12 FINITE STATE MACHINES

12.1 INTRODUCTION

What to Expect

Finite State Machines (FSMs) are a model of a real-world application. System designers often start with a FSM in order to determine what a digital logic system must do and then use that model to design the system. This chapter introduces the two most common forms of FSMs: Moore and Mealy. It includes the following topics.

- Analyzing a circuit using a Moore FSM
- Analyzing a circuit using a Mealy FSM
- Developing state tables for a circuit
- Creating a FSM for an elevator simulation

12.2 FINITE STATE MACHINES

12.2.1 *Introduction*

Sequential logic circuits are dynamic and the combined inputs and outputs of the circuit at any given stable moment is called a *state*. Over time, a circuit changes states as triggering events take place. As an example, a traffic signal may be green at some point but change to red because a pedestrian presses the "cross" button. The current state of the system would be "green" but the triggering event (the "cross" button) changes the state to "red."

The mathematical model of a sequential circuit is called a FSM. A FSM is an abstract model of a sequential circuit where each state of the circuit is indicated by circles and various triggering events are used to sequence a process from one state to the next. The behavior of many devices can be represented by a FSM model, including traffic signals, elevators, vending machines, and robotic devices. FSMs are an analysis tool that can help to simplify sequential circuits.

There are two fundamental FSM models: Moore and Mealy. These two models are generalizations of a state machine and differ only in the way that the outputs are generated. A Moore machine generates an output as a function of only the current state while a Mealy machine generates an output as a function of the current state plus the inputs into that state. Moore machines tend to be safer since they are only activated on a clock pulse and are less likely to create unwanted feedback when two different modules are interconnected; however, Mealy machines tend to be simpler and have fewer states than a Moore machine. Despite the strengths and weaknesses for each of these two FSMs, in reality, they are so similar that either can be effectively used to model any given circuit and the actual FSM chosen by the designer is often little more than personal preference.

12.2.2 *Moore Finite State Machine*

The Moore FSM is named after Edward F. Moore, who presented the concept in a 1956 paper, *Gedanken-experiments on Sequential Machines*. The output of a Moore FSM depends only on its current state. The Moore FSM is typically simpler than a Mealy FSM so modeling hardware systems is usually best done using a Moore FSM.

As an example of a Moore FSM imagine a simple candy vending machine that accepts either five cents or ten cents at a time and vends a handful of product when 15 cents has been deposited (no change is returned). Figure 12.1 is a Moore FSM diagram for this vending machine.

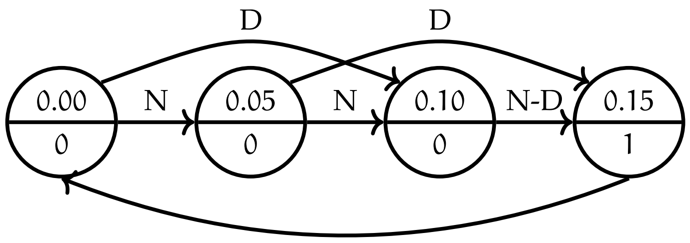

Figure 12.1: Moore Vending Machine FSM

In Figure 12.1, imagine that five cents is deposited between each state circle (that action is indicated by the arrows labeled with an N, for Nickel). The output at each state is zero (printed at the bottom of each circle) until the state reaches 0.15 in the last circle, then the output changes to one (the product is vended). After that state is reached the system resets to state 0.00 and the entire process starts over. If a user deposits ten cents, a Dime, then one of the nickel states is skipped.

12.2.3 *Mealy Finite State Machine*

The Mealy machine is named after George H. Mealy, who presented the concept in a 1955 paper, *A Method for Synthesizing Sequential Circuits*. The Mealy FSM output depends on both its current state and the inputs. Typically a Mealy machine will have fewer states than a Moore machine, but the logic to move from state to state is more complex.

As an example of a Mealy FSM, the simple candy vending machine introduced in Figure 12.1 can be redesigned. Recall that the machine accepts either five cents or ten cents at a time and vends a handful of product when 15 cents has been deposited (no change is returned). Figure 12.2 is a Mealy FSM diagram for this vending machine.

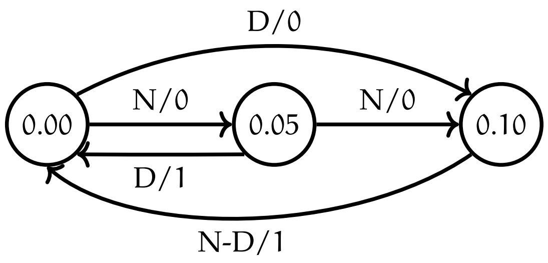

Figure 12.2: Mealy Vending Machine FSM

In Figure 12.2 the states are identified by the amount of money that has been deposited, so the first state on the left (0.00) is when no money has been deposited. Following the path directly to the right of the first state, if five cents (indicated by "N" for a nickel) is deposited, then the output is zero (no candy is dispensed) and the state is changed to 0.05. If another five cents is deposited, the output remains zero and the state changes to 0.10. At that point, if either five or ten cents ("N" or "D") is deposited, then the output changes to 1 (candy is dispensed) and the state resets to 0.00. By following the transition arrows various combinations of inputs and their resulting output can be traced.

Because the Mealy FSM reacts immediately to any input it requires one less state than the Moore machine (compare Figures 12.1 and 12.2). However, since Moore machines only change states on a clock pulse they tend to be more predictable (and safer) when integrated into other modules. Finally, Mealy machines tend to react faster than Moore machines since they do not need to wait for a clock pulse and, generally, are implemented with fewer gates. In the end, whether to design with a Mealy or a Moore machine is left to the designer's discretion and, practically speaking, most designers tend to favor one type of FSM over the other. The simulations in this book use Moore machines because they tend to be easier to understand and more stable in operation.

12.2.4 *Finite State Machine Tables*

Many designers enjoy using Moore and Mealy FSM diagrams as presented in Figures 12.1 and 12.2; however, others prefer to design with a finite state machine table that lists all states, inputs, and outputs.

As an example, imagine a pedestrian crosswalk in the middle of a downtown block. The crosswalk has a traffic signal to stop traffic and a *Cross / Don't Cross* light for pedestrians. It also has a button that pedestrians can press to make the light change so it is safe to cross.

This system has several states which can be represented in a *State Table*. The traffic signal can be Red, Yellow, or Green and the pedestrian signal can be Walk or Don't Walk. Each of these signals can be either zero (for *Off*) or one (for *On*). Also, there are two triggers for the circuit; a push button (the *Cross* button that a pedestrian presses) and a timer (so the walk light will eventually change to don't walk). If the button is assumed to be one when it is pressed and zero when not pressed, and the timer is assumed to be one when a certain time interval has expired but zero otherwise, then State Table 12.1 can be created.

	Current State					Trigger		Next State				
	R	Y	G	W	D	Btn	Tmr	R	Y	G	W	D
1	0	0	1	0	1	0	0	0	0	1	0	1
2	0	0	1	0	1	1	0	0	1	0	0	1
3	0	1	0	0	1	X	0	1	0	0	1	0
4	1	0	0	1	0	X	1	0	0	1	0	1

Table 12.1: Crosswalk State Table

The various states are R (for *Red*), Y (for *Yellow*), and G (for *Green*) traffic lights, and W (for *Walk*) and D (for *Don't Walk*) pedestrian lights. The *Btn* (for *Cross Button*) and *Tmr* (for *Timer*) triggers can, potentially, change the state of the system.

Row One on this table shows that the traffic light is Green and the pedestrian light is Don't Walk. If the button is not pressed (it is zero) and the timer is not active (it is zero), then the next state is still a Green traffic light and Don't Walk pedestrian light; in other words, the system is quiescent. In Row Two, the button was pressed (*Btn* is one); notice that the traffic light changes state to Yellow, but the pedestrian light is still Don't Walk. In Row Three, the current state is a Yellow traffic light with Don't Walk pedestrian light (in other words, the *Next State* from Row Two), the X for the button means it does not matter if it is pressed or not, and the next state is a Red traffic light and Walk pedestrian light. In Row Four, the timer expires (it changes to one at the end of the timing cycle), and the traffic light changes back to Green while the pedestrian light changes to Don't Walk.

While Table 12.1 represents a simplified traffic light system, it could be extended to cover all possible states. Since Red, Yellow, and Green can never all be one at one time, nor could Walk and Don't Walk, the designer must specifically define all of the states rather than use a simple binary count from 00000 to 11111. Also, the designer must be certain that some combinations never happen, like a Green traffic light and a Walk pedestrian light at the same time, so those must be carefully avoided.

State tables can be used with either Mealy or Moore machines and a designer could create a circuit that would meet all of the requirements from the state table and then realize that circuit to actually build a traffic light system.

12.3 SIMULATION

One of the important benefits of using a simulator like *Logisim-evolution* is that a circuit designer can simulate digital systems to determine logic flaws or weaknesses that must be addressed before a physical IC is manufactured.

12.4 ELEVATOR

As a simple example of a IC, imagine an elevator control circuit. For simplicity, this elevator is in a five story building and buttons like "door open" will be ignored. There are two ways the elevator can be called to a given floor: someone could push the floor button in the elevator car to ride to that floor or someone could push the call button beside the elevator door on a floor.

Figure 12.3 is the Moore FSM for this circuit:

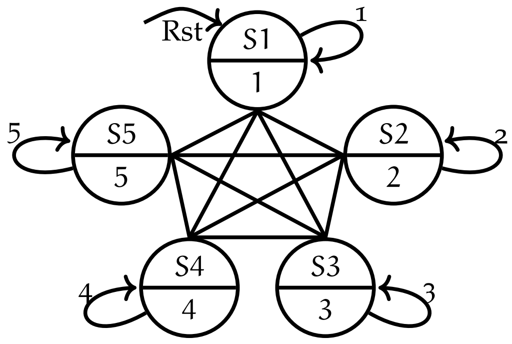

Figure 12.3: Elevator

In Figure 12.3 the various floors are represented by the five states (S1-S5). While the elevator is at a floor then the output from the circuit is the floor number. The elevator is quiescent while in any one state and if a user pushes the button for that same floor then nothing will happen, which is illustrated by the loops starting and coming back to the same state. The star pattern in the middle of the FSM illustrates that the elevator can move directly from one state to any other state. Finally, a Reset signal will automatically move the elevator to state S1.

Using the above state diagram, a simulated elevator could be constructed with *Logisim-evolution* and all of the functions checked using the simulator. Once that is done, the circuit could be realized and implemented on an IC for use in the physical elevator control circuit.