

Supplement to

Logic and Computer
Design Fundamentals
4th Edition¹

VLSI PROGRAMMABLE LOGIC DEVICES: ALTERA FLEX 10K FAMILIES WITH BACKGROUND APPENDIX

This reading supplement covers specific families of VLSI programmable logic devices from Altera® Corporation. The device families chosen are those most likely to be used in beginning logic laboratories that use PLDs. This supplement is referenced at the end of Chapter 6 of the text but can be covered earlier for use in laboratories that run as part of a course or concurrently with the course. In order to permit this early coverage, a number of concepts covered in Chapters 3 through 5 are needed as background. For convenience, a brief introduction to these background concepts is provided as an appendix at the end of this supplement. If this supplement is used before completing Chapter 5, this appendix should be studied first as preparation.

Coverage of a VLSI PLD family is strongly recommended if the course has an associated laboratory component using the family. Coverage of one or both of the VLSI PLD families is recommended as a basic introduction to VLSI PLDs.

The advantage of using a PLD in the design of digital systems is that it can be programmed to incorporate complex logic functions within a single IC. But for larger or more complex functions, VLSI technology is appropriate. VLSI (Very Large Scale Integrated) refers to digital systems that contain thousands to millions of gates within a single IC chip.

In the last two decades, VLSI approaches have been developed for PLDs to handle designs that in the past were implemented by many small chips or with gate arrays having from 1,000 to millions of gates. The new approaches yield high-capacity programmable logic devices typically sharing the following properties:

¹© Pearson Education 2004. All right reserved.

1. substantial amounts of uncommitted combinational logic;
2. pre-implemented flip-flops and/or latches;
3. programmable interconnections between the combinational logic, flip-flops, and the chip input/outputs;
4. memories for storing information; and
5. a volatile or non-volatile programming technology.

ALTERA® FLEX 10K® EMBEDDED PROGRAMMABLE LOGIC DEVICES²

The Altera Flex 10K Embedded Programmable Logic Devices (EPLDs) includes two major families: 1) The basic 10K family and the enhanced 10KE family. There are many important differences in these families, including electrical characteristics and propagation delays. However, these parts with the same basic part numbers (except for the E) have similar architectures and logical structures. As a consequence, the discussion given here, while based on the basic 10K family, applies as well to most of the features of the enhanced 10KE family as well. Significant differences will be pointed out in the discussion.

Altera Flex 10K EPLDs use SRAM technology to store the programming information. After power is applied to the circuit, the program data defining the logic configuration must be loaded into the EPLD SRAM. There are a number of different ways of loading the information, a process referred to as *configuration*. Once the programming information is loaded, the EPLD switches from the programming mode to the operational mode in which the logic is available for use. The logic remains until either the EPLD is reprogrammed or the power is turned off. The ability to reprogram the EPLD allows different logic to be implemented in a system by the same EPLD at different times, leading to the concept of *reconfigurable systems*.

Values loaded into SRAM bits during configuration control the logic implemented in an Altera EPLD. Three techniques, illustrated in Figure 1 (multiplexer control, gate control, and lookup table implementation) are used to convert the stored 0's and 1's into logic. In addition, a portion of the SRAM bits reside in the embedded array blocks (EABs) and can be used as stored information that is fixed or dynamic during normal EPLD operation. An embedded array block with fixed contents is used as a ROM, which can implement complex combinational logic, and with variable contents is used as a RAM.

In Figure 1(a), an SRAM cell represented by a small circle is attached to the select input *S* of a 2-to-1 multiplexer. If the SRAM cell contains a 0, then the value on the I0 input of the multiplexer is passed to the multiplexer output. If the

² Altera and Flex 10K are registered trademarks of Altera Corporation. FastTrack is a trademark of Altera Corporation. This material in this section is intended for educational use only. For all other uses, consult the documentation available from the Altera Corporation website listed in the References.

SRAM cell contains a 1, then the value on the I1 input is passed to the multiplexer output. The structure is used to make selections between two signals. Sometimes there are two SRAM cells driving a 4-to-1 multiplexer.

In Figure 1(b), an OR gate with one input attached to an SRAM cell is shown. If the SRAM cell contains a 0, then the signal on the other input to the OR gate is pass through to the gate output. If the SRAM cell contains a 1, then the output to the OR gate is a fixed 1. This is one form of enabling circuit, as introduced in Chapter 3. An AND gate with one input attached to an SRAM cell is also used.

The final specialized use of SRAM cells is to build a lookup table, as in Figure 1(c). In the figure, a lookup table for a three-variable function $F(A, B, C)$ is illustrated (The actual lookup tables in a Flex 10K EPLD implement four-variable functions). The SRAM cells in the table store the actual truth table of the function, so each cell contains the value of function F for the corresponding minterm. The lookup table is functionally equivalent to a multiplexer with the SRAM bits applied to the data inputs and the input variables A , B , and C on the selection inputs. For example, if $(A, B, C) = 0 1 0$, the value in SRAM cell 2 (binary 010) appears on the output of the circuit. So the lookup table is actually a multiplexer implementation of combinational logic, as discussed in Chapter 3, with the SRAM cells providing the data inputs.

Architecture

The Altera® Flex 10K™ EPLD structure is shown in Figure 2. The logic within the EPLD is implemented in an array of Logic Array Blocks (LABs) and SRAMs are implemented within Embedded Array Blocks (EABs). An EAB also contains logic that permits it to be treated as a ROM, a memory composed of latches, or a memory composed of flip-flops. Inputs to and outputs from the EPLD are handled by Input/Output Elements (IOEs) along the edges of the EPLD. The LABs and IOEs are interconnected using horizontal rows of fixed wires and vertical columns

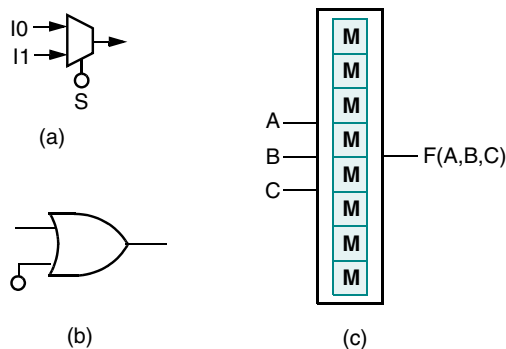


FIGURE 1
SRAM Bit Use in Altera® EPLDs

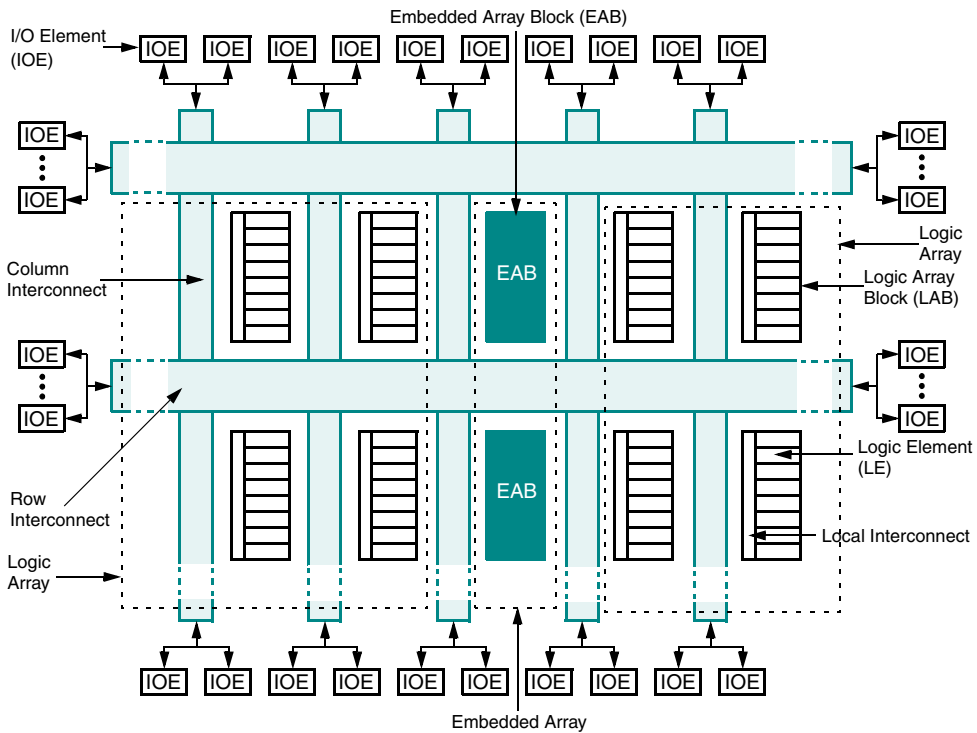


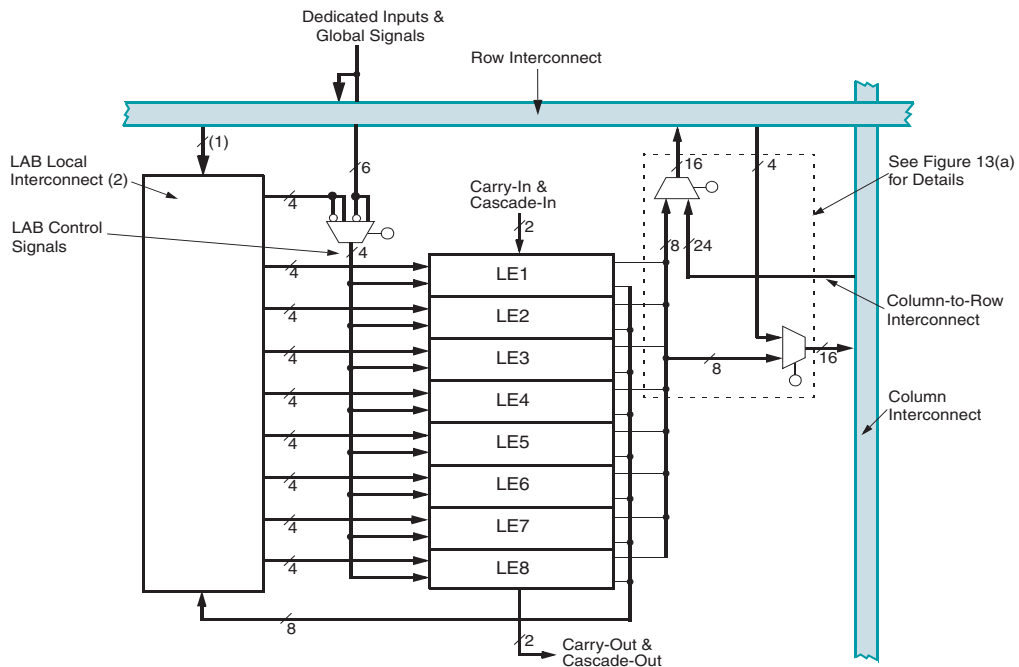
FIGURE 2
 Altera Flex 10K EPLD Structure (Reprinted with permission of Altera Corporation. © 2004 Altera Corporation)

of fixed wires. A set of wires is referred to as a *channel*. There are programmable connections between the channels and the LABs, the channels and EABs, and the channels and the IOEs. In addition, there are programmable connections between the row interconnect channels and the column interconnect channels.

Logic

The logic circuits in the Altera Flex 10K EPLD lie within the LABs and the IOEs. Both of these structures are internally programmable and fairly complex. We will look in detail at the LAB and then sketch the main features of the IOE.

LOGIC ARRAY BLOCK (LAB) A diagram of a LAB appears in Figure 3. The core of the LAB consists of eight logic elements (LEs), each of which contains an identical block of implementation logic. In order to interconnect the LEs to each other, LAB local interconnect is provided. In addition, there is a set of four signals that enter all of the LEs which are used for controlling the LE storage element. These four signals can be driven externally or from the LAB local interconnect. The outputs of the LEs can be attached to row interconnect above and the column interconnect to the right of LAB. Connections can also be made within the LAB from



Notes:

- (1) EPF10K10, EPF10K10A, EPF10K20, EPF10K30, EPF10K30A, EPF10K40, EPF10K50, and EPF10K50V devices have 22 inputs to the LAB local interconnect channel from the row; EPF10K70, EPF10K100, EPF10K100A, EPF10K130V, and EPF10K250A devices have 26.
- (2) EPF10K10, EPF10K10A, EPF10K20, EPF10K30, EPF10K30A, EPF10K40, EPF10K50, and EPF10K50V devices have 30 LAB local interconnect channels; EPF10K70, EPF10K100, EPF10K100A, EPF10K130V, and EPF10K250A devices have 34.

FIGURE 3

Diagram of an Altera Flex 10K Logic Array Block (LAB) (Reprinted with permission of Altera Corporation. © 2004 Altera Corporation)

column interconnect to row interconnect and from row interconnect to column interconnect. Finally, there are a Carry-in and a Carry-out for implementing arithmetic functions and a Cascade-in and Cascade-out for implementing a limited class of functions with large numbers of inputs.

LOGIC ELEMENT (LE) A diagram of the Flex 10K LE appears in Figure 4. A lookup table (LUT) is provided for implementing a 4-input, 1-output combinational function. The lookup table is attached to Carry Chain block which also has a Carry-in from the LE above it and a Carry-out to the LE below it. The combination of the LUT and the Carry Chain block provides two 3-input functions that implement the sum and carry functions of a full-adder. In the Cascade Chain block, the LUT output is ANDed with Cascade-in.

Each LE contains a storage element that can be configured to be either an edge-triggered D flip-flop or a level-sensitive latch. The D input is driven by a multiplexer that can be programmed to select between the output of the Cascade

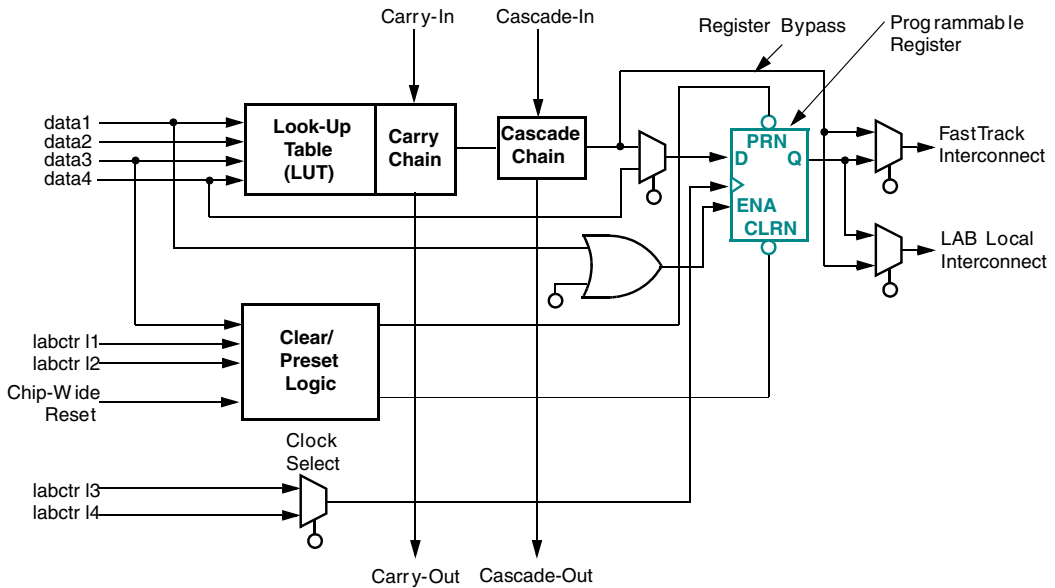


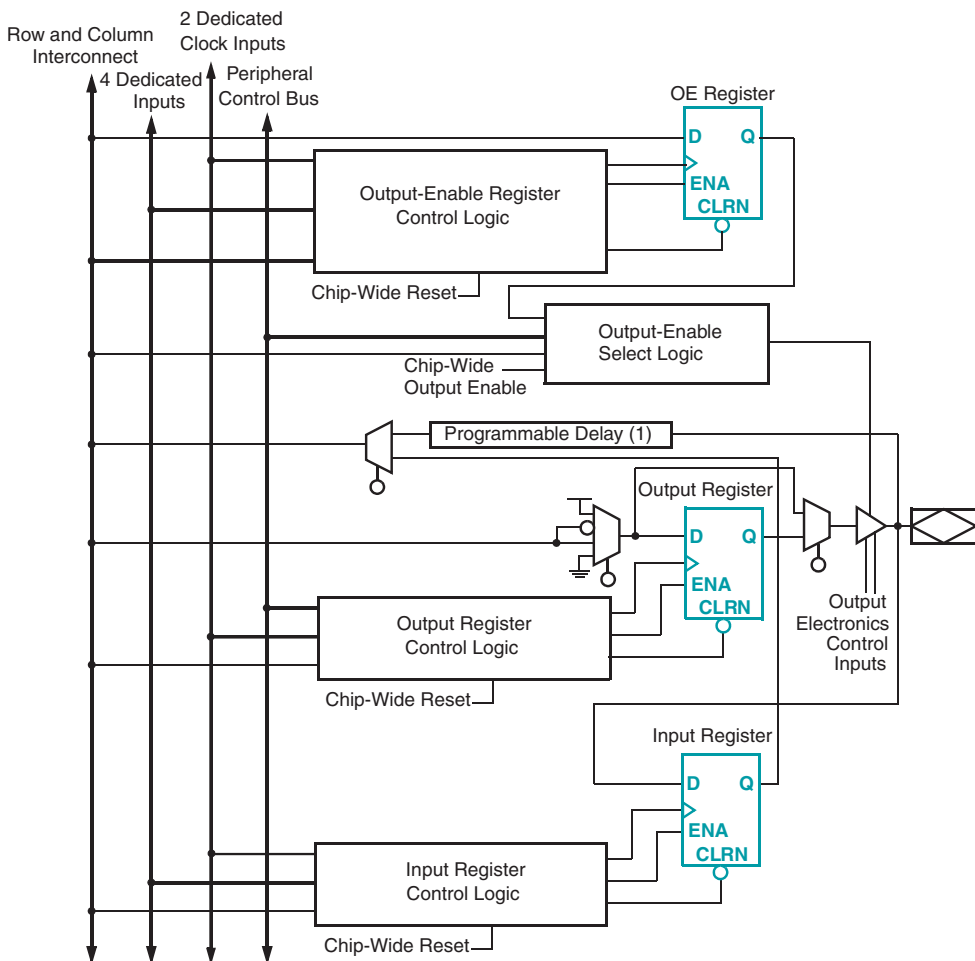
FIGURE 4
 Diagram of an Altera Flex 10K Logic Element (LE) (Reprinted with permission of Altera Corporation. © 2004 Altera Corporation)

Chain block and the LE input data, permitting the storage element to be used with or independently of the LE combinational logic. The clock for the storage element can be selected by a programmed multiplexer from two control inputs. The storage element also has Clear and Preset inputs that are control by Clear/Preset logic having a mixture of data and control inputs.

Two of the outputs from the LE attach to the row and column interconnect channels (called FastTrack™ interconnect) and to the local interconnect respectively. Two outputs are used to allow independent selection using programmed multiplexers from two sources. One source, an output of the Cascade Chain, is combinational and the other, the output of the storage element, is sequential. The other two outputs of the LE are Carry-out and Cascade-out.

INPUT/OUTPUT ELEMENT (IOE) The Flex 10K Input/Output Element (IOE) shown in Figure 5 has its I/O pin driven by a three-state output buffer and permits the signal on the pin to be used as an input. This permits the pin to be configured as an output, an input, or a bidirectional connection. There are three primary internal signals associated with I/Os: 1) the input signal from the I/O pin, 2) the output signal that goes to the 3-state buffer, and 3) the output enable signal for the 3-state buffer. Each of these signals may be optionally driven by a local storage register or may be driven by signals from elsewhere within the EPLD. The storage registers each have a clock, a load enable input, and a direct clear input.

The combinational logic within an IOE selects 1) whether or not each of these three internal signals is driven by a local storage register or by a signal from



Note:

- (1) All FLEX 10KE devices (except the EPF10K50E and EPF10K200E devices) have a programmable input delay buffer on the input path.

FIGURE 5

Simplified Diagram of an Altera Flex 10K Input/Output Element (IOE) (Adapted with permission of Altera Corporation. © 2004 Altera Corporation)

elsewhere and 2) selects the sources for the signals that control the local storage registers. A simplified version of this logic appears in Figure 5. The inputs to this logic includes data signals, control signals, and SRAM-configuration bits. Beginning at the input signal from the I/O pin, there are two paths to the row or column interconnect selected by a multiplexer controlled by an SRAM bit. One of the multiplexer inputs is the input signal itself. For some parts in the 10K family and most of the parts within the 10KE family, there is a programmable delay available in this path that is used to delay the input so that the hold time with respect to the clock is zero. The other multiplexer input is the output of the Input register. In addition,

there is control logic for selection of the input register clock, enable, and clear signals. The inputs to this logic come from row and column interconnect, four dedicated inputs, two dedicated clock lines, and a constant 1. The selection process is controlled by SRAM bits. In addition, the result of the selection for the clear is ANDed with the Chip-wide Reset to produce the clear output signal from this logic.

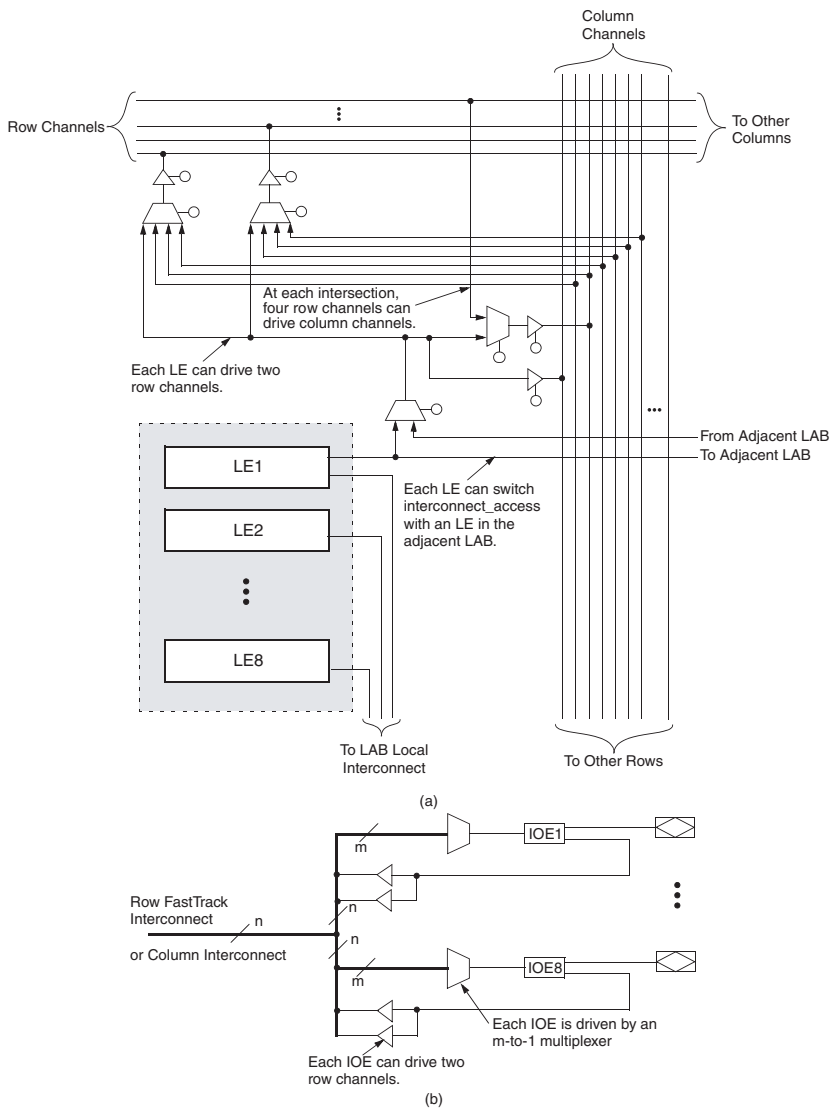
Beginning at the input to the 3-state buffer, two multiplexers controlled by SRAM bits are used to select whether or not the output signal is stored in the output register and to select one of four signals to act as the output and the input to the output register. These four signals are a constant 1, a constant 0, and true and inverted versions of a signal from the row and column interconnect. In addition, there is control logic for selection of the output register clock, enable, and clear signals. The inputs to this logic come from the row or column interconnect, the peripheral control bus, two dedicated clock lines, and a constant 1. The selection process is controlled by SRAM bits. In addition, the result of the selection for the clear is ANDed with the Chip-wide Reset to produce the clear output signal from this logic. Finally, for the output control enable input to the 3-state buffer, there is selection logic controlled by SRAM bits for selecting from the row or column interconnect, the peripheral control bus, or the output of the OE register. The input of the OE register can come only from the row or column interconnect. Also, the Chip-wide Reset is ANDed with the result of the selection for the clear to produce the clear output signal from this logic. In addition, there is control logic for selection of the output register clock, enable, and clear signals. The inputs to this logic come from the row or column interconnect, 4-dedicated inputs, two dedicated clock lines, and a constant 1. The selection process is controlled by SRAM bits.

Embedded Array Block

The Flex 10K parts contain several Embedded Array Blocks (EABs). Each EAB in the 10K parts contains $2^{11} = 2,048$ bits of storage and in the 10KE parts contains $2^{12} = 4,096$ bits of storage. The 10K EABs can be configured to have 2^m words of 2^n bits with $m + n = 11$ and $0 \leq n \leq 3$ and the 10KE parts can be configured to have 2^m words of 2^n bits with $m + n = 12$ and $1 \leq n \leq 4$. The EABs can be configured as a number of useful functional blocks for system design. The simplest is as a ROM, effectively using the EAB as a large LUT. A second configuration is as a RAM which has a single port consisting of control, address, and data inputs and one data output. A dual-port RAM configuration, which has two sets of control, address, data inputs and data outputs for reading and writing data, is available in the 10KE family. Data inputs and data outputs for a port can be merged into a single bidirectional data input/output if desired. The clock signals, read and write enable signals, and use of registers on inputs and outputs is flexible and programmable.

FastTrack™ Interconnections

The row interconnection channels are divided into two groups, full-length channels and half-length channels. A full length channel can be connected to all LABs in a



■ **FIGURE 6**

Diagrams of Altera Flex 10K Interconnect (Reprinted with permission of Altera Corporation. © 2004 Altera Corporation)

row. A half-length channel can be connected to all LABs in the left half or right half of the array. Two diagrams showing the connections associated with the row and column interconnection structure are given in Figure 6.

The programmable connections between an LE, an interconnect row and an interconnect column all lie within a LAB are shown in Figure 6(a). The LE output can be programmed to from zero to two row channels and to from zero to two column channels. In addition, for flexibility in routing signals on the channels, the

same connections for an LE can be made from the adjacent LAB. If the connections are to be made for the corresponding LE to channels, then due to the limits of the structure shown, the connection access can only be switched and not mixed between the two LABs. The connections to the row channels for the LE are shared with connections from column channels to row channels as shown. Likewise, one of the connections from the LE to the column channels is shared with a connection from a row channel. In each LAB, this overall structure appears eight times, once for each of the eight LEs. The inputs to the LAB from the row channels are shown in Figure 6. There are from 22 to 26 such signals entering the LAB Local interconnect depending on the size of the particular FLEX 10K part.

The programmable connections between an interconnect row and IOEs are shown in Figure 6(b). An IOE can drive two separate row channels as shown. Each IOE is driven by a multiplexer that selects from a subset of the row channels. The structure for connections between an interconnect column and IOEs is similar except that there are only two IOEs at each end of a column.

Design Methodology

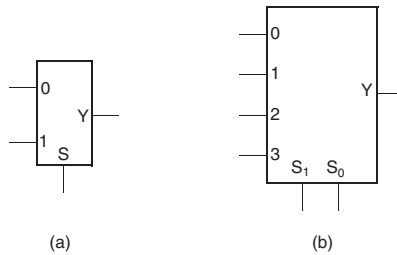
The overall structure of the interconnections, LABs, and LEs, IOEs, and EAB is complicated. A designer having to deal with 10's of LABs, hundreds of IOEs, and thousands of interconnection points in an EPLD has a very difficult job. As a consequence, CAD tools are provided that take a design in the form of a schematic or HDL description, automatically partition the design into pieces that fit within the LEs of a LAB, place the pieces into specific LEs and LABs and route the connections between the LABs. The end result of this process is thousands of bits of programming information that can be loaded into the EPLD to implement the desired logic.

APPENDIX: CONCEPTS FOR UNDERSTANDING VLSI PLDs

This appendix provides an overview of concepts provided after Chapter 3 of the text that are needed for a basic understanding of VLSI PLDs. The concepts to be overviewed include:

- 1) multiplexers, (Chapter 3),
- 2) arithmetic circuits, (Chapter 4),
- 3) latches, clocks, and flip-flops (Chapter 5), and
- 4) SRAMs (Static Random Access Memories) (Chapter 8).

These concepts are covered in detail in the chapter listed in parentheses. If you have already studied a chapter listed, then you may skip the corresponding overview on that material in this appendix. In general, the overviews give an external view of the object rather than the detailed views of internal construction given in the regular text chapters.



□ **FIGURE 7**

Multiplexer Examples: (a) a multiplexer with two information inputs; (b) a multiplexer with four information inputs

Multiplexers

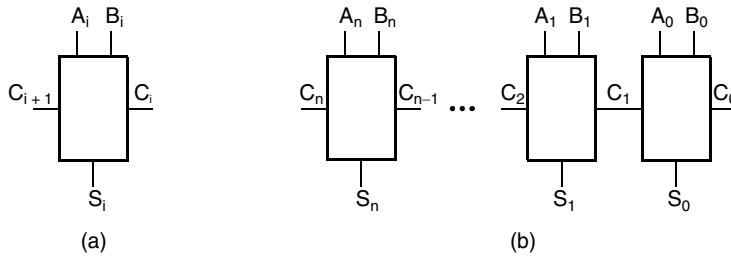
Within VLSI programmable logic, it is useful to be able to select the values on an output from among a set of different information inputs. The circuit which performs a selection operation is a *multiplexer*. In general, a multiplexer with m information inputs requires n selection signals (with $2^n \leq m < 2^{n+1}$) to select the information input to connect to its output. Two multiplexer examples are shown in Figure 7. The multiplexer in Figure 7(a) has two information inputs, requiring one selection signal S . For $S = 0$, information on input 0 is selected, and for $S = 1$, information on input 1 is selected. The multiplexer in Figure 7(b), has four information inputs, 0 through 3, requiring two selection signals S_1 and S_0 . An accompanying table shows the input, 0 through 3, selected for each of the combinations on (S_1, S_0) .

A detailed discussion of multiplexers and their implementation is given in Chapter 3.

Arithmetic Circuits

Arithmetic operations such as addition, subtraction, and counting are frequently performed within VLSI PLDs. These operations can be implemented by the programmable logic blocks provided. Unfortunately, such an approach leads to long propagation delays for a large number of operand bits (for example, 16 to 64 bits). In this section, we will discuss the origin of this delay and give an overview of the approach used to reduce it in VLSI PLDs.

To illustrate the delay problem, we consider an addition of two n -bit operands A and B to form a sum S . One way of implementing this operation is to build a hierarchical circuit made of interconnected subcircuits, using one such subcircuit for performing addition in each of the bit positions of the operands. This subcircuit, with the symbol shown in Figure 8(a), is called a full adder. Its inputs are A_i and B_i , the i th bit of the respective operands A and B . In addition, there is a carry input C_i which receives the carry output from the full adder to its right. The outputs are the



□ **FIGURE 8**

Adder circuits: (a) the symbol for a full adder; (b) an n -bit ripple carry adder

i th bit, S_i of the sum S , and carry output C_{i+1} which provides the carry input to the full adder on its left. The full adder can be implemented by one (or two) programmable logic block(s).

In Figure 8(b), n full adders have been connect carry out to carry in to form an n -bit adder referred to as a *ripple carry adder*. For simplicity in our discussion, we assume that the worst case propagation delay path through this full adder circuit runs from carry input C_0 to carry output C_n . For a propagation delay from C_i to C_{i+1} of t_{cc} , the worst case propagation delay for adding two n bit operands A and B is $n \times t_{cc}$ which grows in proportion to the number of bits n in A and B . In VLSI PLDs, the delay is even longer since the carry connections must pass from logic block to logic block through programmable interconnects which add additional delays t_i that vary depending on the physical location of each pair of connected logic blocks within the PLD. This increases the worst case adder propagation delay to:

$$n \times t_{cc} + \sum_{i=1}^{n-1} t_i$$

For large n , this delay is too high for many applications. As a consequence, this approach to implementing addition and other similar arithmetic functions is not used in VLSI PLDs. Instead, logic specifically dedicated to handling the carry function is employed. This dedicated logic accomplishes two goals. It replaces the variable time delays t_i contributed by the programmable interconnections with fixed connections with insignificant delay. Also, it replaces the programmable logic blocks with faster fixed logic in implementing most of the carry logic. By use of these techniques, this dedicated logic substantially reduces the carry delay and speeds up the arithmetic operations performed in the VLSI PLDs discussed in this supplement.

Latches, Clocks, and Flip-flops

The programmable logic blocks used in VLSI PLDs require storage elements. In addition, because of the nature of logic implementation in the technologies used, the design process becomes tractable only with the use of a clock to provide syn-

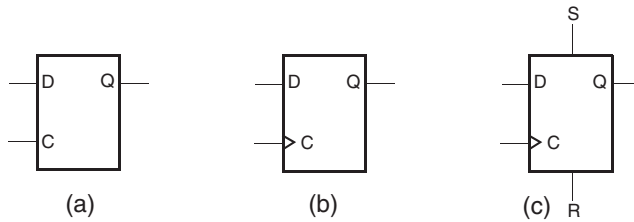


FIGURE 9
D Latch and D Flip-flop Symbols

chronous operation. In this section, we introduce the concepts of latches, clocks, and flip-flops to satisfy the storage element needs in VLSI PLDs.

THE D LATCH A latch is a basic circuit that stores a single bit of information. A symbol for a D latch is shown in Figure 9(a). The D latch has a data input D, a control input C, and an output Q. For the control input $C = 0$, the Q provides stored value stored which is fixed until C becomes 1. For the control input $C = 1$, Q follows the value appearing on D (after a propagation delay) until C become 0. From this behavior, it is apparent that control input C controls whether the value on D is being “loaded” into the latch, or the latch is storing a value that previously was “loaded” from input D. By alternating the value on C between 1 and 0, and applying a value to be stored on D during the interval when C is equal to 1, the D latch acts as a storage element.

If the alternating signal applied to C is periodic, it is referred to as a *clock*. The use of a clock provides a mechanism to control the times at which a set of D-latches is loaded. A circuit that uses a clock to synchronize its storage elements is referred to as a *synchronous circuit*. Knowing the interval of time between the loading of the storage elements in a synchronous circuit makes the design of the circuit considerably easier. As a consequence, most VLSI PLDs use a synchronous design technique.

• **EXAMPLE 1 Synchronous Operation of a D Latch**

In Figure 10, the operation of the D Latch from Figure 9 using a periodic clock Clock is shown in the waveforms which are plots of signal values along a time axis.

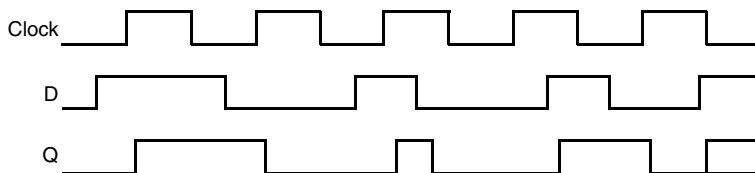
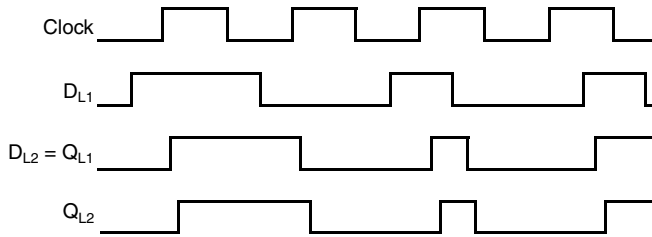


FIGURE 10
Waveforms for Example 1



□ **FIGURE 11**
Waveforms for Example 2

It is assumed that the latch initially contains 0 before the first clock pulse ($C = 1$). The following observations can be made:

1) for the first two clock pulses, D is a 1 or 0 throughout the clock pulse, and Q assumes the value on D a propagation delay after $Clock$ goes to 1 and remains there while $Clock = 0$.

2) for the second two clock pulses, D changes while $Clock = 1$, and after a propagation delay, Q takes on the value(s) on D with a propagation delay. When $Clock$ changes to 0, the value on Q remains at the last value on D before $Clock$ changes to 0.

3) for fifth clock pulse, with a change in D too close to the time at which $Clock$ changes to 0, the stored value on Q can be become either 0 or 1, arbitrarily. •

CLOCKED BEHAVIOR IN A SYNCHRONOUS CIRCUIT The behavior illustrated in case 2 in Example 1 unfortunately becomes problematic in the operation of a synchronous circuit with a single clock as illustrated in Example 2.

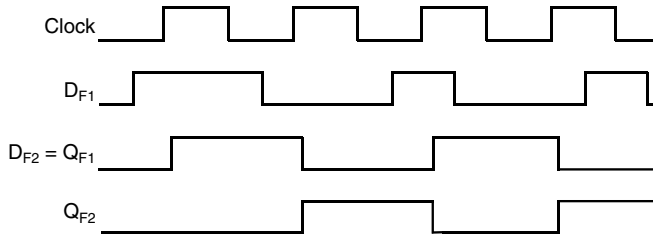
• **EXAMPLE 2 Synchronous Operation of a Circuit with Two D Latches**

In Figure 11, the operation of a circuit containing two D latches, L_1 and L_2 with the output Q of latch L_1 connected to the input D of latch L_2 is illustrated. It is assumed that both latches initially contain 0 before the first clock pulse ($Clock = 1$). The following observations can be made:

1) With a $D = 1$ applied to its input, the value 1 is loaded into and stored in latch L_1 as expected.

2) Initially, the value $D = 0$ applied to the latch L_2 , the value 0 remains in L_2 . But when the output of latch L_1 becomes 1, the value $D = 1$ is applied to latch L_2 and its stored value (after a propagation delay) changes to 1. •

The behavior in case 1 of Example 2 is correct, but what about the behavior in case 2? In order to decide whether or not this is correct, suppose that the circuit is extended to contain L_3, L_4, \dots, L_n , with n large. How far down this chain of latches does the stored value in the latches change to 1? Since the 1 propagates down the chain until $Clock$ changes to 0, and it takes a propagation delay for the



□ **FIGURE 12**
Waveforms for Example 3

value to propagate through each latch, the last of the flip-flops to change to 1 is determined by the length of time Clock is at 1 and the propagation delay of the latches. Clearly, we do not want the resulting values stored in the latches to depend on two timing values! So, by definition, in a synchronous circuit, for the application of a clock pulse, a difference between the value on D and the stored value on Q for a latch is to propagate only through that latch and not through other latches in the circuit. The only way that this can be achieved is by controlling delay values in the circuit so that a change in the output of a latch cannot reach another latch before the clock changes from 1 to 0. This kind of detailed timing control is difficult at best and, in a VLSI PLD with propagation delays associated with the interconnects, impossible. Thus, a different approach to storage is needed.

THE D FLIP-FLOP The solution in VLSI PLDs to the clocked behavior problem just described is replacement of each D latch with a D flip-flop, a more complex circuit having the symbol shown in Figure 9(b). Instead of responding, with a change in output Q dictated by an opposite value on D, to the presence of a clock pulse on C, the D flip-flop responds instead to a clock edge on C. In this case, it responds to a change from 0 to 1 on its input, called a *positive edge*. Example 3 illustrates how this solves the problem with clocked latch behavior illustrated in Example 2.

• **EXAMPLE 3 Synchronous Operation of a Circuit with Two D Flip-flops**

In Figure 12, the operation of a circuit replacing the two D latches in Example 2, D1 and D2, with two D flip-flops is illustrated. It is assumed that both flip-flops initially contain 0 before the first clock pulse (Clock = 1). The following observations can be made:

- 1) With a D = 1 applied to its input, the value 1 is loaded into and stored in flip-flop D1 as expected in response to the positive edge on C when Clock changes from 0 to 1.
- 2) The value D = 0 applied to the flip-flop D2 is loaded into D2 in response to the positive edge on C when Clock changes from 0 to 1. The change in the value of C on D2 to 1 due to the change of the output Q of flip-flop D1 is delayed until after the positive edge by the propagation delay of flip-flop D1. As a consequence,

the change on input D of D2 is not “seen” for the current clock pulse since the edge has already occurred. As a consequence, the stored value in D2 remains at 0 which is correct since the change propagates only through flip-flop D1. •

Returning to Example 1, case 3, the D value on the latch input changes too close to the change of control input C from 1 to 0 causing the value stored to be indeterminate. The same thing happens for a flip-flop triggered by an edge. As a consequence, D must become fixed at the value to be stored in a flip-flop a time interval before the positive edge called the *setup time*. It must also be held that the value to be stored for a time interval after the positive edge called the *hold time*. For many modern flip-flop designs, the hold time is often zero.

Our final coverage in the section deals with the flip-flop shown in Figure 9(c). This is a positive-edge triggered D flip-flop with two additional inputs S and R. Output changes in response to these inputs demonstrate a latch behavior and are entirely independent of the clock. Further, they have precedence over any edge-triggered changes. S acts as a signal that sets the stored value Q to 1 and R acts as a signal that resets the stored value Q to 0. Thus, regardless of the values on D and C:

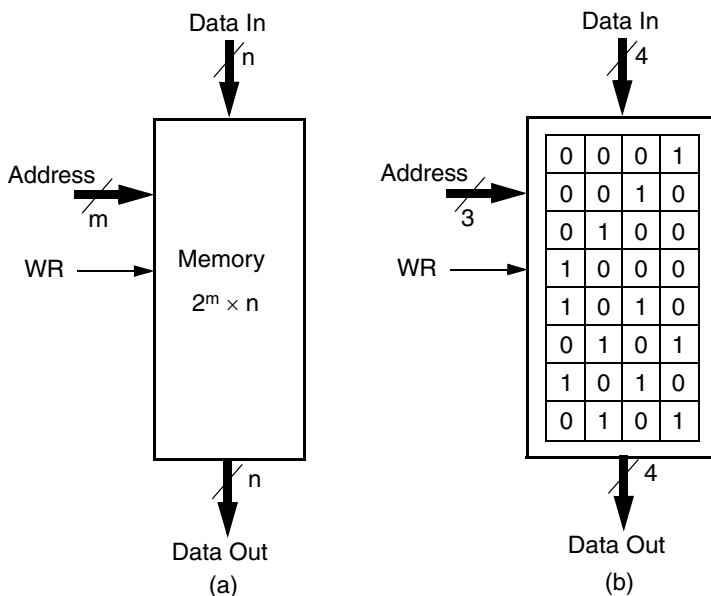
- 1) If $S = 1$ and $R = 0$, Q becomes 1, and
- 2) If $S = 0$ and $R = 1$, Q becomes 0.

If $S = R = 0$, then the value of Q is determined by the behavior of C and D, and $S = R = 1$ is a forbidden input combination.

SRAMs

Conceptually, a memory is a two-dimensional array of storage elements each of which stores the value of a single bit. Figure 13(a) represents a memory containing with n elements per row and 2^m elements per column. A specific memory with $n = 4$ and $m = 3$ is shown in Figure 13(b). Example values are shown as the contents of this memory. The memory is conceptually viewed as storing information in the rows. The contents of a row is referred to as a *word*. In order to access a word in the memory, the *address* of the word must be applied to the memory. The address of a word is the number associated with the position of the row in the memory in which the word is stored. These positions are numbered 0 through $2^m - 1$. In Figure 13(b), address 0 applies to the top word and address $2^m - 1 = 7$ applies to the bottom word.

The memory has two basic modes of operation, read and write, controlled by one or more memory control signals. Associated with these memory operations are inputs and outputs on the memory called the Address, Input, and Output ports. In addition, we assume that a memory has a single control signal, WR. If WR is equal to 1, the memory operation is a write, and if WR = 0, the memory operation is a read. For a *read* operation, with WR = 0, the word that appears on the Output of the memory is the contents of the row selected by the address value applied to the Address input. For a write operation, with WR = 1, the word applied to the Input



□ **FIGURE 13**
A Generic Memory Example

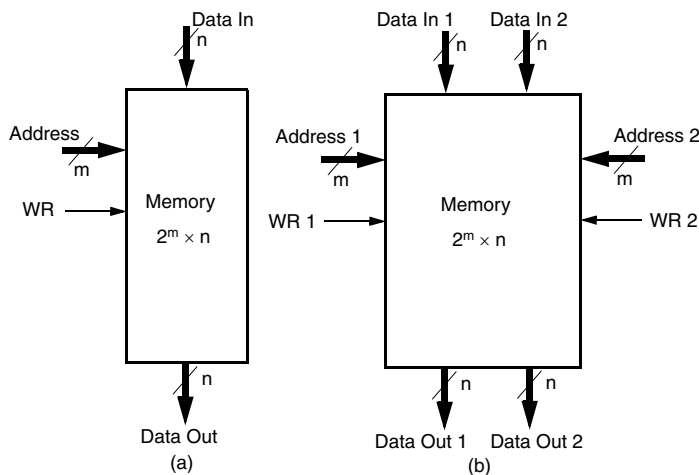
of the memory replaces the contents of the location selected by the address value applied to the Address input.

• **EXAMPLE 4 Read and Write Operations**

This example illustrates read and write operations using the memory example in Figure 13.

Suppose that a read operation on the memory with the contents shown in Figure 13(b) is to be performed from address 5 (101). The address 101 is applied to the Address input and 0 is applied to the WR input. After a time delay, called the *access time*, the value of location 5, 1010, appears on the Data Output.

Next, suppose that a write operation on the memory with the contents shown in Figure 13(b) is to be performed. This operation is to write the word 0111 into the address 2 (010). First, the address 010 is applied to the Address input and 0 is applied to the WR input. After a specified time, called the *address setup time*, WR is changed to 1. Then the word to be written, 0111, is applied to the Data Input. After both the *data setup time* from application of the word and the *minimum write pulse width time* after the change to 1 in WR, WR can be changed to 0, completing the write operation of word 0111 into address 010. Finally, after the *address hold time* has elapsed, the address can be changed, if desired. Likewise, the word applied to the Data Input can be changed after the *data hold time*. The various times specified in this example are necessary to insure that the word is written cor-



□ **FIGURE 14**
Single-Port and Dual-Port Memory Symbols

rectly into the addressed location and that the words stored in other locations are not disturbed during the write operation. •

With this introduction to memories and their operation, SRAM can now be defined. First of all, RAM is an abbreviation for random access memory. A *random access memory* is a memory with the property that the access time for a word is the same regardless of the location addressed. SRAM is an abbreviation for static RAM. This is an integrated circuit RAM that will retain its stored information as long as the power is applied. In contrast, DRAM (Dynamic RAM) will lose its stored information after a few milliseconds unless the information is (typically internally) read and restored.

The memory we illustrated in Figure 13 is a *single-port SRAM* since it has just one set of address, data, and control input and a single data output. In Figure 14, the single port RAM is contrasted with a *dual-port SRAM*. The dual port SRAM has two sets of inputs and of outputs, permitting a pair of reads, a read and a write, or a pair of writes to be performed concurrently. For a pair of writes, the write addresses must be different. Also, if a write is occurring to a given location, a concurrent read from that location may not give a correct result.

REFERENCES

1. Altera Corporation, FLEX 10K Embedded Programmable Logic Device Family Data Sheet, January 2003, version 4.2 (<http://www.altera.com/literature/ds/dsf10k.pdf>).
2. Altera Corporation, FLEX 10KE Embedded Programmable Logic Device Data Sheet, January 2003, version 2.5 (<http://www.altera.com/literature/ds/dsf10ke.pdf>).