DIGITAL LOGIC

Chapter 6: Synchronous Sequential Circuits

Ru Han
Fall

OUTLINE

Synchronous Sequential Circuits

- Design techniques for circuits that use flip-flops
- The concept of states and their implementation with flip-flops
- Synchronous control by using a clock signal
- Sequential behavior of digital circuits
- A complete procedure for designing synchronous sequential circuits
- Verilog specification of sequential circuits
- The concept of finite state machines

SOME CONCEPTS

- ***Combinational circuit***

In a *combinational* circuit, the values of the outputs are determined solely by the present values of its inputs.

- ***Sequential circuit***

In a *sequential* circuit, the values of the outputs depend on the past behavior of the circuit, as well as the present values of its inputs.

Sequential circuits are also called *finite state machines* (FSMs) or simply *machine* .

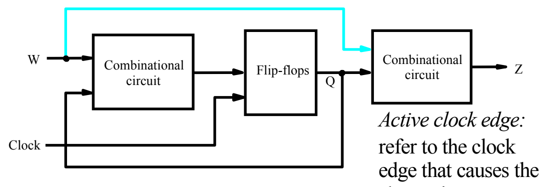
SEQUENTIAL CIRCUITS

- Sequential circuits can be:
  - *Synchronous* – where flip-flops are used to implement the states, and a clock signal is used to control the operation.
  - *Asynchronous* – where no common clock is used.

SYNCHRONOUS SEQUENTIAL CIRCUITS

Synchronous sequential circuits are realized using combinational logic and one or more flip-flops.

SYNCHRONOUS SEQUENTIAL CIRCUITS

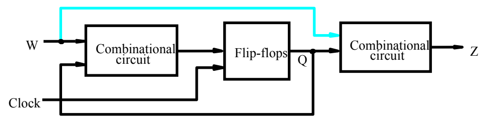


Active clock edge: refer to the clock edge that causes the change in state.

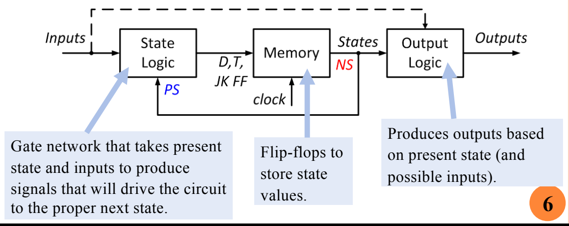
Figure 6.1. The general form of a synchronous sequential circuit.

- If the outputs depend only on the present state, the circuit is said to be of Moore type.
- If the outputs depend on both the present state and the present values of the inputs, the circuit is said to be of Mealy type.

SYNCHRONOUS SEQUENTIAL CIRCUITS



Finite State Machine: General Block Diagram


Inputs
State Logic
PS
D,T, JK FF
Memory
clock
States
NS
Output Logic
Outputs
Gate network that takes present state and inputs to produce signals that will drive the circuit to the proper next state.
Flip-flops to store state values.
Produces outputs based on present state (and possible inputs).

6.1 Basic Design Steps

- Illustrated by a simple example
- An application: the regulated speed of an automatically-controlled vehicle.

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.

Basic Design Steps

- Design a circuit that 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 immediately preceding clock cycles the input *w* was equal to 1. Otherwise, the value of *z* is equal to 0.

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	0	1	0	0	1	1	0

Figure 6.2. Sequences of input and output signals.

Basic Design Steps

- The design steps:

(1) **State Diagram** :

- The first step is to determine how many states are needed and which transitions are possible from one state to another.

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	0	1	0	0	1	1	0

Figure 6.2. Sequences of input and output signals.

Basic Design Steps

(1) **State Diagram** :

A pictorial representation:

which is a graph that depicts states of the circuit as nodes (circles) and transitions between states as directed arcs.

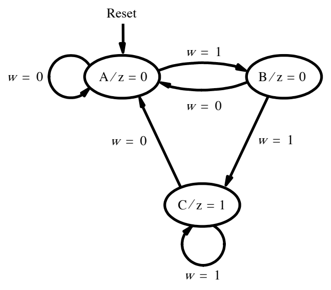


Figure 6.3. State diagram of a simple sequential circuit.

Basic Design Steps

(2) **State Table** :

- Truth table showing, for each combination of state and input values, what the next state will be.

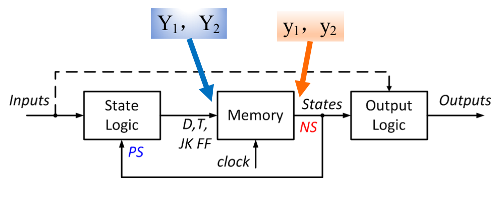
Present state	Next state		Output
	$w = 0$	$w = 1$	z
A	A	B	0
B	A	C	0
C	A	C	1

Figure 6.4. State table for the sequential circuit in Figure 6.3.

Basic Design Steps

(3) **State Assignment** :

- The state table defines the three states in terms of letters *A*, *B*, and *C*. So, it is sufficient to use two state variables to realize three states.



BASIC DESIGN STEPS

(3) **State Assignment** :

- The state table defines the three states in terms of letters *A*, *B*, and *C*. So, it is sufficient to use two state variables to realize three states.

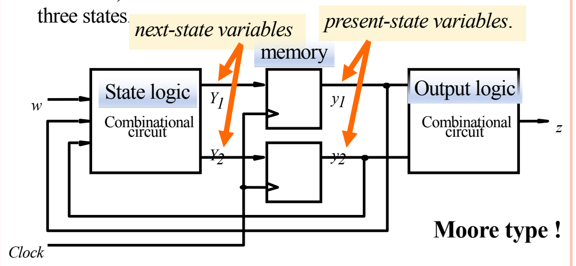


Figure 6.5. A general sequential circuit with input *w*, output *z*, and two state flip-flops.

BASIC DESIGN STEPS

(3) **State Assignment** :

- Create a truth table that defines this circuit.

PS	NS w=0	NS w=1	Out z
A	A	B	0
B	A	C	0
C	A	C	1

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

Figure 6.6. State-assigned table for the sequential circuit in Figure 6.4.

BASIC DESIGN STEPS

(4) **Choice of Flip-Flops and Derivation of Next-State and Output Expressions** :

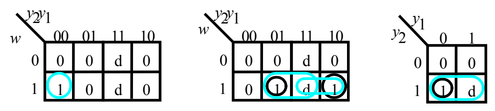
The most straightforward choice is to use D-type flip-flops.
Design the combinational circuits:

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

BASIC DESIGN STEPS

(4) **Choice of Flip-Flops and Derivation of Next-State and Output Expressions** :

The most straightforward choice is to use D-type flip-flops.
Design the combinational circuits:



Ignoring don't cares

$Y_1 = w\bar{y}_1\bar{y}_2$ $Y_2 = wy_1\bar{y}_2 + w\bar{y}_1y_2$ $z = \bar{y}_1y_2$

Using don't cares

$Y_1 = w\bar{y}_1\bar{y}_2$ $Y_2 = wy_1 + wy_2 = w(y_1 + y_2)$ $z = y_2$

Fig. 6.7 Derivation of logic expressions for the sequential circuit in Fig. 6.6.

BASIC DESIGN STEPS

$Y_1 = w\bar{y}_1\bar{y}_2$

$Y_2 = wy_1 + wy_2 = w(y_1 + y_2)$

$z = y_2$

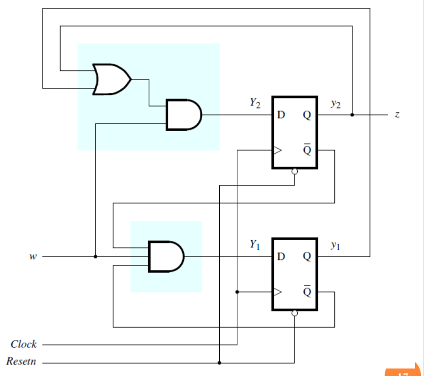


Fig. 6.8. Final implementation of the sequential circuit derived in Fig. 6.7

BASIC DESIGN STEPS

(5) **Timing Diagram** :

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	0	1	0	0	1	1	0

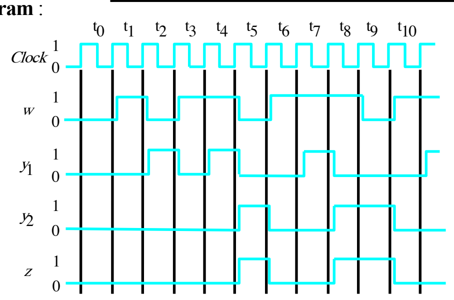


Figure 6.9. Timing diagram for the circuit in Figure 6.8.

Basic Design Steps

- **Summary of Design Steps**

1. Obtain the specification of the desired circuit.
2. Derive a state diagram.
3. Derive the corresponding state table.
4. Reduce the number of states if possible.
5. Decide on the number of state variables.
6. Choose the type of flip-flops to be used.
7. Derive the logic expressions needed to implement the circuit.

6.2 State-Assignment Problem

- But can the FSM of Figure 6.4 be implemented with an even simpler circuit by using a different state assignment?

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

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

Figure 6.16. Improved state assignment for the sequential circuit in Figure 6.4

State-Assignment Problem

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

$$Y_1 = D_1 = w$$
$$Y_2 = D_2 = wy_1$$
$$z = y_2$$

$$Y_1 = w\bar{y}_1\bar{y}_2$$
$$Y_2 = wy_1 + wy_2$$
$$= w(y_1 + y_2)$$
$$z = y_2$$

State-Assignment Problem

$$Y_1 = D_1 = w$$
$$Y_2 = D_2 = wy_1$$
$$z = y_2$$

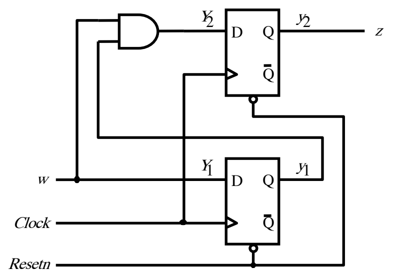


Figure 6.17. Final circuit for the improved state assignment in Figure 6.16.

One-Hot Encoding

- Use as many state variables as there are states.
- In this method, for each state all but one of the state variables are equal to 0.
- The approach is known as the *one-hot encoding* method.

	Present state	Nextstate		Output
		$w = 0$	$w = 1$	
	$y_3y_2y_1$	$Y_3Y_2Y_1$	$Y_3Y_2Y_1$	z
A	001	001	010	0
B	010	001	100	0
C	100	001	100	1

$$Y_1 = \bar{w}$$
$$Y_2 = wy_1$$
$$Y_3 = w\bar{y}_1$$
$$z = y_3$$

Figure 6.20. One-hot state assignment for the sequential circuit in Figure 6.4.

6.3 Mealy State Model

Finite State Machine

- A sequential logic circuit defined by progression through a *finite* number of states, depending on the sequence of input values.

- ***Moore machine:*** outputs depend only on present state (not inputs).
- ***Mealy machine:*** outputs depend on both the present state and inputs.

Mealy State Model

- Moore Mechine: Generate an output $z = 1$ whenever a second occurrence of the input $w = 1$ is detected in consecutive clock cycles.
- Mealy Mechine: The output z should be equal to 1 in the same clock cycle when the second occurrence of $w = 1$ is detected.

Clock cycle:	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 6.22. Sequences of input and output signals.

Mealy State Model

- State diagram

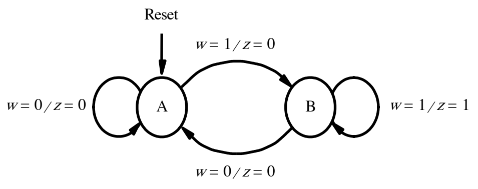


Figure 6.23. State diagram of an FSM that realizes the task in Figure 6.22.

Mealy State Model

- State table

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 6.24. State table for the FSM in Figure 6.23.

- State-assigned table

	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

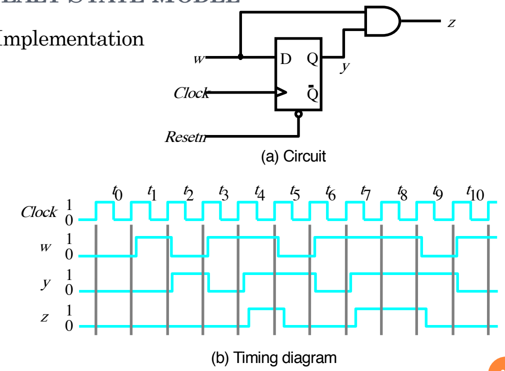
Y=D=w

$z = wy$

Figure 6.25. State-assigned table for the FSM in Figure 6.24.

Mealy State Model

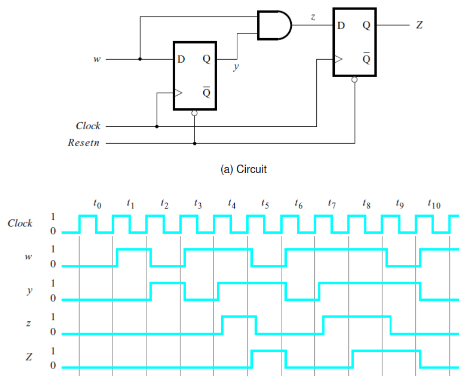
- Implementation



z
w
D
Q
y
Clock
Q̄
Resetn
(a) Circuit
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
Clock
w
y
z
1
0
(b) Timing diagram

Figure 6.26. Implementation of FSM in Figure 6.25.

Mealy Machine to Moore Machine

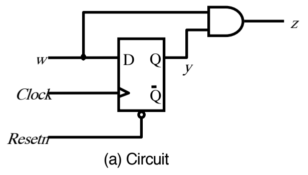


z
Z
w
D
Q
Q̄
y
Clock
Resetn
(a) Circuit
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10}
Clock
w
y
z
Z
1
0

Figure 6.27. Circuit that implements the specification in Figure 6.2

Specification of Mealy FSMs Using Verilog

All changes in the input w occur immediately following a positive clock edge.



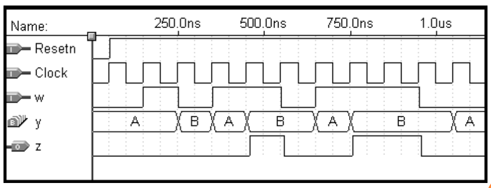
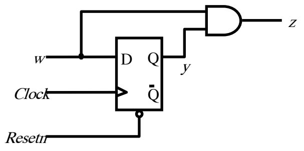


Figure 6.37. Simulation results for the Mealy machine.

Specification of Mealy FSMs Using Verilog

If the changes in *w* take place at the negative edge of the clock.



(a) Circuit

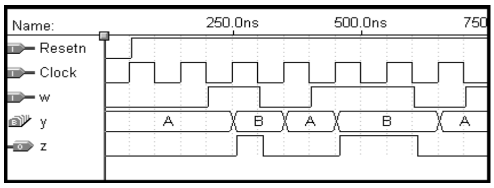


Figure 6.38. Potential problem with asynchronous inputs to a Mealy FSM.

6.4 Design of Finite State Machines Using CAD Tools

A rudimentary way of using CAD tools for FSM design could be as follows:

- The designer employs the manual techniques described previously to derive a circuit that contains flip-flops and logic gates from a state diagram.
- This circuit is entered into the CAD system by drawing a schematic diagram or by writing structural hardware description language (HDL) code.
- The designer then uses the CAD system to simulate the behavior of the circuit and uses the CAD tools to automatically implement the circuit in a chip, such as a PLD.

6.4 Design of Finite State Machines Using CAD Tools

- It is tedious to manually synthesize a circuit from a state diagram. A better approach is to directly enter the state diagram into the CAD system and perform the entire synthesis process automatically.
- CAD tools automatically support the entire synthesis process in two main ways:

(1) One method is to allow the designer to draw the state diagram using a graphical tool similar to the schematic capture tool.

The designer draws circles to represent states and arcs to represent state transitions and indicates the outputs that the machine should generate.

(2) Another and more popular approach is to write HDL code that represents the state diagram, as described below.

Verilog Code for Moore-Type FSMs

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.

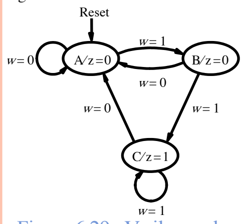


Figure 6.29. Verilog code for the FSM in Figure 6.3.

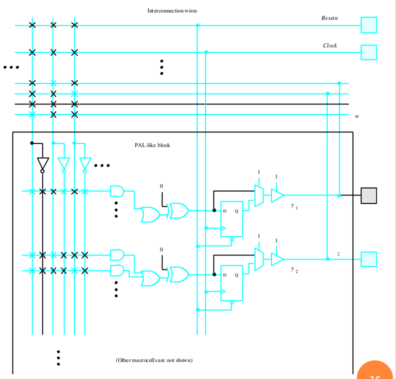
```
module simple (Clock, Resetn, w, z);
   input Clock, Resetn, w;
   output z;
   reg [2:1] y, Y;
   parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;
// Define the next state combinational circuit
   always @(w, y)
      case (y)
         A: if (w) Y = B;
            else   Y = A;
         B: if (w) 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
```

In general, it is always necessary to include a **default** clause when using a **case** statement that does not cover all of its alternatives.

Synthesis of Verilog Code

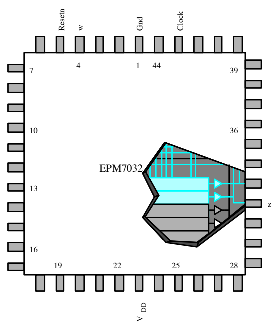
$$Y_1 = w\bar{y}_1\bar{y}_2$$
$$Y_2 = wy_1 + wy_2$$
$$z = y_2$$



CPLD: Complex Programmable Logic Device

Figure 6.30. Implementation of the FSM of Figure 6.3 in a CPLD.

Synthesis of Verilog Code



a *global wire* distributes the clock signal to all of the flip-flops in the chip.

Figure 6.31. The circuit from Figure 6.30 in a small CPLD.

Simulating and Testing the Circuit

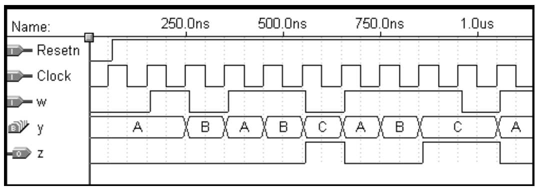


Figure 6.32. Simulation results for the circuit in Figure 6.30.

Alternative Styles of Verilog Code

we specified the output *z* inside the **always** block that defines the required combinational circuit.

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

Figure 6.33. Second version of code for the FSM in Figure 6.3.

```
module simple (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output z;
    reg [2:1] y;
    parameter [2:1] A = 2'b00, B = 2'b01, C = 2'b10;

    // Define the sequential block
    always @(negedge Resetn, posedge Clock)
        if (Resetn = = 0)        y <= A;
        else
            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 output
    assign z = (y = = C);

endmodule
```

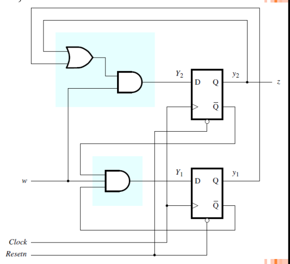


Figure 6.34. Third version of code for the FSM in Figure 6.3.

Summary of Design Steps When Using CAD Tools

CAD tools can automatically perform much of the work, however, they have not replaced *all* manual steps.

- The machine specification and a state diagram still have to be done manually.
- Given the state diagram information as input, the CAD tools then automatically perform the tasks needed to generate a circuit with logic gates and flip-flops.
- The testing and simulation stage can not be ignored.

Specifying the State Assignment in Verilog Code

- An obvious objective of the state-assignment process is to minimize the cost of implementation.
- A particular state assignment can be specified in Verilog code by means of a **parameter** statement as done in Figures 6.29 through 6.34.
- The user can either allow the compiler to use its FSM-handling capability, or surpress it in which case the compiler simply deals with the Verilog statements in the usual way.

Specification of Mealy FSMs Using Verilog

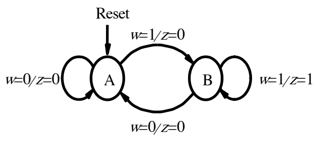
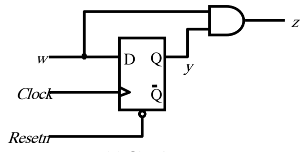


Figure 6.36. Verilog code for the Mealy machine of Figure 6.23.

```
module mealy (Clock, Resetn, w, z);
    input Clock, Resetn, w;
    output reg z;
    reg y, Y;
    parameter A = 1'b0, B = 1'b1;
// Define the next state and output combinational circuits
    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
```

Specification of Mealy FSMs Using Verilog

All changes in the input *w* occur immediately following a positive clock edge.



(a) Circuit

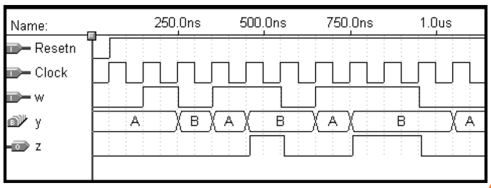


Figure 6.37. Simulation results for the Mealy machine.

END