PDHonline Course G349 (2 PDH)

State Machines

Instructor: Mark A. Strain, P.E.

2020

PDH Online | PDH Center

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone: 703-988-0088
www.PDHonline.com

An Approved Continuing Education Provider

Table of Contents

Introduction 1
Description of a State Machine 1
State Machine Model 2
  Components of a State Machine 2
  State Diagram 3
  State Table 3
  Block Diagram 4
Mealy and Moore Machines 5
  Mealy Machine 5
  Moore Machine 8
Implementation 11
  Hardware Implementation 11
  Software Implementation 13
Summary 15
References 16

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.

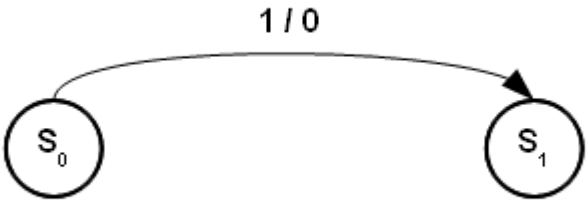

Figure 3 - the output is shown on the transition after the input: input/output

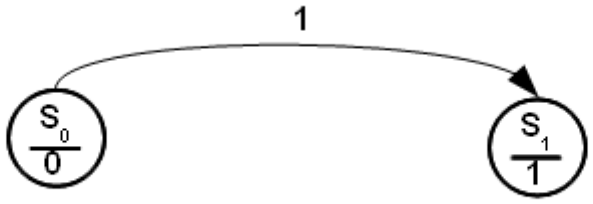

Figure 4 - the output is shown within the state - output is a function of the current state

State Diagram

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

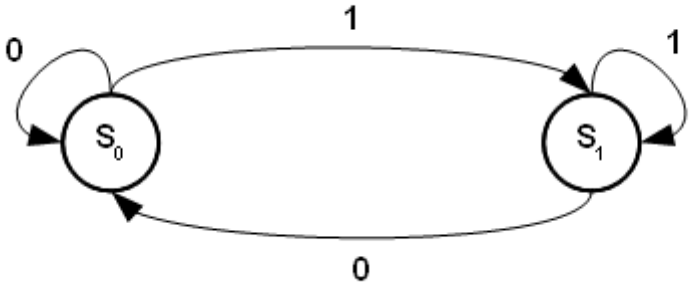

Figure 5 - state diagram

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.

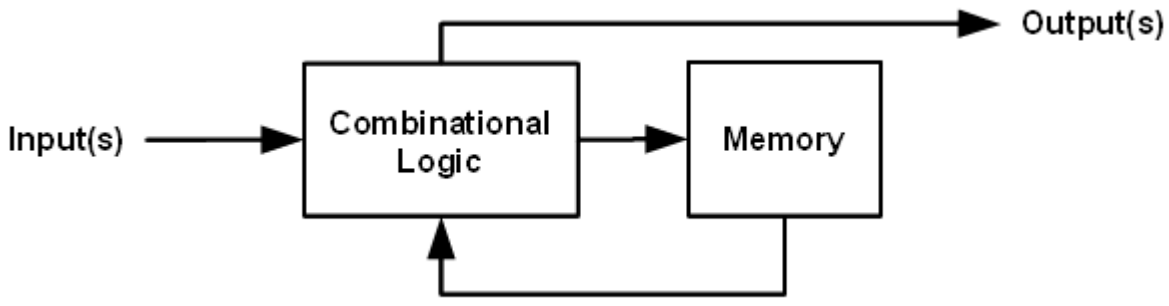

Figure 7 - general block diagram of a state machine

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].

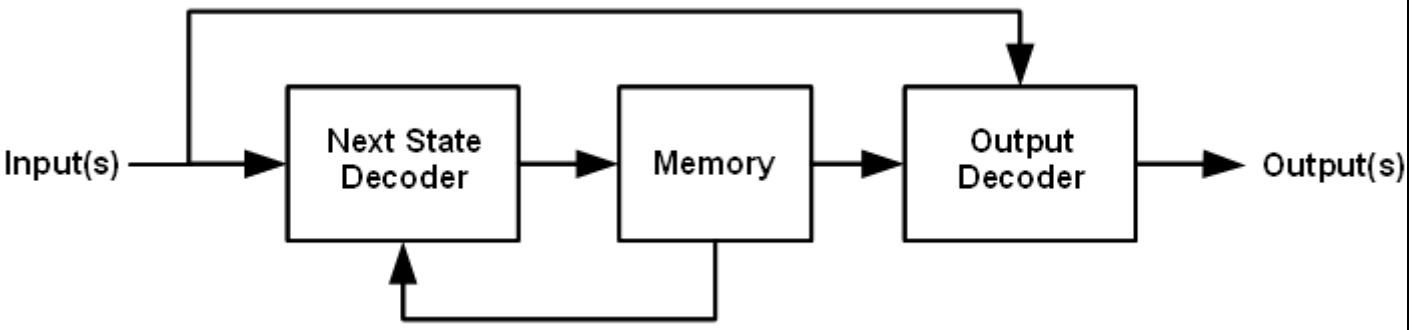

Figure 8 - block diagram of a state machine showing the next state and output decoders

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.

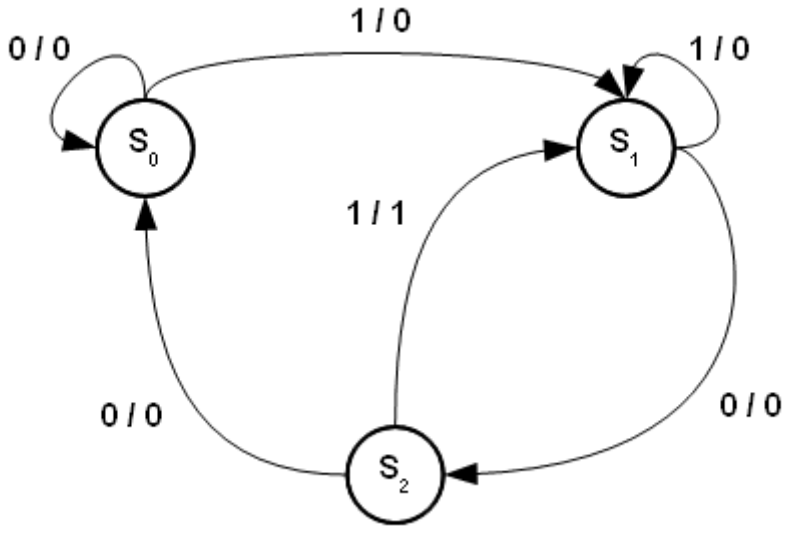

Figure 9 - state diagram of {1 0 1} sequence detector implemented with a Mealy machine

Present State	Next State		Output	
	x = 0	x = 1	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		Z	
	x = 0	x = 1	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)

Figure 11 - state table encoded in binary

For D flip-flops, the characteristic equation translates to $Q^+ = D$.

In another form the state table becomes

Q_1Q_2x	D_1	D_2	Z
0 0 0	0	0	0
0 0 1	0	1	0
0 1 0	1	0	0
0 1 1	0	1	0
1 0 0	0	0	0
1 0 1	0	1	1
1 1 0	-	-	-
1 1 1	-	-	-

Figure 12 - another form of state table

Using Karnaugh maps to reduce the minterms and simplify the equations:

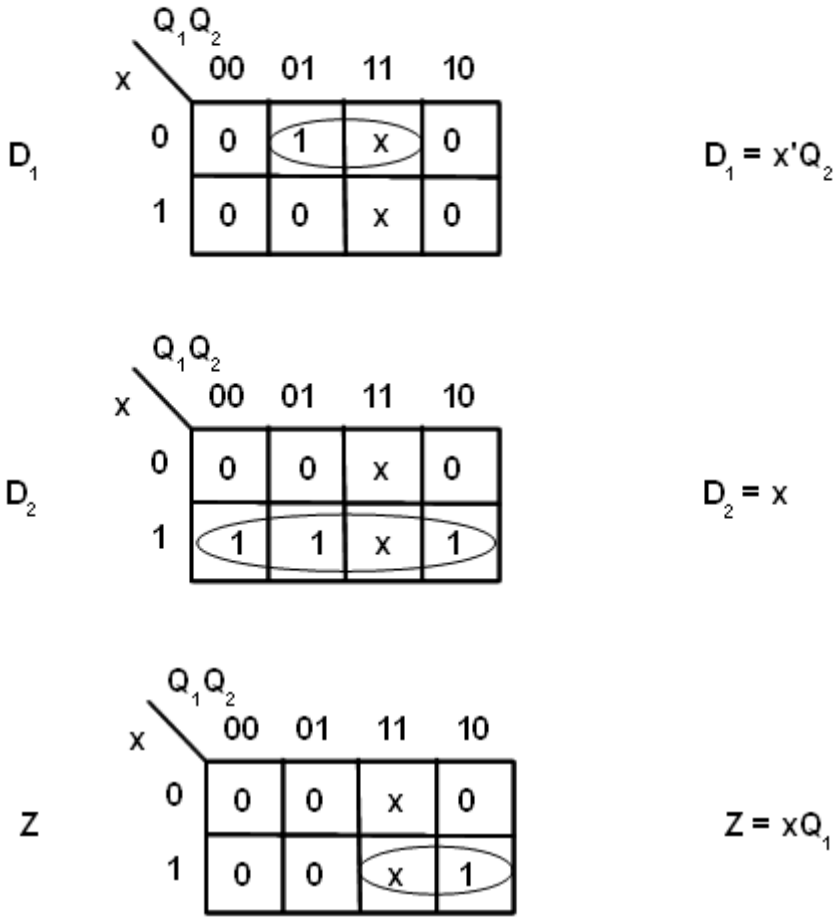

Figure 13 - Karnaugh map of D1, D2, and Z equations

The implementation of the sequence detector {1 0 1} using a Mealy machine architecture becomes

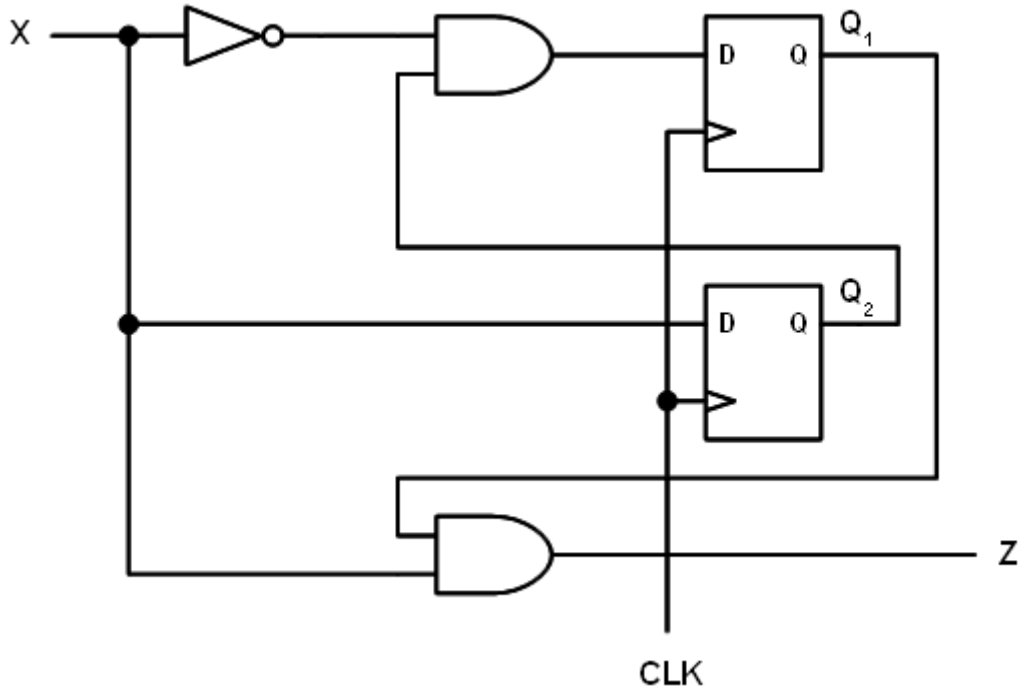

Figure 14 - hardware implementation of Mealy machine

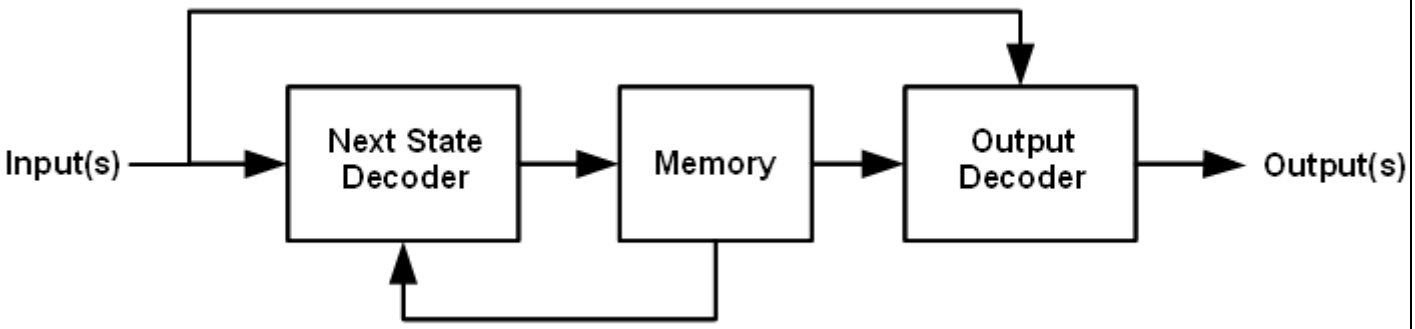

Figure 15 - block diagram of Mealy machine

Moore Machine

The advantage of a Moore machine is a simplification of behavior. The output of a Moore machine depends only on the current state. Therefore, the output is coupled with a state and is depicted within the bubble of the state as shown in Figure 4. The following example is the same {1 0 1} sequence detector as shown above but implemented here as a Moore machine.

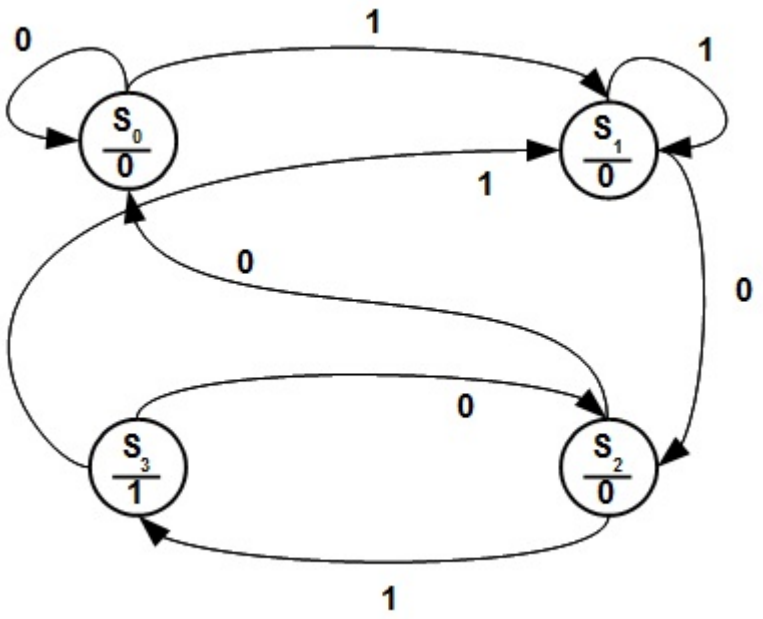

Figure 16 - state diagram of {1 0 1} sequence detector implemented with a Moore machine

Present State	Next State x = 0	Next State x = 1	Output
S_0	S_0	S_1	0
S_1	S_2	S_1	0
S_2	S_0	S_3	0
S_3	S_2	S_1	1

(where x is the input)

Figure 17 - state table for above sequence detector

As in the other example consider a hardware implementation using combinational logic and D flip-flops. Here there are four states: S_0, S_1, S_2 and S_3, and still only 2 bits are needed to define all of the states. This will require two D flip-flops. The present state will be represented as Q_1Q_2 and the next state will be represented by the flip-flop equations D_1D_2. The output will be represented by the variable Z. The state transition table now encoded in binary becomes

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

Figure 18 - state table encoded in binary

or simply

Q_1Q_2x	D_1	D_2	Z
0 0 0	0	0	0
0 0 1	0	1	0
0 1 0	1	0	0
0 1 1	0	1	0
1 0 0	0	0	0
1 0 1	1	1	0
1 1 0	1	0	1
1 1 1	0	1	1

Figure 19 - another form of state table

Using Karnaugh maps to simplify the equations:

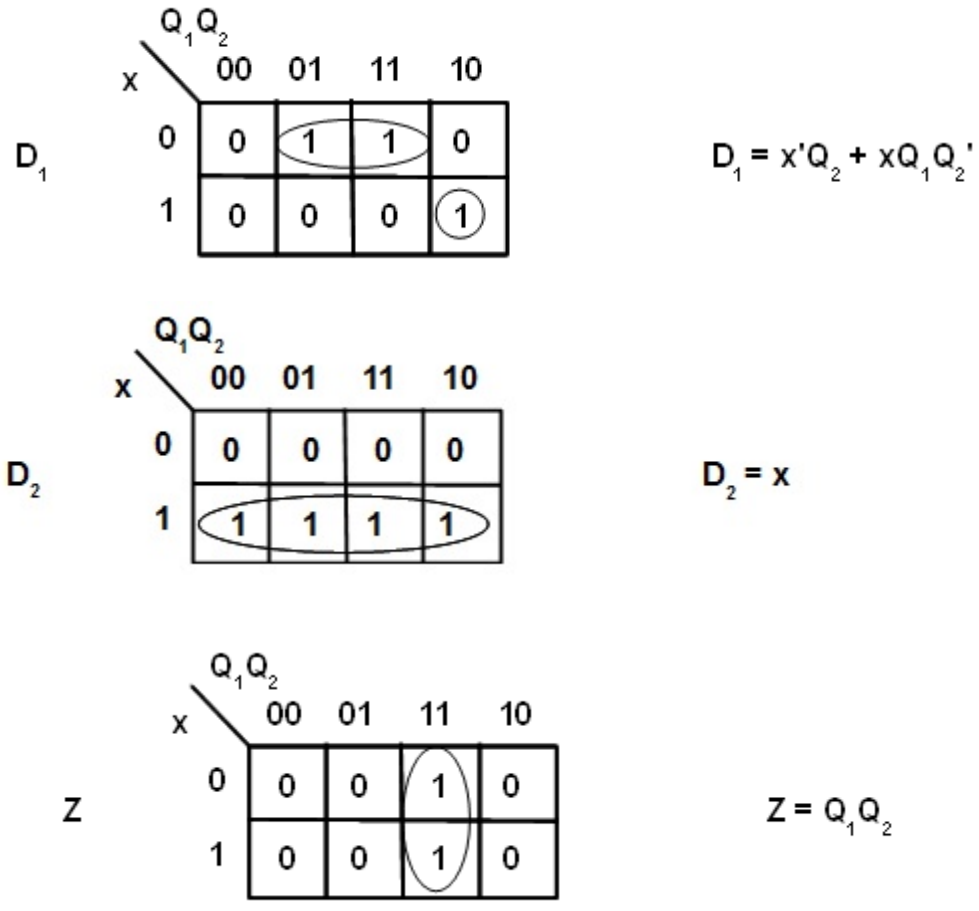

Figure 20 - Karnaugh map of D1, D2, and Z equations

The implementation of the {1 0 1} sequence detector using a Moore machine architecture becomes

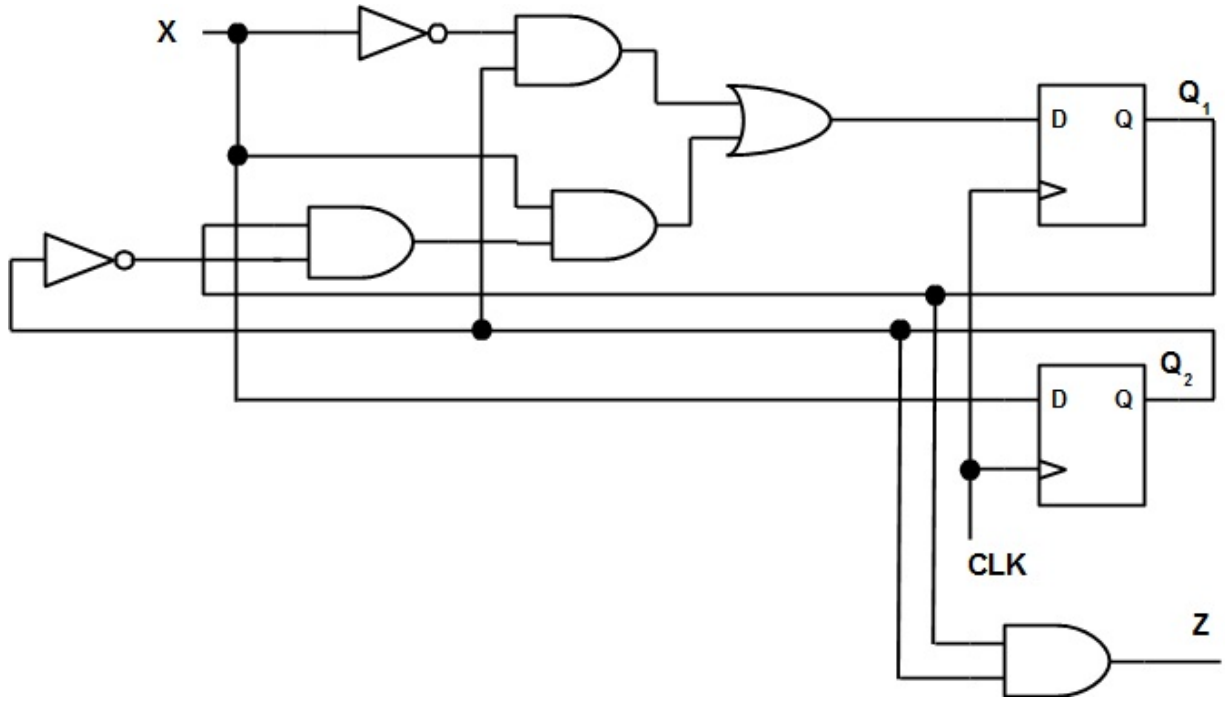

Figure 21 - hardware implementation of Moore machine

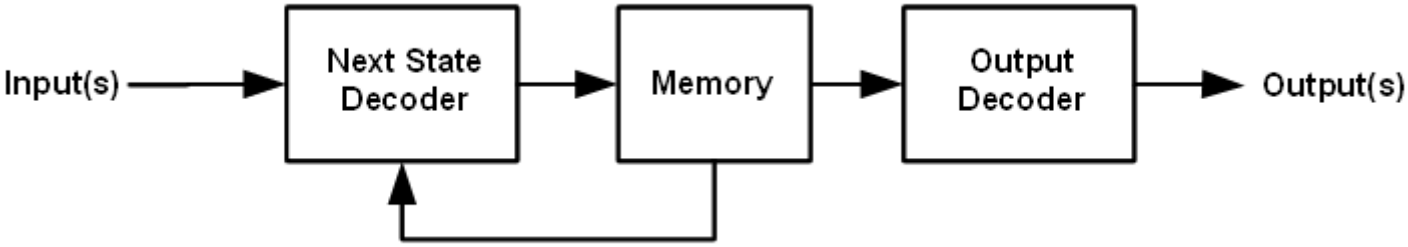

Figure 22 - block diagram of Moore machine

Implementation

Hardware Implementation

A disadvantage of the pure hardware implementation of the state machine using hardwired gates and flip-flops is that the design is difficult to modify once it is committed to copper. Another drawback of the discrete hardware implementation is that it requires significant circuit board area. It is also difficult to debug if there is anomalous behavior. The greatest advantage of a pure hardware implementation is that a hardware realization is very fast compared to a software implementation.

Another form of hardware implementation uses a schematic capture program or a Verilog implementation to produce a binary file which is loaded onto a field programmable gate array (FPGA). This architecture typically requires less board space. However, the schematic capture architecture is sometimes difficult to debug. It is easier to modify the design after it has been

committed to copper. To modify, the schematic or the Verilog firmware is modified and rebuilt and the FPGA is reprogrammed.

The following state table is the Mealy implementation of the {1 0 1} sequence detector

Present State	Next State		Output	
	x = 0	x = 1	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 23 - {1 0 1} sequence detector example

Here is an example of a Verilog realization of the above state machine

```
module mealy_fsm ( clk, reset, out )

input  clk;
input  reset;
output out;

reg [1:0] state;
reg       out;

parameter S0 = 2'd0,
          S1 = 2'd1,
          S2 = 2'd2;


always @(posedge clk)
begin
    if (reset) // this is sync reset, not async reset
    begin
        state <= S0;
        out   <= 1'b0;
    end
    else
    begin
        case (state)
            S0:
                if (x == 1'b1)
                begin
                    state <= S1;
                    out   <= 1'b0;
                end
                else
                begin
                    state <= S0;
                    out   <= 1'b0;
                end
            S1:
                if (x == 1'b1)
                begin
                    state <= S1;
                    out   <= 1'b0;
                end
```

```
                else
                begin
                    state <= S2;
                    out   <= 1'b0;
                end
            S2:
                if (x == 1'b1)
                begin
                    state <= S1;
                    out   <= 1'b1;
                end
                else
                begin
                    state <= S0;
                    out   <= 1'b0;
                end
            default:
                begin
                    state <= S0;
                    out   <= 1'b0;
                end
        endcase
    end
end

endmodule
```

Software Implementation

State machines may also be implemented in software using the C programming language. The code is compiled with a compiler resulting in a binary file which is loaded onto a microprocessor or microcontroller.

A state machine implementation using a software architecture is significantly easier to debug than a hardware implementation using discrete flip-flops and combinational logic. Software is easier to modify than hardware. Simply modify the code, recompile and reload the binary on the microprocessor. Also, the software implementation may be more flexible than the hardware design; it may be ported to different hardware platforms.

The main disadvantage of the software implementation is that it may be slower than the hardware implementation.

The following state table is the Mealy implementation of the same {1 0 1} sequence detector shown above.

Present State	Next State		Output	
	x = 0	x = 1	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 24 - {1 0 1} sequence detector example

Here is an example of a C programming language realization of the above state machine

```
typedef enum
{
    S0,
    S1,
    S2
} StateType;

void MealyFSM(StateType *state,
              int x,
              int *out)
{

    switch (*state)
    {
        case S0:
            if (x == 0)
            {
                *state = S0;
                *out = 0;
            }
            else
            {
                *state = S1;
                *out = 0;
            }
            break;

        case S1:
            if (x == 0)
            {
                *state = S2;
                *out = 0;
            }
            else
            {
                *state = S1;
                *out = 0;
            }
            break;

        case S2:
            if (x == 0)
            {
                *state = S0;
                *out = 0;
            }
            else
            {
                *state = S1;
                *out = 1;
            }
            break;
```

```
            default:
                *state = S0;
                *out = 0;
                break;

        }
    }
```

Summary

A state machine is a model used to describe the behavior of a real world system. State machines are used to solve a large number of problems. They are used to model the behavior of various types of devices such as electronic control devices, parsing of communications protocols and programs that perform text or pattern searches.

State machines may be described using a state diagram and a state table. A state diagram is composed of states, inputs, outputs and transitions between states. A state table describes a state machine with the present state and input on the left and the next state and output on the right.

State machines may be implemented using either a hardware architecture or a software architecture. The advantage of a hardware implementation is that it operates very fast, but it is difficult to modify and usually requires more circuit board space. The advantage of a software implementation is that it is easier to design and modify, but can be slower than the hardware equivalent.

References

1. "A New Paradigm for Synchronous State Machine Design in Verilog." visited 1 November 2010 <http://ideaconsulting.com/smv.pdf>

2. "Finite State Machine – National Institute of Standards and Technology." 12 May 2008 <http://xw2k.nist.gov/dads/HTML/finiteStateMachine.html>

3. "Finite-State Machine – Wikipedia, the Free Encyclopedia." 8 July 2010 <http://en.wikipedia.org/wiki/Finite-state_machine>

4. "State Machine Design." June 1993 <http://www.mil.ufl.edu/4712/docs/PLD_Basics/StateMachineDesign.pdf>

5. "State Machines." 8 September 2010 <http://www.xilinx.com/itp/xilinx4/data/docs/xst/hdlcode15.html>

6. "UML Tutorial: Finite State Machines." June 1998 <http://www.objectmentor.com/resources/articles/umlfsm.pdf>