display manufacturer to fix bugs in the circuit, and even upgrade the circuit, without requiring the high cost of creating a new ASIC. Third, the displays themselves are fairly large, expensive, and consume much power, and therefore the larger size, higher cost, and more power consumption of FPGAs compared to ASICs do not impact the overall display's size, cost, and power too significantly.

▶ 7.7 CHAPTER SUMMARY

Section 7.1 discussed the idea that circuits must be mapped to a physical implementation so that those circuits can be inserted into a real system. Section 7.2 introduced some IC types that require that a new chip be fabricated to implement our circuit. A full-custom IC type gives the most optimized implementation, but is expensive and time-consuming to design. Semicustom IC types give very good implementations while costing less and taking less time to design, through the predesigning of the gates or cells that will be used on the IC. Section 7.3 described the increasingly popular IC type of FPGAs, and showed how a circuit could be mapped onto a set of programmable lookup tables and switch matrices. Section 7.4 highlighted several other IC types, including off-the-shelf SSI/MSI ICs, and programmable logic devices. Section 7.5 provided some data showing the relative popularity of the IC types described in the chapter.

An interesting trend in physical implementation is the trend toward programmable ICs (FPGAs in particular). Implementing functionality on an FPGA involves the task of downloading a bitstream into the FPGA IC device. One might notice the similarity of that task with the task of implementing functionality on a microprocessor, which also involves downloading bits into an IC device. Thus, the difference between software on a microprocessor and custom digital circuits continues to be blurred—especially when one considers that modern FPGAs can also include one or several microprocessors within the same IC. For more information on the blurring, see "The Softening of Hardware," F. Vahid, *IEEE Computer*, April 2003, and also "It's Time to Stop Calling Circuits Hardware," F. Vahid, *IEEE Computer*, September 2007.

▶ 7.8 EXERCISES

SECTION 7.2: MANUFACTURED IC TYPES

7.1 Explain why a gate array IC type has a shorter production time than a full-custom IC type.

7.2 Explain why the use of NAND or NOR gates in a CMOS gate array circuit implementation is typically preferred over an AND/OR/NOT implementation of a circuit.

7.3 Draw a gate array IC having three rows, the first row having four 2-input AND gates, the second row having four 2-input OR gates, and the third row having four NOT gates. Show how to instantiate wires to the gate array to implement the function `F(a,b,c) = abc + a'b'c'`.

7.4 Assume that a standard cell library has a 2-input AND gate, a 2-input OR gate, and a NOT gate. Use a drawing to show how to instantiate and place standard cells on an IC and wire them together to implement the function in Exercise 7.3. Draw your cells the same size as the gates in Exercise 7.3, and be sure your rows are of equal size.

7.5 Draw a gate array IC having three rows, the first row having four 2-input AND gates, the second row having four 2-input OR gates, and the third row having four NOT gates. Show

how to instantiate wires to the gate array to implement the equation `F(a,b,c,d) = a'b + cd + c'`.

7.6 Assume that a standard cell library has a 2-input AND gate, a 2-input OR gate, and a NOT gate. Use a drawing to show how to instantiate and place standard cells on an IC and wire them together to implement the function in Exercise 7.5. Be sure to draw your cells the same size as the gates in Exercise 7.5, and be sure your rows are of equal size.

7.7 Consider the implementations of a half-adder with a gate array in Figure 7.7 and with standard cells in Figure 7.5. Assume that each gate or cell (including inverters) has a delay of 1 ns. Also assume that every inch of wire (for each inch in your drawing, not on an actual IC) has a delay of 3 ns (wires are relatively slow in the era of tiny fast transistors). Estimate the delay of the gate array and the standard cell circuits.

7.8 For your solutions to Exercise 7.3 and Exercise 7.4, assume that each gate and cell has a delay of 1 ns, and that every inch of wire (for each inch in your drawing, not on an actual IC) corresponds to a delay of 3 ns. Estimate the delays of the gate array and standard cell circuits.

7.9 Draw a circuit using AND, OR, and NOT gates for the following equation: `F(a, b, c) = a'bc + abc'`. Place inversion bubbles on that circuit to convert the circuit to:
(a) NAND gates only,
(b) NOR gates only.

7.10 Draw a circuit using AND, OR, and NOT gates for the following equation: `F(a, b, c) = abc + a' + b' + c'`. Place inversion bubbles on that circuit to convert the circuit to:
(a) NAND gates only,
(b) NOR gates only.

7.11 Draw a circuit using AND, OR, and NOT gates for the following equation: `F(a, b, c) = (ab + c)(a' + d) + c'`. Convert the circuit to a circuit using:
(a) NAND gates only,
(b) NOR gates only.

7.12 Draw a circuit using AND, OR, and NOT gates for the following equation: `F(w, x, y, z) = (w + x)(y + z) + wy + xz`. Convert the circuit to a circuit using:
(a) NAND gates only,
(b) NOR gates only.

7.13 Draw a circuit using AND, OR, and NOT gates for the following equation: `F(a, b, c, d) = (ab)(b' + c) + (a'd + c')`. Convert the circuit to a circuit using:
(a) NAND gates only,
(b) NOR gates only.

7.14 Show how to convert the following gates into circuits having only 3-input NAND gates:
(a) A 3-input AND gate.
(b) A 3-input OR gate.
(c) A NOT gate.

7.15 Assume that a standard cell library consists of 2-input and 3-input NAND gates with a delay of 1 ns each, 2-input and 3-input AND and OR gates with a delay of 1.8 ns each, and a NOT gate with a delay of 1 ns. Compare the number of transistors and the delay of an implementation using only AND/OR/NOT gates with an implementation using only NAND gates for the function: `F(a, b, c)=ab'c + a'b`. For calculating the size of an implementation, assume that each gate input requires two transistors.

7.16 Assume that a standard cell library consists of 2-input AND and OR gates with a delay of 1 ns each, 3-input AND and OR gates with a delay of 1.5 ns each, and a NOT gate with a delay of 1 ns. Compare the number of transistors and the delay of an implementation using only 2-input AND/OR gates and NOT gates with an implementation using only 3-input AND/OR

gates and NOT gates for the function: `F(a, b, c)= abc + a'b'c + a'b'c'`. For calculating the size of an implementation, assume that each gate input requires two transistors.

7.17 Assume a standard cell library consisting of 2-input NAND and NOR gates with a delay of 1 ns each, and 3-input NAND and NOR gates with a delay of 1.5 ns each. Compare the number of transistors and the delay of an implementation using only 2-input NAND/NOR gates with an implementation using only 3-input NAND/NOR gates for the function: `F(a, b, c)= a'bc + ab'c + abc'`. For calculating the size of an implementation, assume that each gate input requires two transistors.

SECTION 7.3: PROGRAMMABLE IC TYPE—FPGA

7.18 Show how to implement on a 3-input 2-output lookup table the function `F(a, b, c) = a + bc`.

7.19 Show how to implement on two 3-input 2-output lookup tables the function `F(a, b, c, d) = ab + cd`. Assume you can connect the lookup tables in a custom manner (i.e., do not use a switch matrix, just directly connect your wires).

7.20 Show how to implement on two 3-input 2-output lookup tables the following function: `F(a, b, c, d) = a'bd + b'cd'`. Assume the two lookup tables are connected in the manner shown in Figure 7.47. You may not need to use every lookup table output.

7.21 Show how to implement on two 3-input 2-output lookup tables the following functions: `F(x, y, z) = x'y + xyz'` and `G(w, x, y, z) = w'x'y + w'xyz'`. Assume the two lookup tables are connected in the manner shown in Figure 7.47.

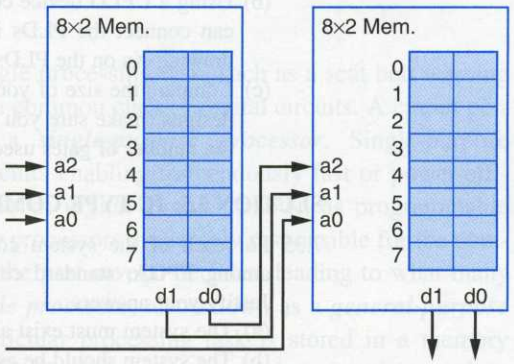


Figure 7.47 Two 3-input 2-output lookup tables implemented using 8x2 memory.

7.22 Show how to implement on two 3-input 2-output lookup tables the following functions: `F(a, b, c, d) = abc + d` and `G = a'`. You must implement both `F` and `G` with only two lookup tables connected in the manner shown in Figure 7.47.

7.23 Implement a 2-bit comparator that compares two 2-bit numbers and has three outputs indicating greater-than, less-than, and equal-to, using any number of 3-input 2-output lookup tables and custom connections among the lookup tables.

7.24 Show how to implement a 4-bit carry-ripple adder using any number of 3-input 2-input lookup tables and custom connections among the lookup tables. Hint: map one full-adder to each lookup table.

7.25 Show how to implement a 4-bit carry-ripple adder using any number of 4-input 1-output lookup tables and custom connections among the lookup tables.

7.26 Show how to implement a comparator that compares two 8-bit numbers and has a single equal-to output, using any number of 4-input 1-output lookup tables and custom connections among the lookup tables.

7.27 Show the bit file necessary to program the FPGA fabric in Figure 7.31 to implement the function `F(a, b, c, d) = ab + cd`, where `a`, `b`, `c`, and `d` are external inputs.

7.28 Show the bit file necessary to program the FPGA fabric in Figure 7.31 to implement the function `F(a, b, c, d) = abcd`, where `a`, `b`, `c`, and `d` are external inputs.

7.29 Show the bit file necessary to program the FPGA fabric in Figure 7.31 to implement the function `F(a, b, c, d) = a'b' + c'd`, where `a`, `b`, `c`, and `d` are external inputs.

SECTION 7.4: OTHER IC TYPES

7.30 Use any combination of 7400 ICs listed in Table 7.1 to implement the function `F(a, b, c, d) = ab + cd`.

7.31 Use any combination of 7400 ICs listed in Table 7.1 to implement the function `F(a, b, c, d) = abc + ab'c' + a'bd + a'b'd'`.

7.32 By drawing Xs on the circuit, program the PLD of Figure 7.38(a) to implement a full-adder.

7.33 By drawing Xs on the circuit, program the PLD of Figure 7.38(a) to implement a 2-bit equality comparator. Assume the PLD has an additional `I4` input.

7.34 *(a)Design a PLD device capable of supporting a 2-bit carry-ripple adder. By drawing Xs on your PLD circuit, program the PLD to implement the 2-bit carry-ripple adder.

(b) Using a CPLD device consisting of several PLDs from Figure 7.38(a) and assuming you can connect the PLDs in a custom manner, implement the 2-bit carry-ripple adder by drawing Xs on the PLDs.

(c) Compare the size of your PLD and the CPLD by determining the gates required for both designs (make sure you compare the number of gates within the PLD and CPLD and not the number of gates used for your implementation).

SECTION 7.5: IC TYPE COMPARISONS

7.35 For each of the system constraints below, choose the most appropriate technology from among FPGA, standard cell, and full-custom IC types for implementing a given circuit. Justify your answers.

(a) The system must exist as a physical prototype by next week.

(b) The system should be as small and low-power as possible. Short design time and low cost are *not* priorities.

(c) The system should be reprogrammable even after the final product has been produced.

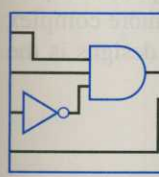
(d) The system should be as fast as possible and should consume as little power as possible, subject to being completely implemented in just a few months.

(e) Only five copies of the system will be produced and we have no more than $1000 to spend on all the ICs.

7.36 Which of the following implementations are *not* possible? (1) A custom processor on an FPGA. (2) A custom processor on an ASIC. (3) A custom processor on a full-custom IC. (4) A programmable processor on an FPGA. (5) A programmable processor on an ASIC. (6) A programmable processor on a full-custom IC. For each, explain your answer.

CHAPTER

▶ 8.1 INTR



Seat bel
warning lig
single-purp
processo

Figure 8.1 Sing