Finite State Machines

An Online Continuing Education Course for Engineers

Course Number: IC-2007

Credit: 2 Hours / 2 PDH / 2 CPD

Finite State Machines

Mark A. Strain, P.E.

Introduction

An electronic lock, a vending machine, a subway turnstile, a control panel for a microwave oven, a spell checker, a text search application, and the core of a microprocessor all embody a common element. Their behavior can be modeled using a finite state machine. Inputs to the system from the real world may affect the state of the system and possibly the output of the system. The behavior of the system is predetermined from its design. All possible outputs and states are designed into the system given any possible input. Therefore, the system is very predictable (assuming all possible state/input/output combinations have been designed into the system). A state machine is one of the most common building blocks of modern digital systems [1].

Description of a State Machine

A finite state machine is a model used to describe the behavior of a real world system. It is a mathematical abstraction used to design digital logic or computer programs [3]. It is a model of behavior composed of a finite number of states, transitions, actions, inputs and outputs [3].
The National Institute of Standards and Technology (NIST) defines a finite state machine as
A model of computation consisting of a set of states, a start state, an input alphabet, and a transition function that maps input symbols and current states to a next state. Computation begins in the start state with an input string. It changes to new states depending on the transition function [2].

Finite state machines are finite in that the number of states used to describe a particular system is limited, i.e., not infinite. The term “finite” is understood since an infinite state machine would be impractical (perhaps even impossible) to model. Hence, they are usually referred to as state machines, also as finite state automaton.

The output of a state machine depends on the history of the system (or current state of the system). However implemented, whether discrete hardware or computer program, a state machine has a finite amount of internal memory to implement the system.
State machines are used to solve a large number of problems. They are used to model the behavior of many different kinds of systems, for example:

- A user interface with a keypad and display (like a microwave oven controller)
- An electronic lock containing a keypad
- A communications protocol that parses the symbols as they are received
- A program that performs a text search (or searches for patterns in strings)

Once the model or state machine is established, the behavior of the system is better understood simply by studying the state diagram.

State Machine Model

Components of a State Machine

A state machine is composed of two or more states. A state stores information about the past and reflects changes from the start of the system to the present state. The current state is determined by past states of the system.

Figure 1 - a single state, S0, in a state machine

A transition indicates a change from one state to another.

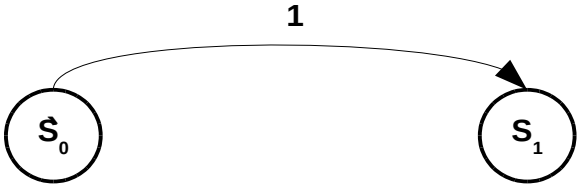

Figure 2 - a transition from state S0 to state S1 as a result of an input of 1

An output, also called an action is a description of an activity that is to be performed as a result of an input and change of state. An output can be depicted either on the transition (the arrow) or within the state.

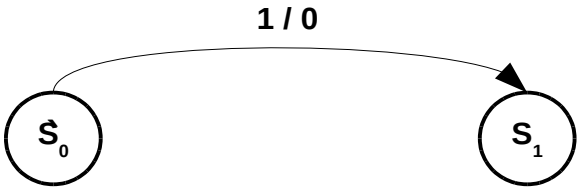

Figure 3 - the output is shown on the transition after the input: input/output

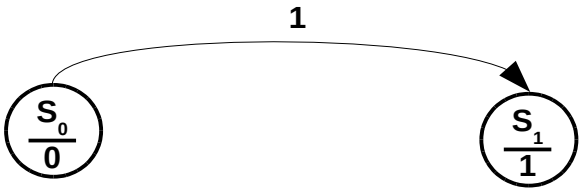

Figure 4 - the output is shown within the state - output is a function of the current state

State Diagram
A state diagram describes a state machine using a graphical representation.

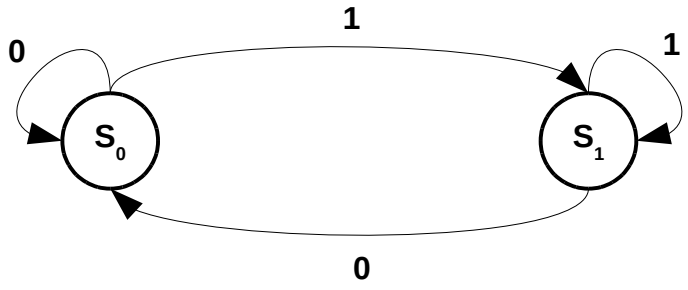

Figure 5 - state diagram

State Table
A state transition table (or state table) describes a state machine in a tabular format.

Present State	Next State x = 0	Next State x = 1
S_0	S_0	S_1
S_1	S_0	S_1

(where x is the input)

Figure 6 - state table

This simple model exemplifies a door lock that embodies two states: LOCKED (S_1) and UNLOCKED (S_0) and two possible inputs: LOCK (1) and UNLOCK (0). If the door is in the UNLOCKED state and an input of LOCK is presented, the state machine progresses to the LOCKED state. If an input of LOCK is presented to the machine in the LOCKED state the machine stays in the LOCKED state.

where

S_0	is unlocked (state)
S_1	is locked (state)
x = 0	to unlock (input)
x = 1	to lock (input)

To summarize, a state machine can be described as:

- A set of possible input events
- A set of possible output events
- A set of states
- An initial state
- A state transition function that maps the current state and input to the next state
- A function that maps states and input to output

Each bubble in a state diagram represents a state, and each arrow represents a transition from one state to another. Inputs are shown next to each transition arrow and outputs are shown under the inputs on the transitions or inside the state bubble.

Block Diagram

Memory is used to store the current state of the state machine. When developing a machine using a hardware architecture, flip-flops are used as the memory device. The number of flip-flops required is proportional to the number of possible states in the state machine.

$\#\text{ of states} \leq 2^x$
(where x is the number of flip-flops required for the state machine)
or
$x \geq \ln(\#\text{ of states}) / \ln 2$

Now, round x up to the nearest integer.

A state machine can be viewed generally as consisting of the following elements: combinational logic, memory (flip-flops or registers), inputs and outputs.

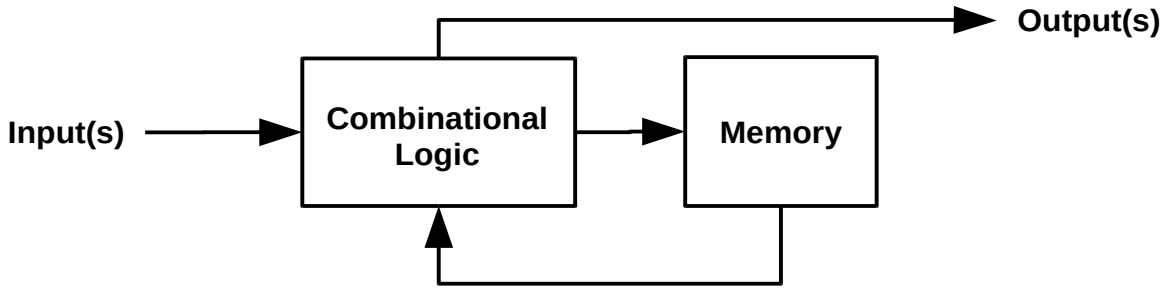

Figure 7 - general block diagram of a state machine

Memory is used to store the state of the system. The combinational logic can be viewed as two distinct functional blocks: a next state decoder and an output decoder [4].

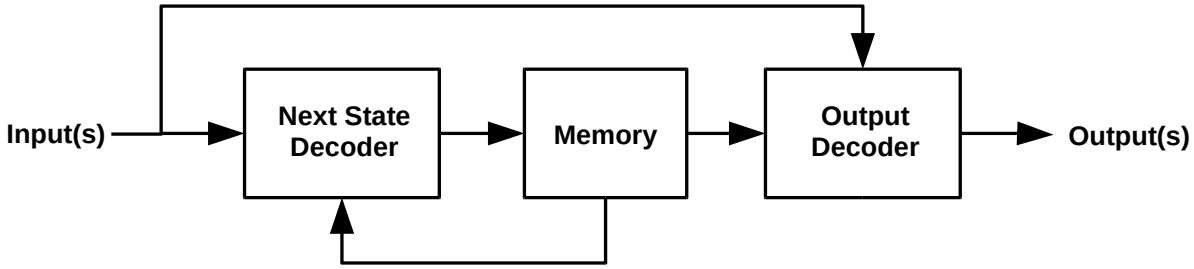

Figure 8 - block diagram of a state machine showing the next state and output decoders

The next state decoder computes the machine's next state and the output decoder computes the output.

Mealy and Moore Machines
Two architectures for state machines include Mealy machines and Moore machines. Each is differentiated by their output dependencies. A Mealy machine's output depends on the input and the current state. A Moore machine's output depends only on the current state.

Mealy Machine
The advantage of a Mealy machine is in its implementation. A Mealy machine often results in a reduced number of states. The output of a Mealy machine depends on the input and the current state. Therefore, the output will be coupled with the input and depicted on the transition between states as shown in Figure 3. The following example is a sequence detector for the sequence {1 0 1}. It is implemented with a Mealy machine.

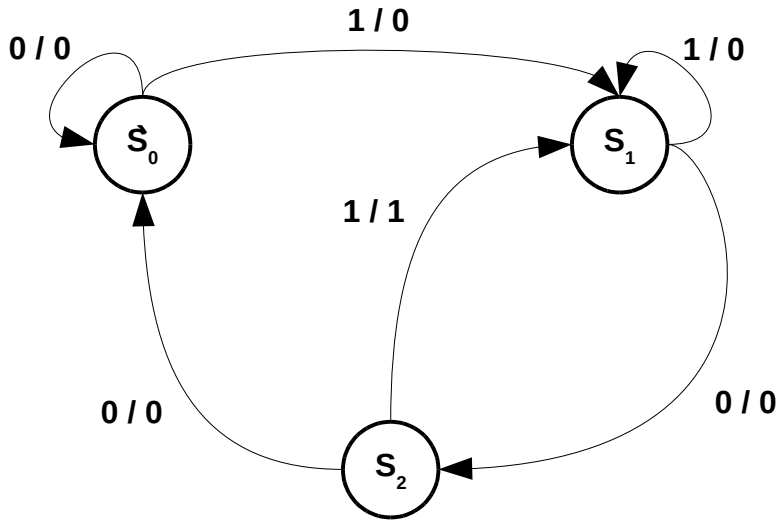

Figure 9 - state diagram of {1 0 1} sequence detector implemented with a Mealy machine

Present State	Next State x = 0 x = 1	Output x = 0 x = 1
S_0	S_0 S_1	0 0
S_1	S_2 S_1	0 0
S_2	S_0 S_1	0 1

(where x is the input)

Figure 10 - state table for above sequence detector

This state machine may be implemented in a number of ways. Consider a hardware implementation using combinational logic and D flip-flops. Since there are three states: S_0, S_1 and S_2, two bits will be required to encode the states thus requiring two D flip-flops. Let the present state be represented by Q_1Q_2 and the next state as the flip-flop equations D_1D_2. The output will be represented as the variable Z. The state table now encoded in binary becomes:

Q_1Q_2	D_1D_2 x = 0	x = 1	Z x = 0	x = 1
0 0	0 0	0 1	0	0
0 1	1 0	0 1	0	0
1 0	0 0	0 1	0	1

(where x is the input,
Q_1Q_2 is the present state,
D_1D_2 is the next state,
and Z is the output)

For D flip-flops, the
In another form the

Using Karnaugh maps t