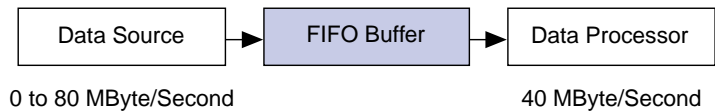


Introduction

Many applications—such as printers, microprocessors, and communications systems—receive data faster than they can process it. These systems require a buffer that can store the data until it is ready for processing. A first-in-first-out (FIFO) buffer is an ideal solution for these applications because it acts as a buffer between a fast, intermittent data source and a relatively slower processor (see [Figure 1](#)). However, the average rate of data coming into the FIFO buffer must be less than or equal to the processor rate.

Figure 1. FIFO Buffer System Block Diagram



Altera's FLEX 10K programmable logic devices (PLDs) have a high memory capacity, which makes them ideal for implementing FIFO buffers. Each FLEX 10K device contains an embedded array consisting of multiple embedded array blocks (EABs) for implementing memory and specialized functions, and a logic array for implementing general logic. Each EAB provides 2,048 bits of memory; multiple EABs can be combined to form larger blocks of memory with no performance penalty.

FIFO buffers implemented in FLEX 10K devices offer several advantages over off-the-shelf devices. You can use FLEX 10K devices to perform data processing as well as data buffering. For example, your FLEX 10K design can generate parity bits while words are written to the FIFO buffer, eliminating the need for an external parity generator. Likewise, your FLEX 10K design can perform funneling and defunneling (matching word widths) or serialization on-the-fly without decreasing performance or increasing board space requirements.

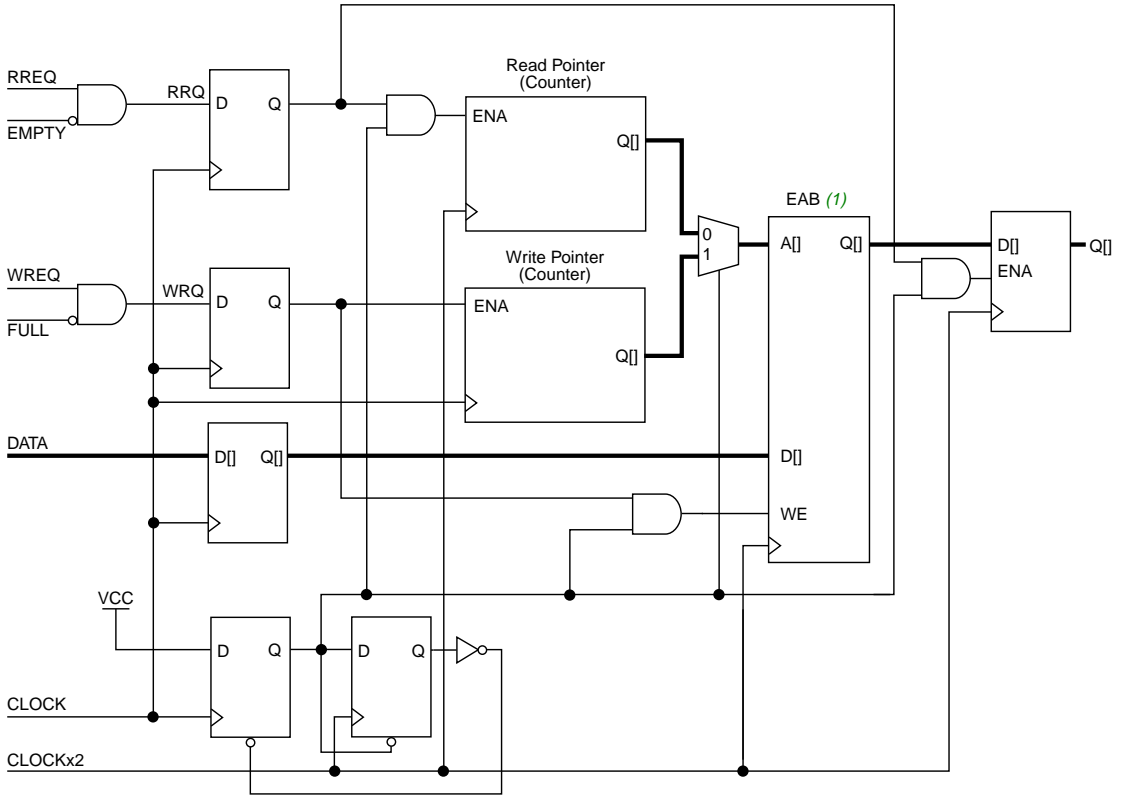
This application note describes how to implement cycle-shared and interleaved-memory FIFO buffers in FLEX 10K devices.

Cycle-Shared FIFO Buffer

In a cycle-shared FIFO buffer, one or more EABs provide the high-speed memory; all other required logic is efficiently implemented in the FLEX 10K logic array.

To implement cycle sharing, the EAB must be clocked at twice the frequency as the rest of the system, allowing both the read and write cycles to occur within one system Clock cycle. [Figure 2](#) shows a cycle-shared FIFO buffer.

Figure 2. Cycle-Shared FIFO Buffer Schematic



Note:

(1) Multiple EABs can be combined to form larger RAM blocks if desired.

The read and write pointers are implemented as counters in the logic array. When a word is written to the FIFO buffer, the write counter is incremented by one; when a word is read from the FIFO buffer, the read counter is incremented by one. A 2-to-1 multiplexer is used to multiplex the address lines from the read and write counters to the address lines of the EAB. The select input of the 2-to-1 multiplexer is continuously toggled by a flipflop synchronized by the Clock.

To request a read, the system asserts Read Request (`rreq`); to request a write, the system asserts Write Request (`wreq`). Reads are performed on the first half of the system Clock cycle, and writes are performed on the second half. This scheme allows the result of the read operation to be available to the system before the start of the next system Clock cycle.

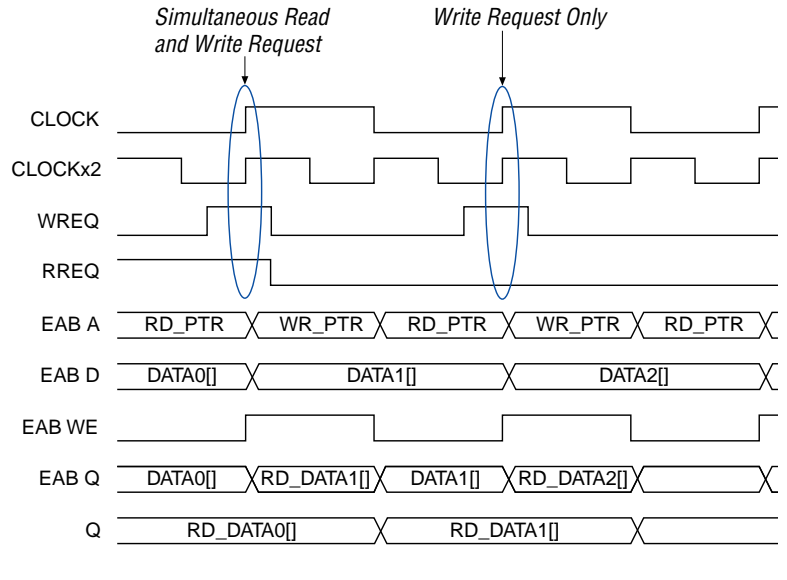
Before the rising edge of the first `clockx2` cycle, the contents of the read counter are multiplexed onto the address lines of the EAB. On the rising edge of `clockx2`, the EAB drives out the data stored at that particular address. If a read is requested, the output of the EAB is latched on the second rising edge of `clockx2`.

On the rising edge of the first `clockx2` cycle, the data to be written to the FIFO buffer is latched into registers so it will be available on the rising edge of the second `clockx2` cycle.

Before the rising edge of the second `clockx2` cycle, the contents of the write counter are multiplexed onto the address lines of the EAB. The rising edge of `clockx2` triggers the write pulse circuit within the EAB, and writes the data bus value to the EAB.

Clock skew between the two user-supplied Clocks lowers the maximum operating frequency of the FIFO buffer. For example, a Clock skew of 2 ns adds 2 ns to the minimum Clock period. To accurately simulate this behavior, the skew must be manually modeled in MAX+PLUS II by inserting a delay between the two Clock sources (a 2-ns delay in this example). MAX+PLUS II simulation does not automatically show this reduction in performance because the Clock skew is generated external to the device. [Figure 3](#) shows the functional waveforms for cycle-shared FIFO buffers in FLEX 10K devices.

Figure 3. FLEX 10K Cycle-Shared FIFO Buffer Functional Waveforms

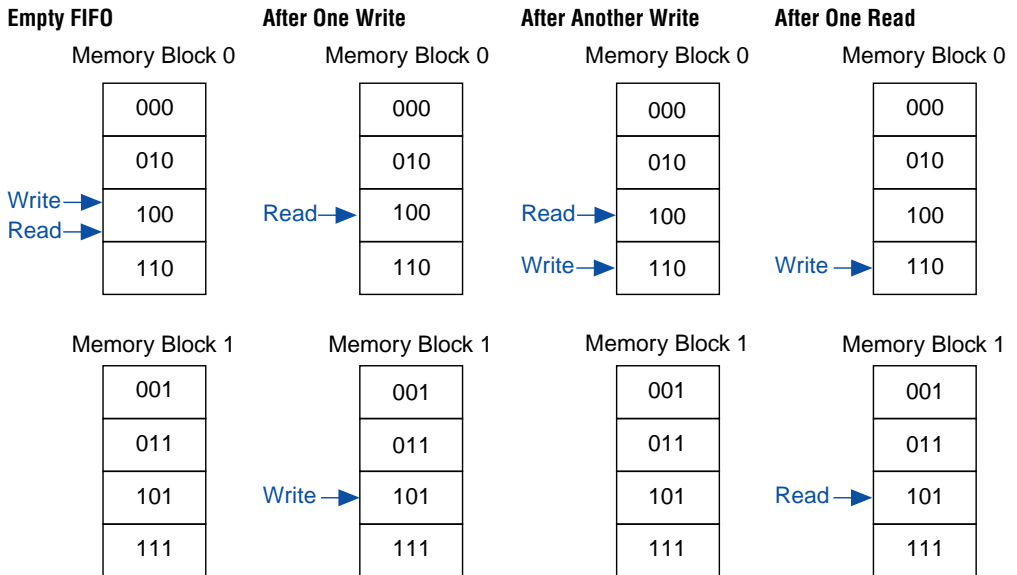


Interleaved-Memory FIFO Buffer

An interleaved-memory FIFO buffer pipelines blocks of memory, thereby increasing the operating frequency. In a FIFO buffer, every memory reference points to a consecutive memory location. The interleaved-memory FIFO buffer always stores consecutive words in different memory blocks, which reduces cycle time. Figure 4 shows how the interleaved-memory FIFO buffer accesses consecutive words from two distinct memory blocks.

Figure 4. Read/Write Memory References from 2 Blocks of Memory

In this figure, the binary numbers represent memory locations (addresses).



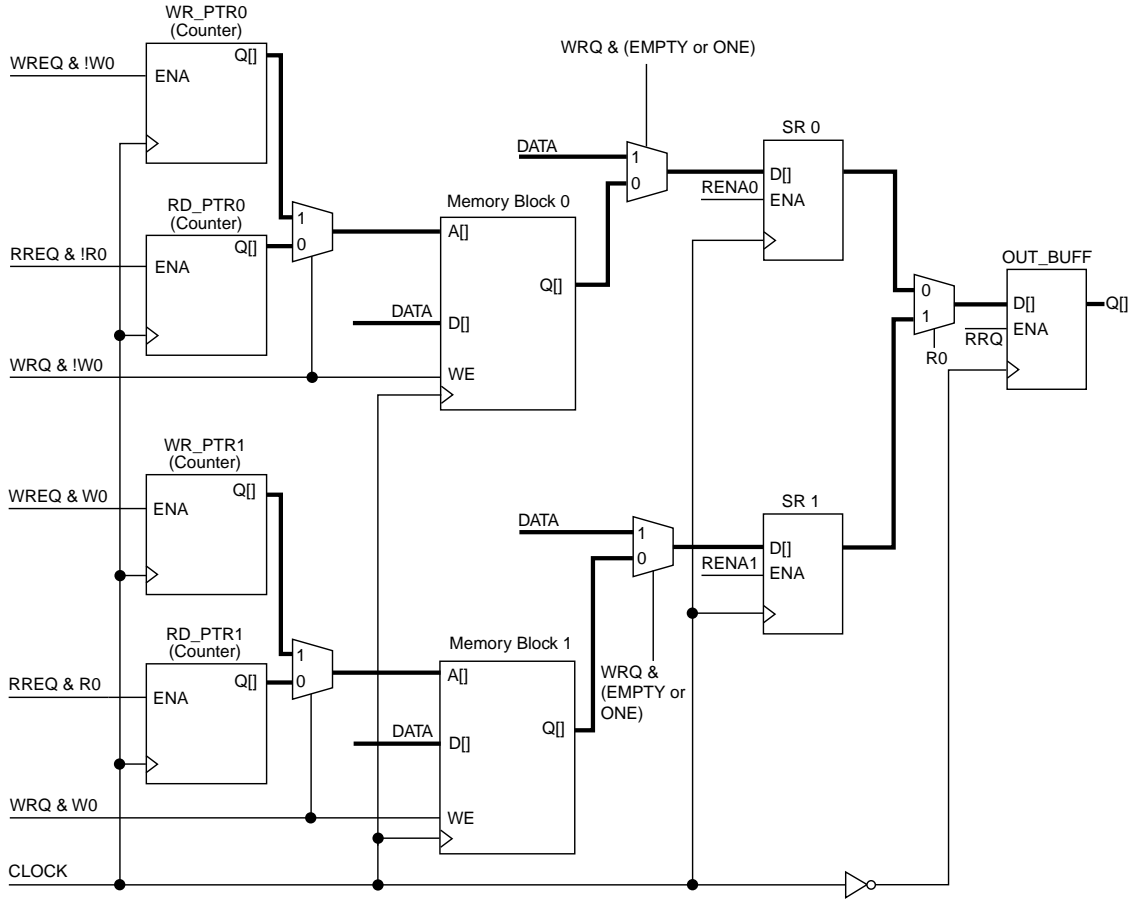
Although an interleaved-memory FIFO buffer can use any number of memory blocks, this example uses two. Using two memory blocks is ideal for simultaneous reads and writes; while a write is performed on one block, a read is performed on the other block. The staging register stores the next word to be read from each block, which prevents a conflict such as the FIFO buffer control circuitry needing to read from and write to the same memory block.

You can use the low order bits of the read and write pointers to select different blocks of memory. If only two memory blocks are used, then only the least significant bit (LSB) of the read and write pointers are needed to select each memory block. If the contents of the read or write pointer are even, Memory Block 0 is selected; if the contents of the read or write pointer are odd, Memory Block 1 is selected.

The interleaved-memory FIFO buffer in [Figure 5](#) also uses two blocks of memory. Each block of memory is implemented in one or more EABs and is accessed with dedicated read and write pointers. Because each memory block has dedicated read and write pointers, two toggle flipflops (designated by `r0` and `w0` in [Figure 5](#)) are used to emulate the LSB of the read and write pointers. The data source writes to Memory Block 0 when `w0` is low and a `wreq` has been granted; the data source writes to Memory Block 1 when `w0` is high and a `wreq` has been granted.

Figure 5. Interleaved-Memory FIFO Buffer

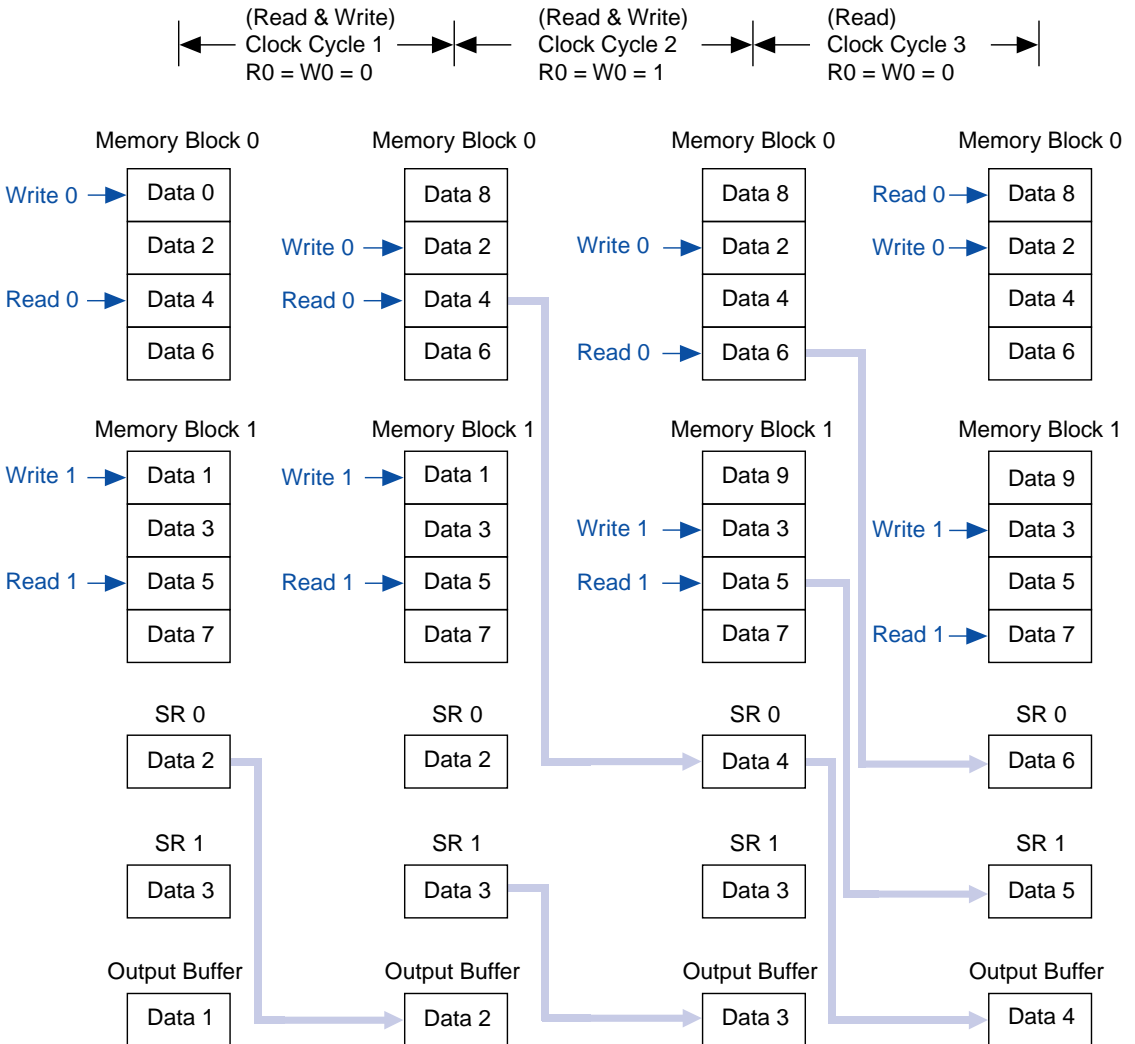
The toggle flipflops that generate R0 and W0 are not shown. RRQ and WRQ are generated as shown in Figure 2. SR 0 and SR 1 are staging registers for Memory Block 0 and Memory Block 1, respectively. RENA0 and RENA1 are the outputs of a simple state machine.



When a read operation is performed on an EAB, the data is read from the staging register for that EAB. If a write operation is being performed, the staging register is updated on the next Clock cycle. If a write operation is not being performed, the staging register is updated immediately. If an EAB is empty during a write operation, the data source simultaneously writes to the EAB and its associated staging register. This scheme ensures that the next word to be read from an EAB is always available.

Figure 6 illustrates this algorithm by showing the state of the interleaved-memory FIFO buffer during three consecutive Clock cycles. On the first Clock cycle, $r0$ and $w0$ are both low, and both read and write requests are granted. Because $r0$ and $w0$ are both low, the data source writes to Memory Block 0, and the content of SR 0 is transferred to the output buffer. On the next Clock cycle, SR 0 will be updated.

Figure 6. Example Operation of Interleaved-Memory FIFO Buffer



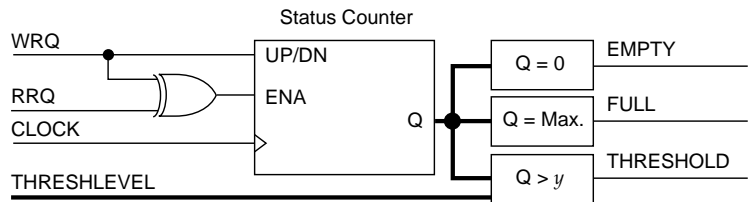
During the second Clock cycle, r_0 and w_0 are both high, and both read and write requests are granted. The data source writes to Memory Block 1 and the contents of SR 1 are transferred to the output buffer. The data processor reads from Memory Block 0, and SR 0 is updated.

During the third Clock cycle, r_0 and w_0 are both low and a read request is granted. SR 0 is transferred to the output buffer, and both SR 0 and SR 1 are updated. The r_0 flipflop is toggled so that it is high on the next Clock cycle.

FIFO Buffer Status Signals

Status signals are needed to monitor the operation of both interleaved-memory and cycle-shared FIFO buffers. The `empty`, `full`, and `threshold` output signals indicate to the controlling system when the FIFO buffer is empty, is full, or has reached a pre-determined level, respectively. These signals can be generated by using a simple up/down counter. When a write occurs, the up/down counter increments by one; when a read occurs, the up/down counter decrements by one. This scheme keeps the current size of the FIFO buffer always available to the system. [Figure 7](#) shows a schematic of the FIFO buffer status signals.

Figure 7. FIFO Buffer Status Signals



The `empty` output signal is asserted when the output of the up/down counter is zero. Read requests are not granted when the FIFO buffer is empty, which prevents the system from reading non-valid data.

The `full` output signal is asserted when the up/down counter reaches a predefined maximum value. When the FIFO buffer is full, requests to write to the FIFO buffer are not granted, which prevents the controlling system from inadvertently overwriting unread data.

FIFO Buffer Macrofunctions

The system needs to know when the FIFO buffer is almost full or almost empty. The designer determines when the system should flag this condition by choosing the value of the `threshlevel` input. When the contents of the up/down counter are greater than the `threshlevel` input ($Q > y$ in [Figure 7](#)), the `threshold` output is asserted.

The cycle-shared FIFO buffer macrofunction, `csfifo`, is currently available in MAX+PLUS II. Contact Altera Applications for availability of the interleaved-memory FIFO buffer macrofunction. Both macrofunctions use functions from the library of parameterized modules (LPM)—a set of architecture-independent modules that completely describes the logical operation of a circuit. You can easily control the size of the macrofunctions by adjusting the parameters of the LPM function. Device resources are not wasted because the size of the FIFO buffer is not fixed.

When you use the FIFO macrofunction, you must specify the value of two parameters, `lpm_width` and `lpm_numwords`, which represent the word size and depth of the FIFO buffer, respectively. [Table 1](#) lists the input and output ports for the FIFO macrofunctions.

Table 1. Ports to FIFO Macrofunctions

Port Type	Ports	Description
Input	<code>rreq</code>	Request to read from the FIFO buffer.
Input	<code>wreq</code>	Request to write to the FIFO buffer.
Input	<code>data</code>	Data to be written into the FIFO buffer.
Input	<code>threshlevel</code>	Level at which the threshold signal is asserted.
Input	<code>clock</code>	System Clock.
Input	<code>clockx2</code>	Internal Clock of the FIFO buffer (must operate at twice the frequency of the system Clock). This port is only available in <code>csfifo</code> .
Output	<code>q</code>	Data from last read of the FIFO buffer.
Output	<code>full</code>	Indicates that the FIFO buffer is full.
Output	<code>empty</code>	Indicates that the FIFO buffer is empty.
Output	<code>threshold</code>	Indicates that the FIFO buffer contains more than the <code>threshlevel</code> number of words.



2610 Orchard Parkway
San Jose, CA 95134-2020
(408) 894-7000

Applications Hotline:
(800) 800-EPLD
Customer Marketing:
(408) 894-7104
Literature Services:
(408) 894-7144

Altera, MAX, MAX+PLUS, and FLEX are registered trademarks of Altera Corporation. The following are trademarks of Altera Corporation: MAX+PLUS II and FLEX 10K. Altera acknowledges the trademarks of other organizations for their respective products or services mentioned in this document. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

U.S. and foreign patents pending.

Copyright © 1996 Altera Corporation. All rights reserved.



I.S. EN ISO 9001