CHAPTER 3 Synchronous Sequential Logic System

OBJECTIVES :

The learning outcome is that student is able to:

- *Differentiate between Moore's and Mealy's finite state machine.*
- *Design and implement any synchronous sequential circuits by constructing state diagrams, state tables and excitation tables.*
- *Write the Verilog codes for any synchronous sequential circuits.*
- *Draw the output waveforms of any synchronous sequential circuits given the input waveforms.*
- *Design minimum cost FSM circuits by proper state assignment using the into-rule, out-of rule and output rule.*
- *Reduce the number of states in an FSMs by partitioning method.*
- *Analyze and describe the function of any given FSM circuits.*

1 Introduction

The coverage for this chapter is a general class of circuits in which the outputs depend on the past behaviour of the circuit, as well as on the present values of inputs. They are called *sequential circuits*. In most case a clock signal is used to control the operation of a sequential circuit; such a circuit is called a *synchronous* sequential circuit. The alternative, in which no clock signal is used, is called an *asynchronous* sequential circuit.

Synchronous sequential circuits are realized using combinational logic and one or more flip-flops. The general structure of such a circuit is shown in Figure 1. The circuit has a set of primary inputs, A, and produces a set of outputs, Z. The stored values in the flip-flops are referred to as the state, Q, of the circuit. Under control of the clock signal, the flip-flops change their state as determined by the combinational logic that feeds the inputs of these flip-flops. Thus the circuit moves from one state to another. To ensure that only one transition from one state to another takes place during one clock cycle, the flip-flops have to be of the *edge-triggered* type. The active clock edge can be PGT or NGT.

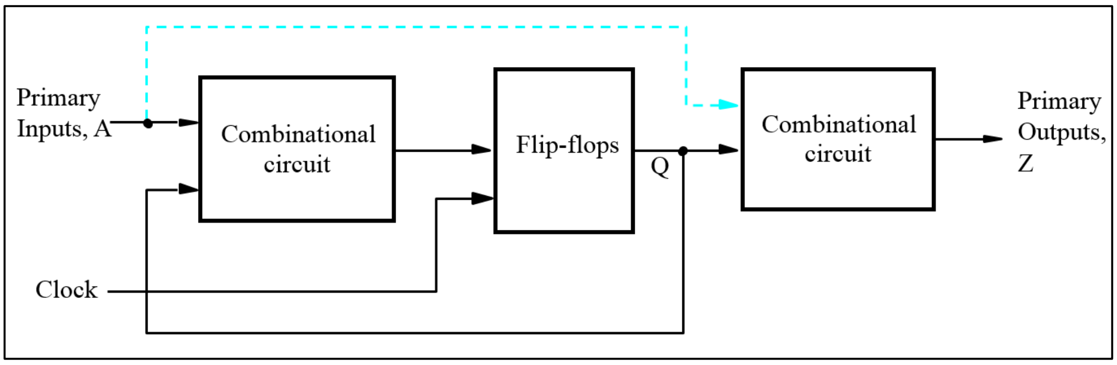


Figure 1. General block diagram of a synchronous sequential circuit.

The dotted line in Figure 1 may or may not exist. As can be seen from Figure 1, the outputs of synchronous sequential circuit always depends on the present state of the flip-flops, Q. However, it may or may not depends on the primary inputs. To distinguish between these two possibilities, it is customary to say that sequential circuits whose outputs depend only on the present state of the circuit are of *Moore* type, while those whose outputs depend on both the present state and the primary inputs are of *Mealy* type.

Sequential circuits are also called *finite state machines* (FSMs), which is a more formal name that is often found in technical literature. The name derive from the fact that the functional behaviour of these circuits can be represented using a finite number of states.

2 General Design Steps

Sequential circuits are often used to control the operation of physical systems. The design of a speed controller of an automatically-controlled vehicle is explained here in order to illustrates the general design steps of any synchronous sequential logic system,

The vehicle is designed to run at some predetermined speed. However, due to some operational conditions the speed may exceed the desirable limit, in which case the vehicle has to be slowed down. To determine when such action is needed, the speed is measured at regular intervals. Let a binary signal, w indicate whether the speed exceeds the required limit, such that $w = 0$ means that the speed is within acceptable range and $w = 1$ indicates excessive speed. The desired control strategy is that if $w = 1$ during two or more consecutive measurements, a control signal z must be asserted to cause the vehicle to slow down. Thus, $z = 0$ allows the current speed to be maintained, while $z = 1$ reduces the speed. Let a signal, *Clock* define the required timing intervals, such that the speed is measured once during each clock cycle. In summary, the design must meets the following specification:

1. The circuit has one input, w, and one output, z.
2. All changes in the circuit occur on the positive edge of the clock signal.
3. The output z is equal to 1 if during two consecutive clock cycles the input w was equal to 1. Otherwise, the value of z is equal to 0.

Figure 2 shows an example of the output, z with a sample of the input, w at different clock cycles.

Clockcycle:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
w:	0	1	0	1	1	0	1	1	1	0	1
z:	0	0	0	0	1	0	0	1	1	0	0

Figure 2.

This design is also known as an "11" *sequence detector* with repeat allowed.

2.1 State Diagram

The first step in designing a finite state machine is to determine how many states are needed and which transitions are possible from one state to another. There is no set procedure for this task. The designer must think carefully about what the machine has to accomplish. A good way to begin is to select one particular state as a starting state; this is the state that the circuit should enter when power is first turned on or when a reset signal is applied. For this example, let us assume that the starting state is called state A. As long as the input, *w* is 0, the circuit need not do anything, and so each active clock edge should result in the circuit remaining in state A. When *w* becomes equal to 1, the machine should recognize this, and move to a different state, B. This transition takes place on the next active clock edge *after* w has become equal to l. In state B, as in state A, the circuit should keep the value of output *z* at 0, because it has not yet seen *w* = l for two consecutive clock cycles. When in state B, if w is 0 at the next active clock edge, the circuit should move back to state A. However, if *w* = 1 when in state B, the circuit should change at the next active clock edge to a third state, C and it should then generate an output *z* = l since *w* = 1 for two consecutive clock cycles. The circuit should remain in state C as long as *w* = 1 and should continue to maintain *z* = l. When w becomes 0, the machine should move back to state A. Since the preceding description handles all possible values of input w that the machine can encounter in its various states, it can be concluded that three states are needed to implement the desired machine.

Figure 3 shows the Moore's type state diagram that shows the behaviour of the machine as describe in the previous paragraph. In the state diagram, ellipse (or circle) represent a state, arrow is a transition from one state to another when the input condition is met on the active clock edge. *Reset* is an input used to force the circuit into the reset state A or upon power-up of the machine.

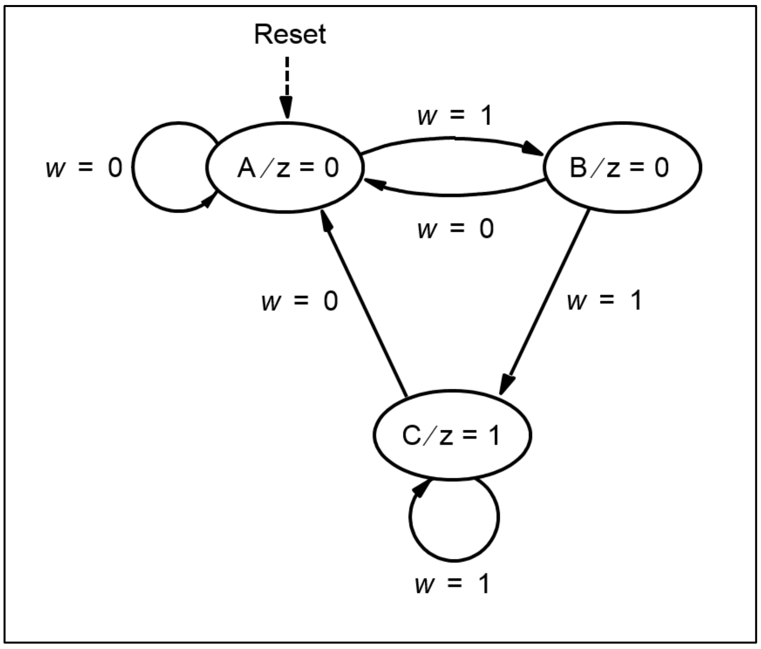


Figure 3. State Diagram

Example 1

Draw the state diagram for a FSM with the following specifications:

1. The circuit has one input, *w*, and one output, *z*.
2. All changes in the circuit occur on the positive edge of the clock signal.
3. The output *z* is equal to 1 if the input sequence for w is 101 for any three consecutive clock cycles. Otherwise, the value of z is equal to 0.

Solution:

2.2 State Table

Although the state diagram provides a description of the behavior of a sequential circuit that is easy to understand, to proceed with the implementation of the circuit it is convenient to translate the information contained in the state diagram into a tabular form. Figure 4 shows the state table for the design. The table indicates all transitions from each present state to the next state for different values of the input signal. Note that the output *z* is specified with respect to the present state, namely, the state that the circuit is in at present time. Note also that it did not include the *Reset* input; instead, it is implicitly assumed that the first state in the table is the starting state.

Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Figure 4. State Table

<u>Example 2</u>

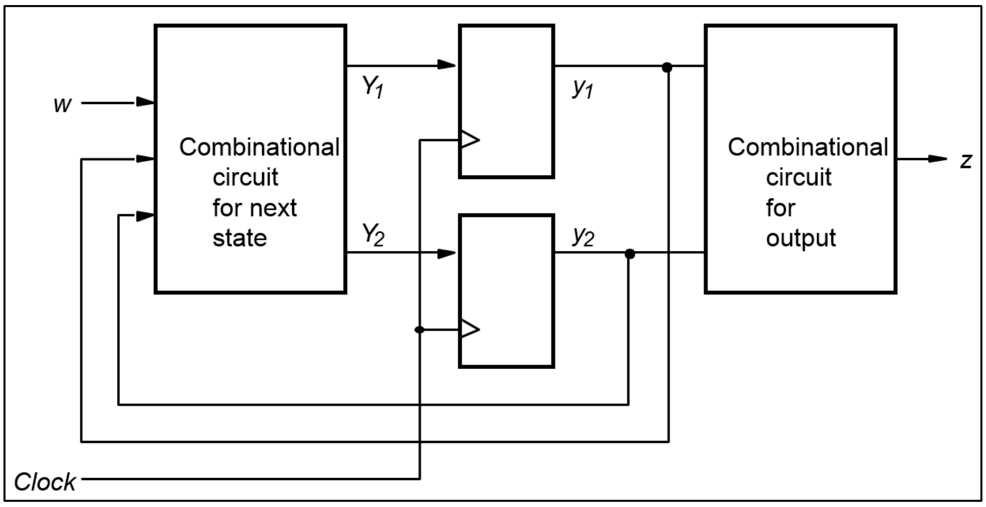
Derive the state table for the state diagram in Example 1.

Solution:

2.3 State Assignment

The state table in Figure 4 defines the three states in terms of letters A, B, and C. When implemented in a logic circuit, each state is represented by a particular combination of state variables. Each state variable may be implemented in the form of a flip-flop. There are a few state assignment scheme available depending on simplicity, speed and hazards due to race conditions of different signal paths. For simplicity, simple binary state assignment is used in this design. Procedure for proper state assignment that leads to a simpler circuit will be covered in Section 7.

Using simple binary state assignment, the number of bit variables, n required is governed by the inequality, $2^n \geq S$, where S is the number of states and n is smallest integer satisfying this inequality. In this design, S=3, hence, n = 2. Let these variables be y_1 and y_2. Two flip-flops represent these state variables as shown in Figure 5.


w
Combinational circuit for next state
Y1
Y2
y1
y2
Combinational circuit for output
z
Clock
Figure 5. Block diagram of sequential Moore's FSM with two flip-flops.

The next step in the design process is to create a truth table that defines both combinational circuits in Figure 5. To produce the desired truth table, a specific combination of variables y_1 and y_2 are assigned to each state. One possible assignment is given in Figure 6, where the states A, B, and C are represented by y_2y_1 = 00, 01, and 10, respectively. The fourth combination, y_2y_1 = 11, is not needed in this case. The type of table given in Figure 6 is usually called a *state-assigned* table. This table can serve directly as a truth table for the output *z* with the inputs y_1 and y_2. Although for the next-state functions Y_1 and Y_2 the table does not have the appearance of a normal truth table, because there are two separate columns in the table for each value of *w*, it is obvious that the table includes all of the information that defines Y_1 and Y_2 in terms of combinations of inputs w, Y_1, and Y_2.

	Present state	Next state		Output
		$w = 0$	$w = 1$	z
	y_2y_1	Y_2Y_1	Y_2Y_1	
A	00	00	01	0
B	01	00	10	0
C	10	00	10	1
	11	*dd*	*dd*	*d*

Figure 6. State-assigned table

2.4 Excitation Table

From the state-assigned table in Figure 6, the logic expressions for the next-state and output functions can be derived. The derivation of logic expressions for the next-state depends on the types of flip-flops used. For D-type flip-flops, the values of Y_1 and Y_2 are simply clocked into the flip-flops to become the new values of y_1 and y_2 . In other words, if the inputs to the flip-flops are called D_1 and D_2, then these signals are the same as Y_1 and Y_2. Note that the diagram in Figure 5 corresponds exactly to the use of D-type flip-flops. In other words, the *excitation* table for D-type flip-flop is same as the state-assigned table. Excitation table tabulates the input combinations to get the corresponding inputs excitation for the flip-flops.

2.5 Derivation of Output and Flip-Flops Inputs logic equations

For implementation using D-type flip-flop, Figure 6 can be used to derive the output and flip-flops inputs logic equations via Boolean algebra or Karnaugh map technique. Making full use of the don't care condition in the present state code, 11 to obtain the simplest circuit, the Karnaugh map method is used to derive the output z, flip-flops inputs, D_1 and D_2 as shown in Figure 7.

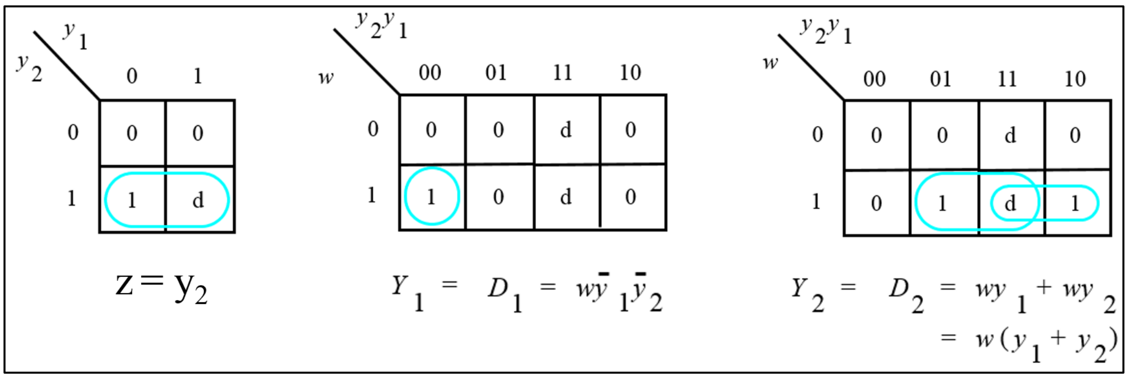


Figure 7. K-map for *z,* D_1 and D_2

Figure 8 shows the circuit designed using D-type flip-flops. Observe that a clock signal is included, and the circuit is provided with an active-low reset capability. Connecting the clear input on the flip-flops to an external *Resetn* signal, as shown in the figure, provides a simple means for forcing the circuit into a known state. If the signal, Resetn = 0, then both flip-flops will be cleared to 0, placing the FSM into the state y_2y_1 = 00.

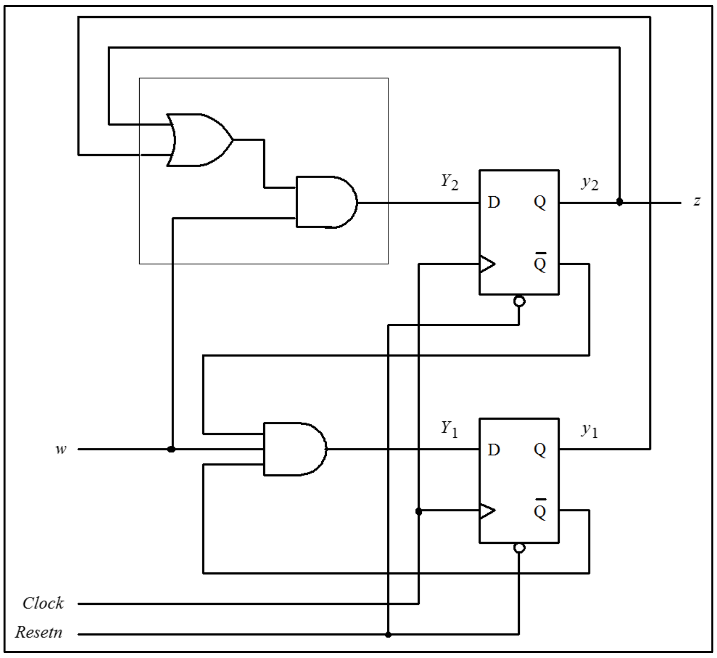


Figure 8. “11” sequence detector implemented using D-type flip-flops

Figure 9 shows timing diagram of the circuits in Figure 8. Since the flip-flops are PGT, all changes in the signals occur shortly after the positive edge of the clock. The amount of delay from the clock edge depends on the propagation delays through the flip-flops. Note that the input signal, *w* is also shown to change slightly after the active edge of the clock. This is a good assumption because in a typical digital system an input such as *w* would be just an output of another circuit that is synchronized by the same clock. A key point to observe is that even though *w* changes slightly after the active clock edge, and thus the value of *w* is equal to I (or 0) for almost the entire clock cycle, no change in the circuit will occur until the beginning of the next clock cycle when the positive edge causes the flip-flops to change their state. Thus the value of *w* must be equal to 1 for two clock cycles if the circuit is to reach state C and generate the output z = 1.

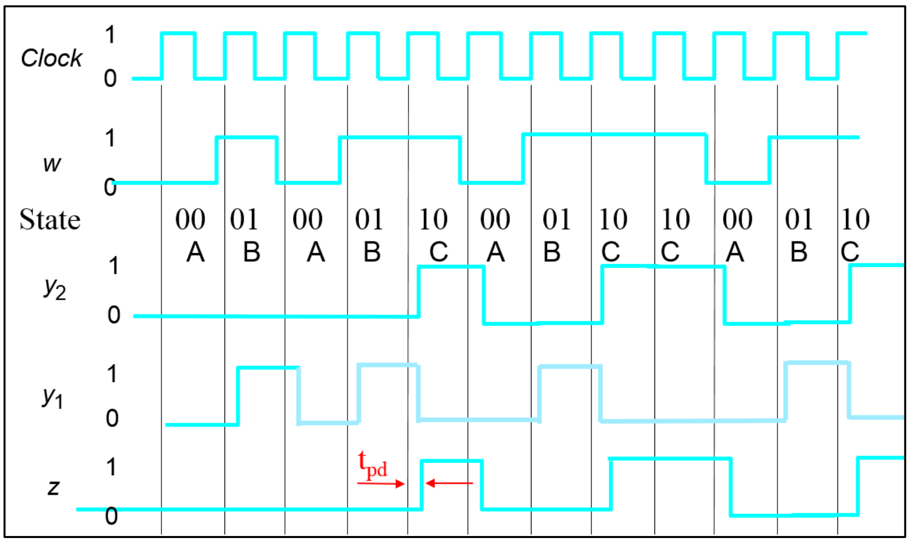


Figure 9. Timing diagram of “11” sequence detector

Example 3

Derive the state-assigned table for the FSM in Example 1 and hence draw the schematic diagram of the FSM using D-type flip-flops.

Solution:

2.6 Summary of general procedure for designing synchronous FSM

The steps involved in designing a synchronous sequential circuit as follows:

1. Obtain the specification of the desired FSM.
2. Draw the state diagram.
3. Derive the state table.
4. Assign state code.
5. Derive the state-assigned table and/or excitation table.
6. Derive the output & flip-flop input equations.
7. Implement the circuit from the equations derived.

3 Verilog for Moore type "11" sequence detector

Verilog does not define a standard way of describing a finite state machine. Hence while adhering to the required Verilog syntax, there is more than one way to describe a given FSM. An example of Verilog code for the Moore type "11" sequence detector of Figure 3 is shown in Figure 10. The code reflects directly the FSM structure in Figure 5. The module *simple* has inputs *Clock*, *Resetn*, and *w*, and output *z*. Two-bit vectors *y* and *Y* represent the present and the next state of the machine, respectively. The parameter statement associates the present-state combinations used in Figure 6 with the symbolic names A, B, and C.

The state transitions are specified by two separate *always* blocks. The first block describes the required combinational circuit. It uses a case statement to give the value of the next state *Y* for each value of the present state *y*. Each case alternative corresponds to a present state of the machine, and the associated *if-else* statement specifies the next state to be reached according to the value of *w*. This portion of the code corresponds to the combinational circuit on the left side of Figure 5.

Since there are only three states in our FSM, the case statement in Figure 10 includes a default clause that covers the unused combination y = 11. This clause specifies Y = 2'bxx, which indicates to the Verilog compiler that the value of Y can be treated as a don't-care when y = 11. It is important to consider the effect on the compiler if the default clause is not included. In that case, the value of Y would not be specified at all for y = 11. The Verilog compiler assumes that a signal's value must be kept unchanged whenever the value is not specified by an assignment, and a memory element should be synthesized. For the code in Figure 10 this means that a latch will be generated in the combinational circuit for Y if the default clause is not included. In general, it is always necessary to include a default clause when using a case statement that does not cover all of its alternatives.

The second *always* block introduces flip-flops into the circuit. Its sensitivity list comprises the *reset* and *clock* signals. Asynchronous reset is performed when the *Resetn* input goes to 0, placing the FSM into state A. The *else* clause stipulates that after each positive clock edge the *y* signal should take the value of the *Y* signal, thus implementing the state changes.

This is a Moore-type FSM in which the output z is equal to 1 only in state C. The output is conveniently defined in a conditional assignment statement that sets z = I if y = C. This realizes the combinational circuit on the right side of Figure 5. Note that there are many ways to write the Verilog code for the same design.

```
module simple (Clock, Resetn, w, z);
   input Clock, Resetn, w;
   output z;
   reg [2: l] y, Y;
   parameter [2:1] A=2'b00, B=2'b01, C=2'b10;
   // Define the next state combinational circuit
   always @(w,y)
     case (y)
       A: if (w) Y=B;
           else Y=A;
       B: if (w) Y=C;
           else Y=A;
       C: if (w) Y=C;
           else Y=A;
       default: Y = 2'bxx;
     endcase
   // Define the sequential block
   always @(negedge Resetn, posedge Clock)
      if (Resetn==0)  y<= A;
      else y <= Y;
   // Define output
   assign z = (y == C);
endmodule
```

Figure 10. Verilog code for Figure 3

4 Mealy type "11" sequence detector

In this section, the same "11" sequence detector in Section 2 is designed but using Mealy-type machines in which the output values are generated based on both the state of the circuit and the present values of its inputs.

4.1 State diagram of Mealy's FSM

Figure 11 shows the Mealy's state diagram for the "11" sequence detector. Again, state A is the starting state. As long as $w = 0$, the machine should remain in state A, producing an output $z = 0$. When $w = 1$, the machine has to move to a new state, B, to record the fact that an input of 1 has occurred. If w remains equal to 1 when the machine is in state B, which happens if $w = 1$ for at least two consecutive clock cycles, the machine should remain in state B and produce an output $z = 1$. As soon as w becomes 0, z should immediately become 0 and the machine should move back to state A at the next active edge of the clock. Only two states are needed because the output value depend on the present value of the input as well as the present state of the machine. The diagram indicates that if the machine is in state A, it will remain in state A if $w = 0$ and the output will be 0. This is indicated by an arrow with the label $w = 0/z = 0$. When w becomes 1, the output stays at 0 until the machine moves to state B at the next active clock edge. This is denoted by the arrow from A to B with the label $w = 1 / z = 0$. In state B the output will be 1 if $w = 1$, and the machine will remain in state B, as indicated by the label w = l/z = l on the corresponding arrow. However, if $w = 0$ in state B, then the output will be 0 and a transition to state A will take place at the next active clock edge. A key point to understand is that during the present clock cycle the output value corresponds to the label on the arrow emanating from the present-state node. Note the difference between the Mealy's and Moore's state diagram. In the Moore's state diagram, the output is stated in the state (circle or ellipse) whereas in the Mealy's state diagram, the output is stated in the transition (arrow).

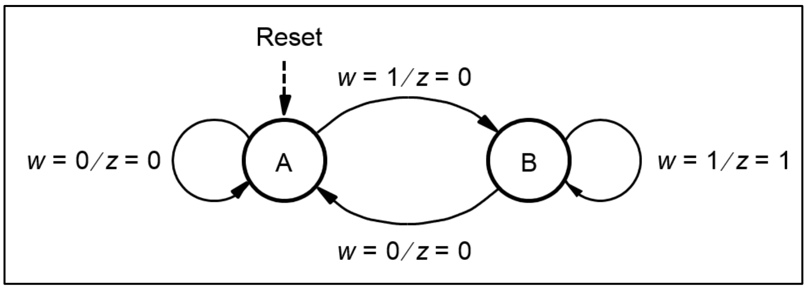


Figure 11. Mealy's State Diagram of "11" sequence detector

4.1 State table of Mealy's FSM

From Figure 11, the state table can be derived as shown in Figure 12. Again note here that the Mealy's state table is different from Moore's state table in that the output z now has two columns – one for $w = 0$ and another for w = 1, since the output z depends on both the present state and input w.

Present state	Next state		Output z	
	$w = 0$	$w = 1$	$w = 0$	$w = 1$
A	A	B	0	0
B	A	B	0	1

Figure 12. Mealy's State Table of "11" sequence detector

4.2 State-assigned table of Mealy's FSM

Since there is only two states, only one state variable, y is required. Representing state A with $y = 0$ and state B with $y = 1$, result in the state-assigned table as shown in Figure 13.

	Present state	Next state		Output	
		$w = 0$	$w = 1$	$w = 0$	$w = 1$
	y	Y	Y	z	z
A	0	0	1	0	0
B	1	0	1	0	1

Figure 13. Mealy's State-assigned Table of "11" sequence detector

4.3 Excitation table of Mealy's FSM

If the FSM is designed using D-type flip-flop, the state-assigned table in Figure 13 can be used to derive the input of the D-type flip-flop.

4.4 Deriving output and next state equations of Mealy's FSM

Figure 13 can be used to derive the output and flip-flops inputs logic equations via Boolean algebra or Karnaugh map technique. If possible, make full use of the don't care condition to obtain the simplest circuit. From Figure 13, it can be seen straight away that z = wy and D = w.

From the derivation of the output z, flip-flop input, D equation, the final circuit is shown in Figure 14. Comparing Figure 14 with Figure 8, it can be concluded that Mealy's FSMs usually used less components than Moore's FSMs. However, Mealy's FSMs is more difficult to design and care must be taken to ensure that the active clock edge must synchronize correctly with the change in input as describe in the next paragraph.

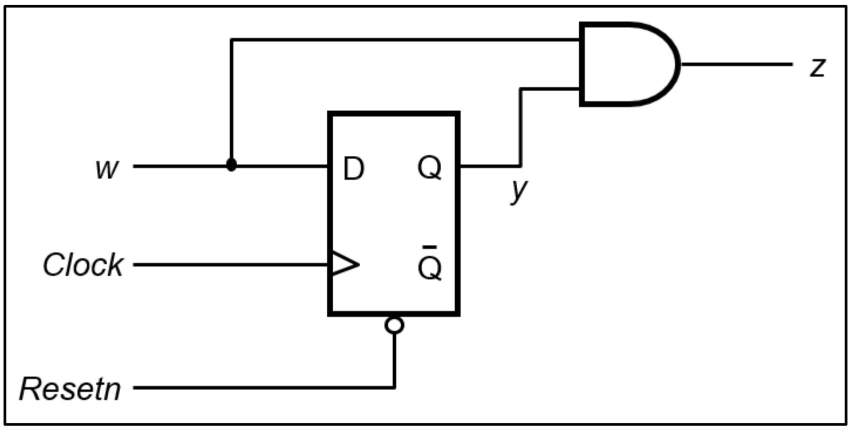


Figure 14. Mealy's circuit for "11" sequence detector

Figure 15 shows the timing diagram of the circuit in Figure 14 when the input w is synchronized with the active clock edge. The circuit functions correctly and output $z = 1$ when w is one for two consecutive clock cycles. Figure 16 shows another timing diagram where input w changes at the negative clock edge. Here, there is an error in the output at time, t=250 ns where $z = 1$ when $w = 1$ only for one clock cycle.

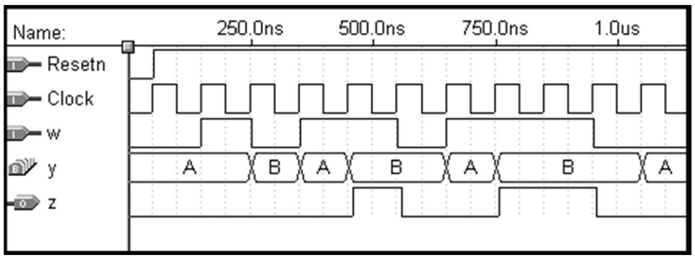


Figure 15. Timing diagram for Mealy's circuit with w synchronized with active clock edge

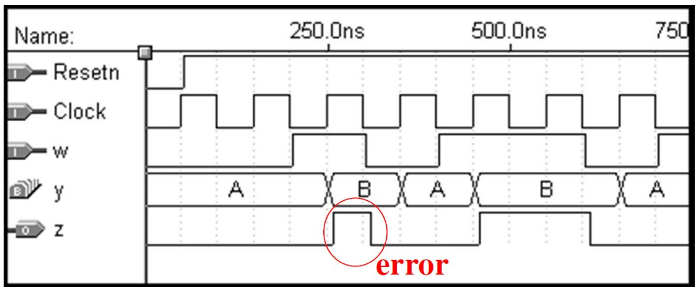


Figure 16. Timing diagram for Mealy's circuit with w changes at negative clock edge

5 Verilog for Mealy type "11" sequence detector

A Mealy-type FSM can be coded in a similar manner as a Moore-type FSM. Figure 17 shows the Verilog code for the FSM in Figure 14. The state transitions are described in the same way as the Verilog code in Figure 10. The variables y and Y represent the present and next states, which can have values A and B. Compared to the code in Figure 10, the main difference is the way in which the code for the output is written.

In Figure 17, the output *z* is defined within the case statement that also defines the state transitions. When the FSM is in state A, *z* should be 0, but when in state B, *z* should take the value of *w*. Since the sensitivity list for the always block includes *w*, a change in *w* will immediately reflect itself in the value of *z* if the machine is in state B, which meets the requirements of the Mealy-type FSM – that is, output *z* depends both on the present state and primary input, *w*.

```
module mealy_Fig13 (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg y, Y;
    parameter A=1'b0, B=1'b1;
    // Define the next state and output combinational circuit
    always @(w,y)
      case (y)
        A: if (w)
            begin
                z=0;
                Y=B;
            end
            else
            begin
               z=0;
               Y=A;
            end
        B: if (w)
            begin
                z=1;
                Y=B;
            end
            else
            begin
               z=0;
               Y=A;
            end
      endcase
    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
       if (Resetn==0)  y<=A;
       else y <= Y;
 endmodule
```

Figure 17. Verilog code for Figure 14