PROGRAMMABLE ASIC LOGIC CELLS

All programmable ASICs or FPGAs contain a basic logic cell replicated in a regular array across the chip (analogous to a base cell in an MGA). There are the following three different types of basic logic cells: (1) multiplexer based, (2) look-up table based, and (3) programmable array logic. The choice among these depends on the programming technology. We shall see examples of each in this chapter.

5.1 Actel ACT

The basic logic cells in the Actel ACT family of FPGAs are called Logic Modules . The ACT 1 family uses just one type of Logic Module and the ACT 2 and ACT 3 FPGA families both use two different types of Logic Module.

5.1.1 ACT 1 Logic Module

The functional behavior of the Actel ACT 1 Logic Module is shown in Figure 5.1 (a). Figure 5.1 (b) represents a possible circuit-level implementation. We can build a logic function using an Actel Logic Module by connecting logic signals to some or all of the Logic Module inputs, and by connecting any remaining Logic Module inputs to VDD or GND. As an example, Figure 5.1 (c) shows the connections to implement the function $F = A \cdot B + B' \cdot C + D$. How did we know what connections to make? To understand how the Actel Logic Module works, we take a detour via multiplexer logic and some theory.

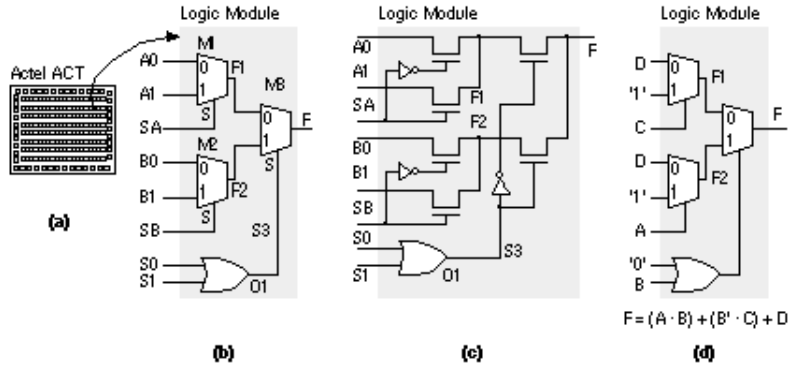


FIGURE 5.1 The Actel ACT architecture. (a) Organization of the basic logic cells. (b) The ACT 1 Logic Module. (c) An implementation using pass transistors (without any buffering). (d) An example logic macro. (Source: Actel.)

5.1.2 Shannon's Expansion Theorem

In logic design we often have to deal with functions of many variables. We need a method to break down these large functions into smaller pieces. Using the Shannon expansion theorem, we can expand a Boolean logic function F in terms of (or with respect to) a Boolean variable A,

$F = A \cdot F(A = '1') + A' \cdot F(A = '0')$,(5.1)

where F (A = 1) represents the function F evaluated with A set equal to '1'.

For example, we can expand the following function F with respect to (I shall use the abbreviation wrt) A,

$F = A' \cdot B + A \cdot B \cdot C' + A' \cdot B' \cdot C$

$= A \cdot (B \cdot C') + A' \cdot (B + B' \cdot C)$.(5.2)

We have split F into two smaller functions. We call F (A = '1') = B · C' the cofactor of F wrt A in Eq. 5.2 . I shall sometimes write the cofactor of F wrt A as F_A (the cofactor of F wrt A' is $F_{A'}$). We may expand a function wrt any of its variables. For example, if we expand F wrt B instead of A,

$F = A' \cdot B + A \cdot B \cdot C' + A' \cdot B' \cdot C$

$= B \cdot (A' + A \cdot C') + B' \cdot (A' \cdot C)$.(5.3)

We can continue to expand a function as many times as it has variables until we reach the canonical form (a unique representation for any Boolean function that uses only minterms. A minterm is a product term that contains all the variables of F-such as A · B' · C). Expanding Eq. 5.3 again, this time wrt C, gives

$F = C \cdot (A' \cdot B + A' \cdot B') + C' \cdot (A \cdot B + A' \cdot B)$.(5.4)

As another example, we will use the Shannon expansion theorem to implement the following function using the ACT 1 Logic Module:

$F = (A \cdot B) + (B' \cdot C) + D$.(5.5)

First we expand F wrt B:

$F = B \cdot (A + D) + B' \cdot (C + D)$

$= B \cdot F2 + B' \cdot F1$.(5.6)

Equation 5.6 describes a 2:1 MUX, with B selecting between two inputs: F (A = '1') and F (A = '0'). In fact Eq. 5.6 also describes the output of the ACT 1 Logic Module in Figure 5.1 ! Now we need to split up F1 and F2 in Eq. 5.6 . Suppose we expand $F2 = F_B$ wrt A, and $F1 = F_B$, wrt C:

$F2 = A + D = (A \cdot 1) + (A' \cdot D)$,(5.7)

$F1 = C + D = (C \cdot 1) + (C' \cdot D)$.(5.8)

From Eqs. 5.6 - 5.8 we see that we may implement F by arranging for A, B, C to appear on the select lines and '1' and D to be the data inputs of the MUXes in the ACT 1 Logic Module. This is the implementation shown in Figure 5.1 (d), with connections: A0 = D, A1 = '1', B0 = D, B1 = '1', SA = C, SB = A, S0 = '0', and S1 = B.

Now that we know that we can implement Boolean functions using MUXes, how do we know which functions we can implement and how to implement them?

5.1.3 Multiplexer Logic as Function Generators

Figure 5.2 illustrates the 16 different ways to arrange '1's on a Karnaugh map corresponding to the 16 logic functions, F (A, B), of two variables. Two of these functions are not very interesting (F = '0', and F = '1'). Of the 16 functions, Table 5.1 shows the 10 that we can implement using just one 2:1 MUX. Of these 10 functions, the following six are useful:

- INV. The MUX acts as an inverter for one input only.
- BUF. The MUX just passes one of the MUX inputs directly to the output.
- AND. A two-input AND.
- OR. A two-input OR.
- AND1-1. A two-input AND gate with inverted input, equivalent to an NOR-11.
- NOR1-1. A two-input NOR gate with inverted input, equivalent to an AND-11.

FIGURE 5.2 The logic functions of two variables.

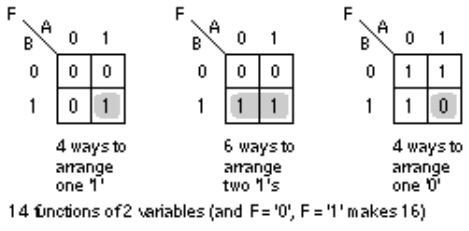


TABLE 5.1 Boolean functions using a 2:1 MUX.

Function, F		F =	Canonical form	Minterms [1]	Minterm code [2]	Function number [3]	M1 [4]		
							A0	A1	SA
1	’0’	’0’	’0’	none	0000	0	0	0	0
2	NOR1-1(A, B)	(A + B’)’	A’ · B	1	0010	2	B	0	A
3	NOT(A)	A’	A’ · B’ + A’ · B	0, 1	0011	3	0	1	A
4	AND1-1(A, B)	A · B’	A · B’	2	0100	4	A	0	B
5	NOT(B)	B’	A’ · B’ + A · B’	0, 2	0101	5	0	1	B
6	BUF(B)	B	A’ · B + A · B	1, 3	1010	6	0	B	1
7	AND(A, B)	A · B	A · B	3	1000	8	0	B	A
8	BUF(A)	A	A · B’ + A · B	2, 3	1100	9	0	A	1
9	OR(A, B)	A + B	A’ · B + A · B’ + A · B	1, 2, 3	1110	13	B	1	A
10	’1’	’1’	A’ · B’ + A’ · B + A · B’ + A · B	0, 1, 2, 3	1111	15	1	1	1

Figure 5.3 (a) shows how we might view a 2:1 MUX as a function wheel , a three-input black box that can generate any one of the six functions of two-input variables: BUF, INV, AND-11, AND1-1, OR, AND. We can write the output of a function wheel as

F1 = WHEEL1 (A, B).(5.9)

where I define the wheel function as follows:

WHEEL1 (A, B) = MUX (A0, A1, SA).(5.10)

The MUX function is not unique; we shall define it as

MUX (A0, A1, SA) = A0 · SA’ + A1 · SA.(5.11)

The inputs (A0, A1, SA) are described using the notation

A0, A1, SA = {A, B, ’0’, ’1’}(5.12)

to mean that each of the inputs (A0, A1, and SA) may be any of the values: A, B, ’0’, or ’1’. I chose the name of the wheel function because it is rather like a dial that you set to your choice of function. Figure 5.3 (b) shows that the ACT 1 Logic Module is a function generator built from two function wheels, a 2:1 MUX, and a two-input OR gate.

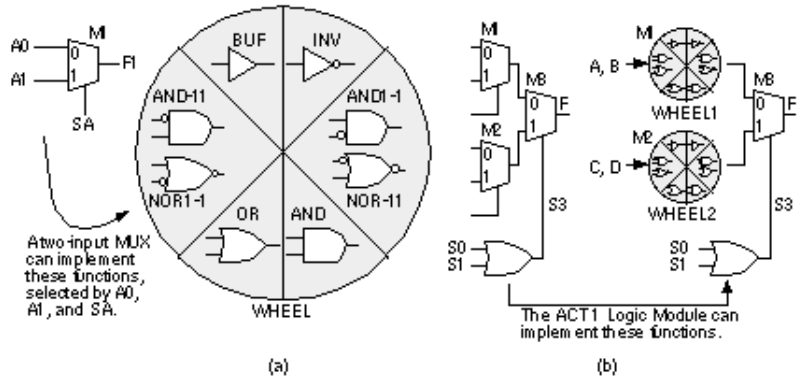


FIGURE 5.3 The ACT 1 Logic Module as a Boolean function generator. (a) A 2:1 MUX viewed as a function wheel. (b) The ACT 1 Logic Module viewed as two function wheels, an OR gate, and a 2:1 MUX.

We can describe the ACT 1 Logic Module in terms of two WHEEL functions:

F = MUX [WHEEL1, WHEEL2, OR (S0, S1)](5.13)

Now, for example, to implement a two-input NAND gate, F = NAND (A, B) = (A · B)', using an ACT 1 Logic Module we first express F as the output of a 2:1 MUX. To split up F we expand it wrt A (or wrt B; since F is symmetric in A and B):

F = A · (B') + A' · ('1')(5.14)

Thus to make a two-input NAND gate we assign WHEEL1 to implement INV (B), and WHEEL2 to implement '1'. We must also set the select input to the MUX connecting WHEEL1 and WHEEL2, S0 + S1 = A-we can do this with S0 = A, S1 = '1'.

Before we get too carried away, we need to realize that we do not have to worry about how to use Logic Modules to construct combinational logic functions-this has already been done for us. For example, if we need a two-input NAND gate, we just use a NAND gate symbol and software takes care of connecting the inputs in the right way to the Logic Module.

How did Actel design its Logic Modules? One of Actel's engineers wrote a program that calculates how many functions of two, three, and four variables a given circuit would provide. The engineers tested many different circuits and chose the best one: a small, logically efficient circuit that implemented many functions. For example, the ACT 1 Logic Module can implement all two-input functions, most functions with three inputs, and many with four inputs.

Apart from being able to implement a wide variety of combinational logic functions, the ACT 1 module can implement sequential logic cells in a flexible and efficient manner. For example, you can use one ACT 1 Logic Module for a transparent latch or two Logic Modules for a flip-flop. The use of latches rather than flip-flops does require a shift to a two-phase clocking scheme using two nonoverlapping clocks and two clock trees. Two-phase synchronous design using latches is efficient and fast but, to handle the timing complexities of two clocks requires changes to synthesis and simulation software that have not occurred. This means that most people still use flip-flops in their designs, and these require two

Logic Modules.

5.1.4 ACT 2 and ACT 3 Logic Modules

Using two ACT 1 Logic Modules for a flip-flop also requires added interconnect and associated parasitic capacitance to connect the two Logic Modules. To produce an efficient two-module flip-flop macro we could use extra antifuses in the Logic Module to cut down on the parasitic connections. However, the extra antifuses would have an adverse impact on the performance of the Logic Module in other macros. The alternative is to use a separate flip-flop module, reducing flexibility and increasing layout complexity. In the ACT 1 family Actel chose to use just one type of Logic Module. The ACT 2 and ACT 3 architectures use two different types of Logic Modules, and one of them does include the equivalent of a D flip-flop.

Figure 5.4 shows the ACT 2 and ACT 3 Logic Modules. The ACT 2 C-Module is similar to the ACT 1 Logic Module but is capable of implementing five-input logic functions. Actel calls its C-module a combinatorial module even though the module implements combinational logic. John Wakerly blames MMI for the introduction of the term combinatorial [Wakerly, 1994, p. 404].

The use of MUXes in the Actel Logic Modules (and in other places) can cause confusion in using and creating logic macros. For the Actel library, setting S = ’0’ selects input A of a two-input MUX. For other libraries setting S = ’1’ selects input A. This can lead to some very hard to find errors when moving schematics between libraries. Similar problems arise in flip-flops and latches with MUX inputs. A safer way to label the inputs of a two-input MUX is with ’0’ and ’1’, corresponding to the input selected when the select input is ’1’ or ’0’. This notation can be extended to bigger MUXes, but in Figure 5.4 , does the input combination S0 = ’1’ and S1 = ’0’ select input D10 or input D01? These problems are not caused by Actel, but by failure to use the IEEE standard symbols in this area.

The S-Module (sequential module) contains the same combinational function capability as the C-Module together with a sequential element that can be configured as a flip-flop. Figure 5.4 (d) shows the sequential element implementation in the ACT 2 and ACT 3 architectures.

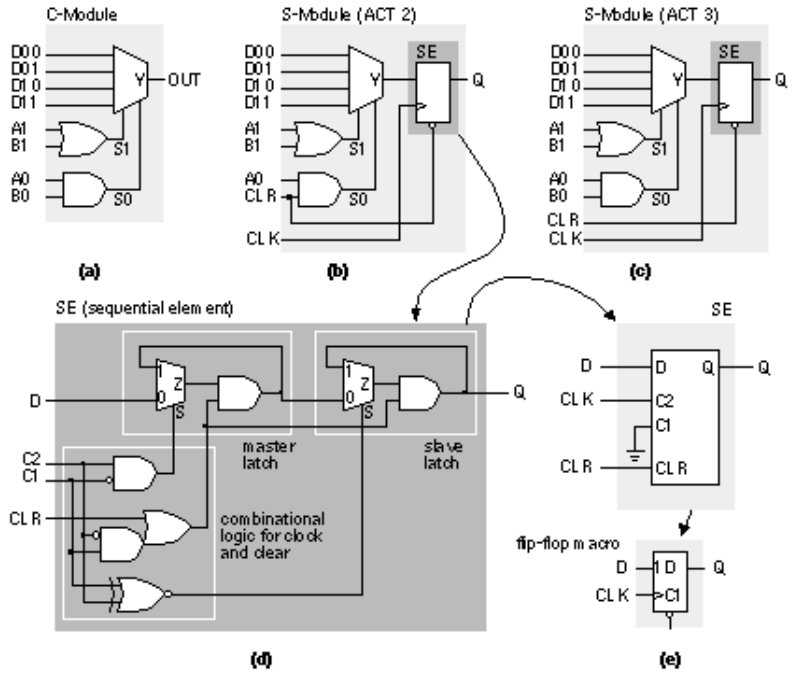


FIGURE 5.4 The Actel ACT 2 and ACT 3 Logic Modules. (a) The C-Module for combinational logic. (b) The ACT 2 S-Module. (c) The ACT 3 S-Module. (d) The equivalent circuit (without buffering) of the SE (sequential element). (e) The sequential element configured as a positive-edge-triggered D flip-flop. (Source: Actel.)

5.1.5 Timing Model and Critical Path

Figure 5.5 (a) shows the timing model for the ACT family. [5] This is a simple timing model since it deals only with logic buried inside a chip and allows us only to estimate delays. We cannot predict the exact delays on an Actel chip until we have performed the place-and-route step and know how much delay is contributed by the interconnect. Since we cannot determine the exact delay before physical layout is complete, we call the Actel architecture nondeterministic .

Even though we cannot determine the preroute delays exactly, it is still important to estimate the delay on a logic path. For example, Figure 5.5 (a) shows a typical situation deep inside an ASIC. Internal signal I1 may be from the output of a register (flip-flop). We then pass through some combinational logic, C1, through a register, S1, and then another register, S2. The register-to-register delay consists of a clock-Q delay, plus any combinational delay between registers, and the setup time for the next flip-flop. The speed of our system will depend on the slowest register-register delay or critical path between registers. We cannot make our clock period any longer than this or the signal will not reach the second register in time to be clocked.

Figure 5.5 (a) shows an internal logic signal, I1, that is an input to a C-module, C1. C1 is drawn in Figure 5.5 (a) as a box with a symbol comprising the overlapping letters "C" and "L" (borrowed from carpenters who use this symbol to mark the centerline on a piece of wood). We use this symbol to describe combinational logic. For the standard-speed grade ACT 3 (we shall look at speed grading in Section 5.1.6) the delay between the input of a C-module and the output is specified in the data book as a parameter, t , with a maximum value of 3.0 ns.

a parameter, t_{PD}, with a maximum value of 3.0 ns.

The output of C1 is an input to an S-Module, S1, configured to implement combinational logic and a D flip-flop. The Actel data book specifies the minimum setup time for this D flip-flop as t_{SUD} = 0.8 ns. This means we need to get the data to the input of S1 at least 0.8 ns before the rising clock edge (for a positive-edge-triggered flip-flop). If we do this, then there is still enough time for the data to go through the combinational logic inside S1 and reach the input of the flip-flop inside S1 in time to be clocked. We can guarantee that this will work because the combinational logic delay inside S1 is fixed.

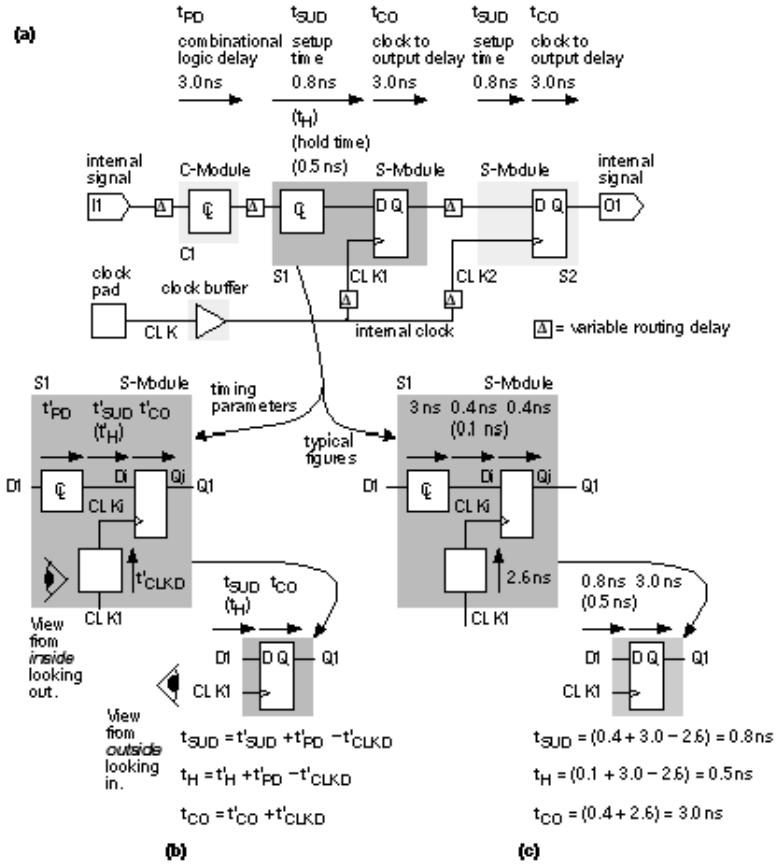


FIGURE 5.5 The Actel ACT timing model. (a) Timing parameters for a 'Std' speed grade ACT 3. (Source: Actel.) (b) Flip-flop timing. (c) An example of flip-flop timing based on ACT 3 parameters.

The S-Module seems like good value-we get all the combinational logic functions of a C-module (with delay t_{PD} of 3 ns) as well as the setup time for a flip-flop for only 0.8 ns? ...not really. Next I will explain why not.

Figure 5.5 (b) shows what is happening inside an S-Module. The setup and hold times, as measured inside (not outside) the S-Module, of the flip-flop are t'_{SUD} and t'_{H} (a prime denotes parameters that are measured inside the S-Module). The clock-Q propagation delay is t'_{CO}. The parameters t'_{SUD}, t'_{H}, and t'_{CO} are measured using the internal clock signal CLKi. The propagation delay of the combinational logic inside the S-Module is t'_{PD}. The delay of the combinational logic that drives the

flip-flop clock signal (Figure 5.4 d) is t'_{CLKD} .

From outside the S-Module, with reference to the outside clock signal CLK1:

$t_{SUD} = t'_{SUD} + (t'_{PD} - t'_{CLKD})$,

$t_{H} = t'_{H} + (t'_{PD} - t'_{CLKD})$,

$t_{CO} = t'_{CO} + t'_{CLKD}$.(5.15)

Figure 5.5 (c) shows an example of flip-flop timing. We have no way of knowing what the internal flip-flop parameters t'_{SUD} , t'_{H} , and t'_{CO} actually are, but we can assume some reasonable values (just for illustration purposes):

$t'_{SUD} = 0.4$ ns, $t'_{H} = 0.1$ ns, $t'_{CO} = 0.4$ ns.(5.16)

We do know the delay, t'_{PD} , of the combinational logic inside the S-Module. It is exactly the same as the C-Module delay, so $t'_{PD} = 3$ ns for the ACT 3. We do not know t'_{CLKD} ; we shall assume a reasonable value of $t'_{CLKD} = 2.6$ ns (the exact value does not matter in the following argument).

Next we calculate the external S-Module parameters from Eq. 5.15 as follows:

$t_{SUD} = 0.8$ ns, $t_{H} = 0.5$ ns, $t_{CO} = 3.0$ ns.(5.17)

These are the same as the ACT 3 S-Module parameters shown in Figure 5.5 (a), and I chose t'_{CLKD} and the values in Eq. 5.16 so that they would be the same. So now we see where the combinational logic delay of 3.0 ns has gone: 0.4 ns went into increasing the setup time and 2.6 ns went into increasing the clock-output delay, t_{CO} .

From the outside we can say that the combinational logic delay is buried in the flip-flop setup time. FPGA vendors will point this out as an advantage that they have. Of course, we are not getting something for nothing here. It is like borrowing money-you have to pay it back.

5.1.6 Speed Grading

Most FPGA vendors sort chips according to their speed (the sorting is known as speed grading or speed binning , because parts are automatically sorted into plastic bins by the production tester). You pay more for the faster parts. In the case of the ACT family of FPGAs, Actel measures performance with a special binning circuit , included on every chip, that consists of an input buffer driving a string of buffers or inverters followed by an output buffer. The parts are sorted from measurements on the binning circuit according to Logic Module propagation delay. The propagation delay, t_{PD} , is defined as the average of the rising (t_{PLH}) and falling (t_{PHL}) propagation delays of a Logic Module

$t = (t + t)/2$.(5.18)

$t_{PD} = (t_{PLH} + t_{PHL})/2$ (5.18)

Since the transistor properties match so well across a chip, measurements on the binning circuit closely correlate with the speed of the rest of the Logic Modules on the die. Since the speeds of die on the same wafer also match well, most of the good die on a wafer fall into the same speed bin. Actel speed grades are: a 'Std' speed grade, a '1' speed grade that is approximately 15 percent faster, a '2' speed grade that is approximately 25 percent faster than 'Std', and a '3' speed grade that is approximately 35 percent faster than 'Std'.

5.1.7 Worst-Case Timing

If you use fully synchronous design techniques you only have to worry about how slow your circuit may be-not how fast. Designers thus need to know the maximum delays they may encounter, which we call the worst-case timing . Maximum delays in CMOS logic occur when operating under minimum voltage, maximum temperature, and slow-slow process conditions. (A slow-slow process refers to a process variation, or process corner , which results in slow p -channel transistors and slow n -channel transistors-we can also have fast-fast, slow-fast, and fast-slow process corners.)

Electronic equipment has to survive in a variety of environments and ASIC manufacturers offer several classes of qualification for different applications:

- Commercial. VDD = 5 V ± 5 %, T_A (ambient) = 0 to +70 °C.
- Industrial. VDD = 5 V ± 10 %, T_A (ambient) = -40 to +85 °C.
- Military: VDD = 5 V ± 10 %, T_C (case) = -55 to +125 °C.
- Military: Standard MIL-STD-883C Class B.
- Military extended: Unmanned spacecraft.

ASICs for commercial application are cheapest; ASICs for the Cruise missile are very, very expensive. Notice that commercial and industrial application parts are specified with respect to the ambient temperature T_A (room temperature or the temperature inside the box containing the ASIC). Military specifications are relative to the package case temperature , T_C . What is really important is the temperature of the transistors on the chip, the junction temperature , T_J , which is always higher than T_A (unless we dissipate zero power). For most applications that dissipate a few hundred mW, T_J is only 5-10 °C higher than T_A . To calculate the value of T_J we need to know the power dissipated by the chip and the thermal properties of the package-we shall return to this in Section 6.6.1, "Power Dissipation."

Manufacturers have to specify their operating conditions with respect to T_J and not T_A , since they have no idea how much power purchasers will dissipate in their designs or which package they will use. Actel used to specify timing under nominal operating conditions: VDD = 5.0 V, and T_J = 25 °C. Actel and most other manufacturers now specify parameters under worst-case commercial conditions: VDD = 4.75 V, and T_J = +70 °C.

Table 5.2 shows the ACT 3 commercial worst-case timing. [6] In this table Actel has included some estimates of the variable routing delay shown in Figure 5.5 (a). These delay estimates depend on the

number of gates connected to a gate output (the fanout).

When you design microelectronic systems (or design anything) you must use worst-case figures (just as you would design a bridge for the worst-case load). To convert nominal or typical timing figures to the worst case (or best case), we use measured, or empirically derived, constants called derating factors that are expressed either as a table or a graph. For example, Table 5.3 shows the ACT 3 derating factors from commercial worst-case to industrial worst-case and military worst-case conditions (assuming $T_J = T_A$). The ACT 1 and ACT 2 derating factors are approximately the same. [7]

TABLE 5.2 ACT 3 timing parameters. [8]

		Fanout				
Family	Delay [9]	1	2	3	4	8
ACT 3-3 (data book)	t_{PD}	2.9	3.2	3.4	3.7	4.8
ACT3-2 (calculated)	t_{PD} /0.85	3.41	3.76	4.00	4.35	5.65
ACT3-1 (calculated)	t_{PD} /0.75	3.87	4.27	4.53	4.93	6.40
ACT3-Std (calculated)	t_{PD} /0.65	4.46	4.92	5.23	5.69	7.38

Source: Actel.

TABLE 5.3 ACT 3 derating factors. [10]

	Temperature T_J (junction) / °C						
V_{DD} / V	-55	-40	0	25	70	85	125
4.5	0.72	0.76	0.85	0.90	1.04	1.07	1.17
4.75	0.70	0.73	0.82	0.87	1.00	1.03	1.12
5.00	0.68	0.71	0.79	0.84	0.97	1.00	1.09
5.25	0.66	0.69	0.77	0.82	0.94	0.97	1.06
5.5	0.63	0.66	0.74	0.79	0.90	0.93	1.01

Source: Actel.

As an example of a timing calculation, suppose we have a Logic Module on a 'Std' speed grade A1415A (an ACT 3 part) that drives four other Logic Modules and we wish to estimate the delay under worst-case industrial conditions. From the data in Table 5.2 we see that the Logic Module delay for an ACT 3 'Std' part with a fanout of four is t_{PD} = 5.7 ns (commercial worst-case conditions, assuming $T_J = T_A$).

If this were the slowest path between flip-flops (very unlikely since we have only one stage of combinational logic in this path), our estimated critical path delay between registers , t_{CRIT} , would be the combinational logic delay plus the flip-flop setup time plus the clock-output delay:

$$t_{CRIT} \text{ (w-c commercial)} = t_{PD} + t_{SUD} + t_{CO}$$

= 5.7 ns + 0.8 ns + 3.0 ns = 9.5 ns .(5.19)

(I use w-c as an abbreviation for worst-case.) Next we need to adjust the timing to worst-case industrial conditions. The appropriate derating factor is 1.07 (from Table 5.3); so the estimated delay is

t_{CRIT} (w-c industrial) = 1.07 ¥ 9.5 ns = 10.2 ns .(5.20)

Let us jump ahead a little and assume that we can calculate that $T_J = T_A + 20\ °C = 105\ °C$ in our application. To find the derating factor at 105 °C we linearly interpolate between the values for 85 °C (1.07) and 125 °C (1.17) from Table 5.3). The interpolated derating factor is 1.12 and thus

t_{CRIT} (w-c industrial, $T_J = 105\ °C$) = 1.12 ¥ 9.5 ns = 10.6 ns ,(5.21)

giving us an operating frequency of just less than 100 MHz.

It may seem unfair to calculate the worst-case performance for the slowest speed grade under the harshest industrial conditions-but the examples in the data books are always for the fastest speed grades under less stringent commercial conditions. If we want to illustrate the use of derating, then the delays can only get worse than the data book values! The ultimate word on logic delays for all FPGAs is the timing analysis provided by the FPGA design tools. However, you should be able to calculate whether or not the answer that you get from such a tool is reasonable.

5.1.8 Actel Logic Module Analysis

The sizes of the ACT family Logic Modules are close to the size of the base cell of an MGA. We say that the Actel ACT FPGAs use a fine-grain architecture . An advantage of a fine-grain architecture is that, whatever the mix of combinational logic to flip-flops in your application, you can probably still use 90 percent of an Actel FPGA. Another advantage is that synthesis software has an easier time mapping logic efficiently to the simple Actel modules.

The physical symmetry of the ACT Logic Modules greatly simplifies the place-and-route step. In many cases the router can swap equivalent pins on opposite sides of the module to ease channel routing. The design of the Actel Logic Modules is a balance between efficiency of implementation and efficiency of utilization. A simple Logic Module may reduce performance in some areas-as I have pointed out-but allows the use of fast and robust place-and-route software. Fast, robust routing is an important part of Actel FPGAs (see Section 7.1, "Actel ACT").

---

1. The minterm numbers are formed from the product terms of the canonical form. For example, A · B’ = 10 = 2.

2. The minterm code is formed from the minterms. A ’1’ denotes the presence of that minterm.

3. The function number is the decimal version of the minterm code.

4. Connections to a two-input MUX: A0 and A1 are the data inputs and SA is the select input (see Eq.

5.11).

5. 1994 data book, p. 1-101.

6. ACT 3: May 1995 data sheet, p. 1-173. ACT 2: 1994 data book, p. 1-51.

7. 1994 data book, p. 1-12 (ACT 1), p. 1-52 (ACT 2), May 1995 data sheet, p. 1-174 (ACT 3).

8. V_{DD} = 4.75 V, T_J (junction) = 70 °C. Logic module plus routing delay. All propagation delays in nanoseconds.

9. The Actel '1' speed grade is 15 % faster than 'Std'; '2' is 25 % faster than 'Std'; '3' is 35 % faster than 'Std'.

10. Worst-case commercial: V_{DD} = 4.75 V, T_A (ambient) = +70 °C. Commercial: V_{DD} = 5 V ± 5 %, T_A (ambient) = 0 to +70 °C. Industrial: V_{DD} = 5 V ± 10 %, T_A (ambient) = -40 to +85 °C. Military V_{DD} = 5 V ± 10 %, T_C (case) = -55 to +125 °C.

5.2 Xilinx LCA

Xilinx LCA (a trademark, denoting logic cell array) basic logic cells, configurable logic blocks or CLBs , are bigger and more complex than the Actel or QuickLogic cells. The Xilinx LCA basic logic cell is an example of a coarse-grain architecture . The Xilinx CLBs contain both combinational logic and flip-flops.

5.2.1 XC3000 CLB

The XC3000 CLB, shown in Figure 5.6 , has five logic inputs (A-E), a common clock input (K), an asynchronous direct-reset input (RD), and an enable (EC). Using programmable MUXes connected to the SRAM programming cells, you can independently connect each of the two CLB outputs (X and Y) to the output of the flip-flops (QX and QY) or to the output of the combinational logic (F and G).

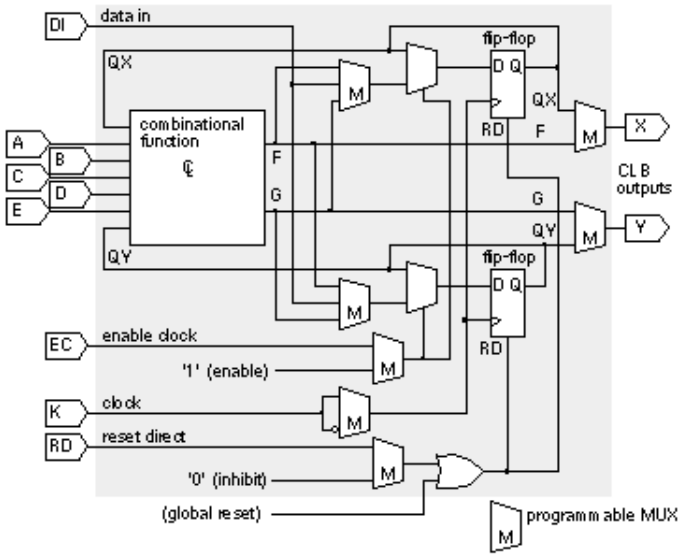


FIGURE 5.6 The Xilinx XC3000 CLB (configurable logic block). (Source: Xilinx.)

A 32-bit look-up table (LUT), stored in 32 bits of SRAM, provides the ability to implement combinational logic. Suppose you need to implement the function F = A · B · C · D · E (a five-input AND). You set the contents of LUT cell number 31 (with address ’11111’) in the 32-bit SRAM to a ’1’; all the other SRAM cells are set to ’0’. When you apply the input variables as an address to the 32-bit SRAM, only when ABCDE = ’11111’ will the output F be a ’1’. This means that the CLB propagation delay is fixed, equal to the LUT access time, and independent of the logic function you implement.

There are seven inputs for the combinational logic in the XC3000 CLB: the five CLB inputs (A-E), and the flip-flop outputs (QX and QY). There are two outputs from the LUT (F and G). Since a 32-bit LUT requires only five variables to form a unique address ($32 = 2^5$), there are several ways to use the LUT:

- You can use five of the seven possible inputs (A-E, QX, QY) with the entire 32-bit LUT. The CLB outputs (F and G) are then identical.
- You can split the 32-bit LUT in half to implement two functions of four variables each. You can choose four input variables from the seven inputs (A-E, QX, QY). You have to choose two of the inputs from the five CLB inputs (A-E); then one function output connects to F and the other output connects to G.
- You can split the 32-bit LUT in half, using one of the seven input variables as a select input to a 2:1 MUX that switches between F and G. This allows you to implement some functions of six and seven variables.

5.2.2 XC4000 Logic Block

Figure 5.7 shows the CLB used in the XC4000 series of Xilinx FPGAs. This is a fairly complicated basic logic cell containing 2 four-input LUTs that feed a three-input LUT. The XC4000 CLB also has special fast carry logic hard-wired between CLBs. MUX control logic maps four control inputs (C1-C4) into the four inputs: LUT input H1, direct in (DIN), enable clock (EC), and a set / reset control (S/R) for the flip-flops. The control inputs (C1-C4) can also be used to control the use of the F’ and G’ LUTs as

32 bits of SRAM.

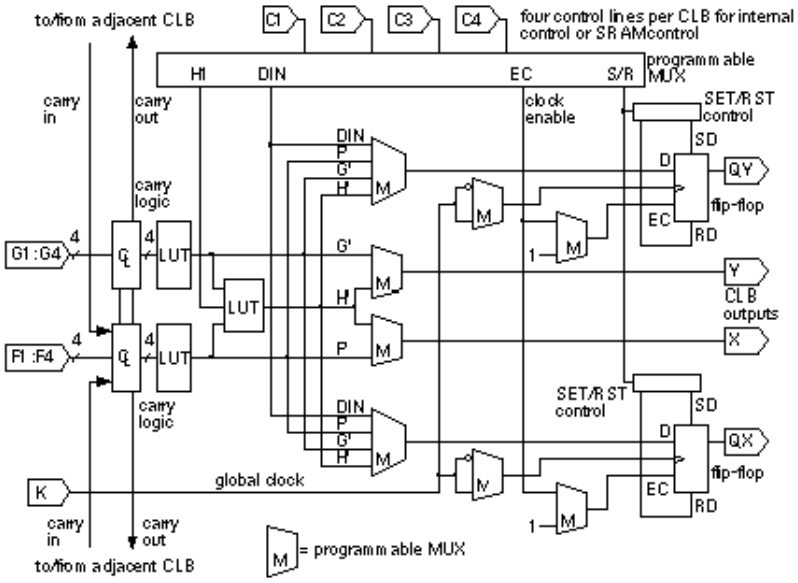


FIGURE 5.7 The Xilinx XC4000 family CLB (configurable logic block). (Source: Xilinx.)

5.2.3 XC5200 Logic Block

Figure 5.8 shows the basic logic cell, a Logic Cell or LC, used in the XC5200 family of Xilinx LCA FPGAs. [1] The LC is similar to the CLBs in the XC2000/3000/4000 CLBs, but simpler. Xilinx retained the term CLB in the XC5200 to mean a group of four LCs (LC0-LC3).

The XC5200 LC contains a four-input LUT, a flip-flop, and MUXes to handle signal switching. The arithmetic carry logic is separate from the LUTs. A limited capability to cascade functions is provided (using the MUX labeled F5_MUX in logic cells LC0 and LC2 in Figure 5.8) to gang two LCs in parallel to provide the equivalent of a five-input LUT.

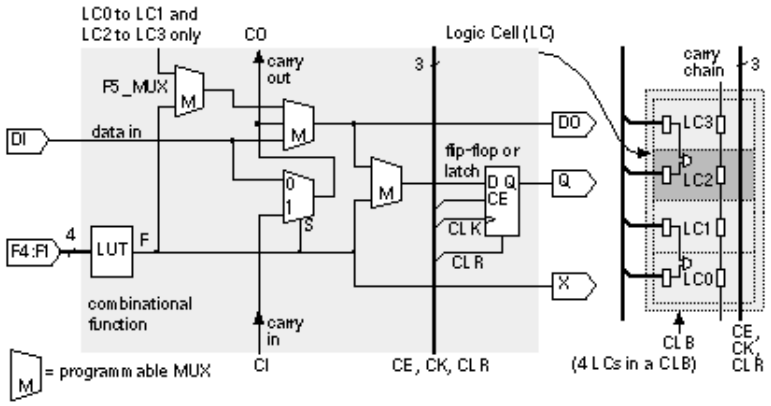


FIGURE 5.8 The Xilinx XC5200 family LC (Logic Cell) and CLB (configurable logic block). (Source: Xilinx.)

5.2.4 Xilinx CLB Analysis

The use of a LUT in a Xilinx CLB to implement combinational logic is both an advantage and a disadvantage. It means, for example, that an inverter is as slow as a five-input NAND. On the other hand a LUT simplifies timing of synchronous logic, simplifies the basic logic cell, and matches the Xilinx SRAM programming technology well. A LUT also provides the possibility, used in the XC4000, of using the LUT directly as SRAM. You can configure the XC4000 CLB as a memory-either two 16 ¥ 1 SRAMs or a 32 ¥ 1 SRAM, but this is expensive RAM.

Figure 5.9 shows the timing model for Xilinx LCA FPGAs. [2] Xilinx uses two speed-grade systems. The first uses the maximum guaranteed toggle rate of a CLB flip-flop measured in MHz as a suffix-so higher is faster. For example a Xilinx XC3020-125 has a toggle frequency of 125 MHz. The other Xilinx naming system (which supersedes the old scheme, since toggle frequency is rather meaningless) uses the approximate delay time of the combinational logic in a CLB in nanoseconds-so lower is faster in this case. Thus, for example, an XC4010-6 has t_{ILO} = 6.0 ns (the correspondence between speed grade and t_{ILO} is fairly accurate for the XC2000, XC4000, and XC5200 but is less accurate for the XC3000).

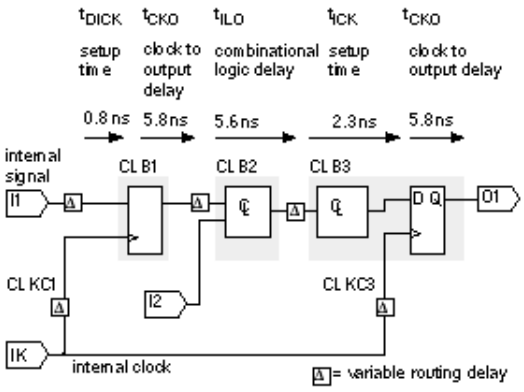


FIGURE 5.9 The Xilinx LCA timing model. The paths show different uses of CLBs (configurable logic blocks). The parameters shown are for an XC5210-6. (Source: Xilinx.)

The inclusion of flip-flops and combinational logic inside the basic logic cell leads to efficient

implementation of state machines, for example. The coarse-grain architecture of the Xilinx CLBs maximizes performance given the size of the SRAM programming technology element. As a result of the increased complexity of the basic logic cell we shall see (in Section 7.2, "Xilinx LCA") that the routing between cells is more complex than other FPGAs that use a simpler basic logic cell.

---

1. Xilinx decided to use Logic Cell as a trademark in 1995 rather as if IBM were to use Computer as a trademark today. Thus we should now only talk of a Xilinx Logic Cell (with capital letters) and not Xilinx logic cells.

2. October 1995 (Version 3.0) data sheet.

5.3 Altera FLEX

Figure 5.10 shows the basic logic cell, a Logic Element (LE), that Altera uses in its FLEX 8000 series of FPGAs. Apart from the cascade logic (which is slightly simpler in the FLEX LE) the FLEX cell resembles the XC5200 LC architecture shown in Figure 5.8 . This is not surprising since both architectures are based on the same SRAM programming technology. The FLEX LE uses a four-input LUT, a flip-flop, cascade logic, and carry logic. Eight LEs are stacked to form a Logic Array Block (the same term as used in the MAX series, but with a different meaning).

5.4 Altera MAX

Suppose we have a simple two-level logic circuit that implements a sum of products as shown in Figure 5.11 (a). We may redraw any two-level circuit using a regular structure (Figure 5.11 b): a vector of buffers, followed by a vector of AND gates (which construct the product terms) that feed OR gates (which form the sums of the product terms). We can simplify this representation still further (Figure 5.11 c), by drawing the input lines to a multiple-input AND gate as if they were one horizontal wire, which we call a product-term line . A structure such as Figure 5.11 (c) is called programmable array logic , first introduced by Monolithic Memories as the PAL series of devices.

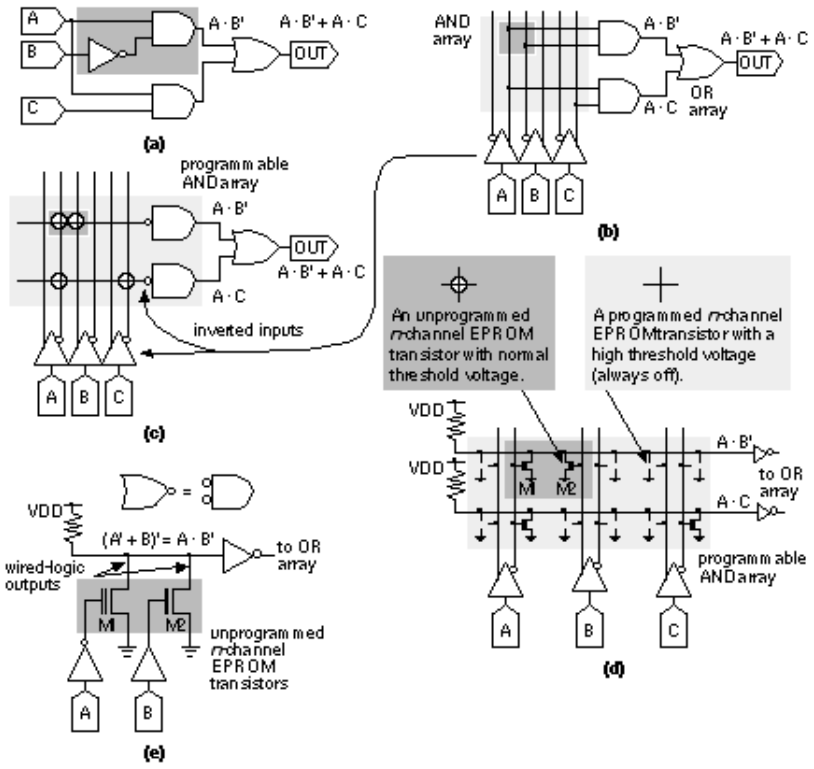


FIGURE 5.11 Logic arrays. (a) Two-level logic. (b) Organized sum of products. (c) A programmable-AND plane. (d) EPROM logic array. (e) Wired logic.

Because the arrangement of Figure 5.11 (c) is very similar to a ROM, we sometimes call a horizontal product-term line, which would be the bit output from a ROM, the bit line . The vertical input line is the word line . Figure 5.11 (d) and (e) show how to build the programmable-AND array (or product-term array) from EPROM transistors. The horizontal product-term lines connect to the vertical input lines using the EPROM transistors as pull-downs at each possible connection. Applying a ’1’ to the gate of an unprogrammed EPROM transistor pulls the product-term line low to a ’0’. A programmed n -channel transistor has a threshold voltage higher than V_{DD} and is therefore always off . Thus a programmed transistor has no effect on the product-term line.

Notice that connecting the n -channel EPROM transistors to a pull-up resistor as shown in Figure 5.11 (e) produces a wired-logic function-the output is high only if all of the outputs are high, resulting in a wired-AND function of the outputs. The product-term line is low when any of the inputs are high. Thus, to convert the wired-logic array into a programmable-AND array, we need to invert the sense of the inputs. We often conveniently omit these details when we draw the schematics of logic arrays, usually implemented as NOR-NOR arrays (so we need to invert the outputs as well). They are not minor details when you implement the layout, however.

Figure 5.12 shows how a programmable-AND array can be combined with other logic into a macrocell that contains a flip-flop. For example, the widely used 22V10 PLD, also called a registered PAL, essentially contains 10 of the macrocells shown in Figure 5.12 . The part number, 22V10, denotes that there are 22 inputs (44 vertical input lines for both true and complement forms of the inputs) to the programmable AND array and 10 macrocells. The PLD or registered PAL shown in Figure 5.12 has an 2

i ¥ jk programmable-AND array.

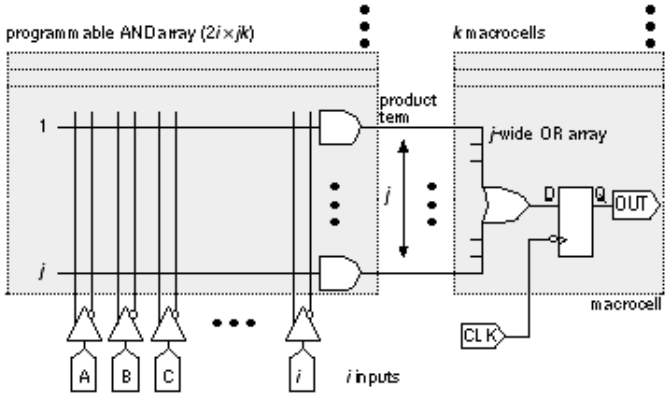


FIGURE 5.12 A registered PAL with i inputs, j product terms, and k macrocells.

5.4.1 Logic Expanders

The basic logic cell for the Altera MAX architecture, a macrocell, is a descendant of the PAL. Using the logic expander , shown in Figure 5.13 to generate extra logic terms, it is possible to implement functions that require more product terms than are available in a simple PAL macrocell. As an example, consider the following function:

F = A’ · C · D + B’ · C · D + A · B + B · C’.(5.22)

This function has four product terms and thus we cannot implement F using a macrocell that has only a three-wide OR array (such as the one shown in Figure 5.13). If we rewrite F as a "sum of (products of products)" like this:

F = (A’ + B’) · C · D + (A + C’) · B

= (A · B)’ (C · D) + (A’ · C)’ · B ;(5.23)

we can use logic expanders to form the expander terms (A · B)’ and (A’ · C)’ (see Figure 5.13). We can even share these extra product terms with other macrocells if we need to. We call the extra logic gates that form these shareable product terms a shared logic expander , or just shared expander .

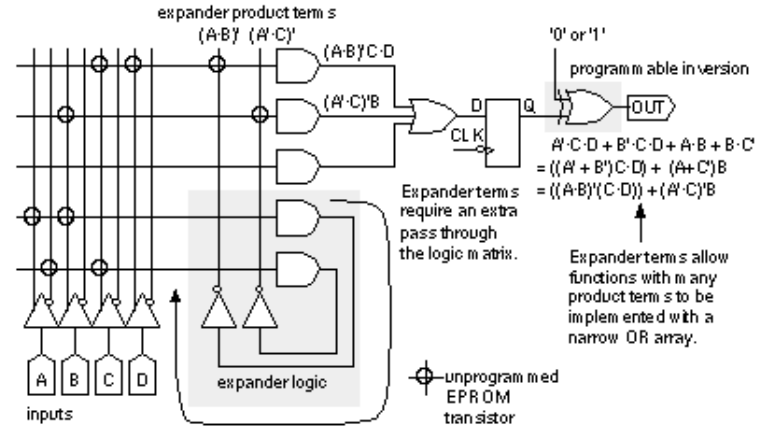


FIGURE 5.13 Expander logic and programmable inversion. An expander increases the number of product terms available and programmable inversion allows you to reduce the number of product terms you need.

The disadvantage of the shared expanders is the extra logic delay incurred because of the second pass that you need to take through the product-term array. We usually do not know before the logic tools assign logic to macrocells (logic assignment) whether we need to use the logic expanders. Since we cannot predict the exact timing the Altera MAX architecture is not strictly deterministic . However, once we do know whether a signal has to go through the array once or twice, we can simply and accurately predict the delay. This is a very important and useful feature of the Altera MAX architecture.

The expander terms are sometimes called helper terms when you use a PAL. If you use helper terms in a 22V10, for example, you have to go out to the chip I/O pad and then back into the programmable array again, using two-pass logic .

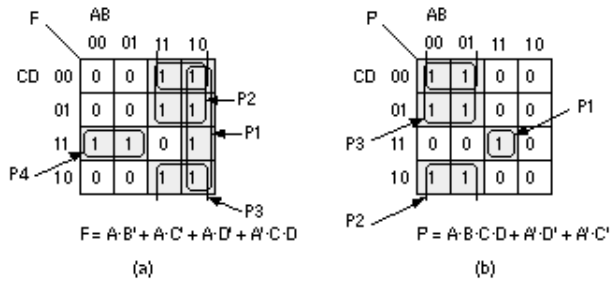


FIGURE 5.14 Use of programmed inversion to simplify logic: (a) The function F = A · B’ + A · C’ + A · D’ + A’ · C · D requires four product terms (P1-P4) to implement while (b) the complement, F ’ = A · B · C · D + A’ · D’ + A’ · C’ requires only three product terms (P1-P3).

Another common feature in complex PLDs, also used in some PLDs, is shown in Figure 5.13 . Programming one input of the XOR gate at the macrocell output allows you to choose whether or not to invert the output (a ’1’ for inversion or to a ’0’ for no inversion). This programmable inversion can reduce the required number of product terms by using a de Morgan equivalent representation instead of a conventional sum-of-products form, as shown in Figure 5.14 .

As an example of using programmable inversion, consider the function

$F = A \cdot B' + A \cdot C' + A \cdot D' + A' \cdot C \cdot D$,(5.24)

which requires four product terms-one too many for a three-wide OR array.

If we generate the complement of F instead,

$F' = A \cdot B \cdot C \cdot D + A' \cdot D' + A' \cdot C'$,(5.25)

this has only three product terms. To create F we invert F ', using programmable inversion.

Figure 5.15 shows an Altera MAX macrocell and illustrates the architectures of several different product families. The implementation details vary among the families, but the basic features: wide programmable-AND array, narrow fixed-OR array, logic expanders, and programmable inversion-are very similar. [1] Each family has the following individual characteristics:

- A typical MAX 5000 chip has: 8 dedicated inputs (with both true and complement forms); 24 inputs from the chipwide interconnect (true and complement); and either 32 or 64 shared expander terms (single polarity). The MAX 5000 LAB looks like a 32V16 PLD (ignoring the expander terms).
- The MAX 7000 LAB has 36 inputs from the chipwide interconnect and 16 shared expander terms; the MAX 7000 LAB looks like a 36V16 PLD.
- The MAX 9000 LAB has 33 inputs from the chipwide interconnect and 16 local feedback inputs (as well as 16 shared expander terms); the MAX 9000 LAB looks like a 49V16 PLD.

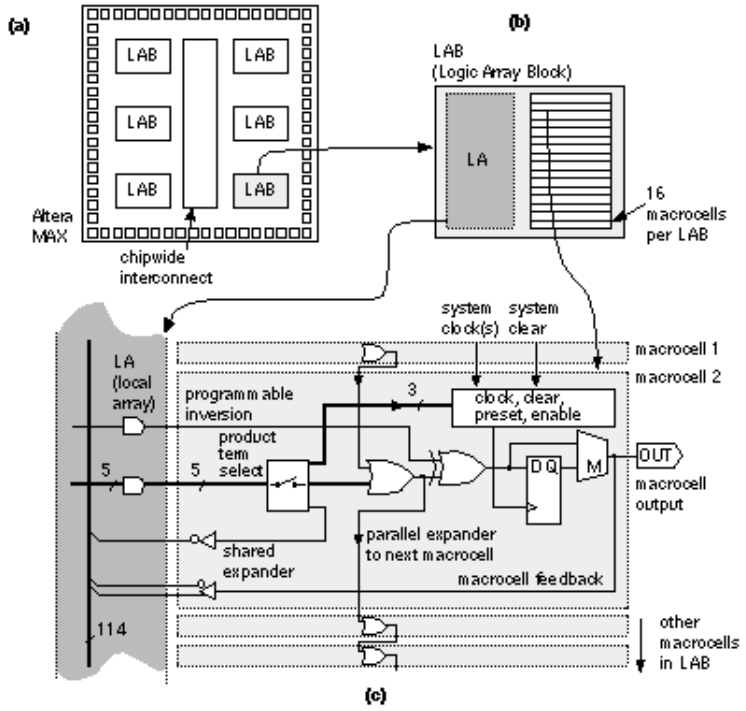


FIGURE 5.15 The Altera MAX architecture. (a) Organization of logic and interconnect. (b) A MAX family LAB (Logic Array Block). (c) A MAX family macrocell. The macrocell details

vary between the MAX families-the functions shown here are closest to those of the MAX 9000 family macrocells.

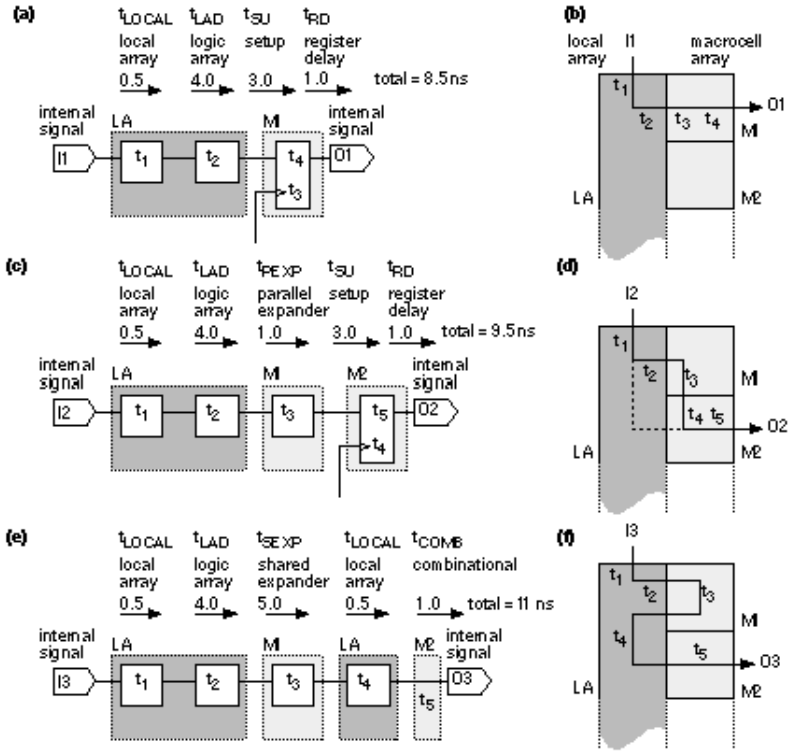


FIGURE 5.16 The timing model for the Altera MAX architecture. (a) A direct path through the logic array and a register. (b) Timing for the direct path. (c) Using a parallel expander. (d) Parallel expander timing. (e) Making two passes through the logic array to use a shared expander. (f) Timing for the shared expander (there is no register in this path). All timing values are in nanoseconds for the MAX 9000 series, '15' speed grade. (Source: Altera.)

5.4.2 Timing Model

Figure 5.16 shows the Altera MAX timing model for local signals. [2] For example, in Figure 5.16 (a) an internal signal, I1, enters the local array (the LAB interconnect with a fixed delay $t_1 = t_{LOCAL} = 0.5$ ns), passes through the AND array (delay $t_2 = t_{LAD} = 4.0$ ns), and to the macrocell flip-flop (with setup time, $t_3 = t_{SU} = 3.0$ ns, and clock-Q or register delay , $t_4 = t_{RD} = 1.0$ ns). The path delay is thus: 0.5 + 4 +3 + 1 = 8.5 ns.

Figure 5.16 (c) illustrates the use of a parallel logic expander . This is different from the case of the shared expander (Figure 5.13), which required two passes in series through the product-term array. Using a parallel logic expander, the extra product term is generated in an adjacent macrocell in parallel with other product terms (not in series-as in a shared expander).

We can illustrate the difference between a parallel expander and a shared expander using an example function that we have used before (Eq. 5.22),

$F = A' \cdot C \cdot D + B' \cdot C \cdot D + A \cdot B + B \cdot C'$.(5.26)

This time we shall use macrocell M1 in Figure 5.16 (d) to implement F1 equal to the sum of the first three product terms in Eq. 5.26 . We use F1 (using the parallel expander connection between adjacent macrocells shown in Figure 5.15) as an input to macrocell M2. Now we can form $F = F1 + B \cdot C'$ without using more than three inputs of an OR gate (the MAX 5000 has a three-wide OR array in the macrocell, the MAX 9000, as shown in Figure 5.15 , is capable of handling five product terms in one macrocell-but the principle is the same). The total delay is the same as before, except that we add the delay of a parallel expander, $t_{PEXP} = 1.0$ ns. Total delay is then 8.5 + 1 = 9.5 ns.

Figure 5.16 (e) and (f) shows the use of a shared expander-similar to Figure 5.13 .

The Altera MAX macrocell is more like a PLD than the other FPGA architectures discussed here; that is why Altera calls the MAX architecture a complex PLD. This means that the MAX architecture works well in applications for which PLDs are most useful: simple, fast logic with many inputs or variables.

5.4.3 Power Dissipation in Complex PLDs

A programmable-AND array in any PLD built using EPROM or EEPROM transistors uses a passive pull-up (a resistor or current source), and these macrocells consume static power . Altera uses a switch called the Turbo Bit to control the current in the programmable-AND array in each macrocell. For the MAX 7000, static current varies between 1.4 mA and 2.2 mA per macrocell in high-power mode (the current depends on the part-generally, but not always, the larger 7000 parts have lower operating currents) and between 0.6 mA and 0.8 mA in low-power mode. For the MAX 9000, the static current is 0.6 mA per macrocell in high-current mode and 0.3 mA in low-power mode, independent of the part size. [3] Since there are 16 macrocells in a LAB and up to 35 LABs on the largest MAX 9000 chip (16 ¥ 35 = 560 macrocells), just the static power dissipation in low-power mode can be substantial (560 ¥ 0.3 mA ¥ 5 V = 840 mW). If all the macrocells are in high-power mode, the static power will double. This is the price you pay for having an (up to) 114-wide AND gate delay of a few nanoseconds ($t_{LAD} = 4.0$ ns) in the MAX 9000. For any MAX 9000 macrocell in the low-power mode it is necessary to add a delay of between 15 ns and 20 ns to any signal path through the local interconnect and logic array (including t_{LAD} and t_{PEXP}).

---

1. 1995 data book p. 274 (5000), p. 160 (7000), p. 126 (9000).

2. March 1995 data sheet, v2.

3. 1995 data book, p. 1-47.

5.5 Summary

Table 5.4 is a look-up table to Tables 5.5 - 5.9 , which summarize the features of the logic cells used by the various FPGA vendors.

TABLE 5.4 Logic cell tables.

Programmable ASIC family		Programmable ASIC family	
Table 5.5	Actel (ACT 1) Xilinx (XC3000) Actel (ACT 2) Xilinx (XC4000) Altera MAX (EPM 5000)	Table 5.8	Actel (ACT 3) Xilinx LCA (XC5200) Altera FLEX (8000/10k) AMD MACH 5
Table 5.6	Xilinx EPLD (XC7200/7300) QuickLogic (pASIC 1) Crosspoint (CP20K)	Table 5.9	Actel 3200DX Altera MAX (EPM 9000)
Table 5.7	Altera MAX (EPM 7000) Atmel (AT6000)		

TABLE 5.5 Logic cells used by programmable ASICs.

	Actel ACT 1	Xilinx XC3000	Actel ACT 2	Xilinx XC4000
Basic logic cell	Logic module (LM)	CLB (Configurable Logic Block)	C-Module (combinatorial-module) and S-Module (sequential module)	CLB (Configurable Logic Block)
Logic cell contents	Three 2:1MUXes plus OR gate	32-bit LUT, 2 D flip-flops, 9 MUXes	C-Module: 4:1 MUX, 2-input OR, 2-input AND S-Module: 4-input MUX, 2-input OR, latch or D flip-flop	32-bit LUT, 2 D flip-flops, 10 MUXes, including fast carry logic E-suffix parts contain dual-port RAM.
Logic path delay	Fixed	Fixed with ability to bypass FF	Fixed	Fixed with ability to bypass FF
Combinational logic functions	Most 3-input, many 4-input functions (total 702 macros)	All 5-input functions plus 2 D flip-flops	Most 3- and 4-input functions (total 766 macros)	Two 4-input LUTs plus combiner with ninth input CLB as 32-bit SRAM (except D-suffix parts)
Flip-flop (FF)	1 LM required for latch, 2	2 D-flip-flops per CLB, latches can	1 S-Module per D flip-flop;	2 D flip-flops per

implementation	for latch, 2 LMs required for flip-flops	CLB, latches can be built from pre-FF logic.	some FFs require 2 modules.	2 D flip-flops per CLB
Basic logic cells in each chip	LMs: A1010: 352 (8R ¥ 44C) = 295 + 57 I/O A1020: 616 (14 R ¥ 44C) = 547 + 69 I/O	64 (XC3020/A/L, XC3120/A) 100 (XC3030/A/L, XC3130/A) 144 (XC3042/A/L, XC3142/A) 224 (XC3064/A/L, XC3164/A) 320 (XC3090/A/L, XC3190/A) 484 (XC3195/A)	A1225: 451 = 231 S + 220 C A1240: 684 = 348 S + 336 C A1280: 1232 = 624 S + 608 C	64 (XC4002A) 100 (XC4003/A/E/H) 144 (XC4004A) 196 (XC4005/A/E/H) 256 (XC4006/E) 324 (XC4008/E) 400 (XC4010/D/E) 576 (XC4013/D/E) 784 (XC4020/E) 1024 (XC4025/E)

TABLE 5.6 Logic cells used by programmable ASICs.

	Altera MAX 5000	Xilinx XC7200/7300	QuickLogic pASIC 1
Basic logic cell	16 macrocells in a LAB (Logic Array Block) except EPM5032, which has 32 macrocells in a single LAB	9 macrocells within a FB (Functional Block), fast FBs (FFBs) omit ALU	Logic Cell (LC)
Logic cell contents	Macrocell: 64-106-wide AND, 3-wide OR array, 1 flip-flop, 2 MUXes, programmable inversion. 32-64 shared logic expander OR terms. LAB looks like a 32V16 PLD.	Macrocell: 21-wide AND, 16-wide OR array, 1 flip-flop, 1ALU FB looks like 21V9 PLD.	Four 2-input and two 6-input AND, three 2:1 MUXes and one D flip-flop
Logic path delay	Fixed (unless using shared logic expanders)	Fixed	Fixed
Combinational logic functions per logic cell	Wide input functions with ability to share product terms	Wide input functions with added 2-input ALU	All 3-input functions
Flip-flop (FF) implementation	1 D flip-flop or latch per macrocell. More can be constructed in arrays.	1 D flip-flop or latch per macrocell FBs:	1 D flip-flop per LC. LCs for other flip-flops not specified.

Basic logic cells in each chip	LABs: 32 (EPM5032) 64 (EPM5064) 128 (EPM5128) 128 (EPM5130) 192 (EPM5192)	4 (XC7236A) 8 (XC7272A) 2 (XC7318) 4 (XC7336) 6 (XC7354) 8 (XC7372) 12 (XC73108) 16 (XC73144)	48 (QL6X8) 96 (QL8X12) 192 (QL12X16) 384 (QL16X24)

TABLE 5.7 Logic cells used by programmable ASICs.

	Crosspoint CP20K	Altera MAX 7k	Atmel AT6000
Basic logic cell	Transistor-pair tile (TPT), RAM-logic Tile (RLT)	16 macrocells in a LAB (Logic Array Block)	Cell
Logic cell contents	TPT: 2 transistors (0.5 gate). RLT: 3 inverters, two 3-input NANDs, 2-input NAND, 2-input AND.	Macrocell: wide AND, 5-wide OR array, 1 flip-flop, 3 MUXes, programmable inversion. 16 shared logic expander OR terms, plus parallel logic expander. LAB looks like a 36V16 PLD.	Two 5:1 MUXes, two 4:1 MUXes, 3:1 MUX, three 2:1 MUXes, 6 pass gates, four 2-input gates, 1 D flip-flop
Logic path delay	Variable	Fixed (unless using shared logic expanders)	Variable
Combinational functions per logic cell	TPT is smaller than a gate, approx. 2 TPTs = 1 gate.	Wide input functions with ability to share product terms	1-, 2-, and 3-input combinational configurations: 44 logical states and 72 physical states
Flip-flop (FF) implementation	D flip-flop requires 2 RLTs and 9 TPTs	1 D flip-flop or latch per macrocell. More can be constructed in arrays.	1 D flip-flop per cell
	TPTs: 1760 (20220)	Macrocells: 32 (EPM7032/V) 64 (EPM7064)	1024 (AT6002)

Basic logic cells in each chip	15,876 (22000) RLTs: 440 (20220) 3969 (22000)	96 (EPM7096) 128 (EPM70128E) 160 (EPM70160E) 192 (EPM70192E) 256 (EPM70256E)	1600 (AT6003) 3136 (AT6005) 6400(AT6010)

TABLE 5.8 Logic cells used by programmable ASICs.

	Actel ACT 3	Xilinx XC5200	Altera FLEX 8000/10k
Basic logic cell	2 types of Logic Module: C-Module and S-Module (similar but not identical to ACT 2)	4 Logic Cells (LC) in a CLB (Configurable Logic Block)	8 Logic Elements (LE) in a Logic Array Block (LAB)
Logic cell contents (LUT = look-up table)	C-Module: 4:1 MUX, 2-input OR, 2-input AND. S-Module: 4:1 MUX, 2-input OR, latch or D flip-flop.	LC has 16-bit LUT, 1 flip-flop (or latch), 4 MUXes	16-bit LUT, 1 programmable flip-flop or latch, MUX logic for control, carry logic, cascade logic
Logic path delay	Fixed	Fixed	Fixed with ability to bypass FF
Combinational functions per logic cell	Most 3- and 4-input functions (total 766 macros)	One 4-input LUT per LC may be combined with adjacent LC to form 5-input LUT	4-input LUT may be cascaded with adjacent LE
Flip-flop (FF) implementation	1 D flip-flop (or latch) per S-Module; some FFs require 2 modules.	1 D flip-flop (or latch) per LC (4 per CLB)	1 D flip-flop (or latch) per LE
			LEs: 208 (EPF8282/V/A /AV) 336 (EPF8452/A) 504 (EPF8636A)

			672 (EPF8820/A)
Basic logic cells in each chip	A1415: 104 S + 96 C	64 CLB (XC5202)	1008 (EPF81188/A)
	A1425: 160 S + 150 C	120 CLB (XC5204)	1296 (EPF81500/A)
	A1440: 288 S + 276 C	196 CLB (XC5206)	
	A1460: 432 S + 416 C	324 CLB (XC5210)	576 (EPF10K10)
	A14100: 697 S + 680 C	484 CLB (XC5215)	1152 (EPF10K20)
			1728 (EPF10K30)
			2304 (EPF10K40)
			2880 (EPF10K50)
			3744 (EPF10K70)
			4992 (EPF10K100)

TABLE 5.9 Logic cells used by programmable ASICs.

	AMD MACH 5	Actel 3200DX	Altera MAX 9000
Basic logic cell	4 PAL Blocks in a Segment, 16 macrocells in a PAL Block	Based on ACT 2, plus D-module (decode) and dual-port SRAM	16 macrocells in a LAB (Logic Array Block)
Logic cell contents	20-bit to 32-bit wide OR array, switching logic, XOR gate, programmable flip-flop	C-Module: 4:1 MUX, 2-input OR, 2-input AND S-Module: 4-input MUX, 2-input OR, latch or D flip-flop D-module: 7-input AND, 2-input XOR	Macrocell: 114-wide AND, 5-wide OR array, 1 flip-flop, 5 MUXes, programmable inversion. 16 shared logic expander OR terms, plus parallel logic expander. LAB looks like a 49V16 PLD.
Logic path delay	Fixed	Fixed	Fixed (unless using expanders)
Combinational functions per logic cell	Wide input functions	Most 3- and 4-input functions (total 766 macros)	Wide input functions with ability to share product terms
Flip-flop (FF)	1 D flip-flop or latch per	1 D flip-flop or latch per S-Module; some	1 D flip-flop or latch per

implementation	1 D flip-flop or latch per macrocell	per S-Module; some FFs require 2 modules.	macrocell. More can be constructed in arrays.
Basic logic cells in each chip	128 (M5-128) 192 (M5-192) 256 (M5-256) 320 (M5-320) 384 (M5-384) 512 (M5-512)	A3265DX: 510 S + 475 C + 20 D A32100DX: 700 S + 662 C + 20 D + 2 kSRAM A32140D): 954 S + 912 C + 24 D A32200DX: 1 230 S + 1 184 C + 24 D + 2.5 kSRAM A32300DX: 1 888 S + 1 833 C + 28 D + 3kSRAM A32400DX: 2 526 S + 2 466 C + 28 D + 4 kSRAM	Macrocells: 320 (EPM9320) 4 ¥ 5 LABs 400 (EPM9400) 5 ¥ 5 LABs 480 (EPM9480) 6 ¥ 5 LABs 560 (EPM9560) 7 ¥ 5 LABs

The key points in this chapter are:

- The use of multiplexers, look-up tables, and programmable logic arrays
- The difference between fine-grain and coarse-grain FPGA architectures
- Worst-case timing design
- Flip-flop timing
- Timing models
- Components of power dissipation in programmable ASICs
- Deterministic and nondeterministic FPGA architectures

5.6 Problems

* = Difficult, ** = Very difficult, *** = Extremely difficult

5.1 (Using the ACT 1 Logic Module, 30 min.) Consider the Actel ACT 1 Logic Module shown in Figure 5.1 . Show how to implement: (a) a three-input NOR gate, (b) a three-input majority function gate, (c) a 2:1 MUX, (d) a half adder, (e) a three-input XOR gate, and (f) a four-input MUX.

5.2 (Worst-case and best-case timing, 10 min.) Seasoned digital CMOS designers do not worry too much when their designs stop working when they get too hot or when they reduce the supply voltage, but an ASIC that stops working either when increasing the supply voltage above normal or when it gets cold causes panic. Why?

5.3 (Typical to worst-case variation, 10 min.) The 1994 Actel data book (p. 1-5) remarks that: "the total derating factor from typical to worst-case for a standard ACT 1 array is only 1.19:1, compared to 2:1 for a masked gate array."

- a. Can you explain why this is when the basic ACT 1 CMOS process is identical to a CMOS process for masked gate arrays?
- b. There is a price to pay for the reduced spread in timing delays from typical to worst-case in an ACT 1 array. What is this disadvantage of the ACT 1 array over a masked gate array?

5.4 (ACT 2/3 sequential element, 30 min.). Show how the Actel ACT 2 and ACT 3 sequential element of Figure 5.4 (used in the S-Module) can be wired to implement:

- a. a positive-edge-triggered flip-flop with clear,
- b. a negative-edge-triggered flip-flop with clear,
- c. a transparent-high latch,
- d. a transparent-low latch, and
- e. how it can be made totally transparent.

5.5 (*ACT 1 logic functions, 40 min.+)

- a. How many different combinational functions of four logic variables are there?
- b. of n variables? Hint: Consider the truth table.
- c. The ACT 1 module can implement 213 of the 256 functions with three variables. How many of the 43 three-input functions that it cannot implement can you find?
- d. (harder) Show that if you have access to both the true and complement form of the input variables you can implement all 256 logic functions of three variables with the ACT 1 Logic Module.

5.6 (Actel and Xilinx, 10 min.) The Actel Logic Modules (ACT 1, ACT 2, and ACT 3) have eight inputs and can implement most three-input logic functions and a few logic functions with four input variables. In contrast, the Xilinx XC5200 CLB, for example, has only four inputs but can implement all logic functions with four or fewer variables. Why would Actel choose these logic cell designs and how can they be competitive with the Xilinx FPGA (which they are)?

5.7 (Actel address decoders, 10 min.) The maximum number of inputs that the ACT 1 Logic Module can handle is four. The ACT 2/ACT 3 C-module increases this to five.

- a. How many ACT 1 Logic Modules do you need to implement a 32-bit wide address decoder (a 32-input AND gate)?
- b. How many ACT 2/ACT 3 C-modules do you need?

5.8 (Altera shared logic expanders, 30 min.) Consider an Altera MAX 5000 logic array with three product-term lines. You cannot directly implement the function $Z = A \cdot B \cdot C + A \cdot B' \cdot C' + A' \cdot B \cdot C' + A' \cdot B' \cdot C$ with a programmable array logic macrocell that has only three product-term lines, since Z has four product terms.

- a. How many Boolean functions of three variables are there that cannot be implemented with a programmable array logic macrocell that has only three product terms? Hint: Use a Karnaugh map

to consider how many Boolean functions of three variables have more than three product terms in their sum-of-products representation.

- b. Show how to use shared logic expanders that feed terms back into the product-term array to implement the function Z using a macrocell with three product terms.
- c. How many shared expander lines do you need to add to be able to implement all the Boolean functions of three variables?
- d. What is the largest number of product terms that you need to implement a Boolean function with n variables?

5.9 (Splitting the XC3000 CLB, 20 min.) In Section 5.2.1 we noted "You can split the (XC3000) 32-bit LUT in half, using one of the seven input variables to switch between the F and G outputs. This technique can implement some functions of six and seven variables."

- a. Show which functions of six and seven variables can, and
- b. which functions cannot, be implemented using this method.

5.10 (Programmable inversion, 20 min.) Section 5.4 described how the Altera MAX series logic cells can use programmable inversion to reduce the number of product terms needed to implement a function. Give another example of a function of four variables that requires four product terms. Is there a way to tell how many product terms a function may require?

5.11 (Table look-up mapping, 20 min.) Consider a four-input LUT (used in the CLB in the Xilinx XC2000, the first generation of Xilinx FPGAs, and in the XC5200 LE). This CLB can implement any Boolean function of four variables. Consider the function

$$Z = (A \cdot (B + C)) + (B \cdot D) + (E \cdot F \cdot G \cdot H \cdot I) \quad (5.27)$$

We can use four CLBs to implement Z as follows:

CLB1: $Z = Z1 + (B \cdot D) + Z3$,

CLB2: $Z1 = A \cdot (B + C)$,

CLB3: $Z3 = E \cdot F \cdot G \cdot Z5$,

CLB4: $Z5 = H \cdot I$.(5.28)

What is the length of the critical path? Find a better assignment in terms of area and critical path.

5.12 (Multiplexer mapping, 10 min.) Consider the function:

$$F = (A \cdot B) + (B' \cdot C) + D \quad (5.29)$$

Use Shannon's expansion theorem to expand F wrt B:

$$F = B \cdot F1 + B' \cdot F2 \quad (5.30)$$

In other words express F in terms of B, B', F1, and F2 (Hint: F1 is a function of A and D only, F2 is a

function of C and D only). Now expand F1 wrt A, and F2 wrt C. Using your answer, implement F using a single ACT 1 Logic Module.

5.13 (*Xilinx hazards, 10 min.) Explain why the outputs of the Xilinx CLBs are hazard-free for input changes in only one variable. Is this important?

5.14 (**Actel S-Modules, 10 min.) Notice that CLR is tied to the input corresponding to B0 of the C-module in the ACT 2 S-Module but the CLR input is separate from the B0 input in the ACT 3 version. Why?

5.15 (**Timing estimates, 60 min.) Using data book values for an FPGA architecture that you choose, and explaining your calculations carefully, estimate the (worst-case commercial) delay for the following functions: (a) 16-bit address decoder, (b) 8-bit ripple-carry adder, (c) 8-bit ripple-carry counter. Give your answers in terms of the data book symbols, and using actual parameters, for a speed grade that you specify, give an example calculation with the delay in ns.

5.16 (Actel logic. 30 min.) Table 5.10 shows how to use the Actel ACT 1 Logic Module to implement some of the 16 functions of two input variables. Complete this table.

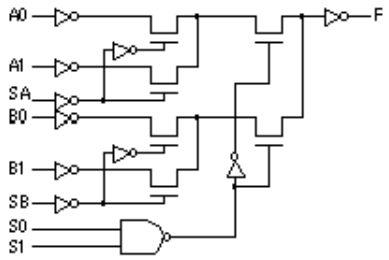
TABLE 5.10 Boolean functions using the ACT 1 Logic Module (Problem 5.16).

	Function, F	F =	Canonical form	Minterms	M1			M2			OR1	
					A0	A1	SA	B0	B1	SB	S0	S1
1	0	0	0	-	0	0	0					
2	AND(A, B)	A · B	A · B	3	0	B	A					
3	AND1-1(A, B)	A · B’	A · B’	2	A	0	B					
4	NOR(A, B)	A + B	A’ · B’	0								
5	NOR1-1(A, B)	A + B’	A’ · B	1	B	0	A					
6	A	A	A · B’ + A · B	2, 3	0	A	1					
7	B	B	A’ · B + A · B	1, 3	0	B	1					
8	NOT(A)	A’	A’ · B’ + A’ · B	0, 1	0	1	A					
9	NOT(B)	B’	A’ · B’ + A · B’	0, 2	0	1	B					
10	EXOR(A, B)	A ? B	A’ · B + A · B’	1, 2								
11	EXNOR(A, B)	(A ? B)	A’ · B’ + A · B	0, 3								
12	OR(A, B)	A + B	A’ · B + A · B’ + A · B	1, 2, 3	B	1	A					
13	OR1-1(A, B)	A + B’	A’ · B’ + A · B’ + A · B	0, 2, 3								
14	NAND(A, B)	(A · B)’	A’ · B’ + A’ · B + A · B’	0, 1, 2								
15	NAND1-1(A, B)	(A · B’)’	A’ · B’ + A’ · B + A · B	0, 1, 3								
16	1	1	A’ · B’ + A’ · B + A · B’ + A · B	0, 1, 2, 3	1	1	1					

5.17 (ACT 1 module implementation, 120 min.)

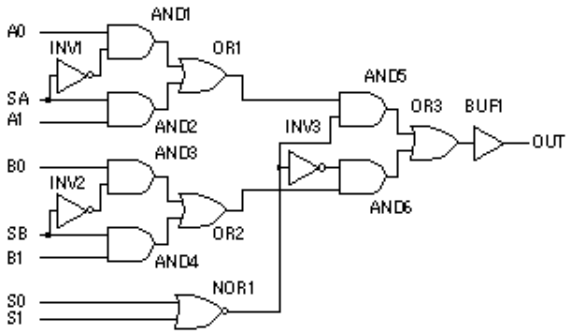
- a. Show that the circuit shown in Figure 5.17 , with buffered inputs and outputs, is equivalent to the one shown in Figure 5.1 .

FIGURE 5.17 An alternative implementation of the ACT 1 Logic Module shown in Figure 5.1 (Problem 5.17).



- b. Show that the circuit for the ACT 1 Logic Module shown in Figure 5.18 is also the same.
- c. Convert the circuit of Figure 5.18 to one that uses more efficient CMOS gates: inverters, AOI, and NAND gates.
- d. (harder) Assume that the ACT 1 Logic Module has the equivalent of a 2X drive and the logic ratio is close to one. Compare your answer to part c against Figure 5.17 in terms of logical efficiency and logical area.

FIGURE 5.18 A schematic equivalent of the Actel ACT 1 Logic Module (Problem 5.17).



5.18 (**Xilinx CLB analysis, 60 min.) Table 5.11 shows some information derived from a die photo in the AT&T ATT3000 series data book that shows the eight by eight CLB matrix on an ATT3020 (equivalent to a XC3020) clearly. By measuring the die size in the photo and knowing the actual die size we can calculate the size of a CLB matrix element (ME) that includes a single XC3000 CLB as approximately 277 mil ^{2} . The ME includes interconnect, SRAM, programming, and other resources as well as a CLB.

TABLE 5.11 ATT3020 die information (Problem 5.18). 1

Parameter	Data book	Die photo	Calculated
3020 die width	183.5 mil	4.1 cm	-
3020 die height	219.3 mil	4.9 cm	-
3000 ME width	-	0.325 cm	14.55 mil = 370 m m
3000 ME height	-	0.425 cm	19.02 mil = 483 m m
3000 ME area	-	-	277 mil ^{2}

3020 pad pitch - 1.6 mm / pad 7.21 mil / pad

- a. The minimum feature size in the AT&T Holmdel twin-tub V process used for the ATT3000 family is 0.9 m m. Using a value of l = 0.45 m m, calculate the Xilinx XC3000 ME size in l^2.
- b. Estimate, explaining your assumptions, the area of the XC4000 ME, and the XC5200 ME (both in l^2).
- c. Table 5.12 shows the ATT3000 die information. Using a value of 277 mil^2 for the ATT/XC3000 ME area, complete this table.

TABLE 5.12 ATT3000 die information (Problem 5.18). 2

Die	Die height mil	Die width mil	Die area mil^2	Die area cm2	CLBs	ME area mil^2	ME area cm2
3020	219.3	183.5	40,242	0.26	8 ¥ 8		
3030	259.8	215.0	55,857	0.36	10 ¥ 10		
3042	295.3	242.5	71,610	0.46	12 ¥ 12		
3064	270.9	366.5	99,285	0.64	16 ¥ 14		
3090	437.0	299.2	130,750	0.84	16 ¥ 20		

---

1. Data from AT&T data book, July 1992, p. 3-76, MN92-024FPGA

2. Data from AT&T data book, July 1992, p. 3-75, MN92-024FPGA. 1 $mil^2 = 10^{-6}$ $in^2 = 2.54^2$ ¥ 10^{-6} $cm^2 = 6.452$ ¥ 10^{-6} cm^2

5.7 Bibliography

The book by Brown et al. [1992] on FPGAs deals with commercially available FPGAs and logic block architecture. There are several easily readable articles on FPGAs in the July 1993 issue of the IEEE Proceedings including articles by Rose et al. [1993] and Greene et al. [1993]. Greene's article is a good place to start digging deeper into the Actel FPGA architecture and gives an idea of the very complex problem of programming antifuses, something we have not discussed. Trimberger, who works at Xilinx, has edited a book on FPGAs [1994]. For those wishing to understand even more about the trade-offs in the different programmable ASIC architectures, a student of Stanford Professor Abbas El Gamal (one of the cofounders of Actel) has completed a Ph.D. on this topic [Kouloheris, 1993]. The best resources for information on FPGAs and their logic cells are the manufacturer's data sheets, data books, and application notes. The data books change every year or so as new products are released, so it is difficult to give specific references, but Xilinx, Actel, and Altera currently produce huge volumes complete with excellent design guides and application notes-you should obtain each of these even if you are not currently using that particular technology. Many of these are also online in Adobe Acrobat and PostScript

5.8 References

Brown, S. D., et al. 1992. Field-Programmable Gate Arrays. Norwell, MA: Kluwer Academic. 206 p. ISBN 0-7923-9248-5. TK7872.L64F54. Introduction to FPGAs, Commercially Available FPGAs, Technology Mapping for FPGAs, Logic Block Architecture, Routing for FPGAs, Flexibility of FPGA Routing Resources, A Theoretical Model for FPGA Routing. Includes an introduction to commercially available FPGAs. The rest of the book covers research on logic synthesis for FPGAs and FPGA architectures, concentrating on LUT-based architectures.

Greene, J., et al. 1993. "Antifuse field programmable gate arrays." Proceedings of the IEEE, vol. 81, no. 7, pp. 1042-1056. Review article describing the Actel FPGAs. (Included in the Actel 1994 data book.)

Kouloheris, J. L. 1993. "Empirical study of the effect of cell granularity on FPGA density and performance." Ph.D. Thesis, Stanford, CA. 114 p. Detailed research study of the different FPGA architectures concentrating on structures similar to the Actel and Altera FPGAs.

Rose, J., et al. 1993. "A classification and survey of field-programmable gate array architectures." Proceedings of the IEEE, vol. 81, no. 7.

Trimberger, S. M. (Ed.). 1994. Field-Programmable Gate Array Technology.