11.1 Introduction to State Machines

Now that Chapter 10 has introduced us to memory cells, we can begin designing circuits that rely on stored values. With a single latch, we can store a one or a zero. By combining latches, we can store larger binary numbers.

Our first design application using these latches will be ***state machines***. A state machine is a digital circuit that relies not only on the circuit's inputs, but also the current state of the system to determine the proper output. For example, assume an elevator is stopped on the eighth floor and someone from the fourth floor presses the elevator call button. The elevator needs to decide whether to go up or down. As long as it remembers its current state, i.e., that it is on the eighth floor, it will know that it needs to go down to access the fourth floor.

To remember the current state of a system, memory will have to be added to our circuits. This memory will act as additional inputs to the circuitry used to create the outputs.

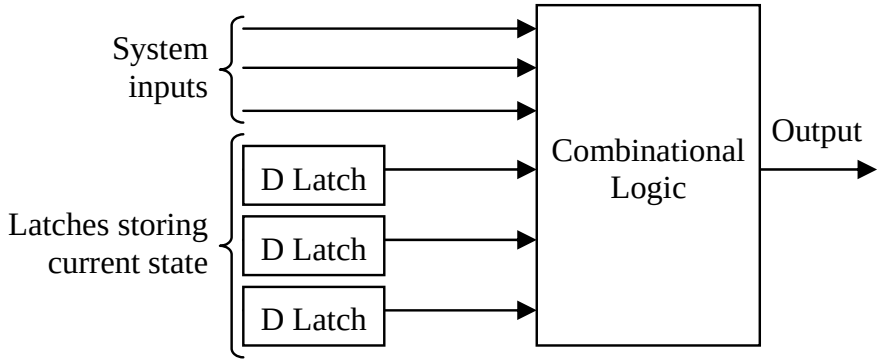


Figure 11-1 Adding Memory to a Digital Logic Circuit

11.1.1 States

So what is a state? A state defines the current condition of a system. It was suggested at the end of Chapter 10 that a traffic signal system is a state machine. The most basic traffic signal controls an intersection with two directions, North-South and East-West for example. There are

certain combinations of lights (on or off) that describe the intersection's condition. These are the system's states.

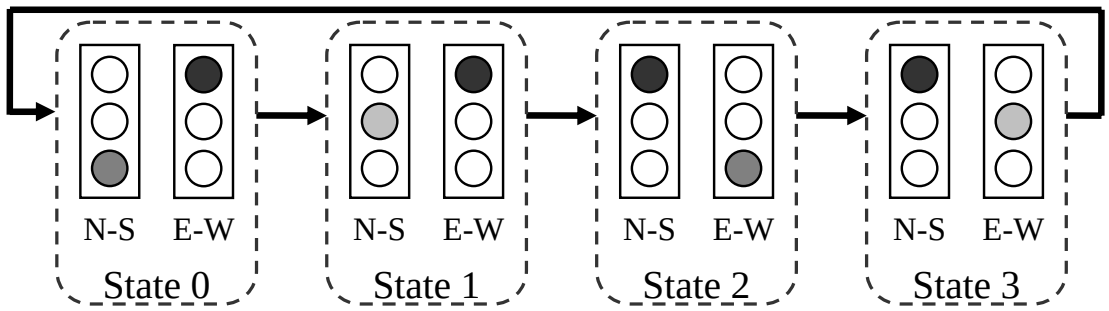


Figure 11-2 States of a Traffic Signal System

A state machine might also be as simple as a light bulb. The light bulb can have two states: on and off. The light switch moves the condition of the bulb from one state to another.

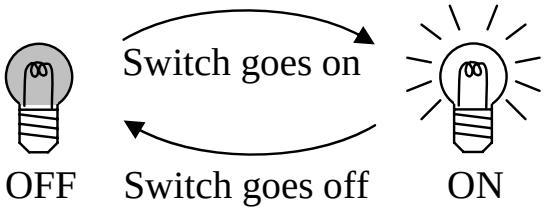


Figure 11-3 States of a Light Bulb

11.1.2 State Diagrams

We will begin our design of state machines by introducing the primary design tool: the ***state diagram***. A state diagram models a state machine by using circles to represent each of the possible states and arrows to represent all of the possible transitions between the states. For example, Figure 11-4 presents the state diagram for the light bulb state machine shown in Figure 11-3.

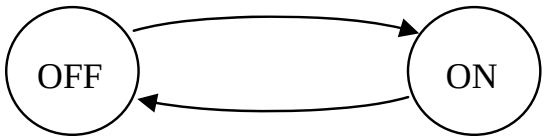


Figure 11-4 State Diagram for Light Bulb State Machine

This state diagram is incomplete. For example, what triggers a change from one state to the other? Even though the words ON and OFF mean something to you, they don't mean much to a computer. We need to include the boolean output associated with each of the states. Figure 11-5 presents the completed state diagram.

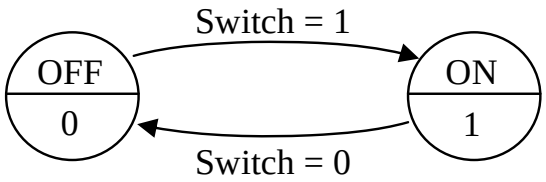


Figure 11-5 Complete State Diagram for Light Bulb State Machine

The upper half of each circle indicates the name of the state. The lower half indicates the binary output associated with that state. In the case of the light bulb state machine, a zero is output while we are in the OFF state and a one is output while we are in the ON state. The arrows along with the input value say that when we are in state OFF and the switch input goes to a 1, move to state ON. When we are in state ON and the switch input goes to a 0, move to state OFF.

Before creating a state diagram, we must define the parameters of the system. We begin with the inputs and outputs. The inputs are vital to the design of the state diagram as their values will be used to dictate state changes. As for the outputs, their values will be determined for each state as we create them.

It is also important to have an idea of what will define our states. For example, what happens to the states of our traffic signal system if we add a crosswalk signal? It turns out that the number of states we have will increase because there will be a difference between the state when the crosswalk indicator says "Walk" and when it flashes "Don't Walk" just before the traffic with the green light gets its yellow.

Here we will introduce an example to illustrate the use of state diagrams. Chapter 10 presented a simple counter circuit that incremented a binary value each time a pulse was detected. What if we wanted to have the option to decrement too? Let's design a state machine that stores a binary value and has as its input a control that determines whether we are incrementing that binary value or decrementing it when a pulse is received.

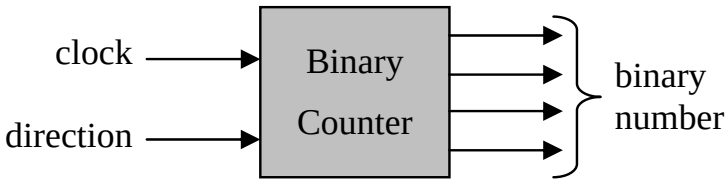


Figure 11-6 Block Diagram of an Up-Down Binary Counter

One of the inputs to this counter is a *clock.* Many state machines have a clock. It is used to drive the system from one state to the next. To do this, it is connected to the clock input of the latches. When a pulse is received, the next state is stored in the latches where it becomes the current state.

The other input to our system is *direction*. The *direction* signal is a binary input telling the system whether the stored value is to be incremented or decremented. Let's make a design decision now and say that when *direction* equals 0 we will be decrementing and when *direction* equals 1 we will be incrementing.

As for the states of our counter, the current state at any time can be represented with the binary number it is outputting. Therefore, the binary number is the state. For example, if the output from a three-bit counter is the binary number 110_2 and a pulse is detected on the *clock* input, the state of the system should change to state 101_2 if *direction* = 0 or state 111_2 if *direction* = 1.

We now have the characteristics of our counter specified well enough for us to begin creating the state diagram for a three-bit up-down counter. This counter has an output that is a three-bit number. Every time a clock pulse occurs, the counter will change state to either increment or decrement the output depending on the value of *direction*. For example, if *direction* equals one, then each clock pulse will increment the output through the sequence 000, 001, 010, 011, 100, 101, 110, 111, 000, 001, etc. If *direction* equals zero, then the output will decrement once for each clock pulse, i.e., 000, 111, 110, 101, 100, 011, 010, 001, 000, 111, 001, etc.

Figure 11-7 presents the state diagram for our system using the binary value to identify the states and the letter D to represent the input *direction*.

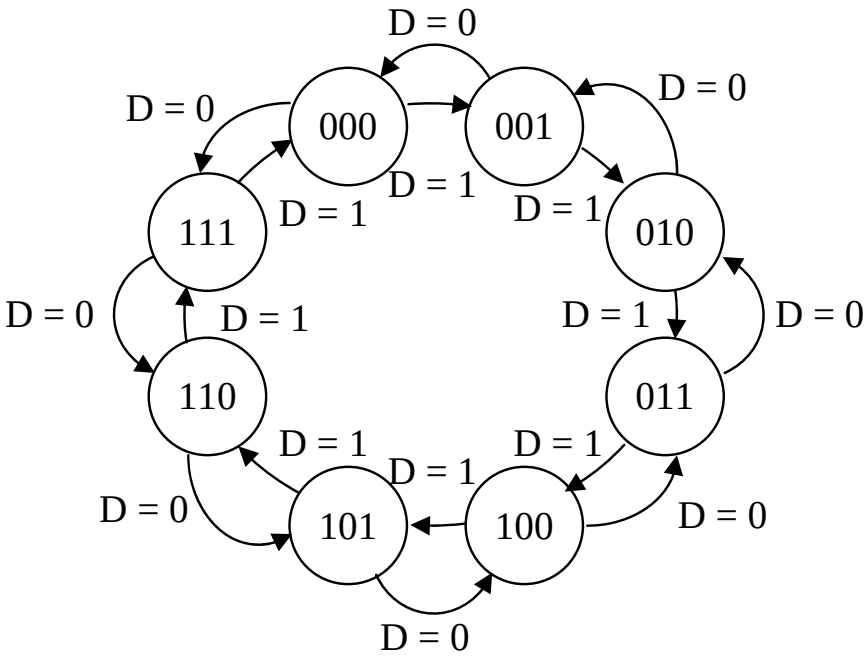


Figure 11-7 State Diagram for a 3-Bit Up-Down Binary Counter

In Figure 11-7, the arrows going clockwise around the inside of the diagram represent the progression through the states at each clock pulse when *direction* equals 1. Notice that each pulse from *clock* should take us to the next highest three-bit value. The arrows going counter-clockwise around the outside of the diagram represent the progression through the states at each clock pulse when *direction* equals zero.

There is an additional detail that must be represented with a state diagram. When a system first powers up, it should be initialized to a reset state. We need to indicate on the diagram which state is defined as the initial state. For example, the up-down counter may be initialized to the state 000_2 when it is first powered up. The state diagram represents this by drawing an arrow to the initial state with the word "reset" printed next to it. A portion of Figure 11-7 is reproduced in Figure 11-8 showing the reset condition.

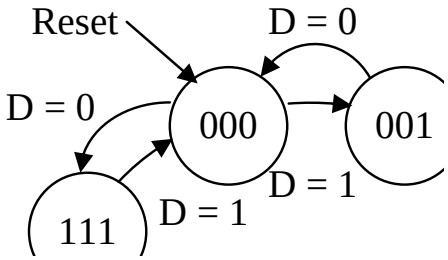


Figure 11-8 Sample of a Reset Indication in a State Diagram

11.1.3 Errors in State Diagrams

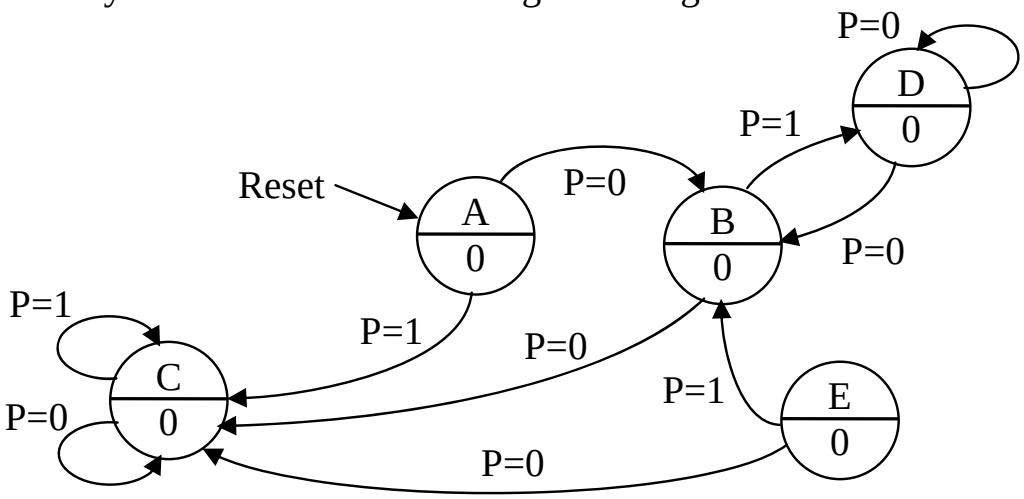
A state diagram must fully describe the operation of a state machine. It is for this reason that we need to watch for any missing or redundant information. Possible errors include the following situations.

- Any state other than an initial state that has no transitions going ***into*** it should be removed since it is impossible to reach that state.
- For a system with n inputs, there should be exactly 2^n transitions coming ***out*** of every state, one for each pattern of ones and zeros for the n inputs. Some transitions may come back to the current state, but every input must be accounted for. Missing transitions should be added while duplicates should be removed.

The following example shows how some of these errors might appear.

Example

Identify the errors in the following state diagram.



Solution

Error 1 – There is no way to get to state E. It should be removed. Although state A has no transitions to it, it is not a problem because it is the initial state.

Error 2 – The transition from state D for P=0 is defined twice while the transition for P=1 is never defined.

11.1.4 Basic Circuit Organization

From the state diagram, we can begin implementing the state machine. Figure 11-1 only revealed a portion of the organization of a

state machine. Figure 11-9 presents a more accurate diagram of the state machine circuitry. The current state is contained in the latches while a block of logic is used to determine the next state from the current state and the external inputs. A second block of logic determines the system output from the current state.

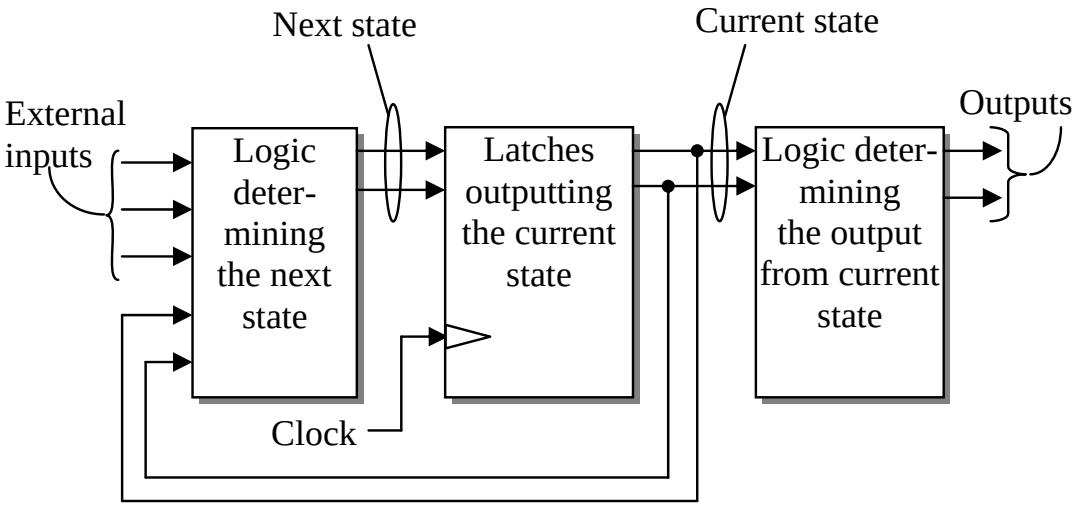


Figure 11-9 Block Diagram of a State Machine

There is a step-by-step process we will present here to design each block of logic. Although the three main pieces are connected, the design of the state machine involves three rather independent designs.

Let's begin with the latches, the center block of the block diagram of Figure 11-9. This component of the system consists only of one or more D latches, the combined outputs of which represent the current state of the state machine. The inputs to the latches represent what the next state would be if the clock were to pulse at that particular moment.

The number of latches in this portion of the circuit is based on the number of states the system can have. If, for example, a state diagram showed ten states for a system, then we would have to have enough latches so that their outputs, Q, could represent at least ten different patterns of ones and zeros. By numbering the ten states in binary, 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, we see that we will need at least four latches, the number of digits it takes to represent the highest value, 1001.

The latches are labeled S_n where n represents the bit position of the number stored in that latch. For example, if a set of four latches stored the binary value 1101_2 indicating that the state machine was currently

in state $1101_2 = 13_{10}$, then latch S_0 contains a 1, latch S_1 contains a 0, latch S_2 contains a 1, and latch S_3 contains a 1.

The D inputs serve to hold the binary value of the next state that the latches, and hence the state machine, will be set to. When a clock pulse occurs, the next state is stored in the latches making it the current state.

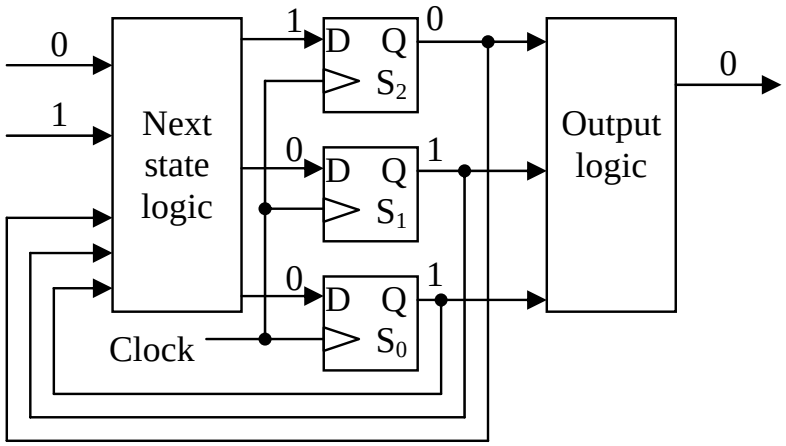
The leftmost block in the block diagram of Figure 11-9 represents the digital logic used to determine the next state of the system. It determines this from the inputs to the system and the current state of the system. We will use the state diagram to construct a truth table describing the operation of this block of logic.

The rightmost block in the block diagram of Figure 11-9 represents the digital logic used to determine the output of the system based on its current state. The state diagrams presented here have an output associated with each state. From this information, a truth table is developed which can be used to determine the output logic.

Example

The block diagram below represents a state machine. Answer the following questions based on the state machine's components and the digital values present on each of the connections.

- What is the maximum number of states this system could have?
- How many rows are in the truth table defining the output?
- How many rows are in the truth table defining the next state?
- What is the current state of this system?
- If the clock were to pulse right now, what would the next state be?



Solution

What is the maximum number of states this system could have? Since the system has 3 latches, then the numbers 000_2, 001_2, 010_2, 011_2, 100_2, 101_2, 110_2, and 111_2 can be stored. Therefore, this state machine can have up to **eight states**.

How many rows are in the truth table defining the output? Since the output is based on the current state which is represented by the latches, and since there are three latches, the logic circuit for the output has three inputs. With three inputs, there are $2^3 = 8$ possible patterns of ones and zeros into the circuit, and hence, **8 rows** in the truth table.

How many rows are in the truth table defining the next state? Since the next state of the state machine, i.e., the value on the input lines to the latches, depends on the current state fed back into the next state logic and the system inputs, then there are five inputs that determine the next state. Therefore, the inputs to the next state logic have $2^5 = 32$ possible patterns of ones and zeros. This means that the next state logic truth table has **32 rows**.

What is the current state of this system? The current state equals the binary value stored in the latches. Remembering that S_0 is the LSB while S_2 is the MSB, this means that the current state is $\mathbf{011_2 = 3_{10}}$.

If the clock were to pulse right now, what would the next state be? The next state is the binary value that is present at the D inputs to the latches. Once again, S_0 is the LSB and S_2 is the MSB. Therefore, the next state is $\mathbf{100_2 = 4_{10}}$.

11.2 State Machine Design Process

This section presents the state machine design process by taking an example through each of the design steps. The example we will be using is a push button circuit used to control a light bulb. When the button is pressed, the light bulb turns on. When the button is released, it stays on. Pushing the button a second time turns off the bulb, and the bulb stays off when the button is released.

First, let's define the system. It has a single input, i.e., the button. We will label this input 'B' and assume that when B = 0 the button is released and when B = 1 the button is pressed. The system has a single output, the light bulb. We will label this output 'L' and assume that L = 0 turns off the light and L = 1 turns on the light.

Now let's design the state diagram. We begin by assuming that the reset state has the light bulb off (L = 0) with the user's finger off of the button. Figure 11-10 presents this initial state labeled as state 0.

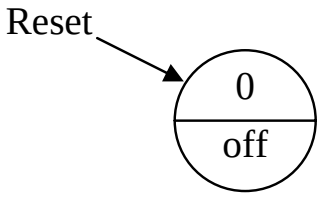


Figure 11-10 Initial State of the Push Button Light Control

The fact that we selected an initial state with the light bulb off might be clear, but it might not be clear why we added the condition that the user's finger is not touching the button. As we go through the design, we will see how the transitions between states depend on whether the button is currently pressed or released. This means that the condition of the button directly affects the state of the system.

So where do we go from this initial state? Well, when a clock pulse occurs, the decision of which state to go to from state 0 depends on the inputs to the system, namely whether the button is pressed (B=1) or released (B=0). Since the button has two possible conditions, then there will be two possible transitions out of state 0. Figure 11-11 shows how these transitions exit state 0 as arrows.

Each of these transitions must pass to a state, so the next step is to determine which state each transition goes to. To do this, we either need to create a new state or have the transition return to state 0.

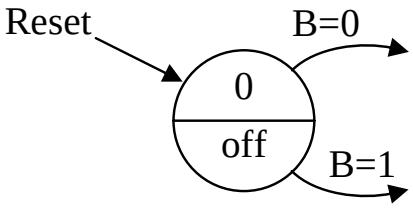


Figure 11-11 Transitions from State 0 of Push Button Circuit

If B=0, the button is not pressed and the light should stay off. We need to pass to a state that represents the condition that the light is off and the button is released. It just so happens that this is the same as the initial state, so the transition for B=0 should just return to state 0.

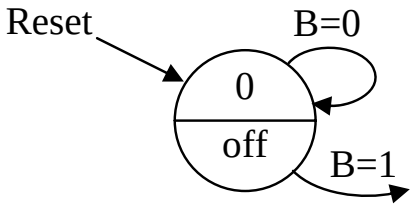


Figure 11-12 B=0 Transition from State 0 of Push Button Circuit

When the button is pressed, the light should come on. Therefore, the transition for B=1 should pass to a state where the light is on and the button is pressed. We don't have this state in our diagram, so we need to add it.

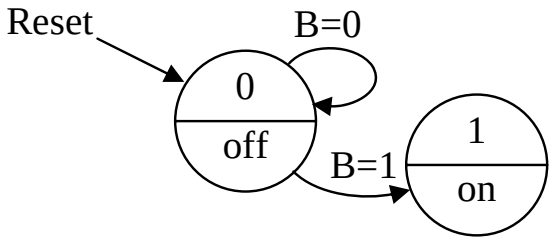


Figure 11-13 B=1 Transition from State 0 of Push Button Circuit

At this point, we have defined all of the transitions out of state 0. In the process, however, we created a new state. We now need to define all of the transitions for the input B for this new state.

If B=0 at the next clock pulse, then the button has been released. Going back to the original definition of the system, we see that if the button is pressed to turn on the light and then it is released, the light should stay on. Therefore, the transition out of state 1 for B=0 should go to a state where the light is on, but the button is released. We don't have this state, so we will need to make it. We will name it state 2.

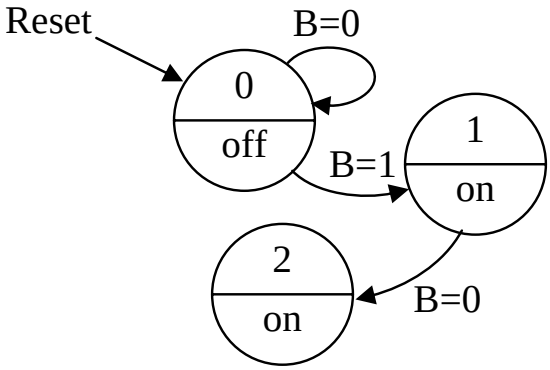


Figure 11-14 B=0 Transition from State 1 of Push Button Circuit

Going back to state 1, if the user's finger is still on the button (B=1) at the next clock pulse, then the light needs to remain on. We therefore need to go to a state where the button is pressed and the light is on. This is state 1, so the transition for B=1 needs to loop back into state 1.

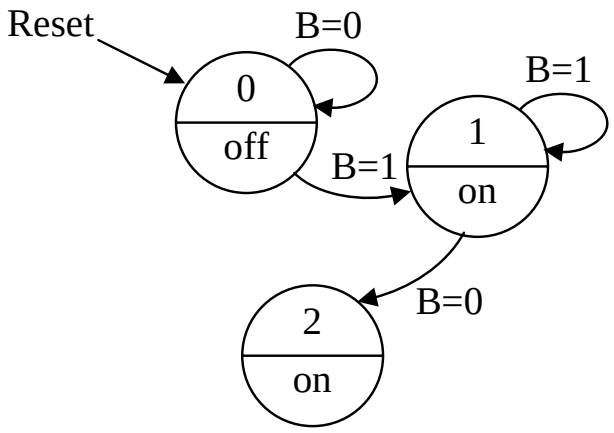


Figure 11-15 B=1 Transition from State 1 of Push Button Circuit

Now that all of the transitions from state 1 have been defined, we need to begin defining the transitions from state 2. If B=0, the button has not been pressed and the current state must be maintained. If the button is pressed, the light is supposed to turn off. Therefore, we need to pass to a state where the light is off and the button is pressed. This state doesn't exist, so we need to create state 3.

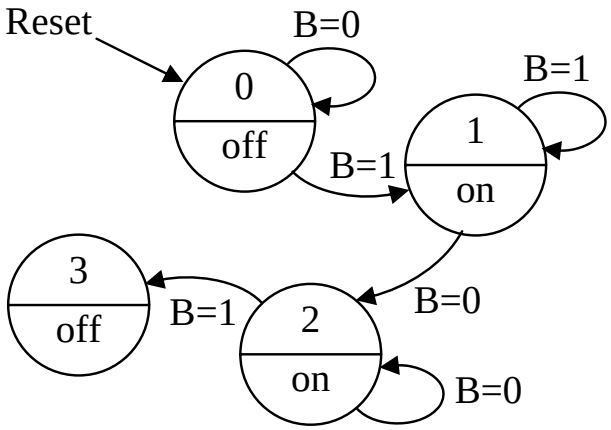


Figure 11-16 Transitions from State 2 of Push Button Circuit

As you can see, each time we create a new state, we need to add the transitions for both B=0 and B=1 to it. This will continue until the addition of all the transitions does not create any new states. The last

step added state 3 so we need to add the transitions for it. If B=0, then the button has been released, and we need to move to a state where the button is released and the light bulb is off. This is state 0. If B=1, then the button is still pressed and the bulb should remain off. This is state 3. Since we didn't create any new states, then the state diagram in Figure 11-17 should be the final state diagram for the system.

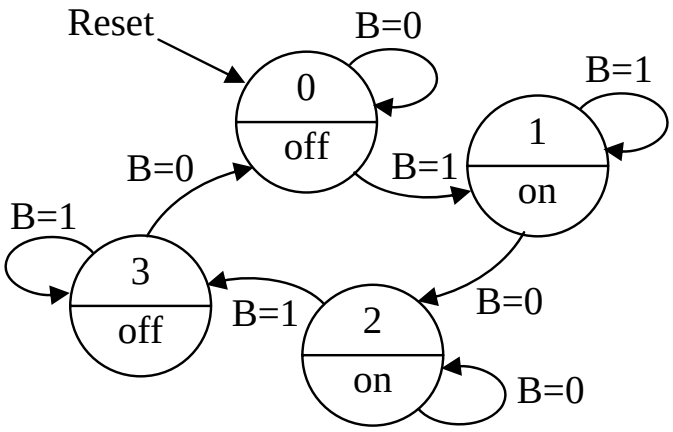


Figure 11-17 Final State Diagram for Push Button Circuit

At this point, there are a couple of items to note. First, as each state was created, it was assigned a number beginning with state 0 for the initial state. The order in which the states are numbered is not important right now. Advanced topics in state machine design examine how the numbering affects the performance of the circuit, but this chapter will not address this issue. It is a good idea not to skip values as doing this may add latches to your design.

The second item to note regards the operation of the state machine. The state diagram shows that to know which state we are going to be transitioning to, we need to know both the current state and the current values on the inputs.

The next step is a minor one, but it is necessary in order to determine the number of latches that will be used in the center block of Figure 11-9. Remember that the latches maintain the current state of the state machine. Each latch acts as a bit for the binary value of the state. For example, if the current state of the system is $2_{10} = 10_2$, then the state machine must have at least two latches, one to hold the '1' and one to hold the '0'. By examining the largest state number, we can

determine the minimum number of bits it will take to store the current state. This is why we begin numbering out states at zero.

For our system, the largest state number is $3_{10} = 11_2$. Since 3 takes two bits to represent, then two latches will be needed to store any of the states the system could enter. Table 11-1 presents each of the states along with their numeric value in decimal and binary.

Table 11-1 List of States for Push Button Circuit

State	Numeric Value	
	Decimal	Binary
Bulb off; button released	0	00
Bulb on; button pressed	1	01
Bulb on; button released	2	10
Bulb off; button pressed	3	11

We will label the two bits used to represent these values S_1 and S_0 where S_1 represents the MSB and S_0 represents the LSB. This means, for example, that when $S_1 = 0$ and $S_0 = 1$, the bulb is on and the button is pressed. Each of these bits requires a latch. Using this information, we can begin building the hardware for our state machine.

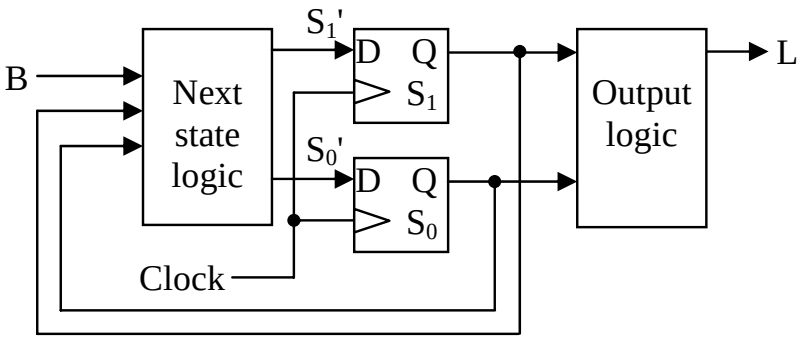


Figure 11-18 Block Diagram for Push Button Circuit

The next step is to develop the truth tables that will be used to create the two blocks of logic on either side of the latches in Figure 11-18. We begin with the "next state logic." The inputs to this logic will be the system input, B, and the current state, S_1 and S_0. The outputs represent the next state that will be loaded into the latches from their D inputs

when a clock pulse occurs. These are represented in Figure 11-18 by the signals S_1' and S_0'.

The ***next state truth table*** lists every possible combination of ones and zeros for the inputs which means that every possible state along with every possible system input will be listed. Each one of these rows represents an arrow or a transition on the state diagram. The output columns show the state that the system will be going to if a clock pulse occurs. For example, if the current state of our push button circuit is state 0 ($S_1 = 0$ and $S_0 = 0$) and the input B equals one, then we are going to state 1 ($S_1' = 0$ and $S_0' = 1$). If the current state is state 0 and the input B equals zero, then we are staying in state 0 ($S_1' = 0$ and $S_0' = 0$). Table 11-2 presents the truth table where each transition of the state diagram in Figure 11-17 has been translated to a row.

We also need to create a truth table for the output logic block of Figure 11-18. The output logic produces the correct output based on the current state. This means that the circuit will take as its inputs S_1 and S_0 and produce the system output, L. The truth table is created by looking at the output (the lower half of each circle representing a state), and placing it in the appropriate row of a truth table based on the values of S_1 and S_0. Table 11-3 presents the output truth table.

Table 11-2 Next State Truth Table for Push Button Circuit

S_1	S_0	B	S_1'	S_0'	
0	0	0	0	0	← State 0 stays in state 0 when B=0
0	0	1	0	1	← State 0 goes to state 1 when B=1
0	1	0	1	0	← State 1 goes to state 2 when B=0
0	1	1	0	1	← State 1 stays in state 1 when B=1
1	0	0	1	0	← State 2 stays in state 2 when B=0
1	0	1	1	1	← State 2 goes to state 3 when B=1
1	1	0	0	0	← State 3 goes to state 0 when B=0
1	1	1	1	1	← State 3 stays in state 3 when B=1

Table 11-3 Output Truth Table for Push Button Circuit

S_1	S_0	L	
0	0	0	← State 0: bulb is off
0	1	1	← State 1: bulb is on
1	0	1	← State 2: bulb is on
1	1	0	← State 3: bulb is off

Now that we have our system fully defined using truth tables, we can design the minimum SOP logic using Karnaugh maps. Figure 11-19 presents the Karnaugh maps for the outputs S_1', S_0', and L.

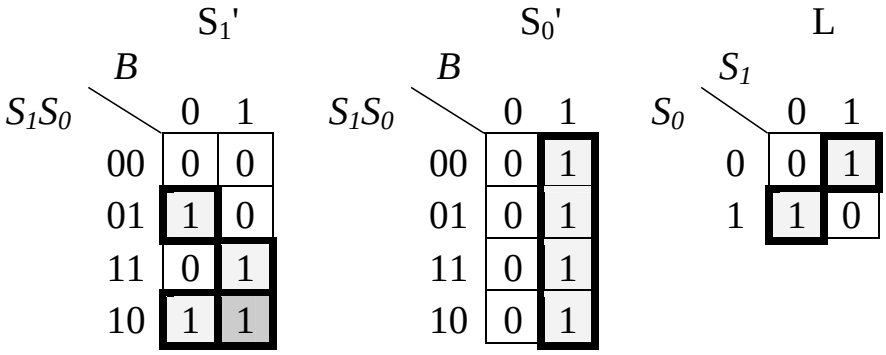


Figure 11-19 K-Maps for S_1', S_0', and L of Push Button Circuit

From these Karnaugh maps, we get the following boolean expressions:

$$S_1' = \overline{S_1} \cdot S_0 \cdot \overline{B} + S_1 \cdot B + S_1 \cdot \overline{S_0}$$

$$S_0' = B$$

$$L = \overline{S_1} \cdot S_0 + S_1 \cdot \overline{S_0} = S_1 \oplus S_0$$

These expressions give us the final implementation of our state machine. By converting the expressions to logic circuits and substituting them for the logic blocks in Figure 11-18, we get the circuit shown in Figure 11-20.

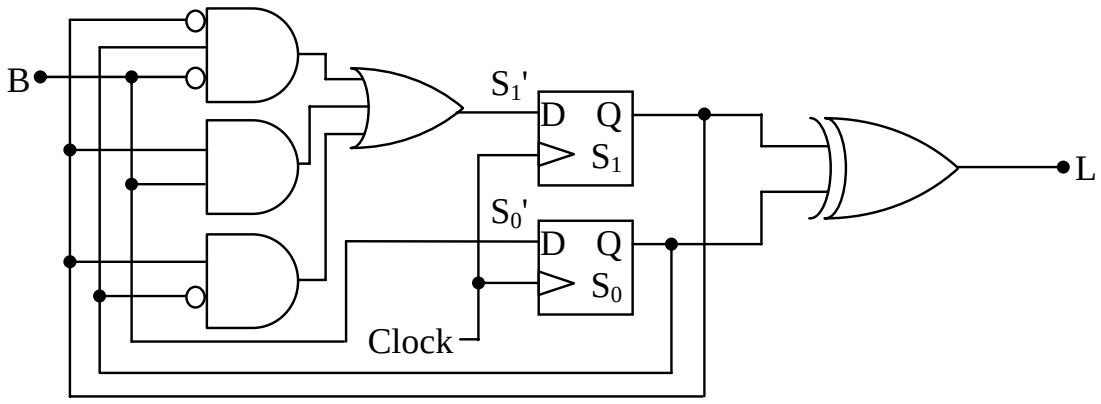


Figure 11-20 Finished Push Button Circuit

As was mentioned earlier, the numbering of the states directly affects the logic that results from a design. Let's use our design to see how this might happen. Assume we were to leave everything in the design of our circuit the same except for swapping the numbers for states 2 and 3. Table 11-4 shows the new binary values for our states.

Table 11-4 Revised List of States for Push Button Circuit

State	Numeric Value	
	Decimal	Binary
Bulb off; button released	0	00
Bulb on; button pressed	1	01
Bulb on; button released	3	11
Bulb off; button pressed	2	10

This modification affects all of the logic, but let's only look at how it affects the output logic that drives the signal L. In this case, the light is on in states 1 and 3, but off for states 0 and 2. Figure 11-21 presents the new output truth table and the resulting Karnaugh map.

S_1	S_0	L
0	0	0
0	1	1
1	0	0
1	1	1

S_0 \ S_1	0	1
0	0	1
1	0	1

Figure 11-21 Revised Truth Table and K Map for Push Button Circuit

This gives us a new boolean expression for L.

$$L = S_1$$

The final SOP expression for L with our previous numbering scheme used two AND gates and an OR gate. (This is the way an XOR gate is implemented.) Our new numbering scheme now consists only of a wire connecting the output of S_1 to the light bulb.

11.3 Another State Machine Design: Pattern Detection

A common application for state machines is to watch for a specific binary pattern within a serial data stream. The binary pattern may be used to indicate the start or end of a message or to alert the receiving device that a control sequence is about to come. Figure 11-22 presents a sample binary stream where the binary bit pattern "101" is identified.

1101001111011001011101010000111101101111001

Figure 11-22 Identifying the Bit Pattern "101" in a Bit Stream

If a clock can be produced that pulses once for each incoming bit, then we can develop a state machine that detects this pattern. The state machine will initially output a zero indicating no pattern match and will continue to output this zero until the full pattern is received. When the full pattern is detected, the state machine will output a 1 for one clock cycle.

The state machine used to detect the bit pattern "101" will have four states, each state representing the number of bits that we have received up to this point that match the pattern: 0, 1, 2, or 3. For example, a string of zeros would indicate that we haven't received any bits for our sequence. The state machine should remain in the state indicating no bits have been received.

If, however, a 1 is received, then it is possible we have received the first bit of the sequence "101". The state machine should move to the state indicating that we might have the first bit. If we receive another 1 while we are in this new state, then we know that the first 1 was not part of the pattern for which we are watching. The second 1, however, might indicate the beginning of the pattern, so we should remain in the state indicating that we might have received the first bit of the pattern. This thought process is repeated for each state.

The list below identifies each of the states along with the states they would transition to based on the input conditions.

- ***State 0*** – This is the initial state representing the condition that no bits of the sequence have been received. As long as zeros are received, we should remain in this state. Since the first bit of the sequence is a 1, whenever a 1 is received we should move to the state indicating that we *might* have received the first bit.

- ***State 1*** – This state is entered when we have received the first bit of the sequence. Receiving a 0, the second bit of the sequence, should move us to a state indicating that we've received "10". Receiving a 1 means that the last 1 was not part of the sequence, but this new 1 might be, so we should remain in State 1.
- ***State 2*** – We are in this state when we believe we *might* have the first two bits of the sequence, i.e., we got here after receiving "10". If we receive a 0, then the last three bits we received were "100". Since it is not possible to pull any part of the sequence out of this pattern, we should go back to state 0 indicating none of the sequence has been received. If, however, we receive a 1 while in this state, then the last three bits were "101" and we should go to the state indicating that the first three bits might have been received.
- ***State 3*** – This is the state we go to when we've received all three bits of the sequence, and therefore, it is the only state where a 1 will be output. If while in this state we receive a 0, then the last four bits we received were "1010". Notice that the last two bits of this sequence are "10" which are the first two bits of the pattern for which we are looking. Therefore, we should go to state 2 indicating that we've received two bits of the sequence. If we receive a 1, then the last four bits we received were "1011". This last 1 could be the first bit of a new pattern of "101", so we should go to state 1.

These states along with the appropriate transitions are presented in Figure 11-23 using the letter 'I' to represent the latest bit received from the input stream.

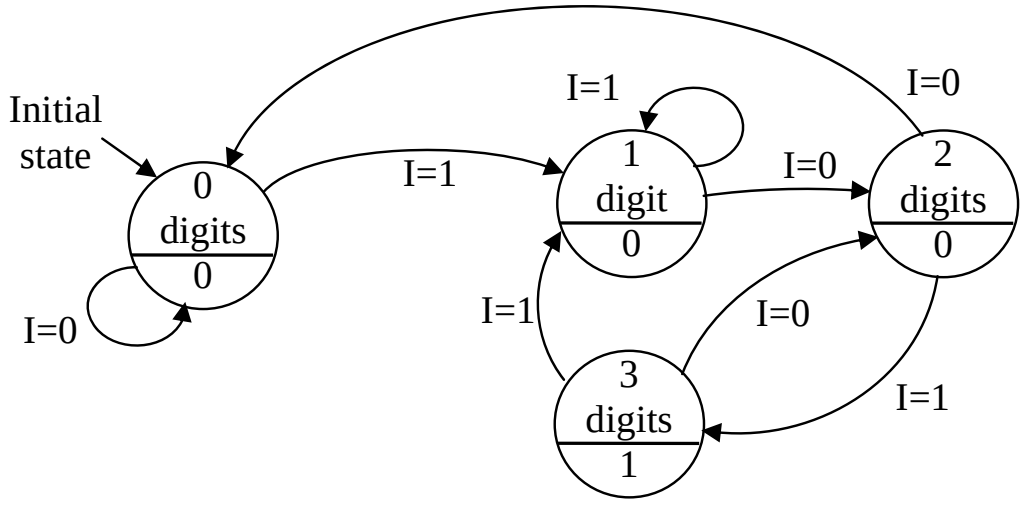


Figure 11-23 State Diagram for Identifying the Bit Pattern "101"

Next, we need to assign binary values to each of the states so that we know how many latches will be needed to store the current state and provide the inputs to the next state logic and the output logic. Table 11-5 presents the list of states along with their decimal and binary values.

From the state diagram and the numbering of the states, we can create the next state truth table and the output truth table. These are presented in Figure 11-24 with S_1 representing the MSB of the state, S_0 representing the LSB of the state, and P representing the output.

Table 11-5 List of States for Bit Pattern Detection Circuit

State	Numeric Value	
	Decimal	Binary
No bits of the pattern have been received	0	00
One bit of the pattern has been received	1	01
Two bits of the pattern have been received	2	10
Three bits of the pattern have been received	3	11

Next State

S_1	S_0	I	S_1'	S_0'
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	0	0
1	0	1	1	1
1	1	0	1	0
1	1	1	0	1

Output

S_1	S_0	P
0	0	0
0	1	0
1	0	0
1	1	1

Figure 11-24 Next State and Output Truth Tables for Pattern Detect

Figure 11-25 shows the Karnaugh maps and resulting equations from the truth tables of Figure 11-24. Figure 11-26 presents the final circuit design.

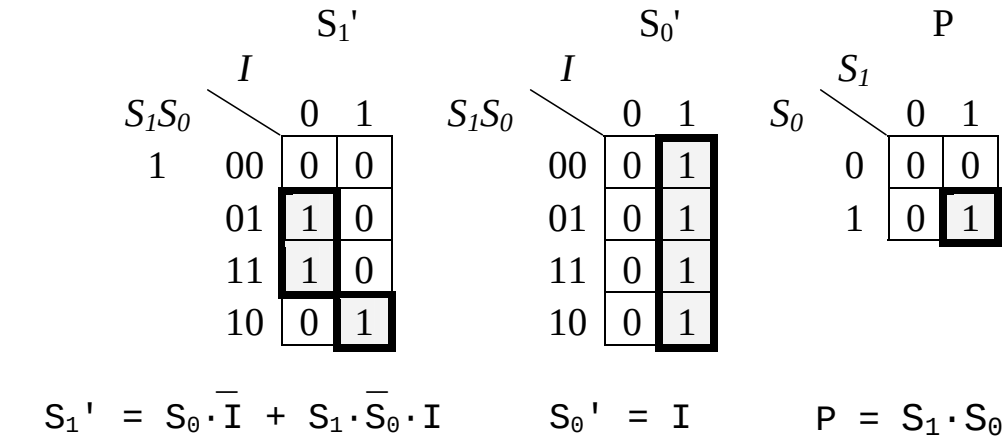


Figure 11-25 K-Maps for S_1', S_0', and P of Pattern Detect Circuit

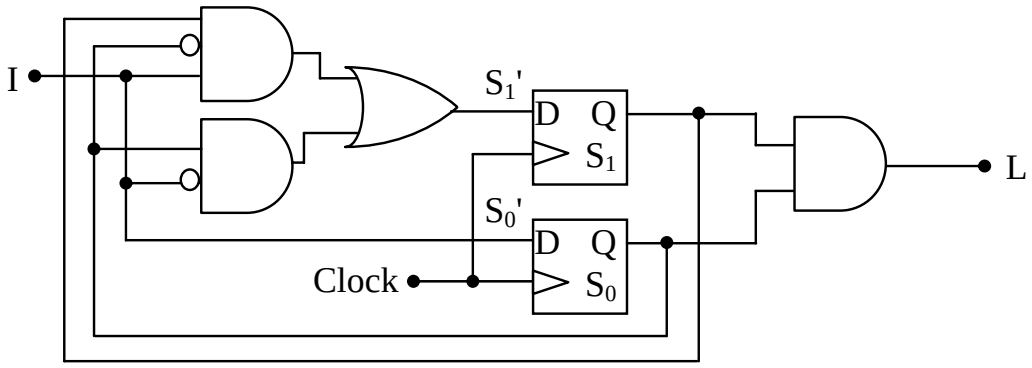


Figure 11-26 Final Circuit to Identify the Bit Pattern "101"

11.4 Mealy Versus Moore State Machines

The state machine design covered in the previous sections is referred to as a ***Moore machine***. The distinguishing characteristic of a Moore machine is that its output is determined only by the current state.

The output of the second type of state machine, the ***Mealy machine***, is based on both the current state of the machine *and* the system's input. Figure 11-27 shows how this is accomplished by including the system's inputs with the current state to drive the output logic.

To do this, the state diagram must be modified. The output values are removed from the states in the state diagram for a Mealy machine. The output values are then associated with each transition. Figure 11-28 shows how each transition is labeled using the format *X/Y* where *X* identifies the input driving the transition and *Y* is the resulting output.

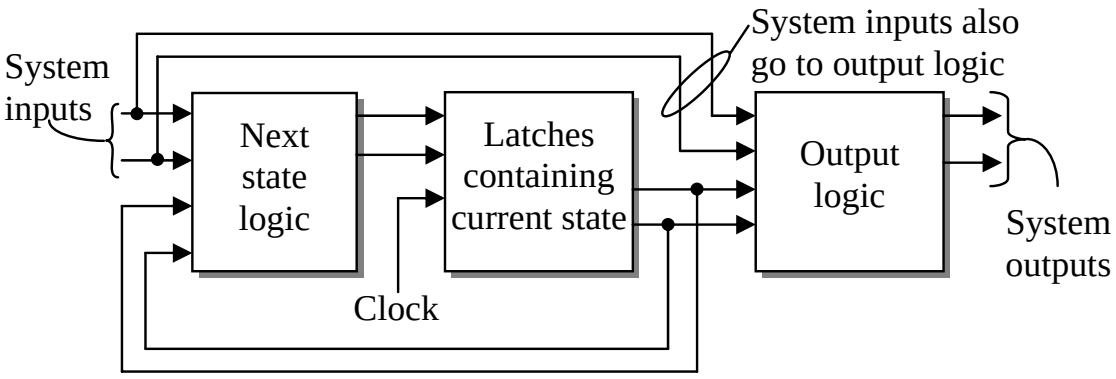


Figure 11-27 Basic Configuration of a Mealy Machine

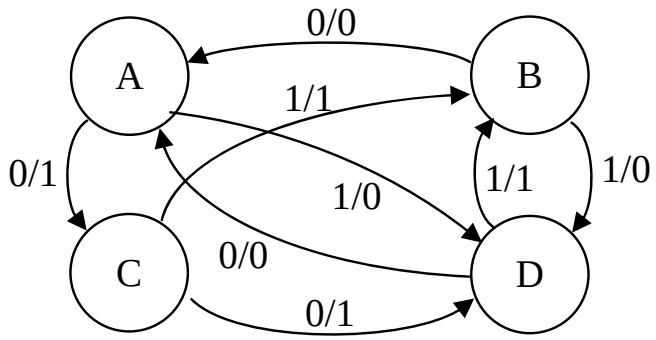


Figure 11-28 Sample State Diagram of a Mealy Machine

The next state truth table for the Mealy machine is the same as that for the Moore machine: the current state and the system input govern the next state. The Mealy machine's output truth table is different, however, since it now uses the system input as one of the truth table's inputs. Figure 11-29 presents the output truth table for the state diagram in Figure 11-28 where state A is $S_0 = 0$, $S_1 = 0$, B is $S_0 = 0$, $S_1 = 1$, C is $S_0 = 1$, $S_1 = 0$, and D is $S_0 = 1$, $S_1 = 1$.

11.5 What's Next?

Chapter 12 expands the application of latches to the arrays used in some forms of memory. It also presents the interface circuitry that the processor uses to communicate with memory. This may seem to be too detailed an examination of hardware for the typical computer science student, but the theories behind these designs are identical to those used to decode the subnet and host IDs of an IP network.

State	S_1	S_0	I	Y
A	0	0	0	1
	0	0	1	0
B	0	1	0	0
	0	1	1	0
C	1	0	0	1
	1	0	1	1
D	1	1	0	0
	1	1	1	1

Figure 11-29 Output Truth Table for Sample Mealy Machine

Problems

1. What is the maximum number of states a state machine with four latches can have?

2. How many latches will a state machine with 28 states require?

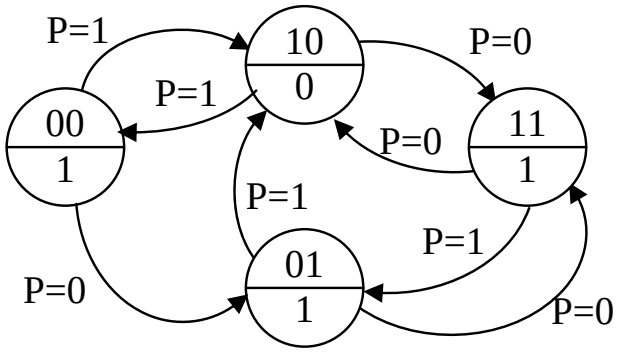
3. Apply the design process presented in Section 11.2 to design a two-bit up/down counter using the input *direction* such that when *direction* = 0, the system decrements (00 → 11 → 10 → 01 → 00) and when *direction* = 1, the system increments (00 → 01 → 10 → 11 → 00).

4. The three Boolean expressions below represent the next state bits, S_1' and S_0', and the output bit, X, based on the current state, S_1 and S_0, and the input I. Draw the logic circuit for the state machine including the latches and output circuitry. Label all signals.

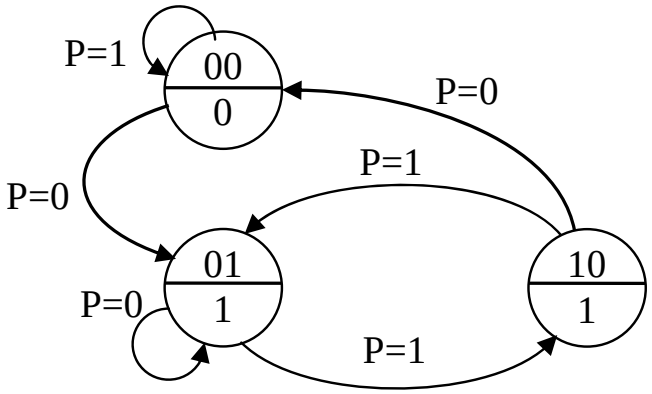
   $S_1' = \overline{S_1} \cdot S_0 \qquad S_0' = S_1 \cdot S_0 \cdot I \qquad X = \overline{S_1} + S_0$

5. Create the next state truth table and the output truth table for the following state diagrams. Use the variable names S_1 and S_0 to represent the most significant and least significant bits respectively of the binary number identifying the state.

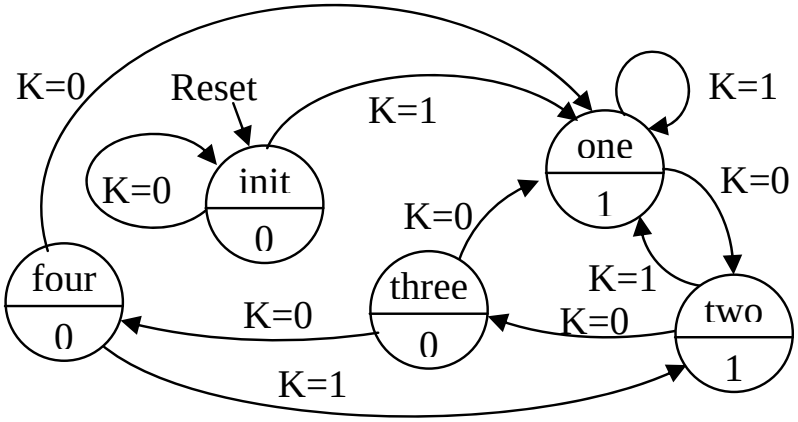
a.)



b.)



6. Identify the error in this state diagram. Be as specific as you can.



7. Design a state machine to detect the bit pattern "110" in a stream of bits.

8. How many latches would be needed in a state machine that detected the pattern "011110" in a stream of bits?

9. Create the logic diagram for the Mealy machine described in Figure 11-28.