▶ 6.9 EXERCISES

SECTION 6.1: INTRODUCTION

6.1 Define the terms "optimization" and "tradeoff."

6.2 A homeowner wishes to increase the amount of light inside the house during the day, with the only criteria of interest being the amount of light and the cost of electricity. Describe how to increase the light via: (a) an optimization, (b) a tradeoff.

SECTION 6.2: COMBINATIONAL LOGIC OPTIMIZATIONS AND TRADEOFFS

6.3 Perform two-level logic size optimization for `F(a,b,c) = ab'c + abc + a'bc + abc'` using (a) algebraic methods, (b) a K-map. Express the answers in sum-of-products form.

6.4 Perform two-level logic size optimization for `F(a,b,c) = a + a'b'c + a'c` using a K-map.

6.5 Perform two-level logic size optimization for `F(a,b,c,d) = a'bc' + abc'd' + abd` using a K-map.

6.6 Perform two-level logic size optimization `F(a,b,c,d) = ab + a'b'd'` using a K-map.

6.7 Perform two-level logic size optimization for `F(a,b,c) = a'b'c + abc`, assuming input combinations `a'bc` and `ab'c` can never occur (those two minterms represent don't cares).

6.8 Perform two-level logic size optimization for `F(a,b,c,d) = a'bc'd + ab'cd'`, assuming that `a` and `b` can never both be `1` at the same time, and that `c` and `d` can never both be `1` at the same time (i.e., there are don't cares).

6.9 Consider the function `F(a,b,c) = a'c + ac + a'b`. Using a K-map: (a) Determine which of the following terms are implicants (but not necessarily prime implicants) of the equation: `a'b'c', a'b', a'bc, a'c, c, bc, a'bc', a'b`. (b) Determine which of those terms are prime implicants of the function.

6.10 For the function `F(a,b,c) = a'c + ac + a'b`, determine all prime implicants and all essential prime implicants: (a) using a K-map, (b) using the tabular method.

6.11 For the equation `F(a,b,c,d) = ab'c' + abc'd + abcd + a'bcd + a'bcd'`, determine all prime implicants and all essential prime implicants: (a) using a K-map, (b) using the tabular method.

6.12 Use repeated application of the expand operation to heuristically minimize the equation `F(a,b,c) = a'b'c + a'bc + abc`. (a) Try expanding each term for each variable. (b) Instead, determine a way to randomly choose an expand operation, and then apply 5 random expands.

6.13 Use repeated application of the expand operation to heuristically minimize the equation `F(a,b,c,d,e) = abcde + abcde' + abcd'e'`. (a) Try expanding each term for each variable. (b) Instead, determine a way to randomly choose an expand operation, and then apply 5 random expands.

6.14 Using algebraic methods, reduce the number of gate inputs for the following equation by creating a multilevel circuit: `F(a,b,c,d,e,f,g) = abcde + abcd'e'fg + abcd'e'f'g'`. Assume only AND, OR, and NOT gates will be used. Draw the circuit for the original equation and for the multilevel circuit, and clearly list the delay and number of gate inputs for each circuit.

SECTION 6.3: SEQUENTIAL LOGIC OPTIMIZATIONS AND TRADEOFFS

6.15 Reduce the number of states for the FSM in Figure 6.88 using the partitioning method.

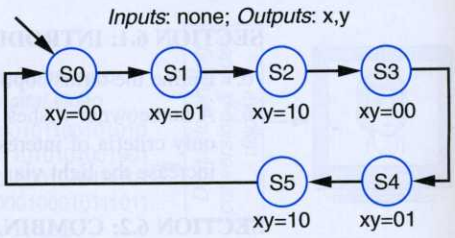

Figure 6.88 FSM example.

6.16 Reduce the number of states for the FSM in Figure 6.89 using the partitioning method.

6.17 Reduce the number of states for the FSM in Figure 6.90 using the partitioning method.

6.18 Compare the logic size (number of gate inputs) and the delay (number of gate-delays) of a straightforward 2-bit binary encoding of the FSM in Figure 6.91 using a 3-bit output encoding versus using a one-hot encoding.

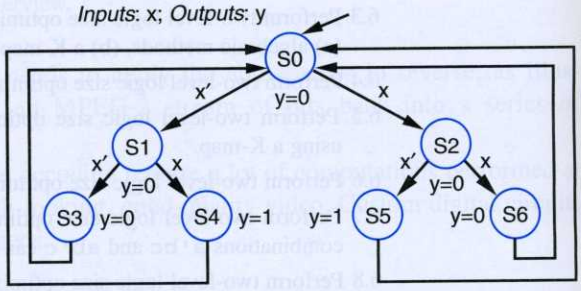

Figure 6.89 Sequence detector for bit patterns `01` and `10`.

6.19 Compare the logic size (number of gate inputs) and the delay (number of gate-delays) of a minimal bitwidth state encoding versus an output encoding for the laser-based distance measurer FSM shown in Figure 5.26.

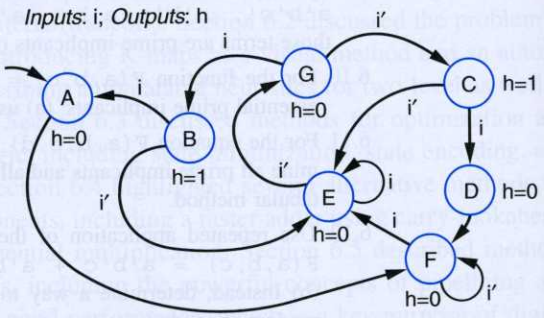

Figure 6.90 FSM example.

6.20 Compare the logic size (number of gate inputs) and the delay (number of gate-delays) of a minimum binary encoding, an output encoding (if it is possible; if not, indicate why not), and a one-hot encoding of the laser timer FSM in Figure 3.47.

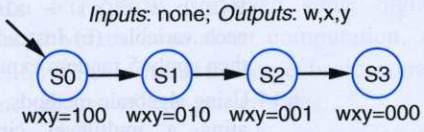

Figure 6.91 FSM example.

6.21 Convert the Moore FSM for the code detector circuit shown in Figure 3.58 to the nearest Mealy FSM equivalent.

6.22 Convert the Moore FSM in Figure 6.92 to the nearest Mealy FSM equivalent.

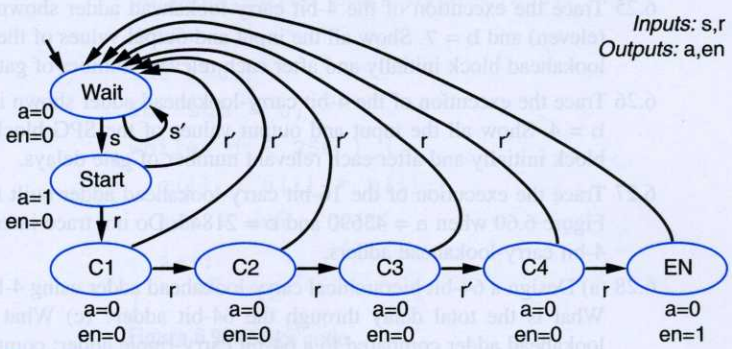

Figure 6.92 FSM example.

6.23 Convert the Mealy FSM in Figure 6.93 to the nearest Moore equivalent.

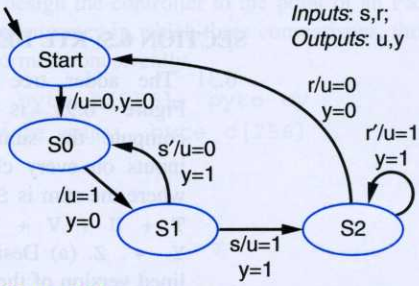

Figure 6.93 FSM example.

6.24 Convert the Mealy FSM in Figure 6.94 to the nearest Moore equivalent.

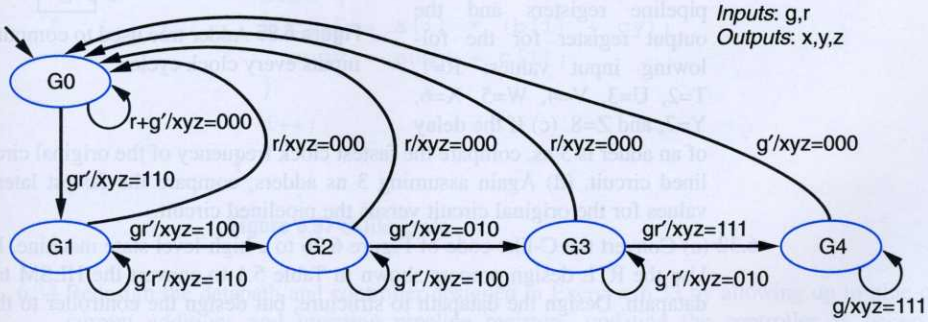

Figure 6.94 FSM example.

SECTION 6.4: DATAPATH COMPONENT TRADEOFFS

6.25 Trace the execution of the 4-bit carry-lookahead adder shown in Figure 6.57 when a = 11 (eleven) and b = 7. Show all the input and output values of the SPG blocks and of the carry-lookahead block initially and after each relevant number of gate delays.

6.26 Trace the execution of the 4-bit carry-lookahead adder shown in Figure 6.57 when a = 5 and b = 4. Show all the input and output values of the SPG blocks and of the carry-lookahead block initially and after each relevant number of gate delays.

6.27 Trace the execution of the 16-bit carry-lookahead adder built from 4-bit adders as shown in Figure 6.60 when a = 43690 and b = 21845. Do not trace internal behavior of the individual 4-bit carry-lookahead adders.

6.28 (a) Design a 64-bit hierarchical carry-lookahead adder using 4-bit carry-lookahead adders. (b) What is the total delay through the 64-bit adder? (c) What is the speedup of the carry-lookahead adder compared to a 64-bit carry-ripple adder; compute speedup as (slower time)/(faster time).

6.29 Design a 24-bit hierarchical carry-lookahead adder using 4-bit carry-lookahead adders.

6.30 Design a 16-bit carry-select adder using 4-bit carry-ripple adders.

SECTION 6.5: RTL DESIGN OPTIMIZATIONS AND TRADEOFFS

6.31 The adder tree shown in Figure 6.95 is used to compute the sum of eight inputs on every clock cycle, where the sum is S = R + T + U + V + W + X + Y + Z. (a) Design a pipelined version of the adder tree to maximize the frequency at which the clock input can operate. (b) Create a timing diagram of the pipelined tree circuit showing the values of pipeline registers and the output register for the following input values: R=1, T=2, U=3, V=4, W=5, X=6, Y=7, and Z=8. (c) If the delay of an adder is 3 ns, compare the fastest clock frequency of the original circuit versus the pipelined circuit. (d) Again assuming 3 ns adders, compare the fastest latency and throughput values for the original circuit versus the pipelined circuit.

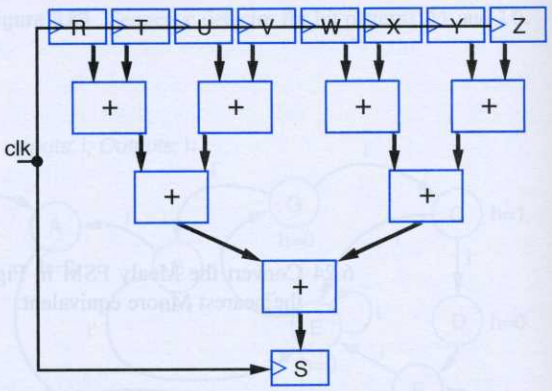

Figure 6.95 Adder tree used to compute the sum of eight inputs every clock cycle.

6.32 (a) Convert the C-like code of Figure 6.96 to a high-level state machine. Ignore overflow. (b) Use the RTL design process shown in Table 5.1 to convert the HLSM to a controller and a datapath. Design the datapath to structure, but design the controller to the point of an FSM only. (c) Redesign the datapath to allow for concurrency in which four multiplications and two additions can be performed concurrently. Assume memory ports can be introduced as needed. (d) Assuming a multiplier delay is 4 ns and an adder delay is 2 ns, list the fastest clock period, latency, and throughput for the original design and for the more concurrent design, assuming the critical path is in the datapath. (e) Introduce more multipliers or adders and pipeline registers as needed to further improve the speed of the design, and compare the clock period, throughput, and latency with the previous two designs.

```
Inputs: byte a[256], b[256]
Outputs: byte sum, byte c[256]
MULT:
int i=0;
int sum = 0;
while( i < 256 ) {
  c[i] = a[i] * b[i];
  sum = sum + c[i];
  i++;
}
```

Figure 6.96 C-like code.

6.33 (a)Convert the C-like code in Figure 6.97 to a high-level state machine. Ignore overflow. (b) Use the RTL design process shown in Table 5.1 to convert the HLSM to a controller and a datapath. Design the datapath to structure, but design the controller to the point of an FSM only. (c) Redesign your datapath to allow for concurrency in which three comparisons, three additions, and three multiplications can be performed concurrently.

```
Inputs: byte a[256], byte b[256], byte cy
Output: byte sumx, byte sumy, byte c[256]
MULT_OR_ADD:
int i=0;
int sumx = 0;
int sumy = 0;
while( i < 256 ) {
  if( a[i] > 128 ) {
    c[i] = a[i] * b[i];
    sumx = sumx + c[i];
  }
  else {
    c[i] = a[i] * (b[i] + cy);
    sumy = sumy + c[i];
  }
  i++;
}
```

Figure 6.97 C-like code.

6.34 Redesign the datapath and controller designed in Exercise 6.33 by allowing up to nine concurrent additions and inserting pipeline registers, updating the controller as necessary. Assuming a comparator has a delay of 4 ns, an adder has a delay of 3 ns, and a multiplier has a delay of 20 ns, how long will the circuit take to finish its computation?

6.35 Given the HLSM in Figure 6.98, create two different designs: one optimized for minimum circuit speed and the other optimized for minimum circuit size. Be sure to clearly indicate the component allocation, operator binding, and operator scheduling used to design the two circuits.

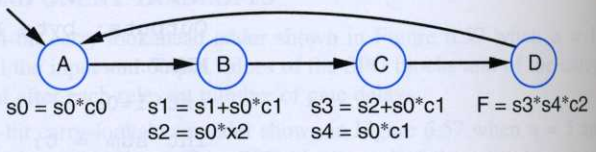

Figure 6.98 High-level state machine for Exercise 6.39.

SECTION 6.6: MORE ON OPTIMIZATIONS AND TRADEOFFS

6.36 Trace through the execution of the binary search algorithm when searching for the number 86 in the following sorted list of 15 numbers: 1, 10, 25, 62, 74, 75, 80, 84, 85, 86, 87, 100, 106, 111, 121. How many comparisons were required to find the number using the binary search and how many comparisons would have been required using a linear search?

6.37 Trace through the execution of the binary search algorithm when searching for the number 99 in the following list of 15 numbers: 1, 10, 25, 62, 74, 75, 80, 84, 85, 87, 99, 100, 106, 111, 121. How many comparisons were required to look for the number using the binary search and how many comparisons are required using a linear search?

6.38 Trace through the execution of the binary search algorithm when searching for the number 121 in the list of numbers from the previous example. How many comparisons were required to find the number using the binary search and how many comparisons are required using a linear search?

6.39 Using the list of 15 numbers from Exercise 6.37, how many numbers can be found faster using a linear search algorithm compared with the binary search algorithm?

6.40 Given the logic gate library in Figure 6.99, optimize the circuit in Figure 6.100 by reducing power consumption without increasing the circuit's delay.

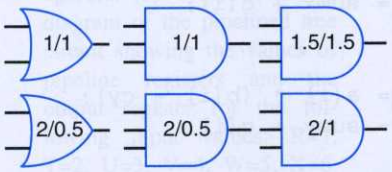

Figure 6.99 Logic gate library. 2/0.5 format means 2 ns delay/0.5 nw power.

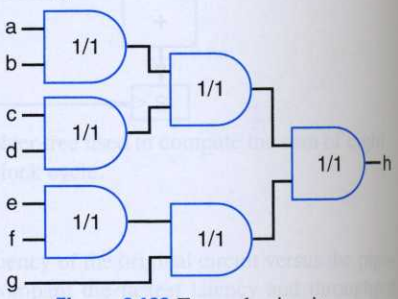

Figure 6.100 Example circuit.

6.41 Given the logic gates shown in Figure 6.99, optimize the circuit in Figure 6.101 by reducing power consumption without increasing the circuit's delay.

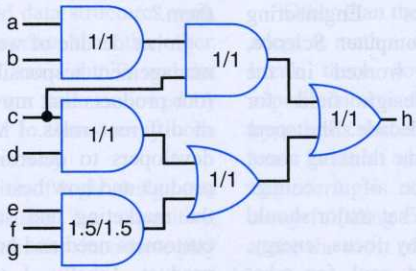

Figure 6.101 Example circuit.

6.42 Given the logic gates shown in Figure 6.99, optimize the circuit in Figure 6.102 by reducing power consumption without increasing the circuit's delay.

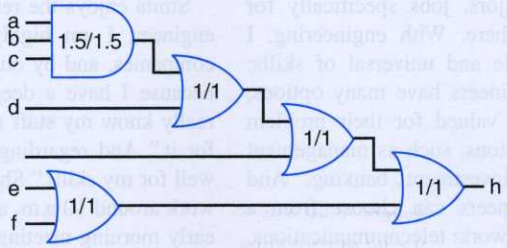

Figure 6.102 Example circuit.

6.43 Given the logic gates shown in Figure 6.99, optimize the circuit in Figure 6.103 by reducing power consumption without increasing the circuit's delay.

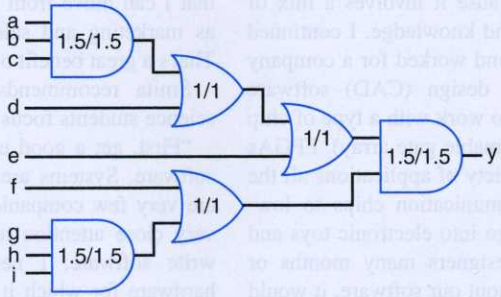

Figure 6.103 Example circuit.

▶ DESIGNER PROFILE

Smita has degrees in Electronics Engineering and in Computer Science, and has worked in the digital design field for nearly a decade. She spent a lot of time thinking about the choice of a college major. "What major should I invest my focus, energy, heart, and soul for what will be some of the most productive years of my life?" She chose engineering, for several reasons. "First, engineering is a career in itself—unlike some other majors, jobs specifically for engineering majors are out there. With engineering, I would learn the most valuable and universal of skills: problem solving. Second, engineers have many options, because engineers are highly valued for their problem solving skills by other professions, such as management consulting, marketing, and investment banking. And electrical and computer engineers can choose from a range of industries in which to work: telecommunications, image processing, medical devices, IC fabrication, and even banking. This was a phenomenal discovery for me!"

Smita continued her education by doing graduate studies in Computer Science, researching methods for automatically designing integrated circuits (IC) or chips—"a fascinating field because it involves a mix of hardware and software skills and knowledge. I continued in this profession after school and worked for a company that develops computer-aided design (CAD) software used by hardware designers who work with a type of chip called an FPGA (field programmable gate array). FPGAs can be used for an amazing variety of applications all the way from high-speed telecommunication chips to low-speed and low-cost chips that go into electronic toys and games. Our software saves designers many months or even years of time. In fact, without our software, it would be absolutely impossible for people to design most chips even if they had a decade or more to do it."

Smita (shown mountain climbing above) loves her work. "My work is intellectually stimulating, and I have an opportunity to innovate, create, and actually build something really useful." She also enjoys the people-aspect of her work. "I work in teams of dynamic people because most projects, hardware or software, are done in teams of 3–8 people these days. The people on my team are also my friends, and it's a lot of fun to work with them."

In her decade of work so far, Smita has taken on some management responsibilities. "As manager of one of the four products that my company develops, I play a variety of different roles. I work with my team of 7 software developers to determine what features to build in the product and how best to build those features. I work with the marketing and sales team to understand what the customers need and how best to message and position our product. Finally, I work with other groups that are involved in releasing a product — technical publications, application engineering, and product engineering. The diversity of my job makes it very interesting.

Smita enjoys the respect that engineers receive. "As an engineer, I am highly respected by customers, partner companies, and by our marketing and sales organizations because I have a deep understanding of our products. I really know my stuff since I built it, and I get recognized for it." And regarding the pay: "I get compensated very well for my skills." She also likes the lifestyle: "I get in to work around 10 a.m. and leave around 7 p.m. I don't have early morning meetings unlike the folks in marketing and sales, and I can work from home once a week or more often if I wish. This is also a great career for women—I can take time off and return to my job without much penalty when I have children. I can tailor my work hours as I need as my children are growing up. Lastly, I realize that I can move from engineering to other functions such as marketing and sales, but not the other way around! That's a great benefit of being an engineer—more options."

Smita recommends that engineering and computer science students focus on certain things while in college.

"First, get a good understanding of both hardware and software. Systems are highly integrated today, and there are very few companies that develop one without paying very close attention to the other. For instance, though I write software, I need to completely understand the hardware for which it will be used. My husband, on the other hand, designs telecommunication chips but works very closely with his software team, especially during the initial design stages when they decide what to implement in hardware versus software and how to design the hardware interface so that the software algorithms work efficiently.

"So, what do I mean by a good understanding of hardware and software? In software, I think it is most important to develop good software 'habits.' Treat your

▶ DESIGNER PROFILE (continued)

program like a well-landscaped garden—you want it to be beautiful and weed-free. Understand data structures well, and know when one is more appropriate than the other. Organize your code, be disciplined, cross the Ts and dot the Is, document diligently, have your code reviewed by friends, and finally, don't be afraid to throw away code and rewrite it if you discover a better way.

"In hardware, understand the basics of logic design and then make sure you also understand the capacitive, inductive, and resistive properties of circuits since these play a big role in designing the high-speed circuits of today.

"Other than these hardware and software skills, become adept at math and analysis. Learn to frame problems and break them down until you can solve them. Be experimental and try different tools and methods. Have a hypothesis and then go about proving or disproving it. If you haven't already, you will soon discover that engineering is not only fun, but also provides you with many fulfilling career opportunities—so stick with it and make the most of it!"