State Machines

Bob Brown
Computer Science Department
Southern Polytechnic State University

Introduction

We were able to model combinational circuits with truth tables. If an input to a combinational circuit changes, the output will change correspondingly. Consider our first sequential circuit, the S-R latch. The truth table isn't an adequate model for a sequential circuit because the output depends upon both the inputs and the state. **Finite state machines**, or FSMs, provide a way of thinking about sequential circuits. FSMs are important in sequential circuit design, programming, compilers, language recognizers, and many other areas of computer science. A major activity in all areas of computer science is the construction of models. State machines are one of the most frequently used model-building tools. It is important for you to learn how to use state machines as models no matter what area of computer science you pursue.

The S-R Latch

Figure 1 shows the digital logic diagram for an S-R latch, a simple circuit with memory. It has inputs S and R and a state Q. The output is identical to the state. Applying a signal momentarily to S causes Q to enter the set state if it wasn't already in that state. When S is removed, Q remains in the set state. Similarly, asserting R momentarily causes a transition of Q to the reset state if it wasn't already in that state. The input S=R=1 is not allowed; the behavior of the S-R latch is undefined for this input. Real S-R latches usually include a second output $\overline{Q}$ as a convenience to the circuit designer.

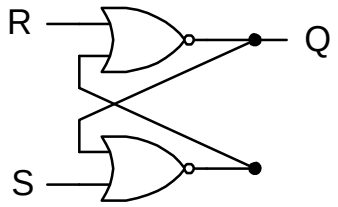

Figure 1. Digital logic diagram for an S-R latch.

One way to represent the actions of the S-R latch (and other sequential circuits) is with a **characteristic table**, as shown in Figure 2. Q, the current state, is one of the inputs. Q^+ represents the next state. The next state is sometimes represented as $Q_{(t+1)}$.

Q	S	R	Q^+
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**Undefined**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**Undefined**

Figure 2. Characteristic table for the S-R latch.

The characteristic table is similar to a truth table except that the state variables appear on both the right and the left. The present state is on the left and the next state is on the right. The input variables must be

repeated for each possible state. For this reason, a characteristic table can become quite large. A characteristic table is a suitable tool for programming a function described as a state machine, but it is difficult to analyze if there are many states or inputs.

Another, more visual way to represent the actions of a sequential circuit is with a **state transition diagram**. A state transition diagram is a directed graph. Nodes represent states and directed edges represent transitions. Each node is labeled with the value of the state it represents. Each edge is labeled by the input values that cause the transition represented by the edge.

Figure 2 shows the state transition diagram for the S-R latch. The two possible states are represented by the two nodes. The inputs S and R are represented by the edges; labels on the edges are the values of S and R in that order. It is important to note that the state transition diagram shows only valid inputs. In particular, the combination S=R=1 does not appear because it is not an input that causes a valid state transition. Valid inputs that do not cause transitions are represented by edges that return to the same state.

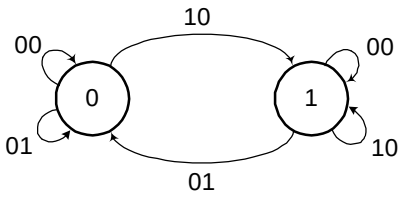

Figure 3. State transition diagram for the S-R latch.

The purpose of a sequential circuit is to generate an output. For the S-R latch, the output is identical to the state. This is not necessarily true for more complex sequential circuits. The output of a state may be shown explicitly by including it in the node labels, separated from the state variable by a slash. For the S-R latch, the node labels would be 0/0 and 1/1. This says that state 0 produces output 0 and state 1 produces output 1. Later we will see a model in which the outputs are associated with transitions rather than states.

Now that we have the idea that a sequential circuit can be represented as a set of states with transition functions based on inputs, let's look at a more formal way of representing that idea.

State Machines

State machines model control processes as functions of time, state variables, inputs, and outputs. A state machine accepts a sequence of inputs. Each input generates a new internal state that is a function of the input and the previous state. Each input also generates an output. Finite state machines are state machines whose state variables, inputs, and outputs can be enumerated. For example, an oil refinery can be modeled by a state machine. A traffic signal can be modeled by a finite state machine.

Finite state machines are of theoretic importance to computer science because they can model many of the concepts of interest in the discipline. You will encounter finite state machines in the study of sequential circuits, compilers, operating systems, and

programming. Finite state machines are of practical importance to students of computer architecture because they can be implemented directly with digital logic.

One definition of a **finite state machine** is an ordered four-tuple, M=(Q,V,t,q_0) where Q is a finite set of states, V is the input alphabet—a finite set of symbols, q_0 is the initial state—a member of Q, The transition function t maps the states and inputs onto a set of state transitions. The output of such a machine is only zero (false) or one (true). We will consider only deterministic finite state machines.* Every nondeterministic finite state machine has an equivalent deterministic finite state machine.

We can add a set of final states to the definition of a finite state machine. Such a machine is called a **finite state acceptor**. A final state that produces an output of true is called an **accepting state.**

The S-R Latch as a Finite State Machine

Given a formal definition of a finite state machine, we see that we must specify one additional condition to make the S-R latch a finite state machine.

The set of states Q of the S-R latch is {0, 1}. The input alphabet V is {00,01,10}; this is the set of values S and R can take on, excluding the undefined case of S=R=1. The output of the S-R latch is a zero or one according to the state. What is missing is the start state, q_0.

You can construct a set of inputs to the S-R latch such that the output is different depending upon the initial state. The trivial case is the input 00. The output will be one if the initial state was one, and zero if the initial state was zero.

In order to treat the S-R latch as an FSM, we must define one of its two states as the start state. We can choose either the zero state or the one state, but the choice results in one of two *different* FSMs.

Finite State Acceptors

There are several set-theoretic models of finite state machines. The choice of which one to use depends upon the specific application for which one is constructing a model.

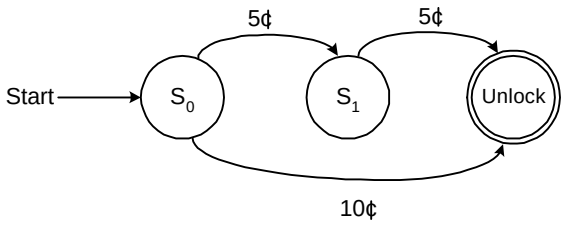

Figure 4. Vending machine example. The final state is an accepting state**.**

Consider a vending machine that accepts nickels and dimes and sells items with a

* Finite state machines are also called *finite automata.* Deterministic finite state machines are also called deterministic finite automata, often abbreviated *DFA.*

price of ten cents. A customer can deposit two nickels or a dime to unlock the dispensing mechanism. We can model such a machine as an **Finite state acceptor**. The start state is the state of no money deposited. The inputs are nickels and dimes. We have introduced a special state—the final or accepting state—that produces an output *true* which unlocks the dispensing mechanism. Final states are shown as nodes with double circles. Such a machine is shown in Figure 4.

Depositing ten cents causes an immediate transition to the unlocked state. Depositing a nickel causes a transition to an intermediate state, S_1. A second nickel causes a transition to the unlocked state. The unlocked state is a final state with output true (unlock the mechanism) so this machine is a finite state acceptor. When an item is dispensed, the machine returns to the start state.

It is important to notice that we have modeled only a part of the operation of a vending machine. This model captures the essence of a successful transaction, but a lot of detail has been suppressed. We don't consider what the machine does if the input is a penny, nor the details of how the unlocking mechanism works nor the mechanism by which the machine returns to the start state. In a real-world design, these details are important.

Moore and Mealy Machines

We have been thinking of finite state machines in the abstract, but it should be clear that FSMs can be implemented directly using combinational and sequential logic. A sequential logic (storage) device called a **state register** provides the necessary state information. Each possible value of the state register is represented by a node on the state transition diagram. A state transition diagram with 2^n nodes requires an n-bit state register. The transition function is implemented as combinational logic that has as its input the inputs to the machine and the state information from the state register.

Let us extend our notion of a finite state machine to one that can produce outputs other than zero and one. The formal definition becomes a sextuple, $M=(Q,V,t,q_0,V¢,o)$ where $V¢$ is the output alphabet, a finite set of output symbols, and o is the output function. We could choose to have the output of the state register serve as the output of the circuit as we did with the S-R latch. However, this definition of a finite state machine allows us more flexibility. The

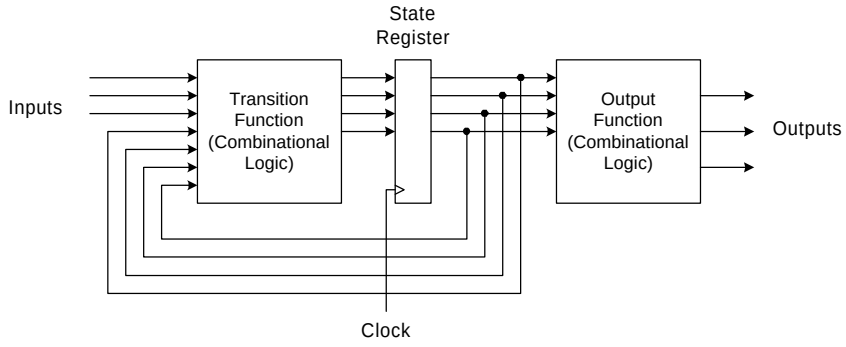

Figure 5. Moore model finite state machine. Output changes only when signaled by the clock.

output function *o* maps the set of states *Q* onto the output alphabet *V*¢. We can express this formally as $o:Q \rightarrow V¢$. This type of output function can be implemented directly in combinational logic driven by the state register. This is called a Moore model finite state machine or **Moore machine**. A conceptual diagram of a Moore machine is shown in Figure 5. The transition function *t* is implemented in combinational logic. The inputs to the machine and the current state from a state register are input to a combinational circuit that computes the next state. State transitions occur when the state register is enabled by the clock signal. Although the output is computed by a combinational circuit, its inputs come from the state register, which can change only when signaled by the clock. This means that the outputs of a Moore machine change only upon clock signals, and the outputs are determined by the state. If you think of a Moore machine in terms of a transition diagram, the outputs are associated with the nodes of the graph.

There is another way of building a finite state machine with an output function: use the same combinational logic circuit to compute both the transition function and the output function. The inputs to the combinational logic are the inputs to the FSM and the present state from the state register. Outputs are a function of both the present state and the input. This is the Mealy model finite state machine or **Mealy machine.** The definition is the same as the Moore machine, but the output function maps both the current state and the input alphabet onto the output alphabet. Formally, the output function is defined as $o:Q \times V \rightarrow V¢$. As you can see from the conceptual diagram in Figure 6, the outputs of a Mealy machine can change independent of the clock. Any change in the input can cause a change in the output even though the state can change only when signaled by the clock. In terms of a state transition diagram, you can think of the outputs as being associated with the edges of the graph, the transitions rather than the states. Since the outputs are associated with the transitions, some functions can be implemented with fewer states using Mealy machines. A potential disadvantage is that the outputs can change at any time. One way to mitigate this disadvantage is to place a register in the output path.

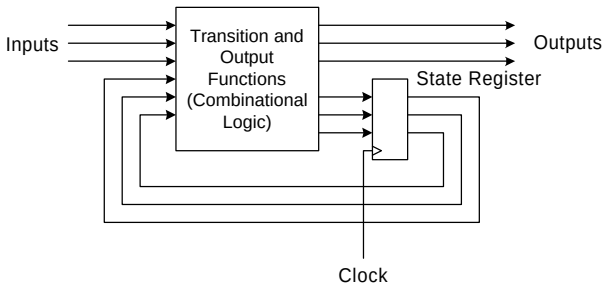

Figure 6. Mealy model finite state machine. Outputs can change with the inputs.

Algorithmic State Machines

Finite state machines are a powerful tool for designing sequential circuits, but they are lacking in that they do not expressly represent the algorithms that compute the transition or output functions, nor is timing information explicitly represented. We can recast the idea of a state machine to include a representation of the algorithms. The result is an **algorithmic state machine,** or **ASM**. "The ASM chart separates the conceptual phase of a design from the actual circuit implementation." [OSBO73] An algorithmic state machine diagram is similar to a flowchart. Square boxes represent the states, diamonds

represent decisions, and ovals represent outputs. States can also have outputs, and the outputs associated with a state are listed in the state box. State boxes are labeled with the state name and possibly with a binary code for the state. The basic unit of an ASM is the **ASM block**. An ASM block contains a single state box, a single entry path, and one or more exit paths to other ASM blocks. Algorithmic state machines capture the timing of state transitions as well. The contents of one ASM block represent the actions during one clock cycle.

Consider a traffic signal that is red in the east-west direction and green in the north-south direction. Call that the RG state. When a timer expires the timer will assert a value of one. That will cause a transition to the RY state and a value of ten seconds to be loaded into the timer. The ASM block for this state and transition is shown in Figure 6.

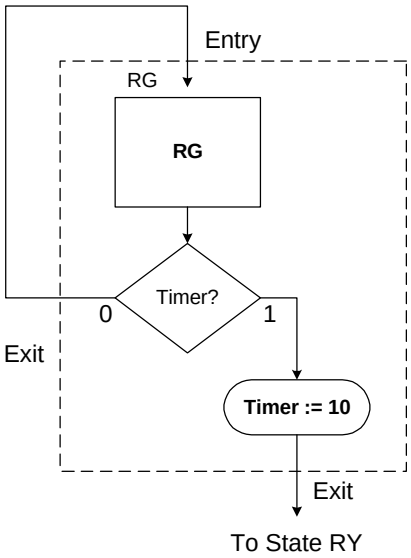

Figure 7. ASM block showing ASM symbols. The ASM block is enclosed within the dashed line.

The ASM block of Figure 7 is enclosed by the dashed line. There is a single entry and a single state named RG. The output **RG** is asserted while in state RG. An exit from state RG occurs when signaled by the clock. The timer is checked and if the value is zero, *i.e.* if the timer has not expired, an exit which will cause re-entry to state RG is taken.

In the clock cycle after the timer expires the timer output value is found to be one. A conditional output causes the timer to be loaded with the new countdown value 10 and an exit to state RY is taken.

You can infer from this description that the ASM of Figure 7 must be a Mealy machine because the output that loads the timer is a conditional output—an output not associated with a state. It is possible to implement ASM blocks using Moore machines. In that case, there could be no conditional outputs. Generally this means more states are required. In the example of Figure 6, a state for "RG and timer = 0" *and* a state for "RG and timer = 1" would be required. Since each ASM block can represent only one state, two ASM blocks would be required.

Summary

Finite state machines can be used to model many of the things we deal with in computer science, from digital sequential circuits to language recognizers. They capture the idea of a finite number of states with an input alphabet, a transition function, and an output alphabet. FSMs are important to computer architecture because they may be implemented directly using sequential and combinational logic. FSMs are one way of thinking about sequential circuit design. There are two models for implementing FSMs

as digital logic circuits. In the Moore model, the outputs can change only on clock signals; outputs are associated with the nodes of a state transition diagram. In the Mealy model, outputs can change with the inputs; the outputs are associated with the edges of the state transition diagram. The algorithmic state machine captures the algorithms and timing associated with the transition and output functions.

References

[OSBO73] Osborne, Thomas E. in forward to Clark, Christopher R., *Designing Logic System s Using State Machines,* McGraw-Hill, 1973.

Bibliography

Aho, Alfred B., Ravi Sethi and Jeffrey D. Ullman, *Compilers, Principles, Techniques, and Tools,* Addison-Wesley, 1986.

Hopcroft, John E., and Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, 1979.

Katz, Randy H., *Contemporary Logic Design*, Benjamin/Cummings, 1994.

Mano, M. Morris and Charles R. Kime, *Logic and Computer Design Fundamentals*, Prentice Hall, 1997.

Wood, Derick, *Theory of Computation,* John Wiley and Sons, 1987.

Exercises

1. Extend the vending machine described in the text to be able to sell items that cost up to fifty cents.

2. Extend the vending machine of exercise 1 to make change.

3. A traffic signal controls a simple cross intersection. Each state has two outputs, EW (east-west) and NS (north-south). The outputs can have the values R, Y, and G, for red, yellow, and green. The rules for the outputs are the familiar rules of the traffic signal. Draw a state transition diagram that models the function of the traffic signal.

4. The state transition diagram of exercise 3 fails to model a real-world traffic signal completely. In what way? (*Hint:* How long does the RY output last in comparison to the RG and GR outputs? How do you know?)

5. You are given a timer which can be loaded with a number *n*, and which emits a signal when *n* seconds have elapsed. Model the traffic light of exercise 3 using an algorithmic state machine that incorporates the timer. Make reasonable assumptions for the time values of outputs R, Y, and G.

6. Draw the state transition diagram for a counter that counts 00, 01, 10, 11, then returns to 00 and repeats. How many nodes do you have? How many nodes would be required for an 8-bit counter? What can you say about the effectiveness of the state transition diagram as a model for a counter?