circuits that store bits,
emory is known as state.
blocks, including the SR
store multiple bits. The
es the loads of registers.
g the desired behavior of
ent FSMs, with an FSM
on 3.4 then described a
. Section 3.5 described
tup time, hold time, and
tastability problems. The
s, and described their use
d the problem of output
blem. Section 3.8 high-
FSM to describe the

desired circuit behavior
a several step process to
quential circuit begins by
following a several-step
ster and a combinational
ptually, the knowledge in
ever, many digital circuits
t inputs representing two
les, or FSMs would be if
e been developed specifi-
ign process—components

3.10 EXERCISES

An asterisk (*) indicates an especially challenging problem.

SECTION 3.2: STORING ONE BIT—FLIP-FLOPS

3.1 Compute the clock period for the following clock frequencies.
- (a) 50 kHz (early computers)
- (b) 300 MHz (Sony Playstation 2 processor)
- (c) 3.4 GHz (Intel Pentium 4 processor)
- (d) 10 GHz (PCs of the early 2010s)
- (e) 1 THz (1 terahertz) (PC of the future?)

3.2 Compute the clock period for the following clock frequencies.
- (a) 32.768 kHz
- (b) 100 MHz
- (c) 1.5 GHz
- (d) 2.4 GHz

3.3 Compute the clock frequency for the following clock periods.
- (a) 1 s
- (b) 1 ms
- (c) 20 ns
- (d) 1 ns
- (e) 1.5 ps

3.4 Compute the clock frequency for the following clock periods.
- (a) 500 ms
- (b) 400 ns
- (c) 4 ns
- (d) 20 ps

3.5 Trace the behavior of an SR latch for the following situation: Q, S, and R have been 0 for a long time, then S changes to 1 and stays 1 for a long time, then S changes back to 0. Using a timing diagram, show the values that appear on wires S, R, t, and Q. Assume logic gates have a tiny nonzero delay.

3.6 Repeat Exercise 3.5, but assume that S was changed to 1 just long enough for the signal to propagate through one logic gate, after which S was changed back to 0—in other words, S did not satisfy the hold time of the latch.

3.7 Trace the behavior of a level-sensitive SR latch (see Figure 3.16) for the input pattern in Figure 3.94. Assume S1, R1, and Q are initially 0. Complete the timing diagram, assuming logic gates have a tiny but nonzero delay.

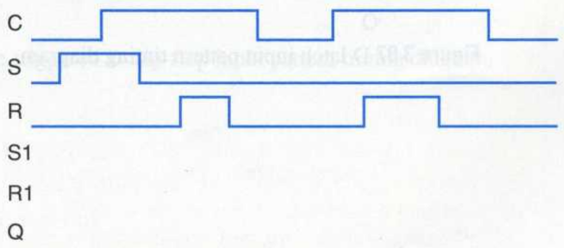


Figure 3.94 SR latch input pattern timing diagram.

3.8 Trace the behavior of a level-sensitive SR latch (see Figure 3.16) for the input pattern in Figure 3.95. Assume S1, R1, and Q are initially 0. Complete the timing diagram, assuming logic gates have a tiny but nonzero delay..

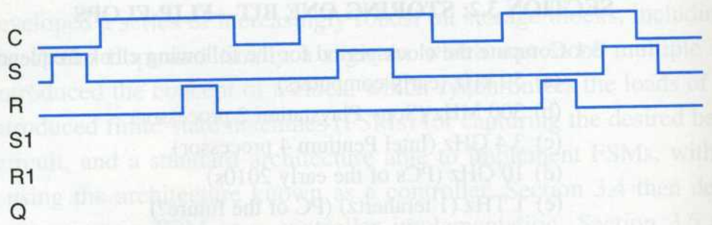


Figure 3.95 SR latch input pattern timing diagram.

3.9 Trace the behavior of a level-sensitive SR latch (see Figure 3.16) for the input pattern in Figure 3.96. Assume S1, R1, and Q are initially 0. Complete the timing diagram, assuming logic gates have a tiny but nonzero delay.

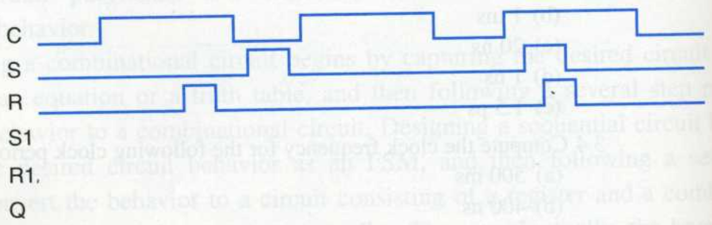


Figure 3.96 SR latch input pattern timing diagram.

3.10 Trace the behavior of a D latch (see Figure 3.19) for the input pattern in Figure 3.97. Assume Q is initially 0. Complete the timing diagram, assuming logic gates have a tiny but nonzero delay.

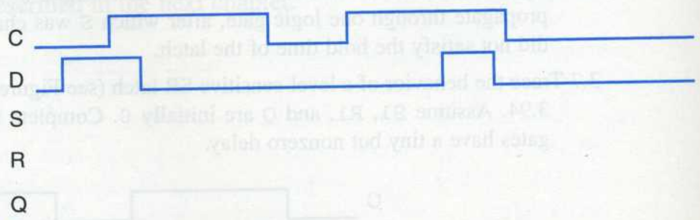


Figure 3.97 D latch input pattern timing diagram.

the input pattern in Figure
diagram, assuming logic

the input pattern in Figure
g diagram, assuming logic

tern in Figure 3.97. Assume
ates have a tiny but nonzero

3.11 Trace the behavior of a D latch (see Figure 3.19) for the input pattern in Figure 3.98. Assume Q is initially 0. Complete the timing diagram, assuming logic gates have a tiny but nonzero delay.

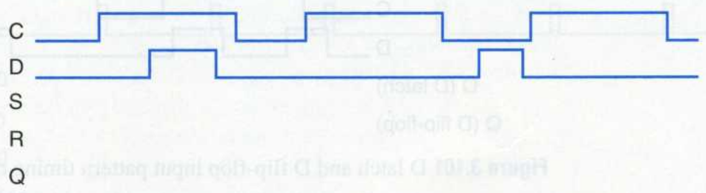


Figure 3.98 D latch input pattern timing diagram.

3.12 Trace the behavior of an edge-triggered D flip-flop using a master-servant design (see Figure 3.25) for the input pattern in Figure 3.99. Assume each internal latch initially stores a 0. Complete the timing diagram, assuming logic gates have a tiny but nonzero delay.

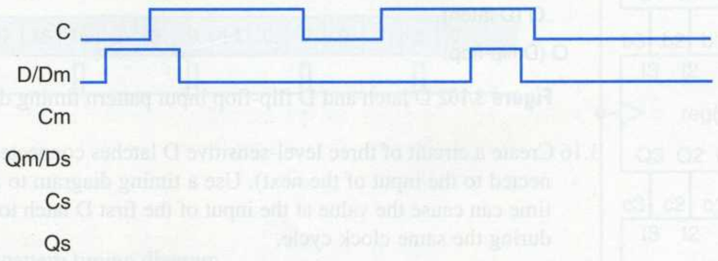


Figure 3.99 Edge-triggered D flip-flop input pattern timing diagram.

3.13 Trace the behavior of an edge-triggered D flip-flop using the master-servant design (see Figure 3.25) for the input pattern in Figure 3.100. Assume each internal latch initially stores a 0. Complete the timing diagram, assuming logic gates have a tiny but nonzero delay.

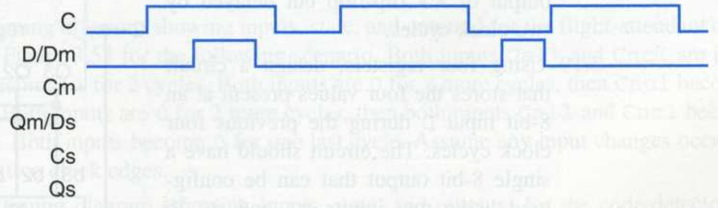


Figure 3.100 Edge-triggered D flip-flop input pattern timing diagram.

3.14 Compare the behavior of D latch and D flip-flop devices by completing the timing diagram in Figure 3.101. Assume each device initially stores a 0. Provide a brief explanation of the behavior of each device.

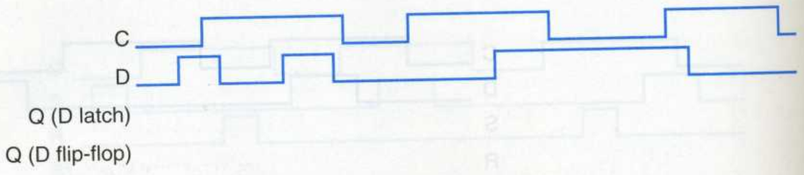


Figure 3.101 D latch and D flip-flop input pattern timing diagram.

3.15 Compare the behavior of D latch and D flip-flop devices by completing the timing diagram in Figure 3.102. Assume each device initially stores a 0. Provide a brief explanation of the behavior of each device.

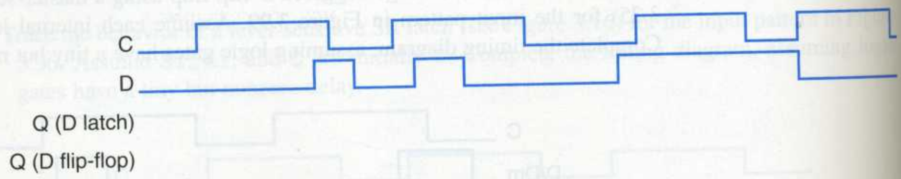


Figure 3.102 D latch and D flip-flop input pattern timing diagram.

3.16 Create a circuit of three level-sensitive D latches connected in series (the output of one is connected to the input of the next). Use a timing diagram to show how a clock with a long high-time can cause the value at the input of the first D latch to trickle through more than one latch during the same clock cycle.

3.17 Repeat Exercise 3.16 using edge-triggered D flip-flops, and use a timing diagram to show how the input of the first D flip-flop does not trickle through to the next flip-flop, no matter how long the clock signal is high.

3.18 A circuit has an input `X` that is connected to the input of a D flip-flop. Using additional D flip-flops, complete the circuit so that an output `Y` equals the output of `X`'s flip-flop but delayed by two clock cycles.

3.19 Using four registers, design a circuit that stores the four values present at an 8-bit input `D` during the previous four clock cycles. The circuit should have a single 8-bit output that can be configured using two inputs `s1` and `s0` to output any one of the four registers. (Hint: use an 8-bit 4x1 mux.)

3.20 Consider three 4-bit registers connected as in Figure 3.103. Assume the initial values in the registers are unknown. Trace the behavior of the registers by completing the timing diagram of Figure 3.104.

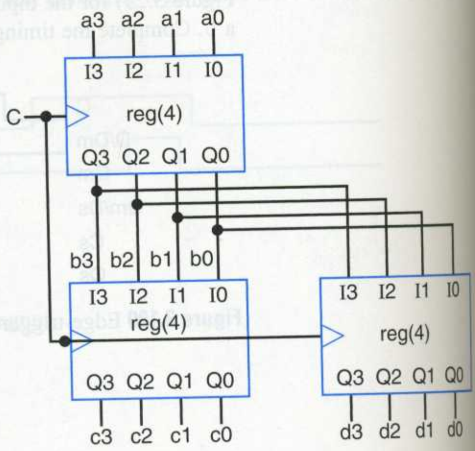
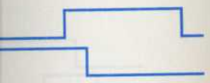
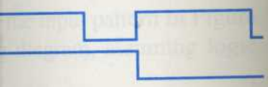


Figure 3.103 Register configuration.

eting the timing diagram in
a brief explanation of the

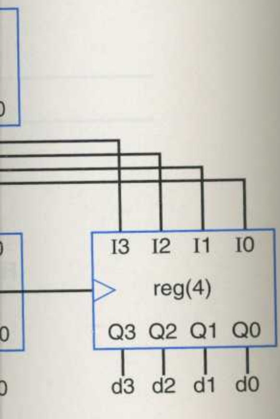


eting the timing diagram in
a brief explanation of the



es (the output of one is con-
w a clock with a long high-
through more than one latch

a timing diagram to show
the next flip-flop, no matter



configuration.

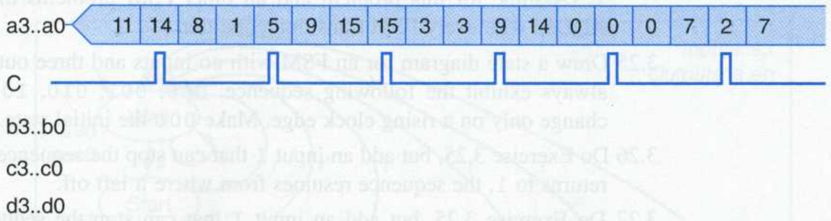


Figure 3.104 4-bit register input pattern timing diagram.

3.21 Consider three 4-bit registers connected as in Figure 3.105. Assume the initial values in the registers are unknown. Trace the behavior of the registers by completing the timing diagram of Figure 3.106.

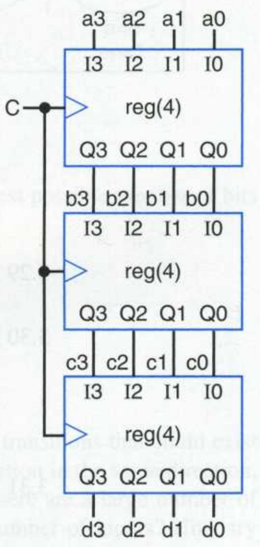


Figure 3.105 Register configuration.

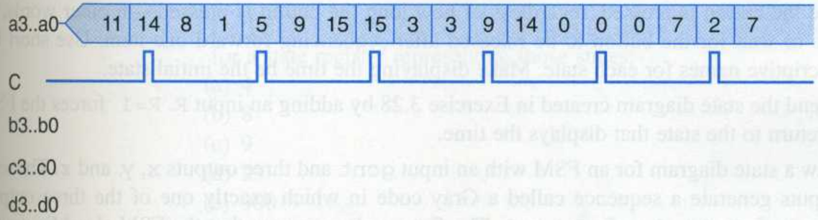


Figure 3.106 4-bit register input pattern timing diagram.

SECTION 3.3: FINITE-STATE MACHINES (FSMS)

3.22 Draw a timing diagram (showing inputs, state, and outputs) for the flight-attendant call-button FSM of Figure 3.53 for the following scenario. Both inputs `Call` and `Cncl` are initially `0`. `Call` becomes `1` for 2 cycles. Both inputs are `0` for 2 more cycles, then `Cncl` becomes `1` for 1 cycle. Both inputs are `0` for 2 more cycles, then both inputs `Call` and `Cncl` become `1` for 2 cycles. Both inputs become `0` for one last cycle. Assume any input changes occur halfway between two clock edges.

3.23 Draw a timing diagram (showing inputs, state, and outputs) for the code-detector FSM of Figure 3.58 for the following scenario. Recall that when a button (or buttons) is pressed, a becomes `1` for exactly 1 clock cycle, no matter how long the button (or buttons) is pressed. Initially no button is pressed. The user then presses buttons in the following order: red, green, blue, red. Noticing the final state of the system, can you suggest an improvement to the system to better handle such incorrect code sequences?

3.24 Draw a state diagram for an FSM that has an input X and an output Y. Whenever X changes from 0 to 1, Y should become 1 for two clock cycles and then return to 0—even if X is still 1. (Assume for this problem and all other FSM problems that an implicit rising clock is ANDed with every FSM transition condition.)

3.25 Draw a state diagram for an FSM with no inputs and three outputs x, y, and z. xyz should always exhibit the following sequence: 000, 001, 010, 100, repeat. The output should change only on a rising clock edge. Make 000 the initial state.

3.26 Do Exercise 3.25, but add an input I that can stop the sequence when set to 0. When input I returns to 1, the sequence resumes from where it left off.

3.27 Do Exercise 3.25, but add an input I that can stop the sequence when set to 0. When I returns to 1, the sequence starts from 000 again.

3.28 A wristwatch display can show one of four items: the time, the alarm, the stopwatch, or the date, controlled by two signals s1 and s0 (00 displays the time, 01 the alarm, 10 the stopwatch, and 11 the date—assume s1 and s0 control an *N*-bit mux that passes through the appropriate register). Pressing a button B (which sets B = 1) sequences the display to the next item. For example, if the presently displayed item is the date, the next item is the current time. Create a state diagram for an FSM describing this sequencing behavior, having an input bit B, and two output bits s1 and s0. Be sure to only sequence forward by one item each time the button is pressed, regardless of how long the button is pressed—in other words, be sure to wait for the button to be released after sequencing forward one item. Use short but descriptive names for each state. Make displaying the time be the initial state.

3.29 Extend the state diagram created in Exercise 3.28 by adding an input R. R=1 forces the FSM to return to the state that displays the time.

3.30 Draw a state diagram for an FSM with an input gcnt and three outputs x, y, and z. The xyz outputs generate a sequence called a Gray code in which exactly one of the three outputs changes from 0 to 1 or from 1 to 0. The Gray code sequence that the FSM should output is 000, 010, 011, 001, 101, 111, 110, 100, repeat. The output should change only on a rising clock edge when the input gcnt = 1. Make the initial state 000.

3.31 Trace through the execution of the FSM created in Exercise 3.30 by completing the timing diagram in Figure 3.107, where C is the clock input. Assume the initial state is the state that sets xyz to 000.

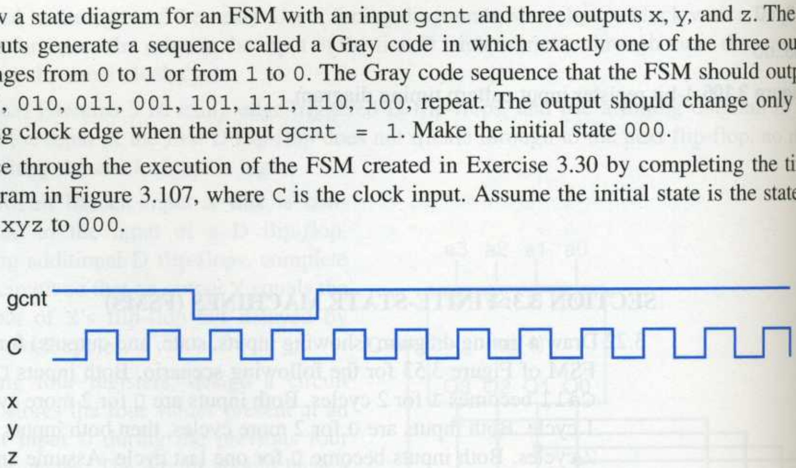


Figure 3.107 FSM input pattern timing diagram.

t Y. Whenever X changes
rn to 0—even if X is still
n implicit rising clock is

x, y, and z. xyz should
epeat. The output should

en set to 0. When input I

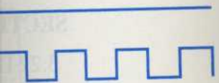
when set to 0. When I

arm, the stopwatch, or the
01 the alarm, 10 the stop-
ux that passes through the
nces the display to the next
e next item is the current
g behavior, having an input
forward by one item each
ressed—in other words, be
rd one item. Use short but
initial state.

ut R. R=1 forces the FSM

tputs x, y, and z. The xyz
ly one of the three outputs
t the FSM should output is
t should change only on a
ate 000.

0 by completing the timing
initial state is the state that



3.32 Draw a timing diagram for the FSM in Figure 3.108 with the FSM starting in state *Wait*. Choose input values such that the FSM reaches state *EN*, and returns to *Wait*..

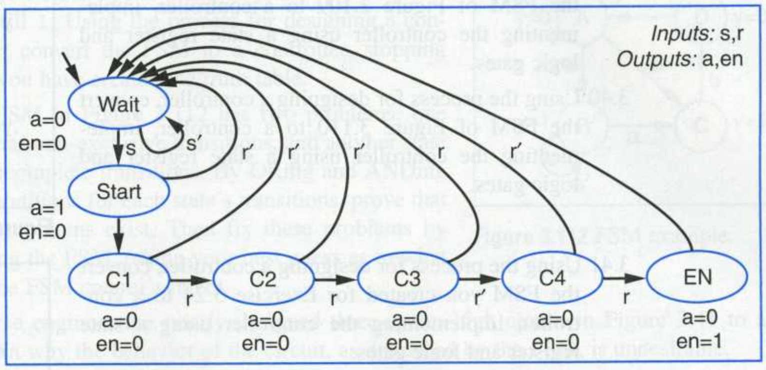


Figure 3.108 FSM.

3.33 For FSMs with the following numbers of states, indicate the smallest possible number of bits for a state register representing those states:

(a) 4
(b) 8
(c) 9
(d) 23
(e) 900

3.34 How many possible states can be represented by a 16-bit register?

3.35 If an FSM has *N* states, what is the maximum number of possible transitions that could exist in the FSM? Assume that no pair of states has more than one transition in the same direction, and that no state has a transition point back to itself. Assuming there are a large number of inputs, meaning the number of transitions is not limited by the number of inputs? Hint: try for small *N*, and then generalize.

3.36 *Assuming one input and one output, how many possible four-state FSMs exist?

3.37 *Suppose you are given two FSMs that execute concurrently. Describe an approach for merging those two FSMs into a single FSM with identical functionality as the two separate FSMs, and provide an example. If the first FSM has *N* states and the second has *M* states, how many states will the merged FSM have?

3.38 *Sometimes dividing a large FSM into two smaller FSMs results in simpler circuitry. Divide the FSM shown in Figure 3.111 into two FSMs, one containing *G0–G3*, the other containing *G4–G7*. You may add additional states, transitions, and inputs or outputs between the two FSMs, as required. Hint: you will need to introduce signals between the FSMs for one FSM to tell the other FSM to go to some state.

SECTION 3.4: CONTROLLER DESIGN

3.39 Using the process for designing a controller, convert the FSM of Figure 3.109 to a controller, implementing the controller using a state register and logic gates.

3.40 Using the process for designing a controller, convert the FSM of Figure 3.110 to a controller, implementing the controller using a state register and logic gates.

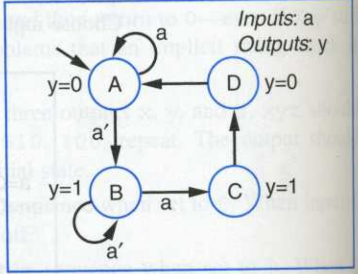


Figure 3.109 FSM example.

3.41 Using the process for designing a controller, convert the FSM you created for Exercise 3.24 to a controller, implementing the controller using a state register and logic gates.

3.42 Using the process for designing a controller, convert the FSM you created for Exercise 3.28 to a controller, implementing the controller using a state register and logic gates.

3.43 Using the process for designing a controller, convert the FSM you created for Exercise 3.30 to a controller, implementing the controller using a state register and logic gates.

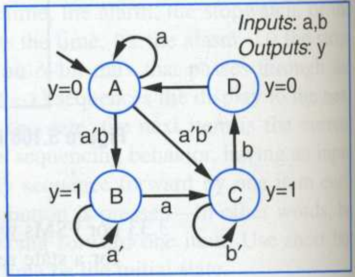


Figure 3.110 FSM example.

3.44 Using the process for designing a controller, convert the FSM in Figure 3.111 to a controller, stopping once you have created the truth table. Note: your truth table will be quite large, having 32 rows—you might therefore want to use a computer tool, like a word processor or spreadsheet, to draw the table.

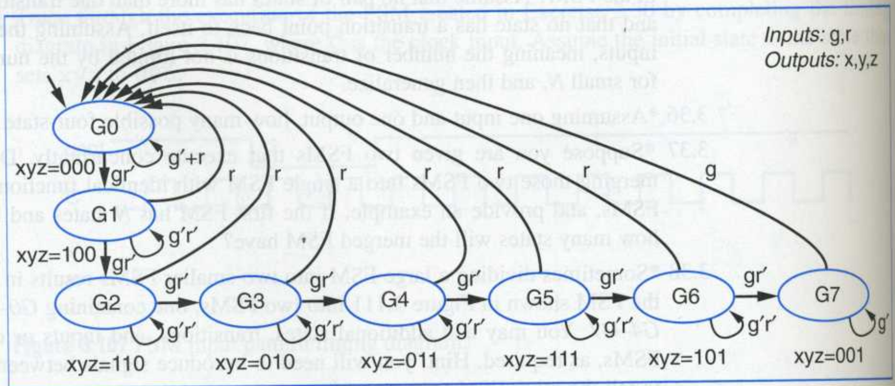
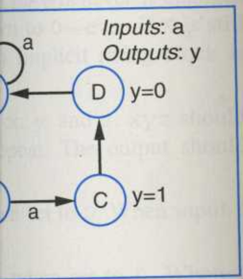
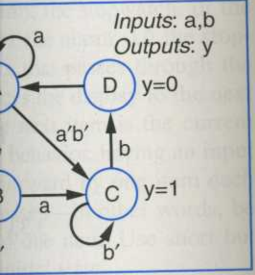


Figure 3.111 FSM example.

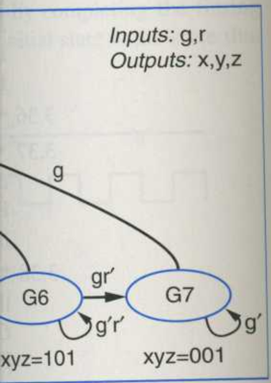


FSM example.



FSM example.

igure 3.111 to a controller,
table will be quite large,
l, like a word processor or



3.45 Create an FSM that has an input X and an output Y. Whenever X changes from 0 to 1, Y should become 1 for five clock cycles and then return to 0—even if X is still 1. Using the process for designing a controller, convert the FSM to a controller, stopping once you have created the truth table.

3.46 The FSM in Figure 3.112 has two problems: one state has non-exclusive transitions, and another state has incomplete transitions. By ORing and ANDing the conditions for each state's transitions, prove that these problems exist. Then fix these problems by refining the FSM, taking your best guess as to what was the FSM creator's intent.

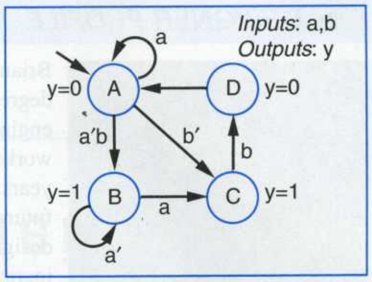


Figure 3.112 FSM example.

3.47 Reverse engineer the poorly designed three-cycles-high circuit in Figure 3.41 to an FSM. Explain why the behavior of the circuit, as described by the FSM, is undesirable.

3.48 Reverse engineer the behavior of the sequential circuit shown in Figure 3.113.

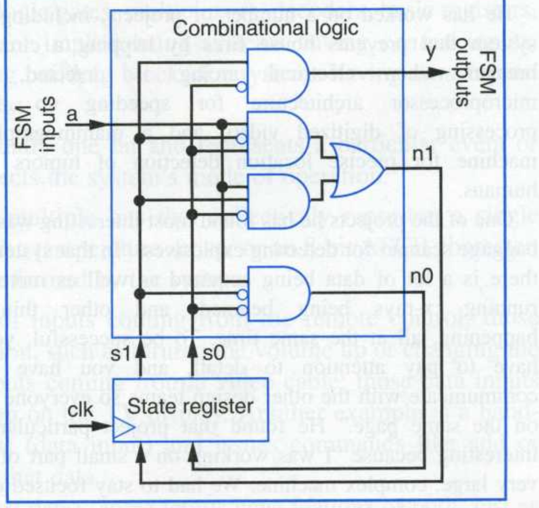


Figure 3.113 A sequential circuit to be reverse engineered.

SECTION 3.5: MORE ON FLIP-FLOPS AND CONTROLLERS

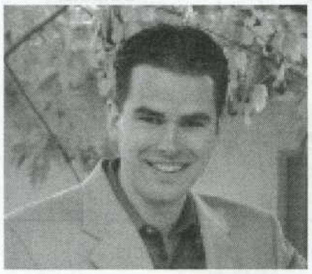
3.49 Use a timing diagram to illustrate how metastability can yield incorrect output for the secure car key controller of Figure 3.69. Use a second timing diagram to show how the synchronizer flip-flop introduced in Figure 3.84 may reduce the likelihood of such incorrect output.

3.50 Design a controller with a 4-bit state register that gets synchronously initialized to state 1010 when an input *reset* is set to 1.

3.51 Redraw the laser-timer controller timing diagram of Figure 3.63 for the case of the output being registered as in Figure 3.88.

3.52 Draw a timing diagram for three clock cycles of the sequence generator controller of Figure 3.68, assuming that AND gates have a delay of 2 ns and inverters (including inversion bubbles) have a delay of 1 ns. The timing diagram should show the incorrect outputs that appear temporarily due to glitching. Then introduce registered outputs to the controller using flip-flops at the outputs, and show a new timing diagram, which should no longer have glitches (but the output may be shifted in time).

▶ DESIGNER PROFILE



Brian got his bachelor's degree in electrical engineering and then worked for several years. Realizing the future demand for digital design targeting an increasingly popular type of digital chip known as FPGAs (see Chapter 7), he returned to school to obtain a master's degree in electrical engineering, with a thesis topic targeting digital design for FPGAs. He has been employed at two different companies, and is now working as an independent digital design consultant.

He has worked on a number of projects, including a system that prevents house fires by tripping a circuit breaker when electrical arcing is detected, a microprocessor architecture for speeding up the processing of digitized video, and a mammography machine for precise location detection of tumors in humans.

One of the projects he has found most interesting was a baggage scanner for detecting explosives. "In that system, there is a lot of data being acquired as well as motors running, x-rays being beamed, and other things happening, all at the same time. To be successful, you have to pay attention to detail, and you have to communicate with the other design teams so everyone is on the same page." He found that project particularly interesting because "I was working on a small part of a very large, complex machine. We had to stay focused on our part of the design, while at the same time being mindful of how all the parts were going to fit together in the end." Thus, being able to work alone as well as in large groups was important, requiring good communication and team skills. And being able to understand not only a part of the system, but also important aspects of the other parts was also necessary, requiring knowledge of diverse topics.

Brian is now an independent digital design consultant, something that many electrical engineers, computer engineers, and computer scientists choose to do after getting experience in their field. "I like the flexibility that being a consultant offers. On the plus side, I get to work on a wide variety of projects. The drawback is that sometimes I only get to work on a small part of a project, rather than seeing a product through from start to finish. And of course being an independent consultant means there's less stability than a regular position at a company, but I don't mind that."

Brian has taken advantage of the flexibility provided by consulting by taking a part-time job teaching an undergraduate digital design course and an embedded systems course at a university. "I really enjoy teaching, and I have learned a lot through teaching. And I enjoy introducing students to the field of embedded systems."

Asked what he likes most about the field of digital design, he says, "I like building products that make people's lives easier, or safer, or more fun. That's satisfying."

Asked to give advice to students, he says that one important thing is "to ask questions. Don't be afraid of looking dumb when you ask questions at a new job. People don't expect you to know everything, but they do expect you to ask questions when you are unsure. Besides, asking questions is an important part of learning."