose, CA 95134-2020
TEL: (408) 894-7900
FAX: (408) 428-0463

SOUTHERN CALIFORNIA
Altera Corporation
15375 Barranca Parkway
Suite B-201
Irvine, CA 92718
TEL: (714) 450-0262
FAX: (714) 587-9789

Altera Corporation
15760 Ventura Blvd.
Suite 700
Encino, CA 91436
TEL: (818) 990-1326
FAX: (818) 990-4876

COLORADO
Altera Corporation
Denver Technology Center
7900 East Union Avenue, # 1100
Denver, CO 80237
TEL: (303) 694-5352
FAX: (303) 694-5351

GEORGIA
Altera Corporation
9755 Dogwood Road
Suite 350
Roswell, GA 30075
TEL: (404) 594-7621
FAX: (404) 998-9830

ILLINOIS
Altera Corporation
150 N. Martingale Road, Suite 838
Schaumburg, IL 60173
TEL: (708) 240-0313
FAX: (708) 240-0266

MASSACHUSETTS
Altera Corporation
238 Littleton Road
Suite 204
Westford, MA 01886
TEL: (508) 392-1100
FAX: (508) 392-1157

NEW JERSEY
Altera Corporation
575 State Highway 28
Raritan, NJ 08869
TEL: (908) 526-9400
FAX: (908) 526-5471

TEXAS
Altera Corporation
5080 Spectrum Drive, Suite 812W
Dallas, TX 75248
TEL: (214) 701-2330
FAX: (214) 701-2331

ONTARIO
Altera Corporation
300 March Road, Suite 601B
Kanata, Ontario K2K 2E2
Canada
TEL: (613) 599-3141
FAX: (613) 599-3143

Altera International Sales Offices

UNITED STATES
(CORPORATE HEADQUARTERS)
Altera Corporation
2610 Orchard Parkway
San Jose, CA 95134-2020
USA
TEL: (408) 894-7000
TLX: 888496
FAX: (408) 433-3943

UNITED KINGDOM
(EUROPEAN HEADQUARTERS)
Altera U.K. Limited
Solar House
Globe Park, Fieldhouse Lane
Marlow, Bucks SL7 ITB
England
TEL: (44) 628 488800
FAX: (44) 628 890078

Altera International Sales Offices
(continued)

FRANCE
Altera France
Zac La Sabliere
4, Rue Maryse Bastie
91430-IGNY
France
TEL: (33) 1 69 85 5630
FAX: (33) 1 69 85 5614

GERMANY
Altera GmbH
Max-Planck-Straße 5
D-85716 Unterschleisshelm
Germany
TEL: (49) 89/3218250
FAX: (49) 89/32182579

ITALY
Altera Italia
Corso Lombardia 75
Autoporto Pescarito
10099 San Mauro, Torinese
(Torino)
Italy
TEL: (39) 11 223 8588
FAX: (39) 11 223 8589

JAPAN
Altera Japan K.K.
Ichikawa Gakugeidai Building
2nd Floor
12-8 Takaban 3-chome
Meguro-ku, Tokyo 152
Japan
TEL: (81) 33 716-2241
FAX: (81) 33 716-7924

North American Distributors

Arrow/Schweber Electronics Group

Future Electronics (Canada only)

Newark Electronics

Pioneer-Standard Electronics

Semad (Canada only)

Wyle Laboratories

U.S. Sales Representatives

ALABAMA
EnVision, Inc.
1009 Henderson Road, Suite 400B
Huntsville, AL 35816
TEL: (205) 721-1788
FAX: (205) 721-1789

ARIZONA
Oasis Sales, Inc.
301 E. Bethany Home Road #A135
Phoenix, AZ 85012
TEL: (602) 277-2714
FAX: (602) 263-9352

ARKANSAS
Technical Marketing, Inc.
3320 Wiley Post Road
Carrollton, TX 75006
TEL: (214) 387-3601
FAX: (214) 387-3605

CALIFORNIA
Addem
1015 Chestnut Street #F2
Carlsbad, CA 92008
TEL: (619) 729-9216
FAX: (619) 729-6408

Altera Corporation
2610 Orchard Parkway
San Jose, CA 95134-2020
TEL: (408) 894-7900
FAX: (408) 428-0463

CALIFORNIA *(continued)*
Infinity Sales
20 Corporate Park, Suite 100
Irvine, CA 92714
TEL: (714) 833-0300
FAX: (714) 833-0303

Sierra Technical Sales
23566 Woodhaven Place
Auburn, CA 95602
TEL: (916) 268-3357
FAX: (916) 268-0192

COLORADO
Compass Marketing and Sales, Inc.
5600 S. Quebec Street, Suite 350D
Englewood, CO 80111
TEL: (303) 721-9663
FAX: (303) 721-0195

CONNECTICUT
Technology Sales, Inc.
237 Hall Avenue
Wallingford, CT 06492
TEL: (203) 269-8853
FAX: (203) 269-2099

DELAWARE
BGR Associates
Evesham Commons
525 Route 73, Suite 100
Marlton, NJ 08053
TEL: (609) 983-1020
FAX: (609) 983-1879

DISTRICT OF COLUMBIA
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
TEL: (410) 995-1900
FAX: (410) 964-3364

FLORIDA
EIR, Inc.
1057 Maitland Center Commons
Maitland, FL 32751
TEL: (407) 660-9600
FAX: (407) 660-9091

GEORGIA
EnVision, Inc.
3220 Pointe Parkway, Suite 1400
Norcross, GA 30092
TEL: (404) 840-1055
FAX: (404) 840-1048

IDAHO
Compass Marketing and Sales, Inc.
5 Triad Center, Suite 320
Salt Lake City, UT 84180
TEL: (801) 322-0391
FAX: (801) 322-0392

Phase II Technical Sales
550 Kirkland Way, Suite 100
Kirkland, WA 98033
TEL: (206) 828-8182
FAX: (206) 828-7472

ILLINOIS
AEM, Inc.
11520 St. Charles Rock Road, Suite 131
Bridgeton, MO 63044
TEL: (314) 298-9900
FAX: (314) 298-8660

Oasis Sales Corporation
1101 Tonne Road
Elk Grove Village, IL 60007
TEL: (708) 640-1850
FAX: (708) 640-9432

INDIANA
Electro Reps, Inc.
7240 Shadeland Station, Suite 275
Indianapolis, IN 46256
TEL: (317) 842-7202
FAX: (317) 841-0230

Electro Reps, Inc.
125 Airport North Office Park
Fort Wayne, IN 46825
TEL: (219) 489-8205
FAX: (219) 489-8408

IOWA
AEM, Inc.
4001 Shady Oak Drive
Marion, IA 52302
TEL: (319) 377-1129
FAX: (319) 377-1539

KANSAS
AEM, Inc.
8843 Long Street
Lenexa, KS 66215
TEL: (913) 888-0022
FAX: (913) 888-4848

KENTUCKY
Electro Reps, Inc.
7240 Shadeland Station, Suite 275
Indianapolis, IN 46256
TEL: (317) 842-7202
FAX: (317) 841-0230

Lyons Corp.
4812 Frederick Road, Suite 101
Dayton, OH 45414
TEL: (513) 278-0714
FAX: (513) 278-3609

LOUISIANA
Technical Marketing, Inc.
2901 Wilcrest Drive, Suite 139
Houston, TX 77042
TEL: (713) 783-4497
FAX: (713) 783-5307

MAINE
Altera Corporation
238 Littleton Road, Suite 204
Westford, MA 01886
TEL: (508) 392-1100
FAX: (508) 392-1157

MARYLAND
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
TEL: (410) 995-1900
FAX: (410) 964-3364

MASSACHUSETTS
Altera Corporation
238 Littleton Road, Suite 204
Westford, MA 01886
TEL: (508) 392-1100
FAX: (508) 392-1157

Technology Sales, Inc.
237 Hall Avenue
Wallingford, CT 06492
TEL: (203) 269-8853
FAX: (203) 269-2099

MICHIGAN
Rathsburg Associates, Inc.
41100 Bridge Street
Novi, MI 48375
TEL: (810) 615-4000
FAX: (810) 615-4001

MINNESOTA
Cahill, Schmitz & Cahill, Inc.
315 N. Pierce
St. Paul, MN 55104
TEL: (612) 646-7217
FAX: (612) 646-4484

U.S. Sales Representatives
(continued)

MISSISSIPPI
EnVision, Inc.
1009 Henderson Road, Suite 400B
Huntsville, AL 35816
TEL: (205) 721-1788
FAX: (205) 721-1789

MISSOURI
AEM, Inc.
11520 St. Charles Rock Road, Suite 131
Bridgeton, MO 63044
TEL: (314) 298-9900
FAX: (314) 298-8660

MONTANA
Compass Marketing
5 Triad Center, Suite 320
Salt Lake City, UT 84180
TEL: (801) 322-0391
FAX: (801) 322-0392

NEBRASKA
AEM, Inc.
4001 Shady Oak Drive
Marion, IA 52302
TEL: (319) 377-1129
FAX: (319) 377-1539

NEVADA
Oasis Sales, Inc.
301 E. Bethany Home Road #A135
Phoenix, AZ 85012
TEL: (602) 277-2714
FAX: (602) 263-9352

Sierra Technical Sales
23566 Woodhaven Place
Auburn, CA 95602
TEL: (916) 268-3357
FAX: (916) 268-0192

NEW HAMPSHIRE
Altera Corporation
238 Littleton Road, Suite 204
Westford, MA 01886
TEL: (508) 392-1100
FAX: (508) 392-1157

NEW JERSEY
BGR Associates
Evesham Commons
525 Route 73, Suite100
Marlton, NJ 08053
TEL: (609) 983-1020
FAX: (609) 983-1879

ERA, Inc.
354 Veterans Memorial Highway
Commack, NY 11725
TEL: (516) 543-0510
FAX: (516) 543-0758

NEW MEXICO
Nelco Electronix
3240 C Juan Tabo Blvd. NE
Albuquerque, NM 87111
TEL: (505) 293-1399
FAX: (505) 293-1011

NEW YORK
ERA, Inc.
354 Veterans Memorial Highway
Commack, NY 11725
TEL: (516) 543-0510
FAX: (516) 543-0758

Technology Sales, Inc.
920 Perinton Hills Office Park
Fairport, NY 14450
TEL: (716) 223-7500
FAX: (716) 223-5526

NORTH CAROLINA
EnVision, Inc.
3200 Wake Forest Rd., Suite 205
Raleigh, NC 27609
TEL: (919) 878-3080
FAX: (919) 878-3090

NORTH DAKOTA
Cahill, Schmitz & Cahill, Inc.
315 N. Pierce
St. Paul, MN 55104
TEL: (612) 646-7217
FAX: (612) 646-4484

OHIO
The Lyons Corporation
4812 Frederick Road, Suite 101
Dayton, OH 45414
TEL: (513) 278-0714
FAX: (513) 278-3609

The Lyons Corporation
4615 W. Streetsboro Road
Richfield, OH 44286
TEL: (216) 659-9224
FAX: (216) 659-9227

The Lyons Corporation
248 N. State Street
Westerville, OH 43081
TEL: (614) 895-1447
FAX: (513) 278-3609

OKLAHOMA
Technical Marketing, Inc.
3320 Wiley Post Road
Carrollton, TX 75006
TEL: (214) 387-3601
FAX: (214) 387-3605

OREGON
Phase II Technical Sales
4900 SW Griffith Drive, Suite 110
Beaverton, OR 97005
TEL: (503) 643-6455
FAX: (503) 626-7442

PENNSYLVANIA
BGR Associates
Evesham Commons
525 Route 73, Suite 100
Marlton, NJ 08053
TEL: (609) 983-1020
FAX: (609) 983-1879

PENNSYLVANIA *(continued)*
The Lyons Corporation
248 N. State Street
Westerville, OH 43081
TEL: (614) 895-1447
FAX: (513) 278-3609

RHODE ISLAND
Altera Corporation
238 Littleton Road, Suite 204
Westford, MA 01886
TEL: (508) 392-1100
FAX: (508) 392-1157

SOUTH CAROLINA
EnVision, Inc.
113 Stonegate Drive
Columbia, SC 29223
TEL: (803) 699-3360
FAX: (803) 699-0377

SOUTH DAKOTA
Cahill, Schmitz & Cahill, Inc.
315 North Pierce
St. Paul, MN 55104
TEL: (612) 646-7217
FAX: (612) 646-4484

TENNESSEE
EnVision, Inc.
8120 Sawyer Brown Rd., Suite 105
Nashville, TN 37221
TEL: (615) 622-7666
FAX: (615) 622-7652

TEXAS
Technical Marketing, Inc.
3701 Executive Center Drive #205
Austin, TX 78731
TEL: (512) 343-6976
FAX: (512) 343-7986

Technical Marketing, Inc.
3320 Wiley Post Road
Carrollton, TX 75006
TEL: (214) 387-3601
FAX: (214) 387-3605

Technical Marketing, Inc.
2901 Wilcrest Drive, Suite 139
Houston, TX 77042
TEL: (713) 783-4497
FAX: (713) 783-5307

UTAH
Compass Marketing and Sales, Inc.
5 Triad Center, Suite 320
Salt Lake City, UT 84180
TEL: (801) 322-0391
FAX: (801) 322-0392

VERMONT
Altera Corporation
238 Littleton Road, Suite 204
Westford, MA 01886
TEL: (508) 392-1100
FAX: (508) 392-1157

VIRGINIA
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
TEL: (410) 995-1900
FAX: (410) 964-3364

WASHINGTON
Phase II Technical Sales
550 Kirkland Way, Suite 100
Kirkland, WA 98033
TEL: (206) 828-8182
FAX: (206) 828-7472

WEST VIRGINIA
Robert Electronic Sales
5525 Twin Knolls Road, Suite 325
Columbia, MD 21045
TEL: (410) 995-1900
FAX: (410) 964-3364

WISCONSIN
Oasis Sales Corporation
1305 N. Barker Road
Brookfield, WI 53005
TEL: (414) 782-6660
FAX: (414) 782-7921

WYOMING
Compass Marketing and Sales, Inc.
5 Triad Center, Suite 320
Salt Lake City, UT 84180
TEL: (801) 322-0391
FAX: (801) 322-0392

Canadian Sales Representatives

ALBERTA
Kaytronics
6815-8th Street NE, Suite 179
Calgary, Alberta T2E 7H7
Canada
TEL: (403) 275-7000
FAX: (403) 295-0732

BRITISH COLUMBIA
Kaytronics
#102-4585 Canada Way
Burnaby, BC V5G 4L6
Canada
TEL: (604) 294-2000
FAX: (604) 294-4585

ONTARIO
Kaytronics
405 Britannia Road E #206
Mississauga, Ontario L4Z 3E6
Canada
TEL: (905) 507-6400
FAX: (905) 507-6444

ONTARIO *(continued)*
Altera Corporation (Canada)
300 March Road, Suite 603
Kanata, Ontario K2K 2E2
Canada
TEL: (613) 564-0080
FAX: (613) 564-0087

QUEBEC
Kaytronics
5800 Thimens Boulevard
Ville St. Laurent, Quebec H4S 1S5
Canada
TEL: (514) 745-5800
FAX: (514) 745-5858

International Distributors

ARGENTINA
YEL S.R.L.
Virrey Cevallos 143
1077 Buenos Aires
Argentina
TEL: (54) 1-372-7140
TLX: (390) 18605 (YEL AR)
FAX: (54) 1-476-2551

AUSTRALIA
Veltek Pty. Ltd.
18 Harker St.
Burwood, Victoria 3125
Australia
TEL: (61) 3-808-7511
FAX: (61) 3-808-5473

AUSTRIA
Eurodis Electronics GmbH
Lamezanstraße 10
A-1232 Wien
Austria
TEL: (43) 1-610620
TLX: (847) 134404 (HIT)
FAX: (43) 1-61062151

BELGIUM
D&D Electronics
VIle Olympiadelaan 93
B-2020 Antwerpen
Belgium
TEL: (32) 3-827-7934
TLX: (846) 73121 (DDELEC BU)
FAX: (32) 3-828-7254

BRAZIL
União Digital Ltda.
Rua Georgia 69
04559-010
São Paulo - SP
Brazil
TEL: (55) 11 536-4121
FAX: (55) 11 533-6780

CHINA
CIDC
No. 1, Gao Jia Yuan
Dongzhimenwai, Chao Yang Qu
Beijing 100015
People's Republic of China
TEL: (86) 1-436-5577
FAX: (86) 1-436-4487

DENMARK
E.V. Johanssen Elektronik A/S
Titangade 15
DK-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
SF-02201 Espoo
Finland
TEL: (358) 0-452-621
TLX: (857) 123212 (YLEOY SF)
FAX: (358) 0-452-62231

FRANCE
Arrow Electronique S.A.
73-79, Rue des Solets
Silic 585
94663 Rungis Cedex
France
TEL: (33) 1 49 784978
FAX: (33) 1 49 780596

Tekelec Airtronic SA
Cité des Bruyères
Rue Carle Vernet
92310 Sèvres
France
TEL: (33) 1 46 23 24 25
TLX: (842) 634018 (TKLEC AF)
FAX: (33) 1 45 07 21 91

GERMANY
Avnet E2000 GmbH
Elektronische Bauelemente
Stahlgruberring 12
D-81829 München
Germany
TEL: (49) 89/45110-01
TLX: (841) 522561 (ELEC D)
FAX: (49) 89/45110-129

Jermyn GmbH
Im Dachsstueck 9
65549 Limburg
Germany
TEL: (49) 64 31 508 000
FAX: (49) 64 31 508 289

SASCO GmbH
Hermann-Oberth-Straße 16
D-85640 Putzbrunn
Germany
TEL: (49) 89/46110
FAX: (49) 89/4611270

GREECE
Micronics Limited
46 Kritis Street
16451-Argyroupolis
Athens
Greece
TEL: (30) 1 991-4786
FAX: (30) 1 995-1814

HONG KONG
Lestina International Ltd.
14/F Park Tower, 15 Austin Road
Tsimshatsui
Hong Kong
TEL: (852) 735-1736
FAX: (852) 730-5260

INDIA
Sritech Information Technology Ltd.
744/51, 2nd Floor, Chintal Plaza
33rd Cross, 10th Main
4th Block, Jayanagar
Bangalore 560 011
India
TEL: (91) 80-6-640-661
TLX: (953) 08458162 (SRIS IN)
FAX: (91) 80-6-633-508

ISRAEL
Active Technologies
80 Express Street
Plainview, NY 11803
USA
TEL: (516) 938-4848
FAX: (516) 938-4141

Vectronics Ltd.
60 Medinat Hayehudim Street
P.O. Box 2024
Herzlia B 46120
Israel
TEL: (972) 9-556-070
TLX: (922) 342579 (VECO IL)
FAX: (972) 9-556-508

ITALY
Lasi Elettronica Div. Silverstar Ltd. S.P.A.
Viale Fulvio Testi, 280
20126 Milano
Italy
TEL: (39) 2 661 431
FAX: (39) 2 661 01385

JAPAN
Altima Corporation
Hakusan High-Tech Park
1-22-2 Hakusan, Midori-Ku
Yokohama City 226
Japan
TEL: (81) 45-939-6113
FAX: (81) 45-939-6114

Paltek Corporation
1-3-3 Azamino-Minami
Midori-Ku, Yokohama City
Kanagawa-Ken, 225
Japan
TEL: (81) 45-910-1381
TLX: (781) 02425205 (PALTEK J)
FAX: (81) 45-910-1390

KOREA
MJL Korea, Ltd.
Hahntoo New Building, 19/F
27-1 Yeouido-Dong
Yungdeungpo-Ku
Seoul 150-010
Korea
TEL: (82) 2-767-2200
FAX: (82) 2-767-2221

International Distributors

(continued)

LATIN AMERICA
Intectra, Inc.
2629 Terminal Blvd.
Mountain View, CA 94043
USA
TEL: (415) 967-8818
TLX: 345545 (INTECTRA MTNV)
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
N.C. ScandComp Norway A/S
Aslakveien 20 F
0753 Oslo 7
Norway
TEL: (47) 22-500650
TLX: (856) 77144 (ELTRO N)
FAX: (47) 22-502777

SINGAPORE
Serial System Pte. Ltd.
11 Jalan Mesin
Standard Industrial Building #06-00
Singapore 1336
TEL: (65) 280-0200
FAX: (65) 286-1812

SPAIN
Selco
CTRA DE LA Coruña KM.18.200
Via de Servicio – Dirección Villalba
28230 Las Rozas DE Madrid
Spain
TEL: (34) 1-637-1011
FAX: (34) 1-637-1506
(34) 1-637-1408

SOUTH AFRICA
Prime Source (Pty) Ltd.
3 Olympia Street
Marlboro, Sandton
P.O. Box 46169
Orange Grove 2119
Republic of South Africa
TEL: (27) 11-444-7237
FAX: (27) 11-444-7298

SWEDEN
Bexab Sweeden AB
P.O. Box 523
S-183 25 Taby
Sweden
TEL: (46) 8 630 8800
TLX: 149 32
FAX: (46) 8 732 7058

SWITZERLAND
Elbatex
Hardstraße 72
CH5430 Wettingen
Switzerland
TEL: (41) 56 27 57 77
FAX: (41) 56 26 14 86

MPI Distribution AG
Täfernstraße 20
5405 Dättwil
Switzerland
TEL: (41) 56 83 55 55
FAX: (41) 56 83 30 20

TAIWAN
Galaxy Far East Corp.
8F-6, 390 Sec. 1
Fu Hsing South Road
Taipei
Taiwan R.O.C.
TEL: (886) 2-705-7266
TLX: (785) 26110 (GALAXYER)
FAX: (886) 2-708-7901

Jeritronics Ltd.
Floor 7B, #267, Sec. 3
Cheng-Teh Road
Taipei
Taiwan R.O.C.
TEL: (886) 2-585-1636
FAX: (886) 2-586-4736

THAILAND
Nu-Era Co. Ltd.
769/8 J.T. Building
Soisoonvijai 4/1
Rama 9 Road, Huaykwang
Bangkok 10310
Thailand
TEL: (66) 2-318-6453
FAX: (66) 2-318-6454

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

Glossary

May 1994

A

Active Parallel Down (APD) A configuration scheme in which a byte-wide parallel PROM loads the design data into a FLEX 8000 device. The FLEX 8000 device first generates an address; the PROM subsequently returns the next byte of data. Addresses are generated by the FLEX 8000 device sequentially in descending order (3FFFFh to 00000h). MAX+PLUS II can generate Hexadecimal (Intel-Format) Files (**.hex**) that contain the data for configuring FLEX 8000 devices in an APD configuration scheme.

Active Parallel Up (APU) A configuration scheme in which a byte-wide parallel PROM loads the design data into a FLEX 8000 device. The FLEX 8000 device first generates an address; the PROM subsequently returns the next byte of data. Addresses are generated by the FLEX 8000 device sequentially in ascending order (00000h to 3FFFFh). MAX+PLUS II can generate Hexadecimal (Intel-Format) Files (**.hex**) that contain the data for configuring FLEX 8000 devices in an APU configuration scheme.

Active Serial (AS) A configuration scheme in which a serial EPROM loads the design data into a FLEX 8000 device. The MAX+PLUS II Compiler automatically generates a Programmer Object File (**.pof**) for programming serial EPROM devices, e.g., the EPC1213 Configuration EPROM, whenever a FLEX 8000 project is compiled.

Altera Hardware Description Language (AHDL) Altera's design entry language. AHDL is completely integrated into MAX+PLUS II, and allows the designer to enter and edit Text Design Files (**.tdf**) with the MAX+PLUS II Text Editor or any standard text editor, then compile, simulate, and program projects within MAX+PLUS II. AHDL supports Boolean equation, state machine, conditional, and decode logic. AHDL also provides access to all Altera macrofunctions.

array Clock A Clock signal that passes through the logic array of a device before arriving at the Clock input of a register.

Assembler The Compiler module that creates one or more Programmer Object Files (**.pof**), SRAM Object Files (**.sof**), and optional JEDEC Files (**.jed**) for programming Altera devices.

B

BitBlaster A cable that allows both PC and workstation users to configure a FLEX 8000 device in a prototype system. This capability functions independently from the MAX+PLUS II Programmer or any other programming hardware. The Altera BitBlaster connects a standard RS-232 serial port on a PC or workstation to a single target FLEX 8000 device in a prototype system.

C

Carry Chain option A FLEX 8000 logic synthesis option that controls the use of carry chain logic. When this option is set to AUTO, it directs the Compiler's Logic Synthesizer module to insert carry chain logic—i.e., insert CARRY buffers—wherever it is useful. When defining a logic synthesis style, designers can specify the maximum allowable length of a chain of these synthesized CARRY buffers. The AUTO setting does not force the Logic Synthesizer to use carry logic and has no effect on CARRY primitives that have been entered in design files.

When the Carry Chain option is set to IGNORE, it directs the Logic Synthesizer to ignore CARRY buffers that have been entered manually in design files. When this option is set to MANUAL, the Logic Synthesizer uses only the CARRY primitives that have been manually entered in design files.

Cascade Chain option A FLEX 8000 logic synthesis option that controls the use of cascade chain logic. When this option is set to AUTO, it directs the Compiler's Logic Synthesizer module to insert cascade logic—i.e., insert CASCADE buffers—wherever it is useful. When defining a logic synthesis style, designers can specify the maximum allowable length of a chain of these synthesized CASCADE buffers. The AUTO setting does not force the Logic Synthesizer to use cascade logic and has no effect on CASCADE primitives that have been entered in design files.

When the Cascade Chain option is set to IGNORE, it directs the Logic Synthesizer to ignore CASCADE buffers that have been entered manually in design files. When this option is set to MANUAL, the Logic Synthesizer uses only the CASCADE primitives that have been manually entered in design files.

CerDIP Ceramic Dual In-Line Package. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

Classic An Altera device family based on Altera's original EPLD architecture. This EEPROM- and EPROM-based family includes EP330, EP610, EP910, and EP1810 devices.

Compiler Netlist Extractor The MAX+PLUS II Compiler module that creates Compiler Netlist Files (**.cnf**), Hierarchy Interconnect Files (**.hif**), and Symbol Files (**.sym**) from the design files for a project. This module includes built-in EDIF, VHDL, and Xilinx Netlist Readers that convert EDIF Netlist Files (**.edf**), VHDL Design Files (**.vhd**), and Xilinx Netlist Format Files (**.xnf**) created with industry-standard CAE software. The Compiler Netlist Extractor also checks each design file in a project for problems such as duplicate node names, missing inputs and outputs, and outputs that are tied together.

Configuration EPROM Altera's family of serial EPROMs, which are designed to configure FLEX 8000 devices. This device family includes the EPC1213 and EPC1064 devices.

configuration scheme The method used to load data into a FLEX 8000 device. Six configuration schemes are available:

Active Serial (AS)
Active Parallel Up (APU)
Active Parallel Down (APD)
Passive Parallel Asynchronous (PPA)
Passive Parallel Synchronous (PPS)
Passive Serial (PS)

For complete information on FLEX 8000 configuration schemes, see *Application Brief 33 (Configuring FLEX 8000 Devices).*

continuity checking A test for open circuits between device pins and programming adapter sockets. This test verifies that a device is properly seated in the socket of the adapter.

CQFP Ceramic Quad Flat Pack. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

D

Database Builder The MAX+PLUS II Compiler module that builds a single, fully flattened database that integrates all files in a project hierarchy. It also examines the logical completeness and consistency of the project and checks for boundary connectivity and syntactical errors.

dedicated input pin A pin that may only be used as an input to the device.

device An Altera programmable logic device, including Classic, MAX 5000/EPS464, MAX 7000, FLEX 8000, EPS448, EPB2001, and Configuration EPROM devices.

device family A group of Altera programmable logic devices with the same fundamental architecture. Altera device families are the Classic, MAX 5000/EPS464, MAX 7000, FLEX 8000, and Configuration EPROM families.

E

EDIF Electronic Design Interchange Format. An industry-standard format for transmitting design data. AN EDIF 2 0 0 or 2 9 0 netlist file is generated from a schematic design or a VHDL or Verilog HDL design that has been processed with an industry-standard synthesis tool. The netlist file is then imported into MAX+PLUS II as an EDIF Input File (**.edf**). The MAX+PLUS II Compiler can generate one or more EDIF Output Files (**.edo**) in EDIF 2 0 0 or 2 9 0 format that contain functional or timing information for simulation with a standard EDIF simulator.

EEPROM Electrically Erasable Programmable Read-Only Memory. A form of reprogrammable semiconductor memory in which the contents (program) can be erased by subjecting the device to appropriate electrical signals.

EPLD Erasable Programmable Logic Device, i.e., an Altera device that is a member of the Classic, MAX 5000/EPS464, or MAX 7000 family.

EPROM Erasable Programmable Read-Only Memory. A form of reprogrammable semiconductor memory in which the contents (program) can be erased by subjecting the device to ultraviolet light of the proper wavelength. See *Operating Requirements for Altera Devices* and *Technology & Reliability* in the current data book for more information.

external timing parameters Factory-tested, guaranteed worst-case values. Examples: t_{PD1}, t_{CO1}, f_{CNT}. In FLEX 8000 device data sheets, external timing parameters are listed under "External Reference Timing Characteristics."

F

family-specific macrofunction An Altera-provided macrofunction that contains logic optimized for the architecture of a specific device family. The functionality of a family-specific macrofunction is always the same, regardless of the device family for which it is designed. However, the actual primitives and nodes used within the macrofunction file can vary from family to family to take advantage of different device architectures, thus providing higher performance and more efficient implementation.

FastTrack Interconnect Dedicated connection paths that span the entire width and height of a FLEX 8000 device. These connection paths allow signals to travel between all Logic Array Blocks (LABs) in a device.

Fitter The MAX+PLUS II Compiler module that fits a project into one or more devices. The Fitter selects appropriate interconnection paths, and pin and logic cell assignments. It also generates part of the Report File (**.rpt**) and Fit File (**.fit**) for the project.

FLEX Download Cable A cable used to download SRAM Object File (**.sof**) data in a passive serial (PS) configuration scheme to a FLEX 8000 device in an in-system circuit. FLEX 8000 devices can be configured with the FLEX Download Cable to allow in-circuit testing and prototyping on the circuit board.

FLEX 8000 An Altera device family based on Flexible Logic Element MatriX architecture. This SRAM-based family offers high-performance, register-intensive, high-gate-count devices. The family includes the EPF8282, EPF8282V, EPF8636, EPF8452, EPF8820, EPF81188, and EPF81500 devices.

flipflop or **register** An edge-triggered, clocked storage unit that stores a single bit of data. A low-to-high transition on the Clock signal changes the output of the flipflop, based on the value of the data input(s). This value is maintained until the next low-to-high transition of the Clock, or until the flipflop is preset or cleared. Depending on the architecture of the device family, a register can be programmed as a level-sensitive flow-through latch or as an edge-triggered D,T, JK, or SR flipflop.

functional simulation A MAX+PLUS II Simulator mode that uses a functional Simulator Netlist File (**.snf**) to simulate the logical performance of a project (without timing information).

Functional SNF Extractor The MAX+PLUS II Compiler module that creates the functional Simulator Netlist File (**.snf**) required for functional simulation.

G

global Clear A signal from a dedicated input pin that does not pass through the logic array before arriving at the Clear input of a register. In FLEX 8000 devices, a global Clear can come from any of the dedicated inputs. MAX 7000 devices have input pins that can be used either as global Clear sources or dedicated inputs to the device.

global Clock A signal from a dedicated input pin that does not pass through the logic array before arriving at the Clock input of a register. In FLEX 8000 devices, a global Clock can come from any of the four dedicated input pins. MAX 7000, MAX 5000, EPS464, and EP1810 devices have input pins that can be used either as global Clock sources or dedicated inputs to the device. EP910, EP610, and EP330 devices have dedicated Clock input pins.

H

Hexadecimal (Intel-Format) File (.hex) A hexadecimal file that supports the Active Parallel Up (APU) and Active Parallel Down (APD) configuration schemes for configuring FLEX 8000 devices.

I

interconnect timing parameters Internal timing parameters for the interconnect in FLEX 8000 devices.

internal timing parameters Worst-case delays based on external timing parameters. Internal timing parameters cannot be measured explicitly, and should be used only for estimating device performance. Post-compilation timing simulation or timing analysis is required to determine actual worst-case performance. Examples: t_{LAD}, t_{CGEN}, t_{CLR}. In FLEX 8000 device data sheets, internal timing parameters are listed under "Internal Timing Characteristics."

J

JEDEC File (.jed) An ASCII file that contains programming information. JEDEC Files provide an industry-standard format for transferring information between a data preparation system and a logic device programmer. The MAX+PLUS II Programmer can optionally save programming data in JEDEC File format and use a JEDEC File to program the following Altera devices: EP330, EP610, EP910, EP1810, EPM5016, and EPM5032 devices.

The Programmer can also use JEDEC Files generated by A+PLUS software to program Classic devices.

JLCC Ceramic J-Lead Chip Carrier. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

JTAG Joint Test Action Group. A set of specifications that enables board- and chip-level functional verification of a board during production.

JTAG boundary-scan testing Testing that isolates a device's internal circuitry from its I/O circuitry. This testing is made possible by the Joint Test Action Group (JTAG) Boundary-Scan Test (BST) architecture that is available in the FLEX 8000 EPF8282, EPF8282V, EPF8636, EPF8820, and EPF81500 devices. Serial data is shifted into boundary-scan cells in the device; observed data is shifted out and externally compared to expected results. Boundary-scan testing offers efficient PC board testing, providing an electronic substitute for the traditional "bed of nails" test fixture.

L

library of parameterized modules (LPM) A technology-independent library of logic functions. Parameterized modules from the LPM support architecture-independent design entry for Altera Classic, MAX 5000/EPS464, MAX 7000, and FLEX 8000 devices. The MAX+PLUS II Compiler's EDIF Netlist Reader module includes built-in compilation support for many parameterized modules in the LPM.

linked simulation A MAX+PLUS II Simulator mode that uses a linked Simulator Netlist File (**.snf**) to simulate the logical performance of a super-project consisting of multiple, linked individual projects. A linked simulation uses the timing and/or functional netlist information from the combined SNFs of the individual linked sub-projects.

Linked SNF Extractor The MAX+PLUS II Compiler module that creates the linked Simulator Netlist File (**.snf**) required for multi-project simulation.

Logic Array Block (LAB) A physically grouped set of logic resources in an Altera device. The LAB consists of a logic cell array and, in some device families, an expander product term array. Any signal that is available to any one logic cell in the LAB is available to the entire LAB. In Classic devices, the logic in the LAB shares a global Clock signal. The LAB is fed by a global bus and a dedicated input bus. In MAX 5000 and MAX 7000 devices, the LAB is fed by a Programmable Interconnect Array (PIA) and a dedicated input bus; in FLEX 8000 devices, the LAB is fed by row interconnect paths and a dedicated input bus.

logic cell The generic term for a basic building block of an Altera general-purpose logic device. In Classic, MAX 5000/EPS464, and MAX 7000 devices, the logic cell is called a macrocell. In FLEX 8000 devices, the logic cell is called a logic element.

logic element (LE) A basic building block of an Altera FLEX 8000 device. A logic element (also known as a logic cell) consists of a look-up table (LUT)—i.e., a function generator that quickly computes any function of four variables—and a programmable flipflop to support sequential functions. The register can be programmed as a flow-through latch; a D, T, JK, or SR flipflop; or bypassed entirely for pure combinatorial logic. The register can feed other logic elements or feed back to the logic cell itself. Some logic cells feed output or bidirectional I/O pins on the device.

Logic elements have "numbers" of the format `LC`*<number>*_*<LAB name>*, where *<number>* ranges from 1 to 8 and *<LAB name>* consists of the row letter and column number of the Logic Array Block (LAB).

logic element timing parameters Internal timing parameters for the logic elements in FLEX 8000 devices.

Logic Programmer card The LP4, LP5, or LP6 expansion card required to run the MAX+PLUS II Programmer and program Altera devices.

Logic Synthesizer The Compiler module that uses several algorithms to minimize gate count, remove redundant logic, and utilize the device architecture as efficiently as possible.

look-up table (LUT) A function that generates outputs based on inputs and a set of stored data. The logic element of FLEX 8000 devices includes a four-input LUT that can be configured to emulate any logical function of four inputs.

M

Master Programming Unit (MPU) A logic device programming box. The MPU works with zero-insertion-force sockets and individual adapters to program and test Altera devices. The PL-MPU base unit and PLM-prefix adapters support both device programming and device testing. The PLE3-12 base unit as well as adapters with other prefixes (e.g., PLE-prefix adapters and the PLAD3-12 adapter) support device programming only.

MAX 5000/EPS464 An Altera device family based on the first generation of Multiple Array MatriX architecture. This EPROM-based device family includes EPM5016, EPM5032, EPM5064, EPS464, EPM5128, EPM5128A, EPM5130, EPM5192, and EPM5192A devices.

MAX 7000 An Altera device family based on the second generation of Multiple Array MatriX architecture. These EPROM- and EEPROM-based devices include EPM7032, EPM7032V, EPM7064, EPM7096, EPM7128, EPM7160, EPM7192, and EPM7256 devices.

MAX 7000E An Altera device family based on the enhanced second-generation Multiple Array MatriX architecture. MAX 7000E devices are function-, pin-, and programming-file-compatible with MAX 7000 devices. These EEPROM-based devices include EPM7192E and EPM7256E devices.

MAX 7000E devices differ from MAX 7000 devices in that they offer up to six pin- or logic-driven Output Enable signals, fast input setup times to logic cells, and multiple global Clocks with optional inversion.

MAX+PLUS II Altera's Multiple Array MatriX Programmable Logic User System. MAX+PLUS II is a set of computer programs and hardware support products that allow design and implementation of custom logic circuits with Classic, MAX 5000/EPS464, MAX 7000, and FLEX 8000 devices.

MPLD Mask-Programmed Logic Device, i.e., a custom Altera device created by converting a design originally created for an EPLD or FLEX 8000 device. Altera offers a program for converting customer designs into MPLDs, which are cost-effective alternatives for high-volume production.

MQFP Metal Quad Flat Pack. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

P

Passive Parallel Asynchronous (PPA) A configuration scheme in which an external controller, e.g., a CPU, loads the design data into a FLEX 8000 device via a common data bus. In this scheme, the FLEX 8000 device accepts a parallel byte of input data. Intelligent handshaking between the external controller and the FLEX 8000 device allows the external controller to configure the device. MAX+PLUS II can generate Tabular Text Files (**.ttf**) that contain the data for configuring FLEX 8000 devices in a PPA configuration scheme.

Passive Parallel Synchronous (PPS) A configuration scheme in which an external controller, e.g., a CPU, loads the design data into a FLEX 8000 device via a common data bus. Data is latched by the FLEX 8000 device on the first rising edge of a CPU-driven Clock signal. The next eight falling Clock edges serialize this latched data within the FLEX 8000 device. The FLEX 8000 latches the next 8-bit byte of data on every eighth rising edge of the Clock signal until the device is completely configured. MAX+PLUS II can generate Tabular Text Files (**.ttf**) that contain the data for configuring FLEX 8000 devices in a PPS configuration scheme.

Passive Serial (PS) A configuration scheme in which an external controller passes configuration data to a FLEX 8000 device via a bit-wide serial data stream. The FLEX 8000 device is treated as a slave device with a 5-wire interface to the external controller. The external controller can be an intelligent host, such as a microcontroller or a CPU, the BitBlaster, or the MAX+PLUS II Programmer used together with the PL-MPU Master Programming Unit, and appropriate device adapter, and the FLEX Download Cable. MAX+PLUS II can generate Tabular Text Files (**.ttf**) that contain the data for sequential or bit-slice PS configuration of multiple FLEX 8000 devices.

PDIP Plastic Dual In-Line Package. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

peripheral register A register that exists on the periphery of a FLEX 8000 device or a fast input-type logic cell that is associated with an I/O pin in a MAX 7000E device. Peripheral Register is also a logic option that specifies that you wish to implement a register in a peripheral register on a FLEX 8000 or MAX 7000E device. This logic option can be applied only to individual logic functions; it cannot be incorporated into a logic synthesis style or applied to an entire project.

PGA Pin-Grid Array. A ceramic device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

PLAD3-12 An adapter that plugs into the PL-MPU Master Programming Unit (MPU). It allows the designer to use PLE-prefix adapters originally designed for use with the PLE3-12A programming unit. It also directly supports programming of EP330 DIP devices.

PLCC Plastic J-Lead Chip Carrier. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

PQFP Plastic Quad Flat Pack. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

product term Two or more factors in a Boolean expression combined with an AND operator constitute a product term, where "product" means "logic product."

Programmer Object File (.pof) A binary file generated by the Compiler's Assembler module. It contains the data used by the MAX+PLUS II Programmer to program an Altera device.

programming file A file containing data for programming Altera devices. Both the MAX+PLUS II Compiler and Programmer can generate programming files. The following programming file formats are available in MAX+PLUS II:

Hexadecimal (Intel-Format) File (**.hex**)
JEDEC File (**.jed**)
Programmer Object File (**.pof**)
SRAM Object File (**.sof**)
Tabular Text File (**.ttf**)

POFs, SOFs, and JEDEC Files are used to program devices with the MAX+PLUS II Programmer. Hex Files and TTFs are used to configure FLEX 8000 devices by other means. JEDEC Files generated by A+PLUS software can also be used to program Classic devices. The Programmer can save data read from an examined device in POF or JEDEC File format.

project A project consists of all files that are associated with a particular design, including all subdesign files and related ancillary files created by the user or by MAX+PLUS II software. The project name is the same as the name of the top-level design file in the project. MAX+PLUS II performs compilation, simulation, timing analysis, and programming on only one project at a time.

R

RQFP Power Quad Flat Pack. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

S

Security Bit A bit that prevents an Altera device from being interrogated or inadvertently reprogrammed. It can be turned on or off for each device in a project, or for the entire project.

software guard A device that attaches to the parallel printer port on a computer. It is required to run MAX+PLUS II software.

SOIC Small-Outline Integrated Circuit. A plastic device package offered by Altera. See *Altera Device Package Outlines and Ordering Information* in the current data book for more information.

SRAM Static Read-Only Memory. A read-write memory that stores data in integrated flipflops. See *Technology & Reliability* in the current data book for more information.

SRAM Object File (.sof) A binary file, generated by MAX+PLUS II, that contains the data for configuring an Altera FLEX 8000 device.

T

Tabular Text File (.ttf) An ASCII text file in tabular format that supports the Passive Parallel Synchronous (PPS) and Passive Parallel Asynchronous (PPA) configuration schemes for configuring FLEX 8000 devices.

timing simulation A MAX+PLUS II Simulator mode that uses a timing Simulator Netlist File (**.snf**) to simulate the logical and timing performance of a project. Because the timing SNF is generated after logic synthesis, partitioning, and fitting are performed, only the nodes that have not been removed by logic optimization are simulated.

Timing SNF Extractor The Compiler module that creates the timing Simulator Netlist File (**.snf**), which contains the functional and timing data for the fully optimized project. This file is used for timing simulation and timing analysis. The Compiler's EDIF Netlist Writer module also uses timing SNFs to generate EDIF Output Files (**.edo**).

TQFP Thin Quad Flat Pack. A device package offered by Altera. See *Altera Device Package Outlines* and *Ordering Information* in the current data book for more information.

U

user I/O The total number of I/O pins and dedicated inputs on a device.

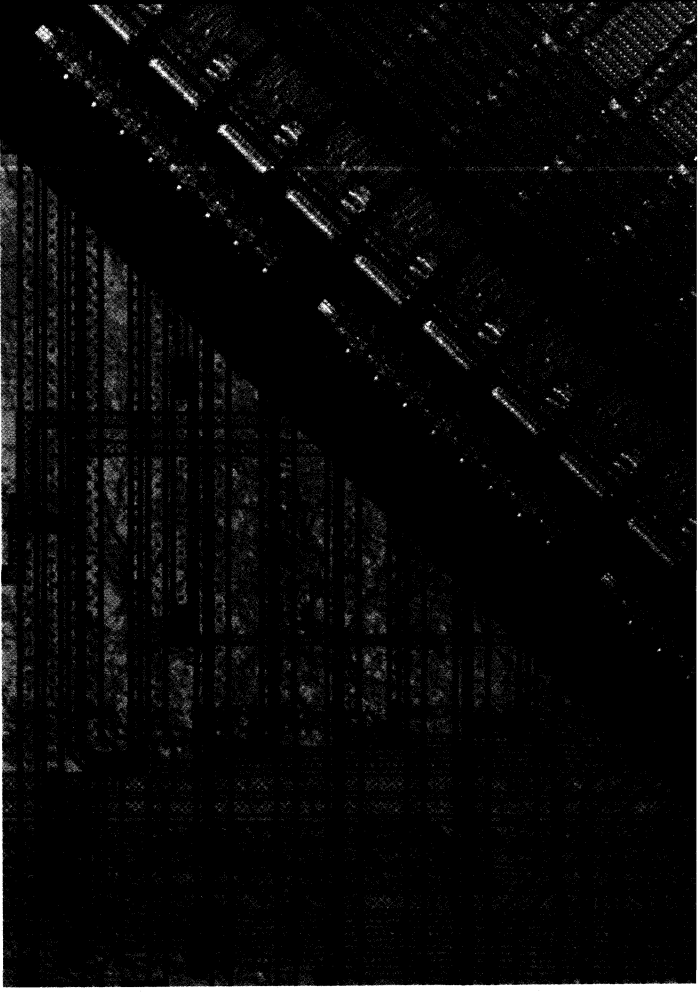
V

Verilog HDL A hardware description language from Cadence. You can generate an EDIF 2 0 0 or 2 9 0 netlist file from a Verilog design that has been processed with a Verilog synthesis tool, then import the file into MAX+PLUS II as an EDIF Input File (**.edf**). The MAX+PLUS II Compiler can also generate a Verilog Output File (**.vo**) that contains functional and timing information for simulation with a standard Verilog simulator.

VHDL Very High Speed Integrated Circuit (VHSIC) Hardware Description Language. You can create a VHDL Design File (**.vhd**) with the MAX+PLUS II Text Editor or any standard text editor and compile it directly with MAX+PLUS II. You can also generate an EDIF 2 0 0 or 2 9 0 netlist file from a VHDL design that has been processed with a VHDL synthesis tool, then import the file into MAX+PLUS II as an EDIF Input File (**.edf**). The MAX+PLUS II Compiler can also generate a VHDL Output File (**.vho**) that contains functional and timing information for simulation with a standard VHDL simulator.

VHDL Design File (.vhd) An ASCII text file written in the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). VHDL Design Files can be compiled by the MAX+PLUS II Compiler.

ALTERA



Altera Corporation
2610 Orchard Parkway
San Jose, CA 95134-2020
Telephone: (408) 894-7000

Altera Europe
Solar House
Globe Park
Fieldhouse Lane
Marlow
Bucks SL7 1TB, U.K.
Telephone: 0628 488800

Altera Japan K.K.
Ichikawa Gakugeidai Bldg.
Second Floor
12-8 Takaban 3-Chome
Meguro-Ku, Tokyo 152
Telephone: 03 (3716) 2241