Supplement to

Logic and Computer Design Fundamentals
3rd Edition[1]

VLSI PROGRAMMABLE LOGIC DEVICES: XILINX SPARTAN II FAMILIES WITH BACKGROUND APPENDIX

This reading supplement covers specific families of VLSI programmable logic devices from Xilinx®, Inc. 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 3 of the text to permit early coverage of the material 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 4 through 6 are needed. For convenience, a brief introduction to these concepts is provided as an appendix at the end of this supplement. If this supplement is used before completing Chapter 6, 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. 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.

Aside from these properties common to all VLSI PLDs, the devices differ from vendor to vendor. Field-Programmable Gate Arrays (FPGAs) has become a generic term referring to SRAM-based VLSI PLDs. To illustrate VLSI PLDs, we present an overview of two medium-density FPGA families that are most frequently used in basic undergraduate course laboratories.

XILINX® SPARTAN™-II FIELD-PROGRAMMABLE GATE ARRAYS[2]

Xilinx Spartan-II Field-Programmable Gate Arrays (FPGAs) include two major families: 1) The basic Spartan-II family and the enhanced Spartan-IIE 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 Spartan-II family, applies as well to most of the features of the enhanced Spartan-IIE family. Significant differences will be pointed out in the discussion.

Xilinx Field-Programmable Gate Arrays (FPGAs) 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 FPGA 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 FPGA switches from the programming mode to the operational mode in which the logic is available for use. The logic remains until either the FPGA is reprogrammed or the power is turned off. The ability to reprogram the FPGA allows different logic to be implemented in a system by the same FPGA at different times, leading to the concept of *reconfigurable systems*.

Values loaded into SRAM bits during configuration control the logic implemented in a Xilinx FPGA. Three techniques, illustrated in Figure 1 (pass transistor control, multiplexer 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 Block RAMs that are accessible to the user. Bits stored in a Block RAM during configuration permits the RAM to be used as a ROM. Otherwise, Block RAM can be can be configured to act to act as a RAM in a user's design.

---

[2] Xilinx is a registered trademark of Xilinx, Inc and Spartan is a trademark of Xilinx, Inc. This material in this section is intended for educational use only. For all other uses, consult the documentation available from the Xilinx, Inc. website listed in the References.

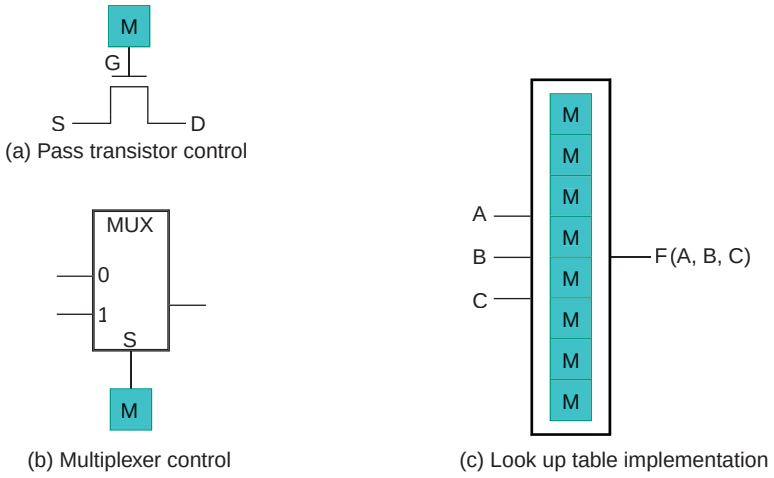


FIGURE 1
SRAM Bit Use in Xilinx® FPGAs

Figure 1(a) shows an SRAM cell driving the gate (*G*) terminal of an n-channel MOS transistor. This transistor acts like a switch. If an H (logic 1) is applied by the SRAM cell, then the path between the two other terminals of the n-channel transistor is CLOSED, permitting a current to flow between the two connected wiring segments. If an L (Logic 0) is applied, the path between the two other terminals is OPEN, preventing current flow. When such a transistor is to make a bidirectional connection for the passage of a signal between two wiring segments, it is called a *pass transistor*. A Xilinx FPGA typically contains thousands to millions of such transistors in its interconnection structure.

In Figure 1(b), an SRAM cell 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 0 input of the multiplexer is passed to the multiplexer output. If the SRAM cell contains a 1, then the value on the 1 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. Finally, in cases where the data inputs to the 0 and 1 inputs are *X* and $\overline{X}$, respectively, the multiplexer symbol is replaced by an XOR gate with *X* applied to one input and the SRAM cell output applied to the other.

The final 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 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 conceptually a multiplexer implementation of combinational logic, (as discussed in Chapter 4), with the SRAM cells providing the data inputs.

Architecture

The Xilinx Spartan-II FPGA structure is shown in Figure 2. The logic within the FPGA is implemented in an array of programmable blocks of logic called configurable logic blocks (CLBs) and Block RAMs for storing information. The Block RAMs are SRAMs that are available for use in implementing RAM and ROM in designs. Inputs to and outputs from the array of blocks are handled by input/output blocks (IOBs) along the edges of the array. The CLBs and IOBs are interconnected by a variety of programmable interconnection structures. By using an array of programmable connection blocks referred to generically as switch matrices, connections to and from CLBs and IOBs can be programmed and wire segments can be interconnected to form paths from one block to another.

On the corners of the structure are four delay-locked loops (DLLs). These are used to align the edges of internal clock signals at the flip-flop inputs with the edges of an external clock. This approach creates a synchronous circuit that includes an FPGA and its environment or multiple FPGAs and their environment. In addition, the DLLs provide multiples of the external clock with different frequencies that are useful in more complex designs.

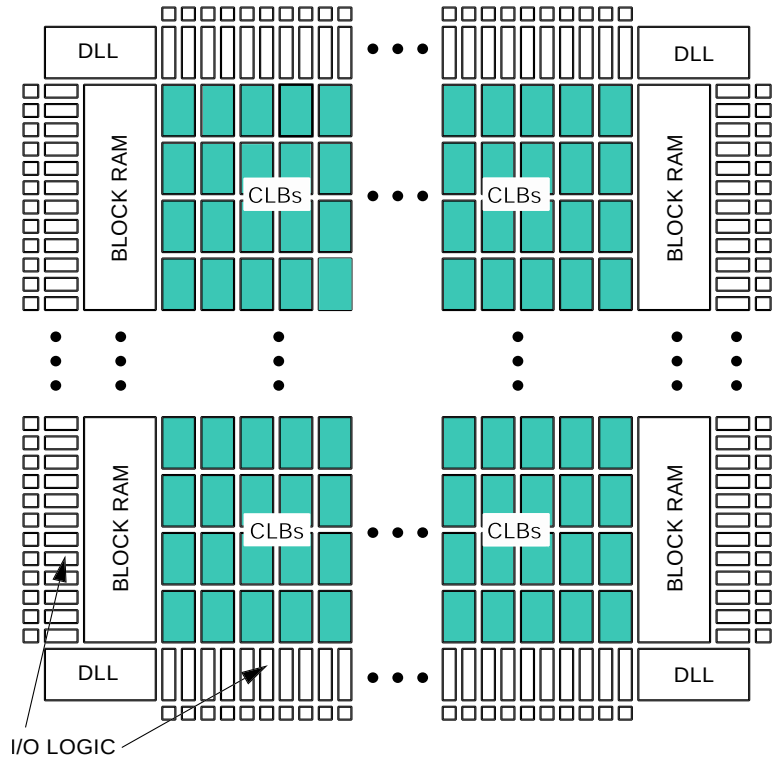


FIGURE 2
Xilinx Spartan-II FPGA Structure (Adapted with Permission of Xilinx, Inc.)

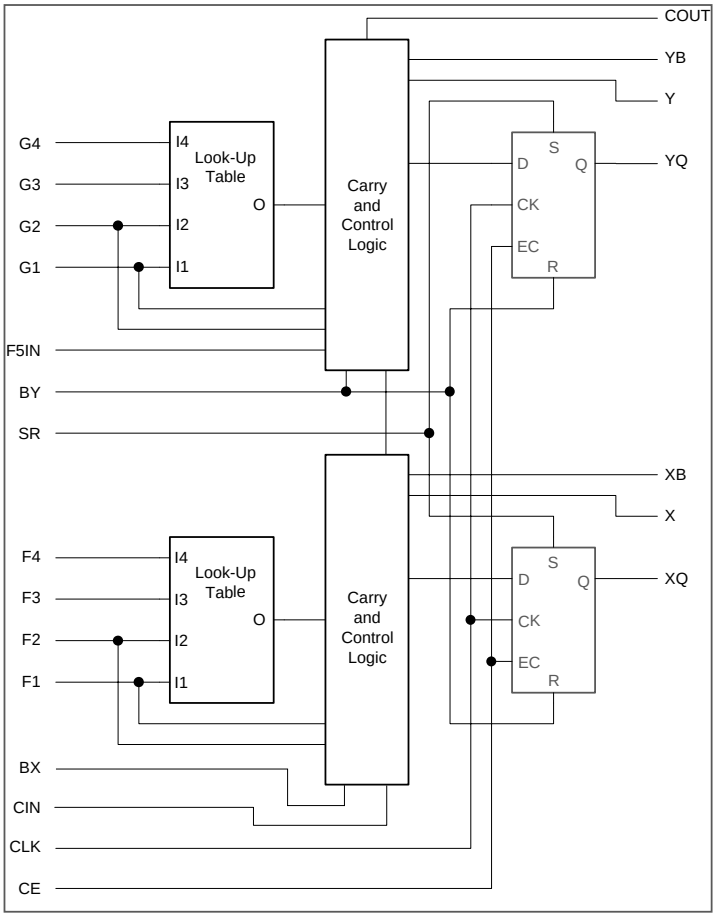


FIGURE 3
Simplified Diagram of a Xilinx Configurable Logic Block (CLB) (Adapted with Permission of Xilinx, Inc.)

Logic

Most of the logic circuits in a Xilinx FPGA lie within the CLBs and the IOBs. Both of these structures are internally programmable and fairly complex. We will look in detail at the CLB and then sketch the main features of the IOB.

CONFIGURABLE LOGIC BLOCKS (CLBS) A simplified diagram of a slice of a Xilinx Spartan-II CLB is shown in Figure 3. There are two slices in each CLB. Each slice has 15 inputs and 7 outputs. A slice contains two lookup tables (LUTs) that implement 4-input, 1-output combinational functions. One has G4, G3, G2, and G1, as inputs. The other has F4, F3, F2, and F1 as inputs. Each lookup table output enters a Carry and Control Logic block which has additional inputs. Each Carry and Con-

trol Logic block contains high-speed carry logic, an XOR gate, and an AND gate that in combination with the LUT implement one bit of arithmetic functions such as addition, counting, and multiplication. In association with this logic, each block has a carry-in and carry-out signal. The carry-in for the lower block and the carry out for the upper block are attached to other CLBs and the carry-out from the lower block is attached to the carry-in for the upper block. These carry signals and logic can also be used to cascade function generator outputs for implementation of logic functions with large numbers of inputs. Each Carry and Control Logic block also contains a number of multiplexers that select the outputs of the block from among the various inputs. Figure 4 shows three additional multiplexers that appear in each CLB. There is one F5 multiplexer in each slice. The two F5 multiplexer outputs in a CLB are the data inputs to the F6 multiplexer. By applying variables BX from each slice to the select inputs of the F5 multiplexers, two additional variables beyond those feeding the four LUTs are introduced. The output of an F5 multiplexer, can be any function of five variables or a restricted class of functions of up to nine variables. The output of the F6 multiplexer can be any function of six variables or a restricted class of functions up to 19 variables. The F5 and F6 multiplexers are included as a part of the Carry and Control Logic blocks in the two CLB slices.

Each CLB slice contains two storage elements that can be configured to be either edge-triggered D flip-flops or a level-sensitive latches. The D inputs are driven by the respective outputs of the Carry and Control Logic blocks. These outputs can be selected from a number of sources by multiplexers within the Carry and Control Logic blocks. In addition to shared Clock and Clock Enable signals, shared signals SR and BY are available for synchronous sets and resets. SR forces the storage element into the initialization state corresponding to the specified configuration and BY forces it into the opposite state if specified by the configuration.

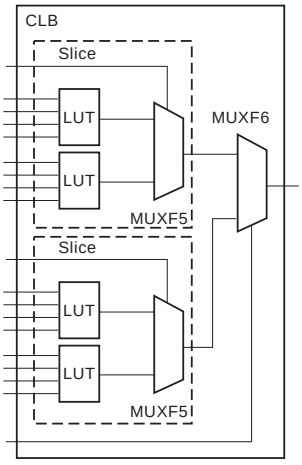


FIGURE 4
F5 and F6 Multiplexers (Adapted with Permission of Xilinx, Inc.)

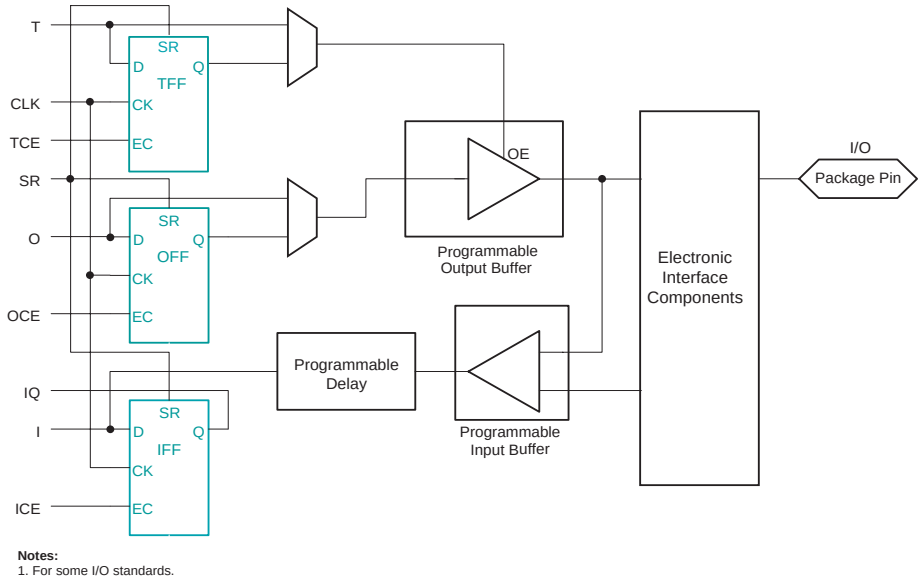


FIGURE 5
Sketch of Xilinx IOB Structure (Adapted with Permission of Xilinx, Inc.)

It is also possible to configure the storage element to provide asynchronous sets and resets using these signals. All of the storage element control inputs can be configured to be inverted or not.

There are seven outputs from a CLB slice, X, XB, XQ, Y, YB, YQ, and COUT. All of these outputs come from multiplexers in the Carry and Control Logic blocks fed by multiple sources. All of the outputs except for XQ and YQ are combinational functions of the inputs. XQ and YQ are storage element outputs. F5 is an additional slice output, used internally to feed the other CLB slice input F5IN and is not a CLB output. Instead the value of the two F5 multiplexers can be configured to appear on X and Y outputs, and the value of the F6 output can be configured to appear on the Y output.

The LUTs can alternatively be used as RAM. With the utilization of write logic, each LUT implements 16 1-bit words of SRAM. Two LUTs within a slice can be combined to give a 16 2-bit word synchronous SRAM, a 32 1-bit word synchronous SRAM, or a 16 1-bit word dual-port synchronous SRAM. Alternatively, an LUT can form a 16-bit chain of flip-flops that can be used to shift data.

INPUT/OUTPUT BLOCKS (IOB) The Xilinx IOB is also programmable and offers the designer a number of choices. We will briefly sketch its primary features. To simplify the explanation, we consider the output and input portions of the IOB separately, as shown in Figure 5. A block labeled Electronic Interface Components lies between the single I/O pin and output and input lines. This block contains many features, a number of which are programmable, that govern electrical characteris-

tics of the output and the sampling of the input value. The details of this block require some understanding of electrical and electronic circuits and are beyond the scope of our coverage.

The output portion of the IOB can provide output data O from the interior of the FPGA on the I/O pin. Alternatively, it can provide a stored value of output data O from a flip-flop with output Q. A 3-state driver on the output allows the I/O pin to be used as an input, an output, or an input/output. The control signal T for the three-state driver has the same alternative output capabilities as the output data signal O.

In the input portion of the IOB, the signal at the I/O pin enters an input buffer. The buffer output optionally passes through a programmable delay that provides a zero hold time on the input to insure that it can be properly captured. The output of the delay appears on input I to the FPGA interior and can be stored in a flip-flop with input IQ provided to the FPGA interior.

Block RAM

The Spartan II contains several Block RAMs organized in two columns along the left and right edges of the diagram in Figure 2. The number of Block RAMs per column ranges from 4 to 36 depending on the size of the FPGA. Each Block RAM contains 2^{12} = 4,096 bits of storage and can be configured to have 2^m words of 2^n bits with $m + n = 12$ and $n \leq 4$. The Block RAMs are synchronous with a clock and can be configured to be ROM or RAM and to have single or dual ports. A single port RAM has just one set of control, address, and data inputs and one data output. A dual port RAM has two sets of control, address, data inputs and data outputs for reading and writing data. In either case, the Block RAM is synchronized with one or two clocks, and cannot be used directly as a combinational component.

Interconnections

There are four major types of interconnections in the Spartan-II FPGA: 1) local routing, 2) general-purpose routing, 3) global routing, and 4) VersaRing routing. Local routing in the vicinity of a single CLB is shown in Figure 6. There are connections within the CLB that connect the internal lookup tables (LUTs) and fast direct connections to the CLBs to the left and right. Local routing connections lie in both directions between each CLB and the General Routing Matrix (GRM) to its upper left. A GRM provides connections between its CLB and the general-purpose routing segments and between general-purpose routing segments attached to the GRM. The general routing matrix is a switch matrix similar to that shown in Figure 7(a). Where four segments meet, there are six pass transistors—one vertical, one horizontal, and four on the diagonals. Each pass transistor is represented by a green line. The connection between two segments is CLOSED for a 1 stored in the SRAM cell driving the gate of the transistor. The connection between two segments is OPEN for a 0 stored in the SRAM cell. Several connections are shown in Figure 7(b). Note that at point 1 all four segments are joined together by closing

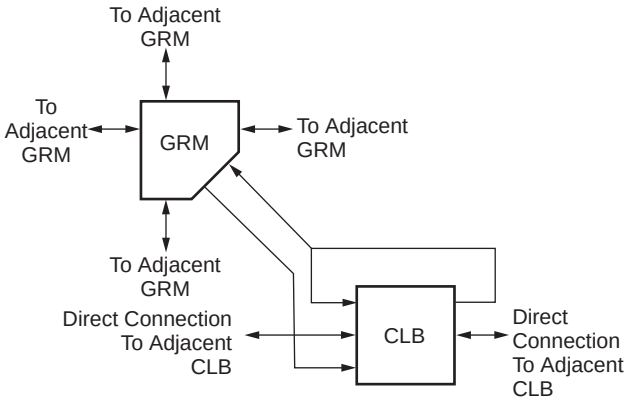
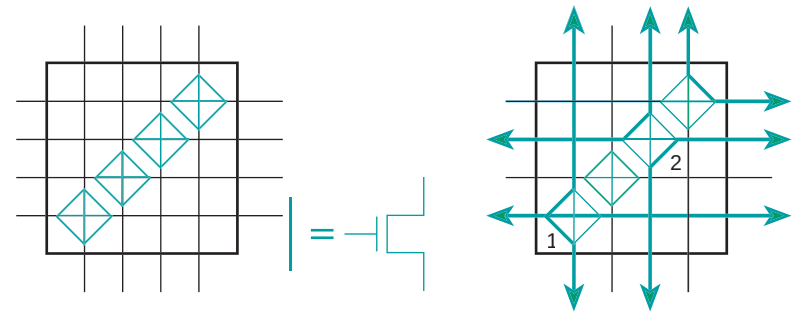


FIGURE 6
Spartan-II Local Routing (Adapted with Permission of Xilinx®, Inc.)

three transistors. In this case, all six transistors could be closed to make a connection with less electrical resistance. At point 2, two distinct signal paths pass through a single set of pass transistors. Wiring segments from the matrix inputs and outputs extend across adjacent wiring channels. SRAM-controlled pass transistors lie at selected intersections between these segments and perpendicular wiring segments in the channels. The GRM connects to wiring segments in both horizontal and vertical channels lying between the CLBs. Some of the segments are very long, spanning the entire length or width of the array. Other segments span a single CLB or six CLBs. In addition to the wiring segments connected together by GRMs, there are global routing interconnections for clocks and, from each CLB, two 3-state buffers as inputs to some of the horizontal lines.



(a) Switch Matrix Transistors (b) Examples of Connections

FIGURE 7
Example of Xilinx Switch Matrix (Adapted with Permission of Xilinx, Inc.)

Design Methodology

The overall structure of the interconnections, CLBs, and IOBs, and Block RAM is clearly quite complicated. A designer having to deal with hundreds of CLBs and IOBs and thousands of interconnection points in such an FPGA would have 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 a CLB, place the pieces into specific CLBs and route the connections between the CLBs, and to and from IOBs. The end result of this process is thousands of bits of programming information that can be loaded into the FPGA 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:

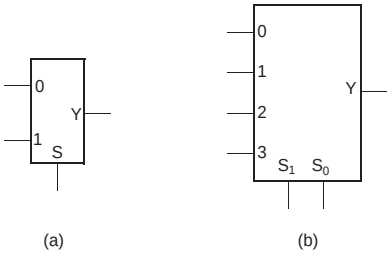
1) multiplexers, (Chapter 4),
2) arithmetic circuits, (Chapter 5),
3) latches, clocks, and flip-flops (Chapter 6), and
4) SRAMs (Static Random Access Memories) (Chapter 9).

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.

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 8. The multiplexer in Figure 8(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 8(b), has four information inputs, 0 through 3, requiring two selection signals S_1 and S_0. An accompanying table shows the input, I_0 through I_3, selected for each of the combinations on (S_1, S_0).

A detailed discussion of multiplexers and their implementation is given in Chapter 4.



S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

FIGURE 8
Multiplexer Examples: (a) a multiplexer with two information inputs; (b) a multiplexer with four information inputs

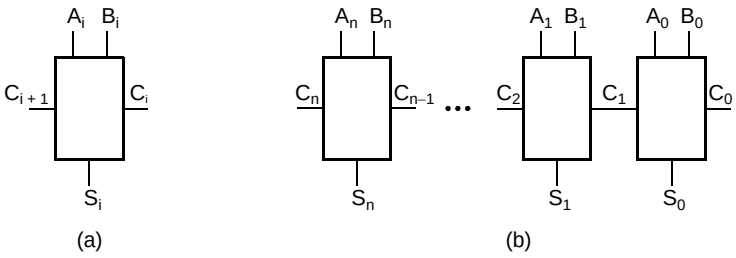


FIGURE 9
Adder circuits: (a) the symbol for a full adder; (b) an n-bit ripple carry adder

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 9(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 that receives the carry output from the full adder to its right. The outputs are the *i*th bit, S_i, of the sum S, and carry output C_{i+1} that 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).

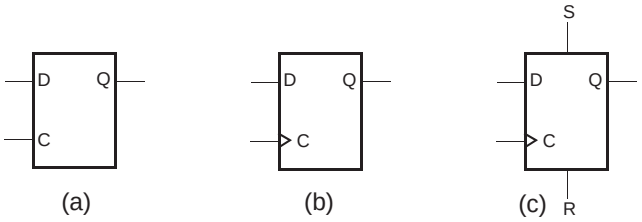


FIGURE 10
D Latch and D Flip-flop Symbols

In Figure 9(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:

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

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 synchronous 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 10(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 the 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 11, the operation of the D Latch from Figure 10 using a periodic clock Clock is shown in the waveforms which are plots of signal values along a time axis. 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.

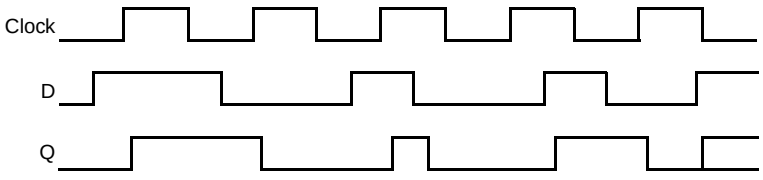


FIGURE 11
Waveforms for Example 1

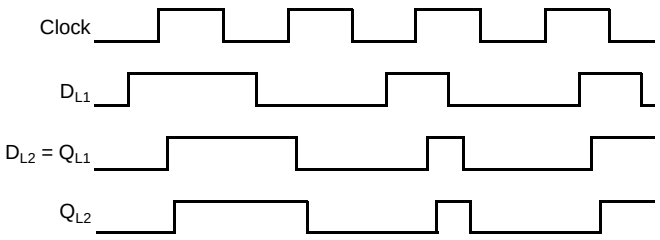


FIGURE 12
Waveforms for Example 2

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

In Figure 12, the operation of a circuit containing two D latches, L1 and L2 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, ..., 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 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 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 10 (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

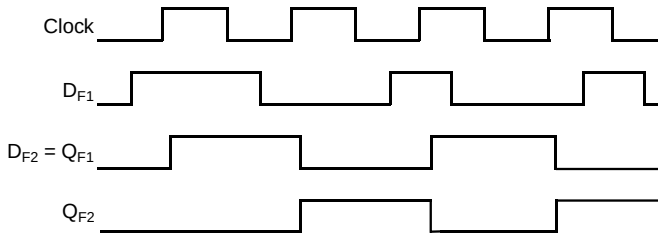


FIGURE 13
Waveforms for Example 3

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 13, 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 at 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 10(c). The is a positive-edge triggered D flip-flop with two additional input 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

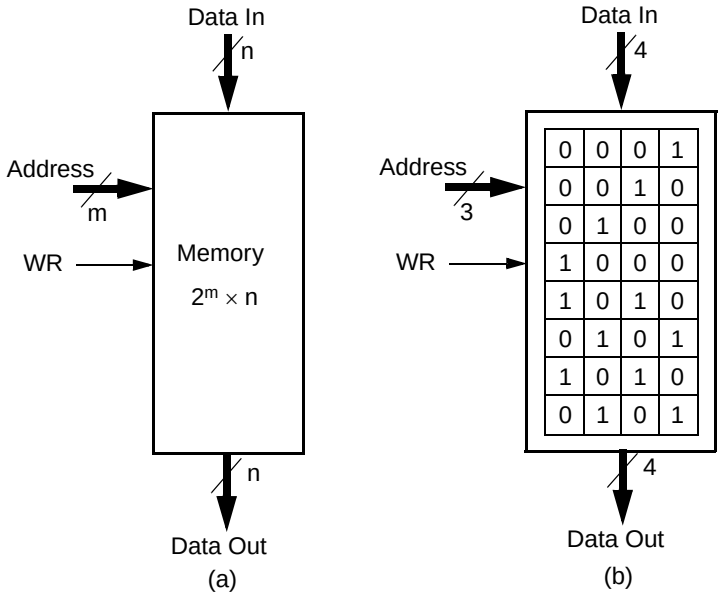


FIGURE 14
A Generic Memory Example

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 14(a) represents a memory containing n elements per row and 2^m elements per column. A specific memory with $n = 4$ and $m = 3$ is shown in Figure 14(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 14(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 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 14.

Suppose that a read operation on the memory with the contents shown in Figure 14(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 14(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 correctly 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 14 is a *single-port SRAM* since it has just one set of address, data, and control input and a single data output. In Figure 15, 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.

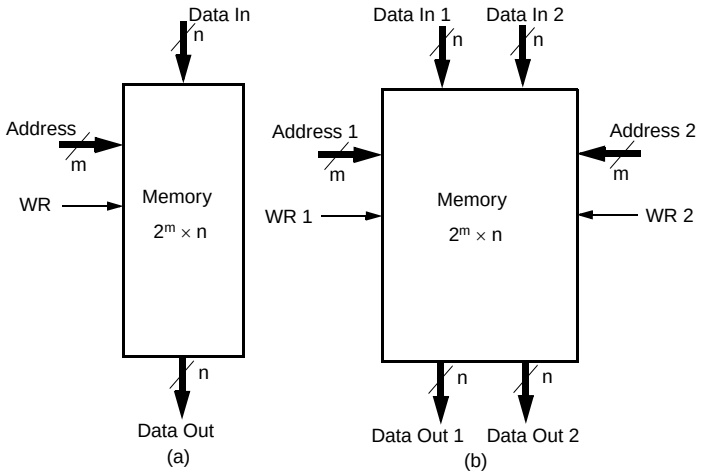


FIGURE 15
Single-Port and Dual-Port Memory Symbols

REFERENCES

1. Xilinx, Inc., Spartan-II 2.5 V FPGA Family Complete Data Sheet, DS001, September 3, 2003 (http://direct.xilinx.com/bvdocs/publications/ds001.pdf).
2. Xilinx, Inc., Spartan-IIE 1.8 V FPGA Family Complete Data Sheet, DS077, July 9, 2003 (http://direct.xilinx.com/bvdocs/publications/ds077.pdf).