CHAPTER 7

PHYSICAL IMPLEMENTATION

7.1 EXERCISES

Section 7.2: Manufactured IC Technologies

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

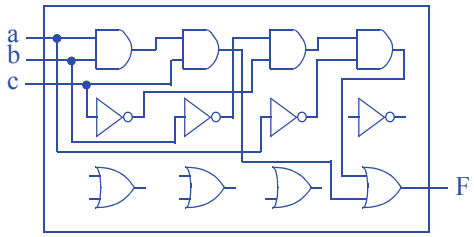
Full-custom IC technology requires that every layer of the chip be manufactured, and each layer takes time to produce. Gate array IC technology only requires the wiring layers to be manufactured, so the lower transistor layers can be pre-manufactured. Furthermore, gate array technology will have fewer errors due to eliminating-errors in the pre-designed transistor layers.

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

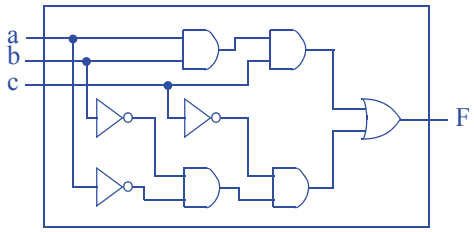
NAND and NOR gates have more efficient CMOS implementations, due to pMOS transistors being efficient at passing 1s and nMOS transistors being efficient at passing 0s. As such, a 2-input NAND gate can be built using two pMOS transistors connected to 1 (power) and two nMOS transistors connected to 0 (ground); an AND gate would then be built be adding an inverter (two more transistors) to the NAND output, yielding more transistors and larger delay.

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 having row four NOT

gates. Show how to instantiate wires to the gate array to implement the function `F(a,b,c) = abc + a'b'c'`.

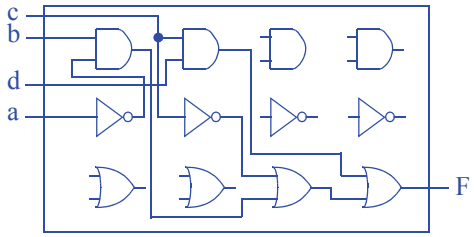


7.4 Assume a standard cell library has a 2-input AND gate, a 2-input OR gate, and a NOT gate. Use a drawing to show how to instantiate and place standard cells on an IC and wire them together to implement the function in Exercise 7.3. Draw your cells the same size as the gates in Exercise 7.3, and be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



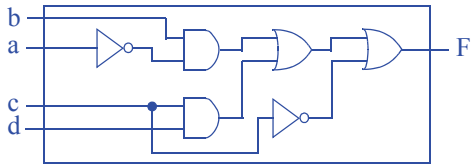
Note that wires are shorter. There are also fewer gates.

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 having row four NOT gates. Show how to instantiate wires to the gate array to implement the function `F(a,b,c,d) = a'b + cd + c'`.



7.6 Assume 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.



Note that wires are shorter. There are also fewer gates.

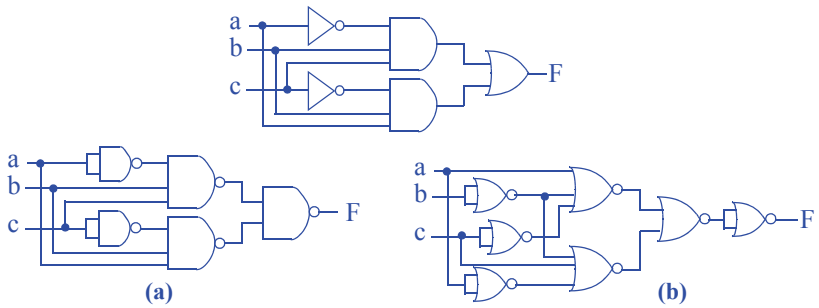
7.7 Consider the implementations of a half adder with a gate array in Figure 7.5 and with standard cells in Figure 7.7. Assume each gate or cell (including inverters) has a delay of 1 ns. Also assume that every inch of wire (for each inch in your drawing, not on an actual IC) in the drawing has a delay 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.

The gate array-based half adder requires 3 levels of gates, contributing 3ns to its delay, and approximately 4.25" of wire, contributing 12.75ns to its delay for a total of 15.75ns. The standard cell-based half adder requires 3 levels of gates (3ns) and approximately 3" of wire (9ns) for a total delay of 12ns.

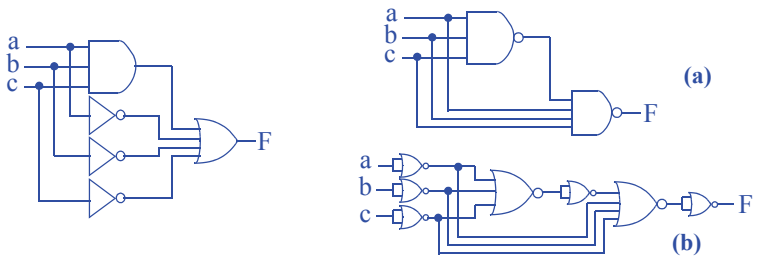
7.8 For your solutions to Exercises 7.3 and 7.4, assume that each gate and cell has a delay of 1 ns, and that every inch of wire (for each inch in your drawing, not on an actual IC) your drawing corresponds to a delay of 3 ns. Estimate the delays of the gate-array and standard cell circuits.

Our solution to Exercise 7.3 required 4 levels of gates (4ns) and approximately 4.5" of wire (13.5ns) for a total delay of 17.5ns. Our solution to Exercise 7.4 required 4 levels of gates (4ns) and approximately 3" of wire (9ns) for a total delay of 13ns.

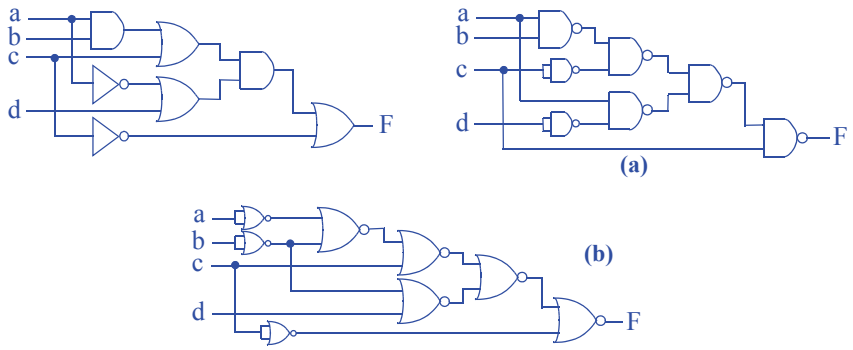
7.9 Draw a circuit using AND, OR and NOT gates for the following function: `F(a,b,c) = a'bc + abc'`. Place inversion bubbles on that circuit to convert that circuit to: (a) NAND gates only, (b) NOR gates only.



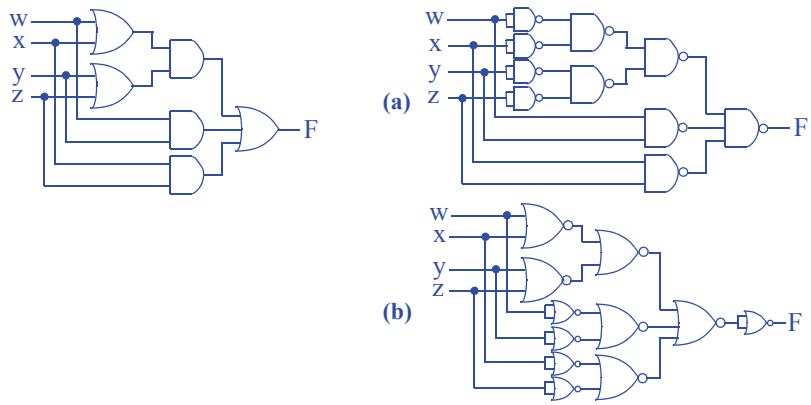
7.10 Draw a circuit using AND, OR and NOT gates for the following function: `F(a,b,c) = abc + a′ + b′ + c′`. Place inversion bubbles on that circuit to convert that circuit to: (a) NAND gates only, (b) NOR gates only.



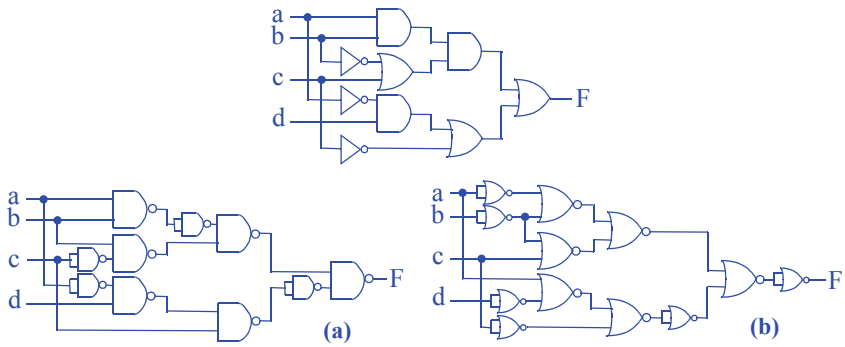
7.11 Draw a circuit using AND, OR, and NOT gates for the following function: `F(a,b,c) = (ab + c)(a′ + d) + c′`. Convert the circuit to a circuit using: (a) NAND gates only, (b) NOR gates only.



7.12 Draw a circuit using AND, OR, and NOT gates for the following function: `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..



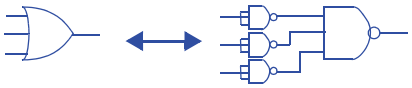
7.13 Draw a circuit using AND, OR, and NOT gates for the following function: `F(a,b,c,d) = (ab)(b' + c) + (a'd + c')`. Convert the circuit to a circuit using: (a) NAND gates only, (b) NOR gates only.



7.14 Show how to convert the following gates into circuits having only 3-input NAND gates:

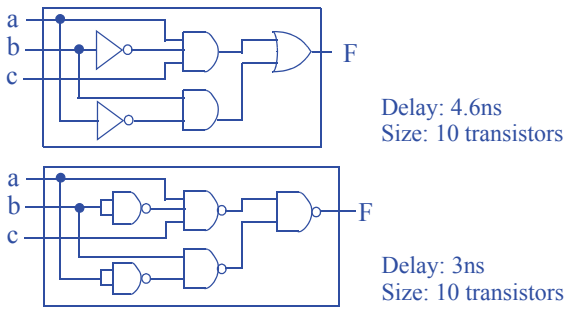
a. a 3-input AND gate

b. a 3-input OR gate.

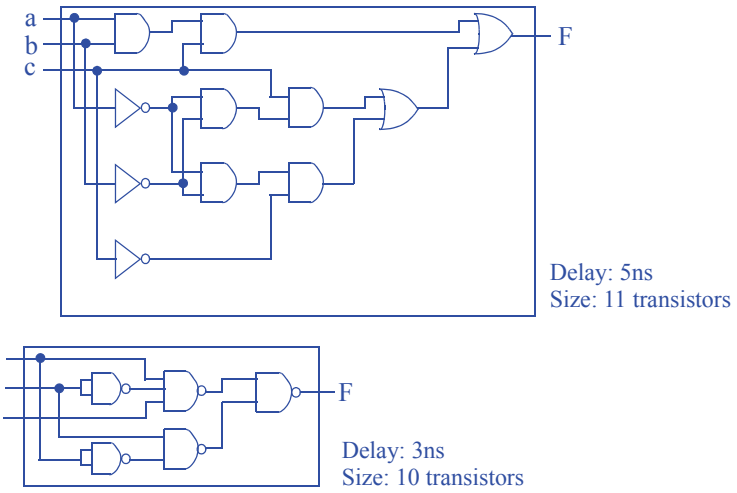


c. a NOT gate.

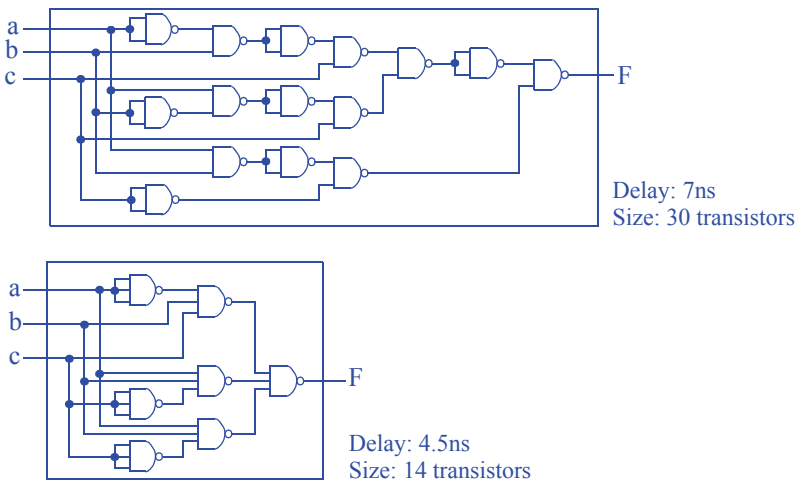
7.15 Assume a standard cell library consisting of 2-input and 3-input NAND gates with a delay of 1 ns each, 2-input and 3-input AND and OR gates with a delay of 1.8 ns each, and a NOT gate with a delay of 1 ns. Compare the number of transistors and the delay of an implementation using only AND/OR/NOT 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 each gate requires two transistors.



7.16 Assume a standard cell library consisting 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 each gate requires two transistors.



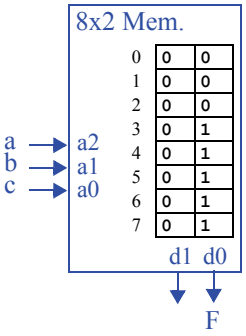
7.17 Assume a standard cell library consisting of 2-input NAND and NOR gates with a delay of 1 ns each, and 3-input NAND and NOR gates with a delay of 1.5 ns each. Compare the number of transistors and the delay of an implementation using only 2-input NAND/NOR gates with an implementation using only 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 each gate requires two transistors.



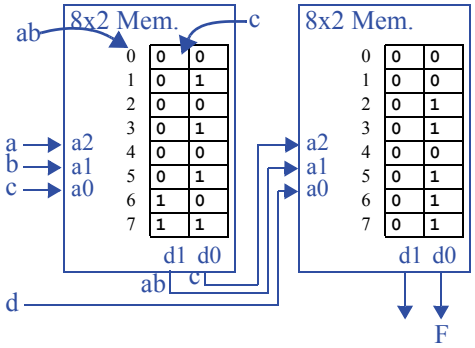
Section 7.3: Programmable IC Technology -- FPGA

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

Inputs			Outputs
a	b	c	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



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.

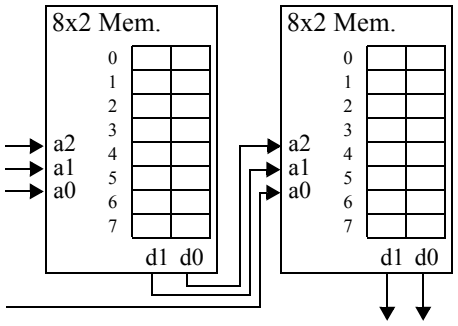
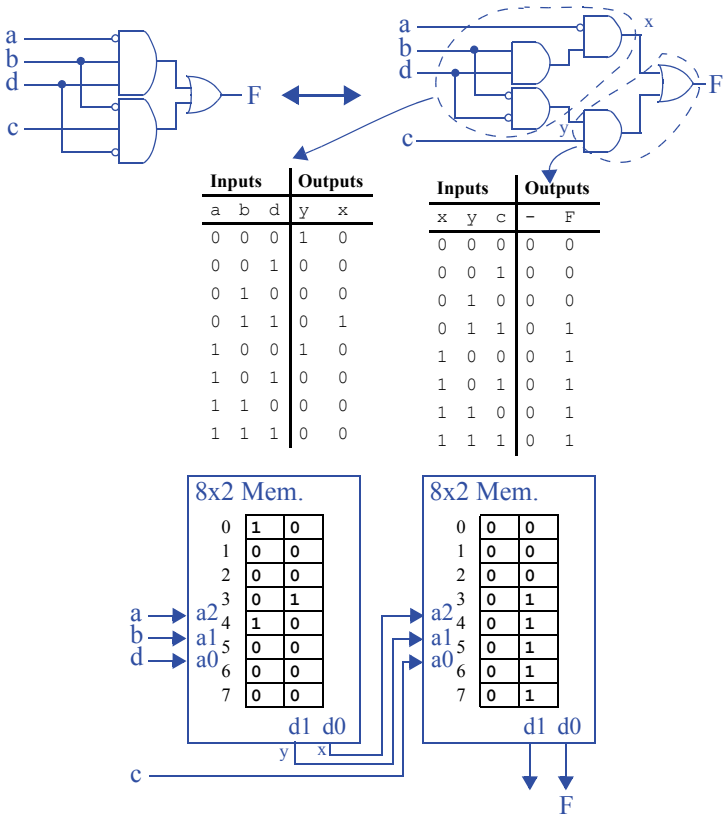
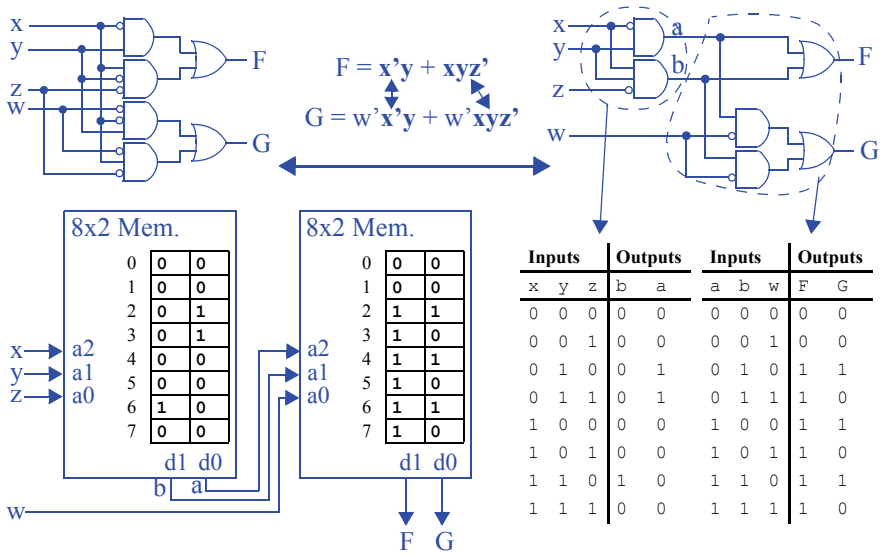


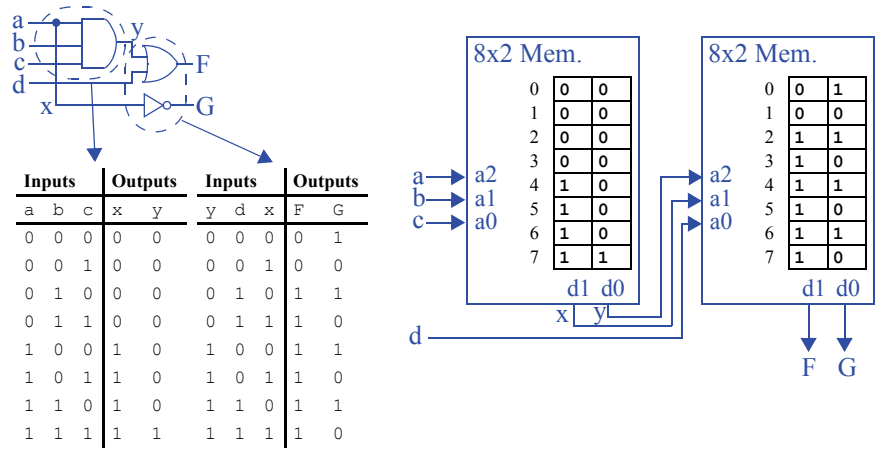
Figure 7.47: Two 3-input 2-output lookup tables implemented using 8x2 memory.


a
b
d
c
F
x
y
Inputs
Outputs
a b d y x
0 0 0 1 0
0 0 1 0 0
0 1 0 0 0
0 1 1 0 1
1 0 0 1 0
1 0 1 0 0
1 1 0 0 0
1 1 1 0 0
Inputs
Outputs
x y c - F
0 0 0 0 0
0 0 1 0 0
0 1 0 0 0
0 1 1 0 1
1 0 0 0 1
1 0 1 0 1
1 1 0 0 1
1 1 1 0 1
8x2 Mem.
a2
a1
a0
d1 d0
8x2 Mem.
a2
a1
a0
d1 d0
F

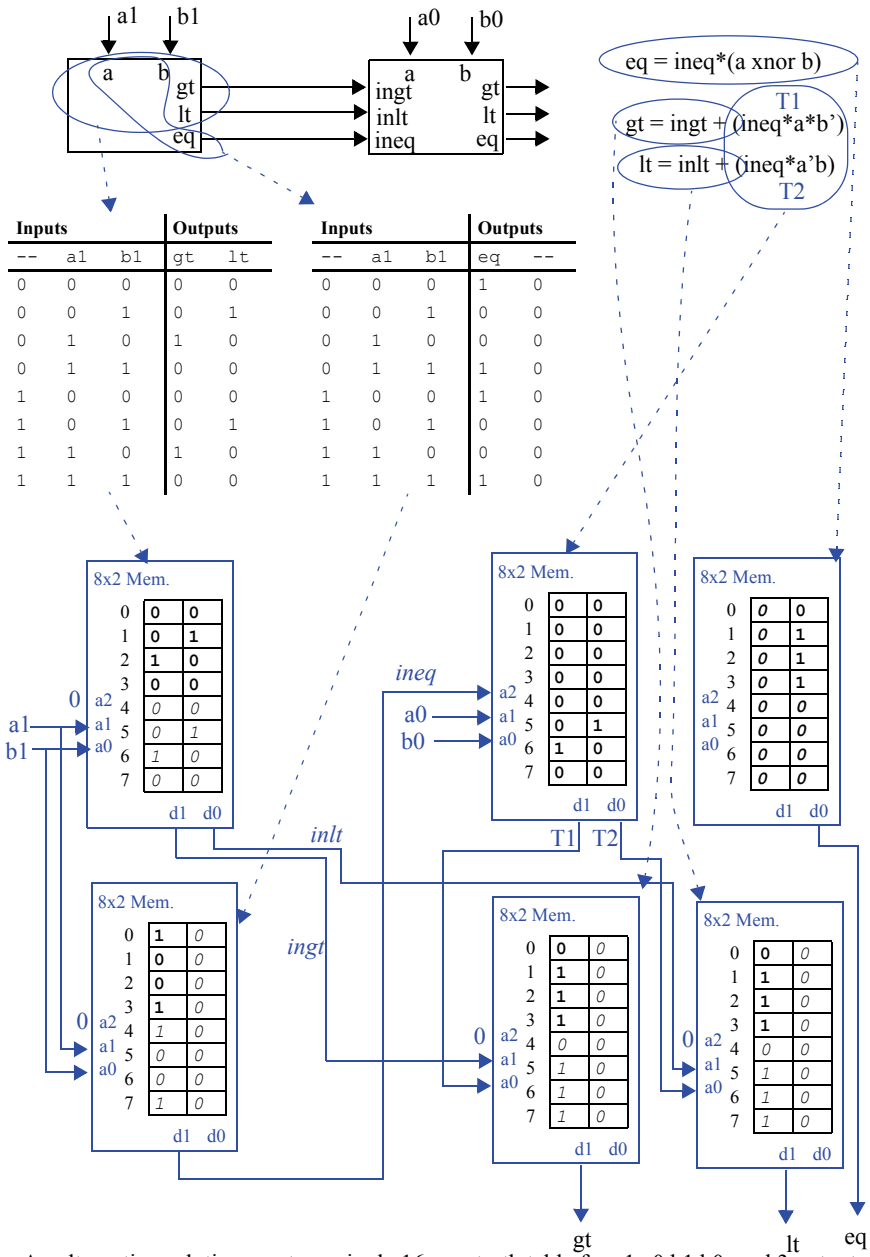
7.21 Show how to implement on two 3-input 2-output lookup tables the following functions: `F(x,y,z) = x'y + xyz'` and `G(w,x,y,z) = w'x'y + w'xyz'`. Assume the two lookup tables are connected in the manner shown in Figure 7.47.



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.



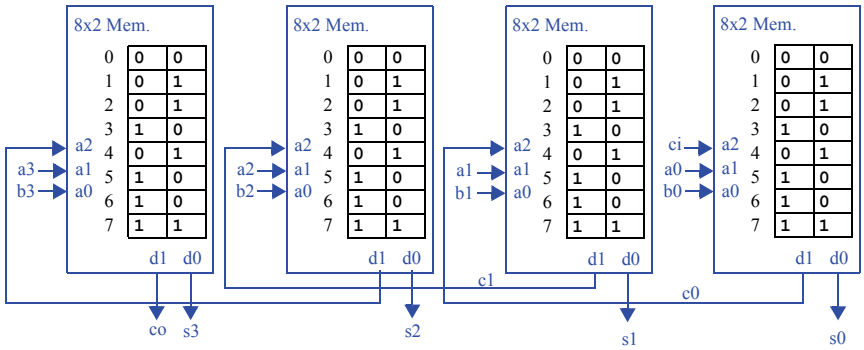
7.23 Implement a 2-bit comparator that compares two 2-bit numbers and has three outputs indicating greater-than, less-than, and equal-to, using any number of 3-input 2-output lookup tables and custom connections among the lookup tables.



An alternative solution creates a single 16-row truth table for a1 a0 b1 b0, and 3 output functions gt, lt, eq; creates minimized equations; and maps equations to LUTs. The above ripple-carry-based approach may be simpler.

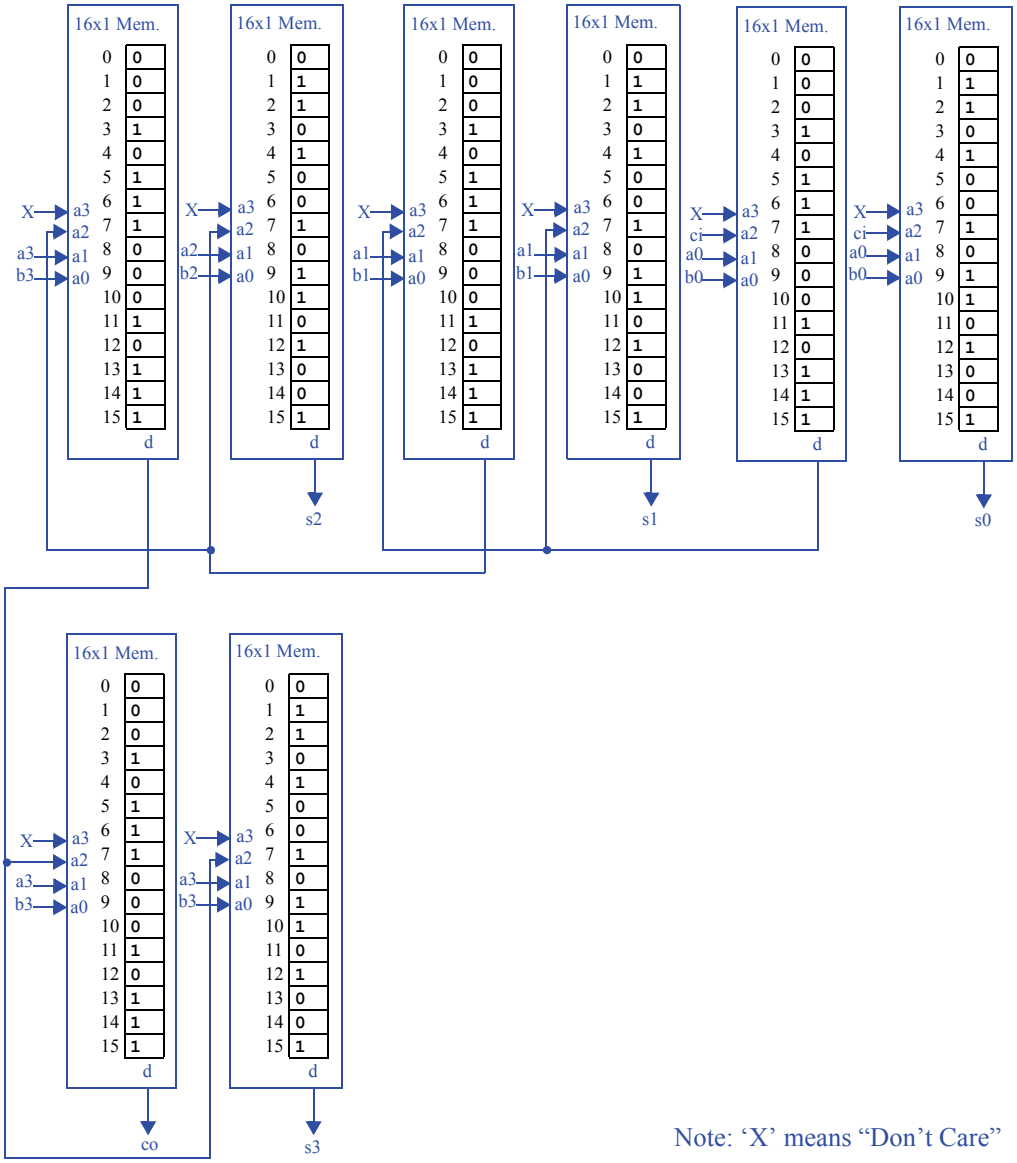
Only the left component need be completed for this exercise. The right component with the ilt, ieq, igt components goes beyond the exercise's problem statement.

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



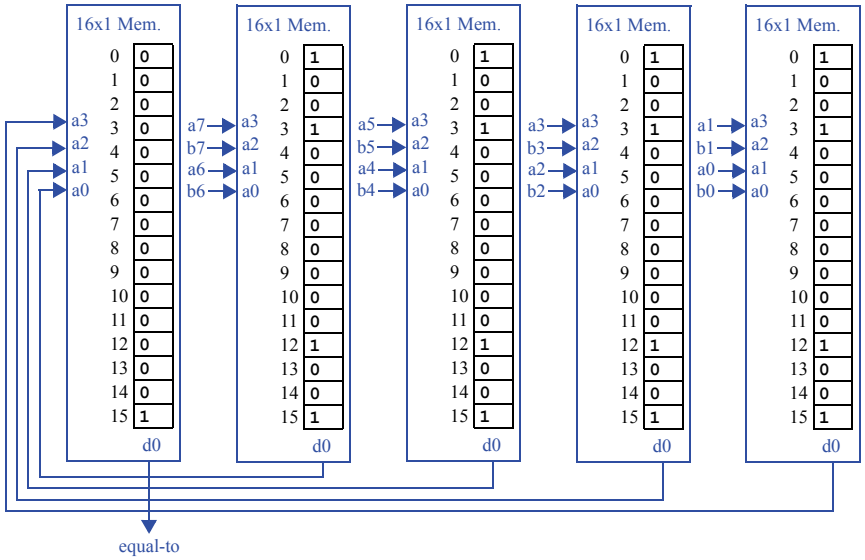
7.25 Show how to implement a 4-bit carry-ripple adder using any number of 4-input 1-output lookup tables and custom connections among the lookup tables.

Similarly to Exercise 7.24, we can simply use one LUT for each output of a full-adder. We can just "ignore" the extra input by repeating the first 8 entries of the table to fill the last 8 entries of the table.

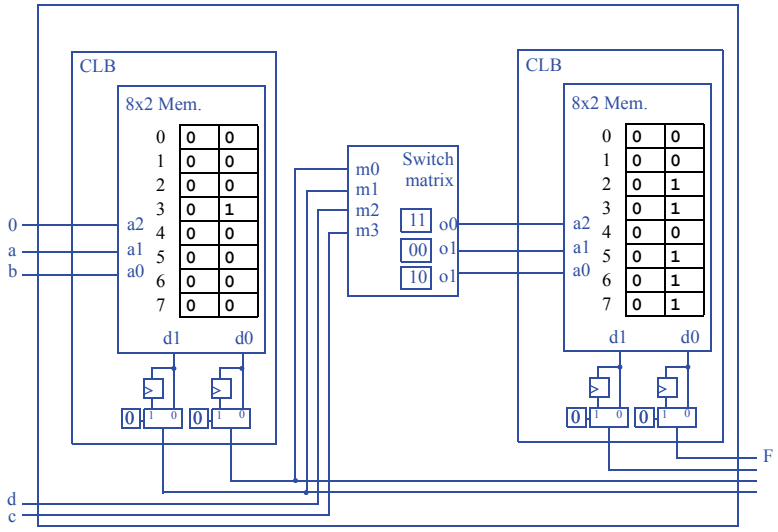


Note: 'X' means "Don't Care"

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.

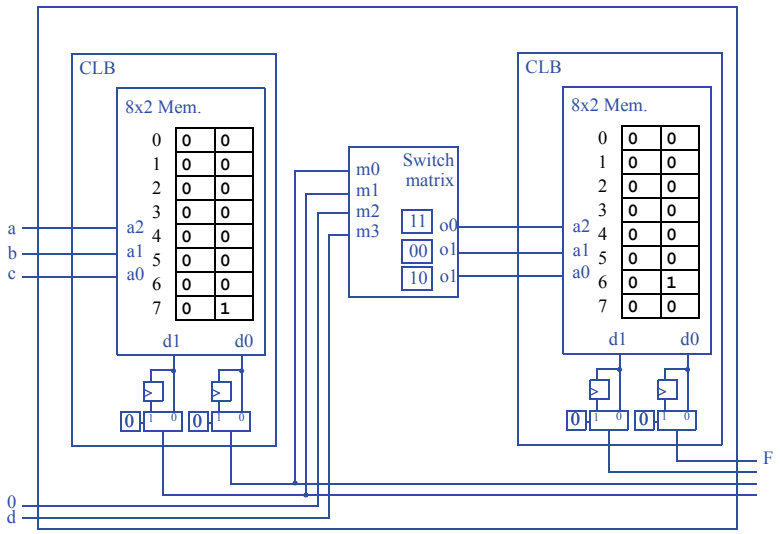


7.27 Show the bitfile necessary to program the FPGA fabric in Figure 7.31 to implement the function `F(a,b,c,d) = ab + cd`, where `a`, `b`, `c` and `d` are external inputs.



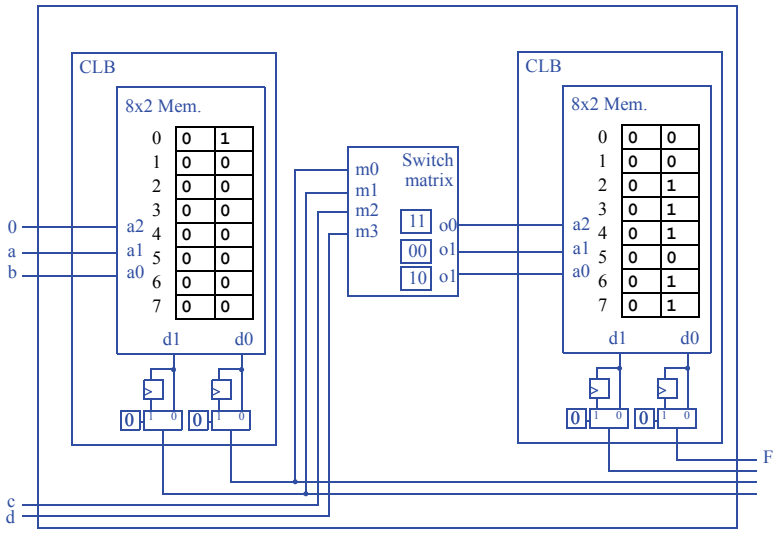
The corresponding bitfile is: 00000000 00010000 0 0 11 00 10 00000000 00110111 0 0

7.28 Show the bitfile 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.



The corresponding bitfile is: 00000000 00000001 0 0 11 00 10 00000000 00000010 0 0

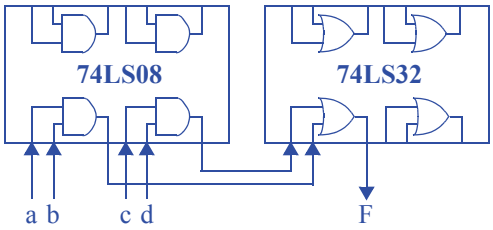
7.29 Show the bitfile necessary to program the FPGA fabric in Figure 7.31 to implement the function `F(a,b,c,d) = a'b' + c'd`, where `a`, `b`, `c` and `d` are external inputs.



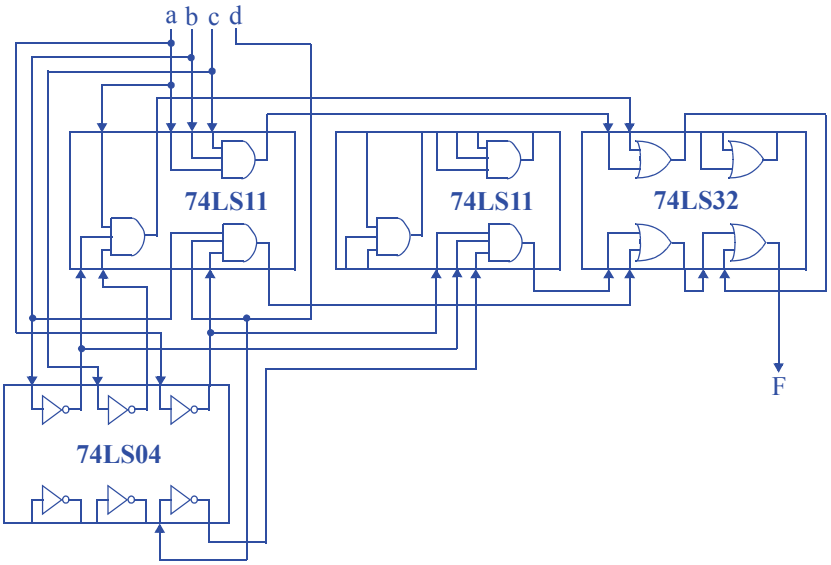
The corresponding bitfile is: 00000000 10000000 0 0 11 00 10 00000000 00111011 0 0

Section 7.4: Other Technologies

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

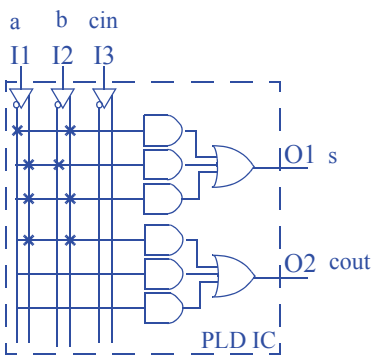


7.31 Use any combination of 7400 ICs listed in Table 7.1 to implement the function `F(a,b,c,d) = abc + ab'c' + a'bd + a'b'd'`.

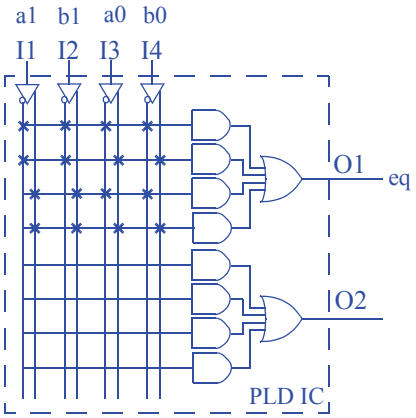


7.32 By drawing Xs on the circuit, program the PLD of Figure 7.38(a) to implement a full-adder.

Inputs			Outputs	
a	b	cin	cout	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



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 and assuming you can connect the PLDs in a custom manner, implement the 2-bit carry-ripple adder by drawing X's 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).

Solution not shown for challenge problems.

Section 7.5: IC Technology Comparisons

7.35 For each of the system constraints below, choose the most appropriate technology from among FPGA, standard cell, and full-custom IC technologies 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 $1,000 to spend on all the ICs.

a) FPGA
b) Full-custom IC
c) FPGA
d) Standard cell
e) FPGA

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. Explain your answer.

None of the above - both a custom processor and a progammable processor can be implemented on either an FPGA, an ASIC, or a full-custom IC. Each implementation has its own strengths and weaknesses, but each implementation is possible.