State Machine Design

10.1 State Machines

- **State Machine:** A synchronous sequential circuit consisting of a sequential logic section and a combinational logic section.

- The outputs and internal flip flops (FF) progress through a predictable sequence of states in response to a clock and other control inputs.

State Machine Types

- **Moore Machine:** A state machine whose outputs is determined only by the Sequential Logic (FF) of the machine.

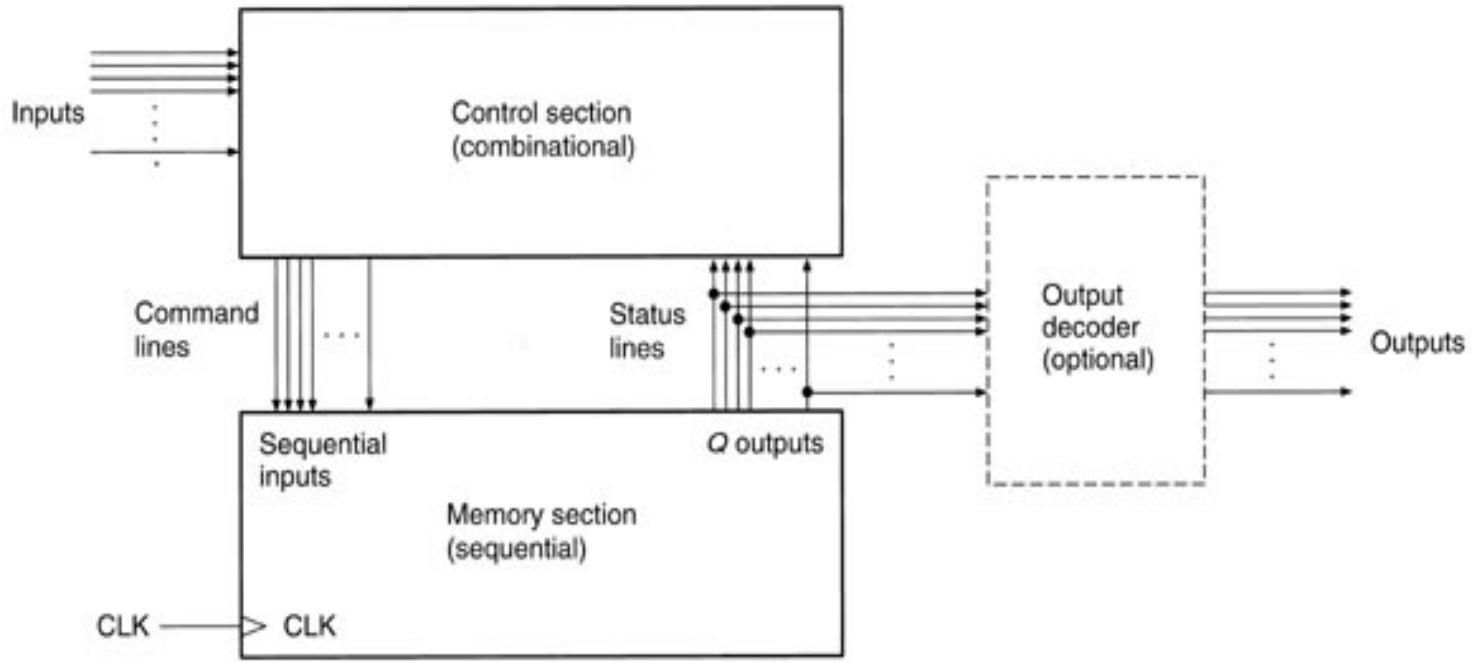
- **Mealy Machine:** A state machine whose outputs are determined by **both** the sequential logic and combinational logic of the machine.

State Machine Basics

- **State Variable:** The variables held in the flip-flops of a state machine that determine its present state. The number of state variables in a machine is equivalent to the number of flip-flops.

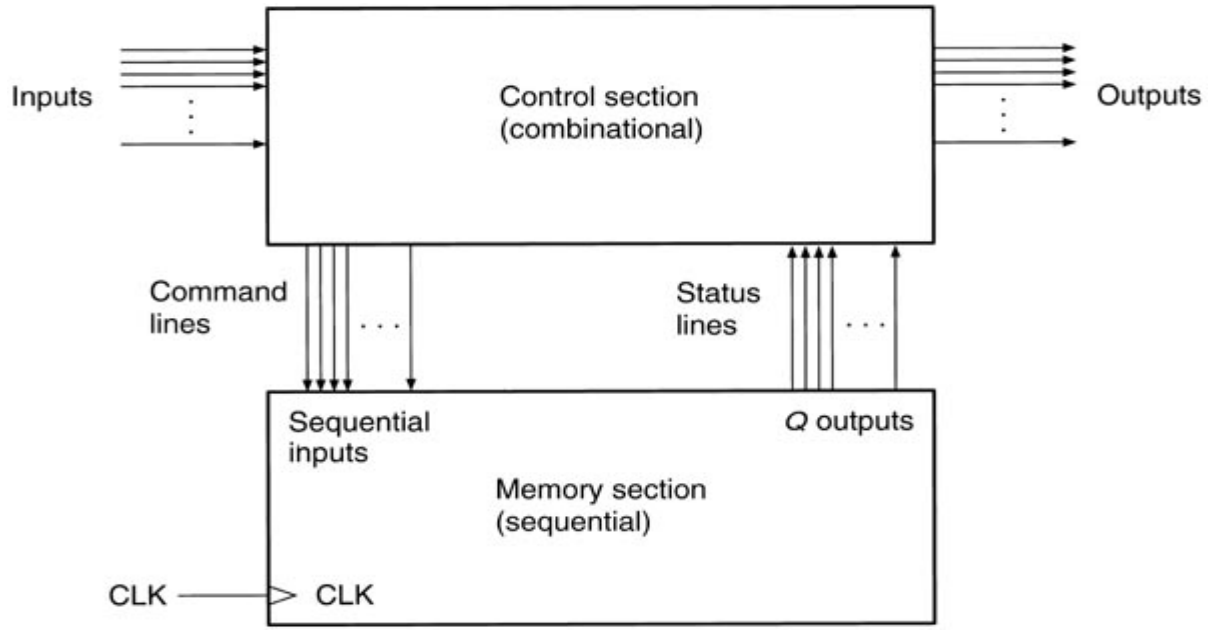
- A basic state machine has a memory section that holds the present state of the machine (stored in FF) and a control section that controls the next state of the machine (by clocks, inputs, and present state).

Moore-Type State Machine



the block diagram of a **Moore machine.** The outputs of a Moore machine are determined solely by the present state of the machine's memory section. The output may be directly connected to the Q outputs of the internal flip-flops, or the Q outputs might pass through a decoder circuit. The output of a Moore machine is synchronous to the system clock, since the output can only change when the machine's internal **state variables** change.

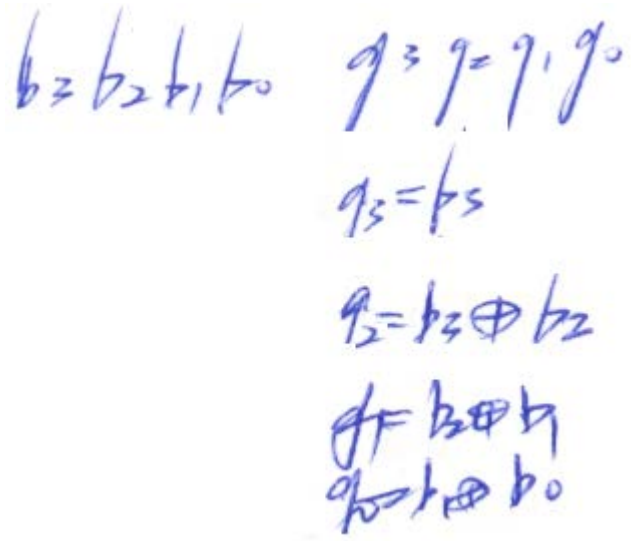
Mealy-Type State Machine



The outputs of the Mealy machine are derived from the combinational (control) section of the machine, as well as the sequential (memory) part of the machine. Therefore, the outputs can change asynchronously when the combinational circuit inputs change out

10.2 State Machine with no Control Inputs

Gray code



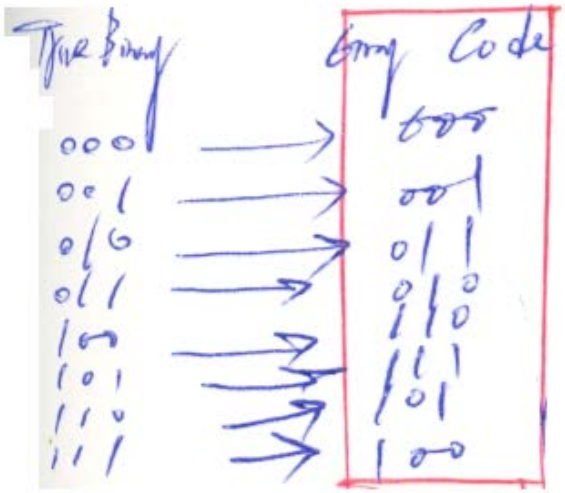


TABLE 10.1 3-Bit Gray Code Sequence

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Classical Design Techniques -1

We can summarize the classical design technique for a state machine as follows:

1. Define the problem.
2. Draw a state diagram.
3. Make a state table that lists all possible present states and inputs, and the next state and output state for each present state/input combination. *List the present states and inputs in* ***binary order.***
4. Use flip-flop excitation tables to determine at what states the flip-flop synchronous inputs must be to make the circuit go from each present state to its next state. *The next state variables are functions of the inputs and present state variables.*
5. Write the output value for each present state/input combination. *The output variables are functions of the inputs and present state variables.*
6. Simplify the Boolean expression for each output and synchronous input.
7. Use the Boolean expressions found in step 6 to draw the required logic circuit.

Classical Design Techniques -2

For example:

design a 3-bit Gray code counter that there are no inputs other than the clock and no outputs that must be designed apart from the counter itself.

1. *Define the problem.* Design a counter whose outputs progress in the sequence

2. *Draw a state diagram.* The state diagram is shown in Figure 10.4.

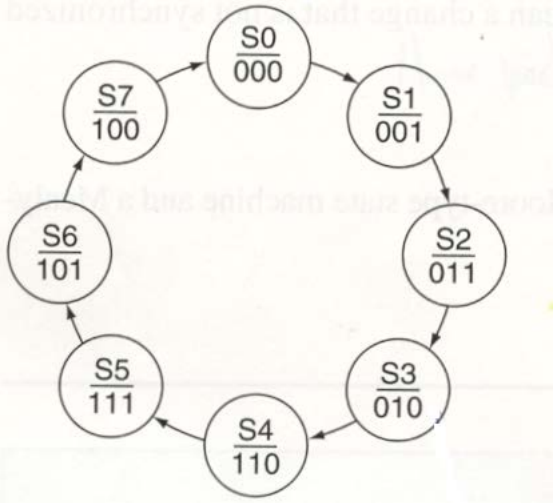


FIGURE 10.4 State Diagram for a 3-Bit Gray Code Counter

Q_2	Q_1	Q_0
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

Classical Design Techniques -3

3. *Make a state table.* The state table, based on D flip-flops, is shown in Table 10.2. *Since there are eight unique states in the state diagram, we require three state variables* ($2^3 = 8$), *and hence three flip-flops.* Note that the present states are in binary-weighted order

TABLE 10.2 State Table for a 3-Bit Gray Code Counter

Present State			Next State			Synchronous Inputs		
Q_2	Q_1	Q_0	Q_2	Q_1	Q_0	D_2	D_1	D_0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	1
0	1	0	1	1	0	1	1	0
0	1	1	0	1	0	0	1	0
1	0	0	0	0	0	0	0	0
1	0	1	1	0	0	1	0	0
1	1	0	1	1	1	1	1	1
1	1	1	1	0	1	1	0	1

Classical Design Techniques -4

4. *Use flip-flop excitation tables to determine at what states the flip-flop synchronous inputs must be to make the circuit go from each present state to its next state.* This is not necessary if we use *D* flip-flops, since *Q* follows *D*. The *D* inputs are the same as the next state outputs. For JK or T flip-flops, we would follow the same procedure as for the design of synchronous counters outlined in Chapter 9.

Classical Design Techniques -5

5. *Simplify the Boolean expression for each synchronous input.* Figure 10.5 shows three Karnaugh maps, one for each D input of the circuit. The K-maps yield three Boolean equations:

$$D_2 = Q_1\bar{Q}_0 + Q_2Q_0$$
$$D_1 = Q_1\bar{Q}_0 + \bar{Q}_2Q_0$$
$$D_0 = \bar{Q}_2\,\bar{Q}_1 + Q_2Q_1$$

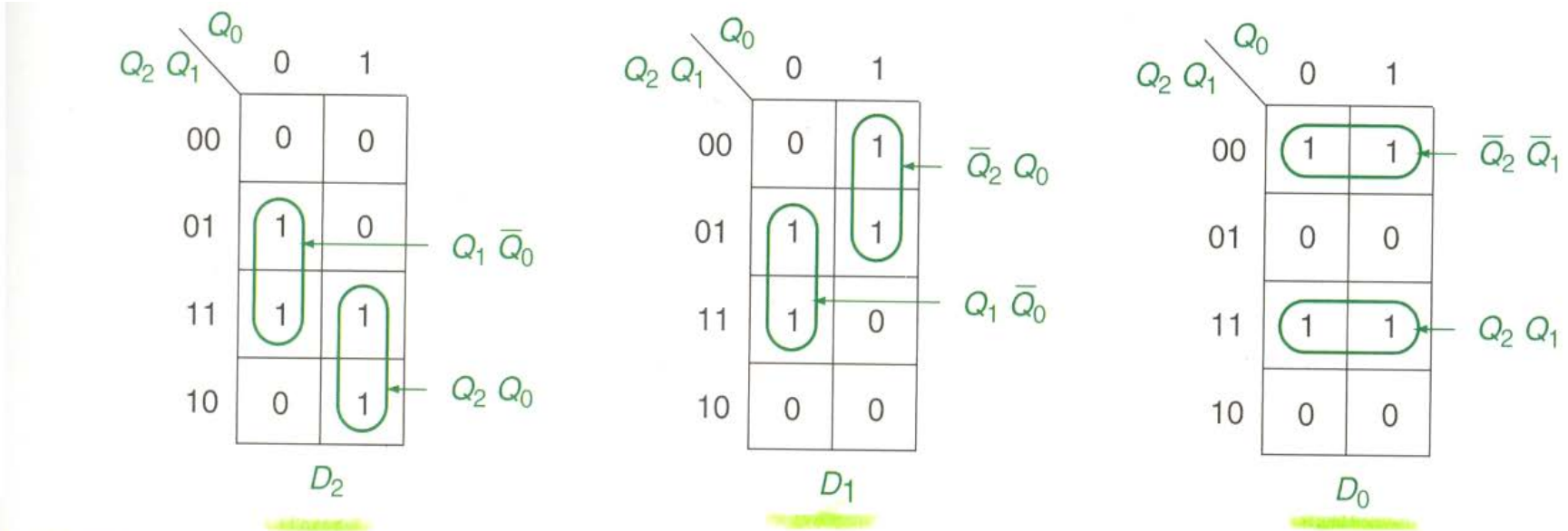


FIGURE 10.5 Karnaugh Maps for 3-Bit Gray Code Counter

Classical Design Techniques -6

6. *Draw the logic circuit for the state machine.* Figure 10.6 shows the circuit for a 3-bit Gray code counter, drawn as a Block Diagram File in Quartus II.

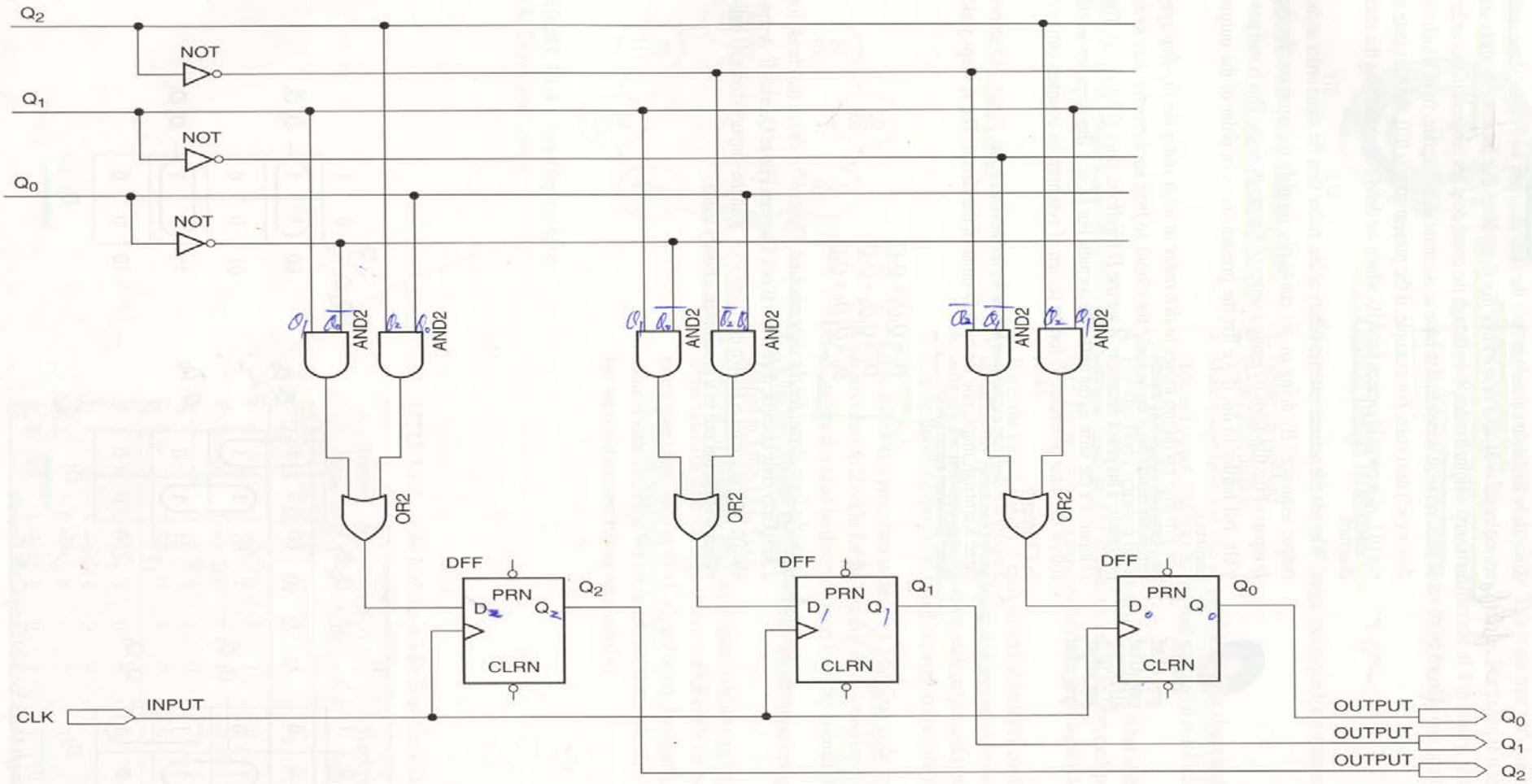
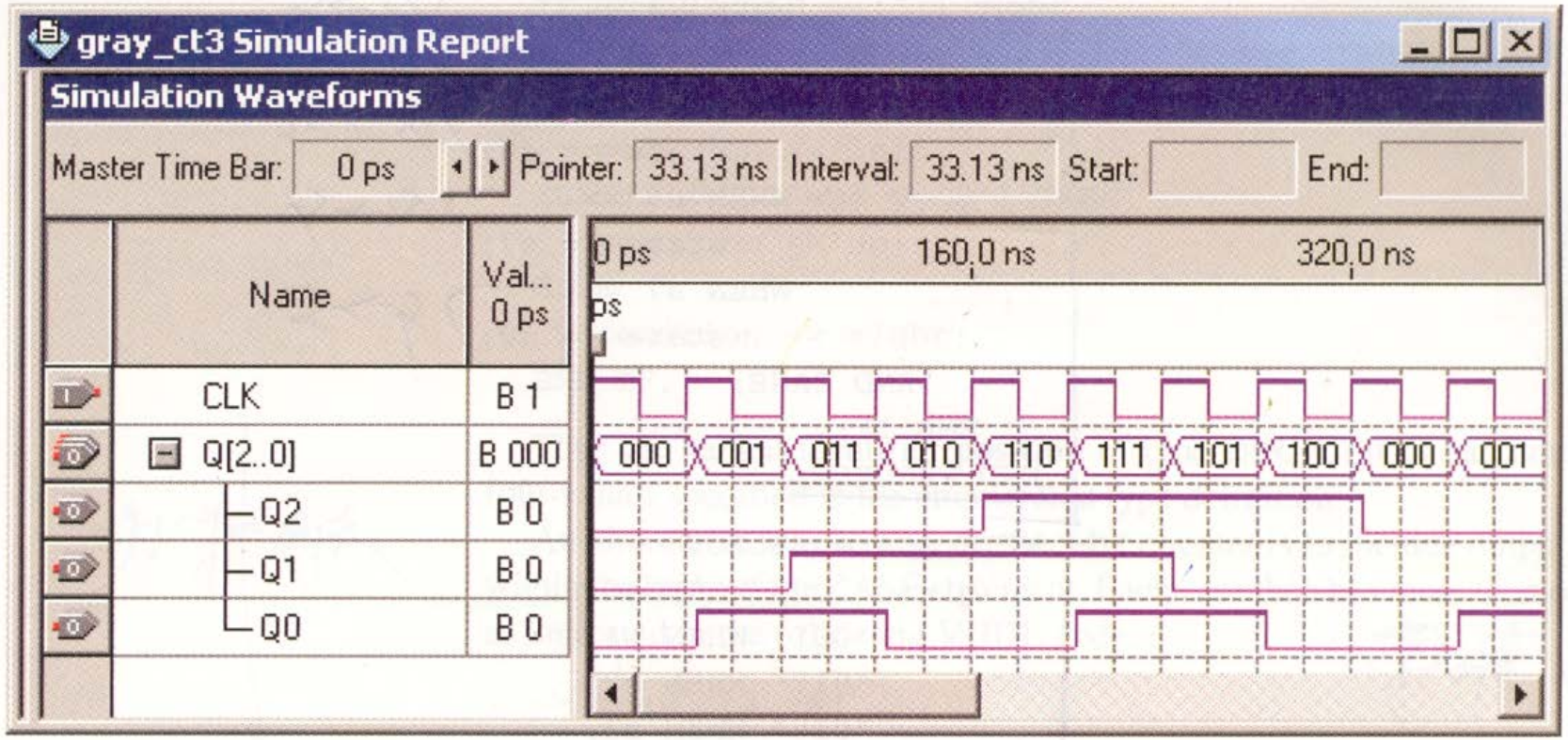


FIGURE 10.6 Logic Diagram of a 3-Bit Gray Code Counter

Classical Design Techniques -7

A simulation for this circuit is shown in Figure 10.7, with the outputs shown as individual waveforms and as a group with a binary value.



Simulation of a 3-Bit Gray Code Counter (from Block Diagram File)

VHDL Design of State Machines -1

KEY TERM

Enumerated Type A user-defined type in VHDL in which all possible values of a named identifier are listed in a type definition statement.

State machines can be defined in VHDL within a CASE statement. The following VHDL code illustrates the principle, using the 3-bit Gray code counter as an example.

```
-- gray_ct1.vhd
-- 3-bit Gray code counter
-- (state machine with decoded outputs)

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY gray_ct1 IS
   PORT(
        clk : IN  STD_LOGIC;
        q   : OUT STD_LOGIC_VECTOR(2 downto 0));
END gray_ct1;

ARCHITECTURE a OF gray_ct1 IS
   TYPE STATE_TYPE IS (s0, s1, s2, s3, s4, s5, s6, s7);
   SIGNAL state: STATE_TYPE;
BEGIN
   PROCESS (clk)
   BEGIN
      IF clk'EVENT AND clk = '1' THEN
         CASE state IS
            WHEN s0 =>
               state <= s1;
            WHEN s1 =>
               state <= s2;
            WHEN s2 =>
               state <= s3;
            WHEN s3 =>
               state <= s4;
```

VHDL Design of State Machines -2

```
            WHEN s4 =>
               state <= s5;
            WHEN s5 =>
               state <= s6;
            WHEN s6 =>
               state <= s7;
            WHEN s7 =>
               state <= s0;
      END CASE;
   END IF;
END PROCESS;

WITH state SELECT
   q <= "000" WHEN s0,
        "001" WHEN s1,
        "011" WHEN s2,
        "010" WHEN s3,
        "110" WHEN s4,
        "111" WHEN s5,
        "101" WHEN s6,
        "100" WHEN s7;
END a;
```

10.3 State Machines with Control Inputs

KEY TERMS

Control Input A state machine input that directs the machine from state to state.

Conditional Transition A transition between states of a state machine that occurs only under specific conditions of one or more control inputs.

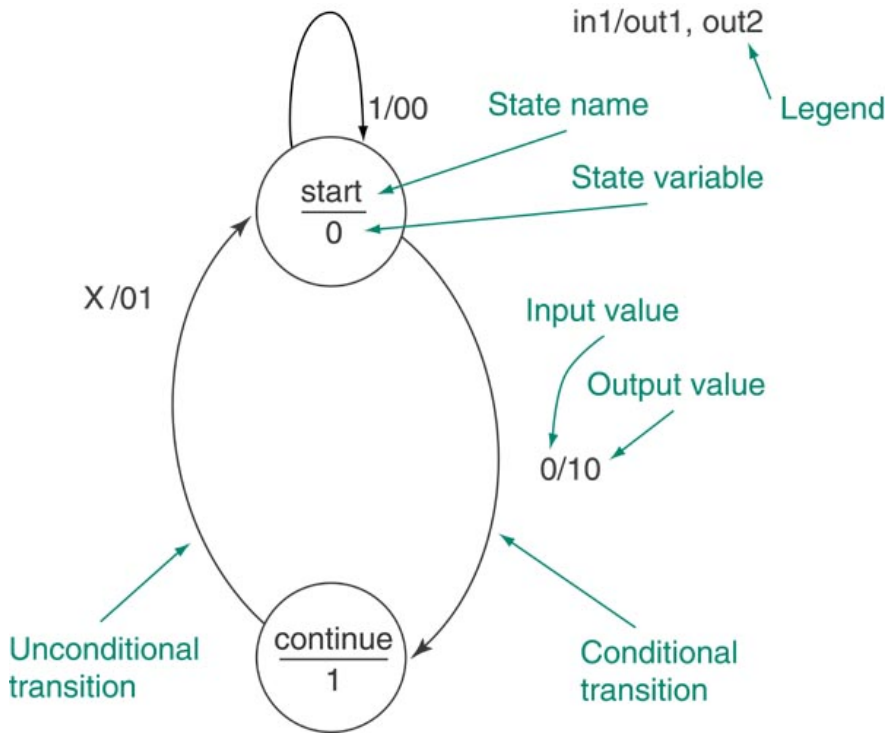
Unconditional Transition A transition between states of a state machine that occurs regardless of the status of any control inputs.

State Machine with Control Inputs -1

- Same design approach used for state machine such as counters.
- Uses the control inputs and clock to control the sequencing from state to state.
- Inputs can also cause output changes not just FF outputs.

State Machine with Control Inputs -2

- Bubbles contain the state name and value (State Name/Value), such as **Start/0**.
- Transitions between states are designated with arrows from one bubble to another.
- Each transition has an ordered Input/Output, such as **in1/out1, out2**.



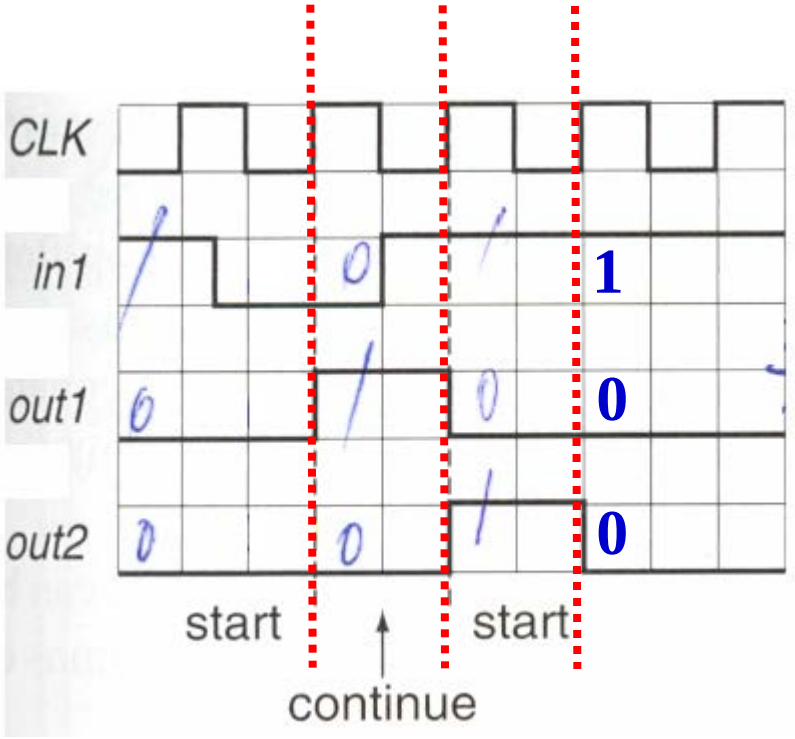
State Machine with Control Inputs -3

1. There are two states, called **start** and **continue.** The machine begins in the **start** state and waits for a LOW input on **in1.** As long as **in1** is HIGH, the machine waits and the outputs **out1** and **out2** are both LOW.
2. When **in1** goes LOW, the machine makes a transition to **continue** in one clock pulse. Output **out1** goes HIGH.
3. On the next clock pulse, the machine goes back to **start.** The output **out2** goes HIGH and **out1** goes back LOW.
4. If **in1** is HIGH, the machine waits for a new LOW on **in1.** Both outputs are LOW again. If **in1** is LOW, the cycle repeats.

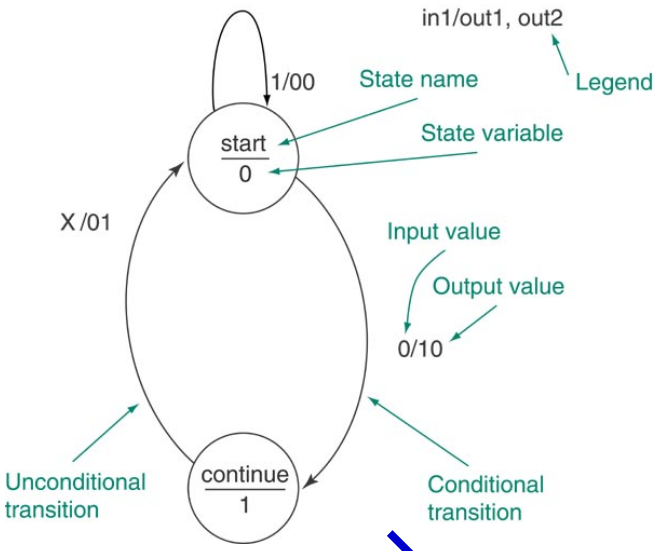
State Machine with Control Inputs -4

Ideal operation of state machine:

out1, out2: synchronous to clock



State Machine with Control Inputs -5



JK Flip-Flop Excitation Table

Transition	*JK*
0→0	0X
0→1	1X
1→0	X1
1→1	X0

State table:

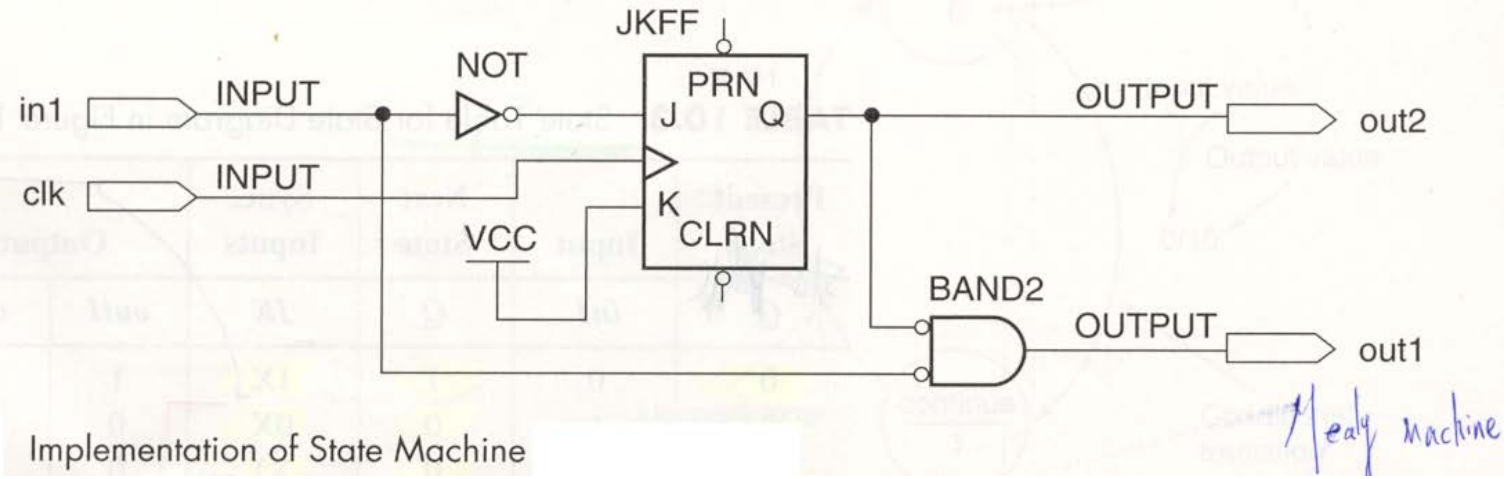
Present State	Input	Next State	Sync. Inputs	Outputs	
Q	*in1*	*Q*	*JK*	*out1*	*out2*
0	0	1	1X	1	0
0	1	0	0X	0	0
1	0	0	X1	0	1
1	1	0	X1	0	1

State Machine with Control Inputs -6

Simplify the Boolean expression for each output and synchronous input. The following equations represent the next state and output logic of the state machine:

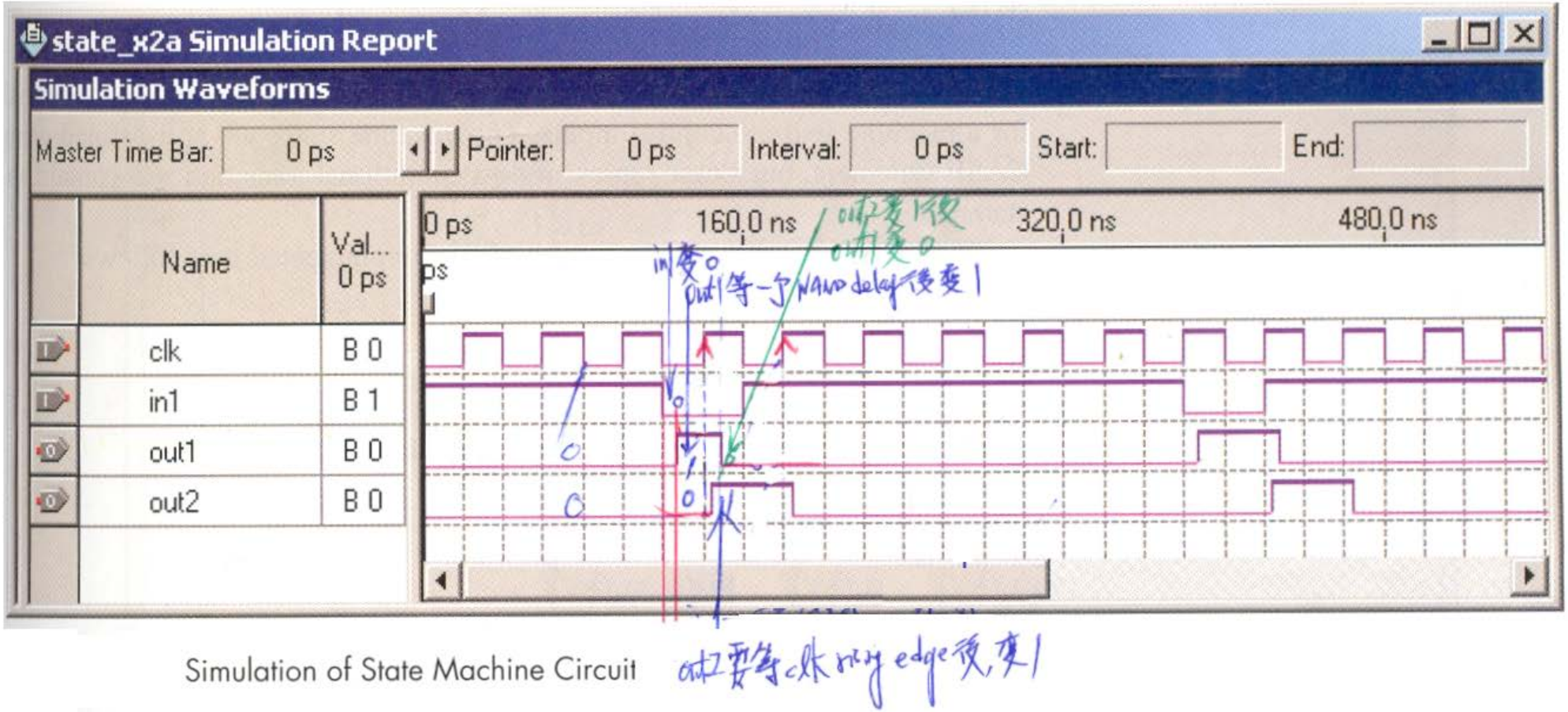
$$J = \overline{Q} \cdot \overline{in1} + Q \cdot \overline{in1} = \overline{in1}$$
$$K = 1$$
$$out1 = \overline{Q} \cdot \overline{in1}$$
$$out2 = Q \cdot \overline{in1} + Q \cdot in1 = Q$$

Use the Boolean expressions to draw the required logic circuit.



Implementation of State Machine

State Machine with Control Inputs -7



Simulation of State Machine Circuit

10.5 Unused States in State Machines -1

For instance, a machine with five states requires three state variables. There are up to eight states available in a machine with three state variables, leaving three unused states.

Unused states can be dealt with in two ways: they can be treated as don't care states, or they can be assigned specific destinations in the state diagram. In the latter case, the safest destination is the first state, in this case the state called **start.**

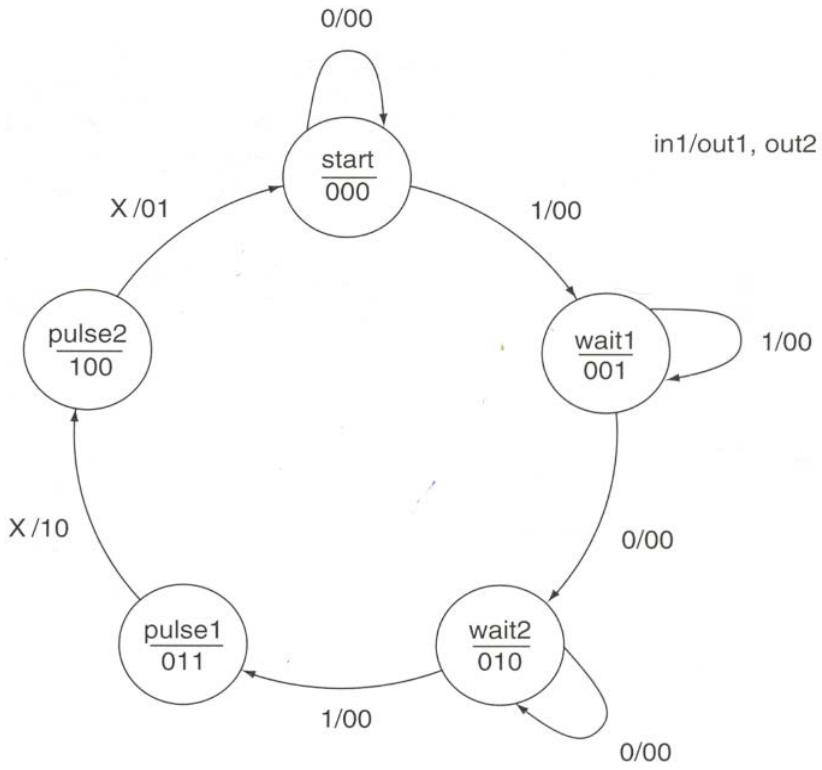


Fig. 10.32 State Diagram for a Two-Pulse Generator

Example 10.4 -1

Redraw the state diagram of Figure 10.32 to include the unused states of the machine's state variables. Set the unused states to have a destination state of **start.** Briefly describe the intended operation of the state machine.

■ Solution

Figure 10.33 shows the revised state diagram.

The machine begins in state **start** and waits for a HIGH on **in1.** The machine then makes a transition to **wait1** and stays there until **in1** goes LOW again. The machine goes to **wait2** and stays there until **in1** goes HIGH and then makes an unconditional transition to **pulse1** on the next clock pulse. Until this point, there is no change in either output.

The machine makes an unconditional transition to **pulse2** and makes **out1** go HIGH. The next transition, also unconditional, is to **start,** when **out1** goes LOW and **out2** goes HIGH. If **in1** is LOW, the machine stays in **start.** Otherwise, the cycle continues as outlined. In either case, **out2** goes LOW again.

Thus the machine waits for a HIGH-LOW-HIGH input sequence and generates a pulse sequence on two outputs.

Example 10.4 -2

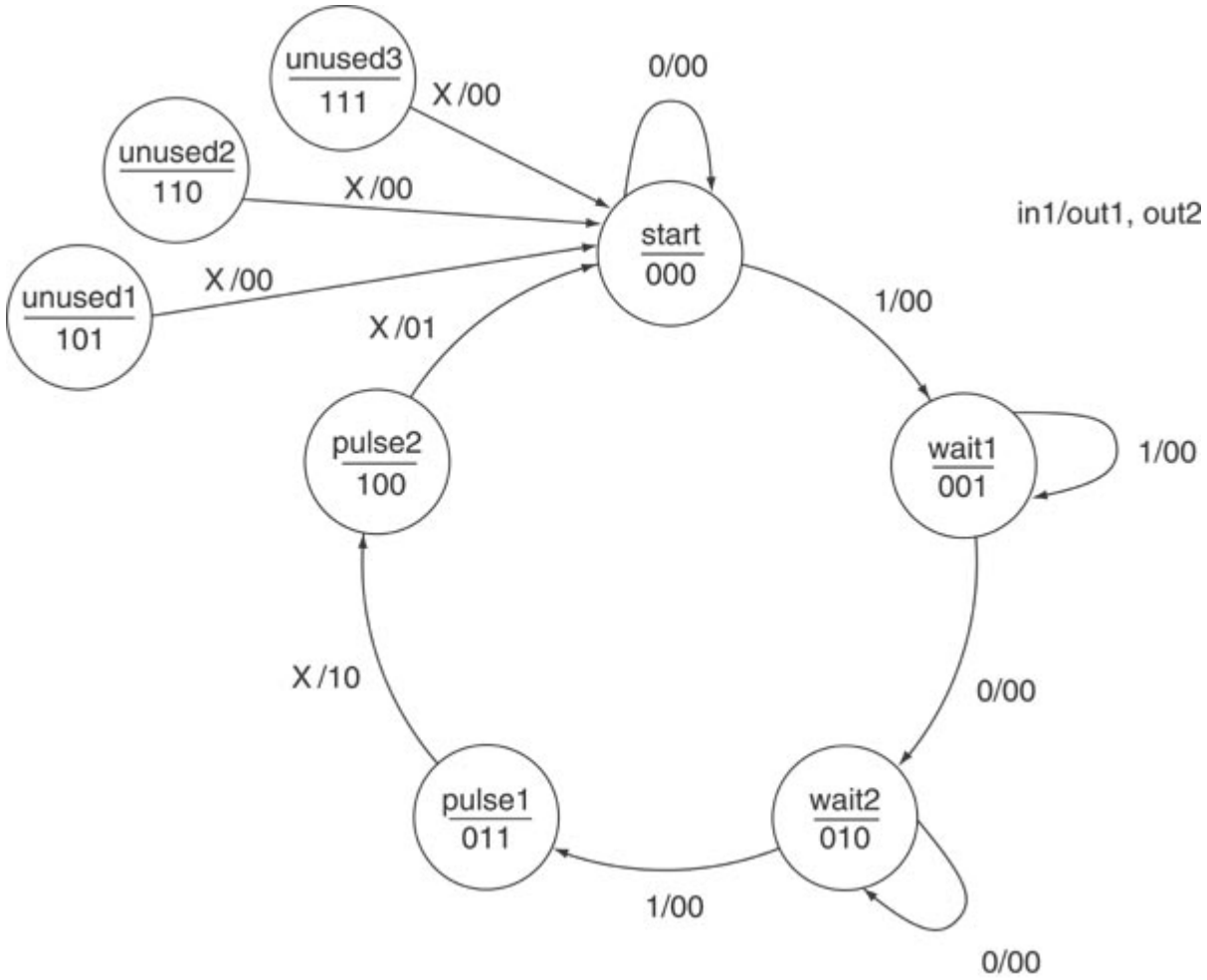


Fig. 10.33

Example 10.5 -1

Use classical state machine design techniques to implement the state machine described in the modified state diagram of Figure 10.33. Draw the state machine as a Block Diagram File in Quartus II and create a simulation to verify its function.

Example 10.5 -2

Solution

Table 10.6 shows the state table of the state machine represented by Figure 10.33.

TABLE 10.6 State Table for State Machine of Figure 10.33

Present State			Input	Next State			Outputs	
Q_2	Q_1	Q_0	*in1*	Q_2	Q_1	Q_0	*out1*	*out2*
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	0	0
0	0	1	0	0	1	0	0	0
0	0	1	1	0	0	1	0	0
0	1	0	0	0	1	0	0	0
0	1	0	1	0	1	1	0	0
0	1	1	0	1	0	0	1	0
0	1	1	1	1	0	0	1	0
1	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	1
1	0	1	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0

Example 10.5 -3

Figure 10.34 shows the Karnaugh maps

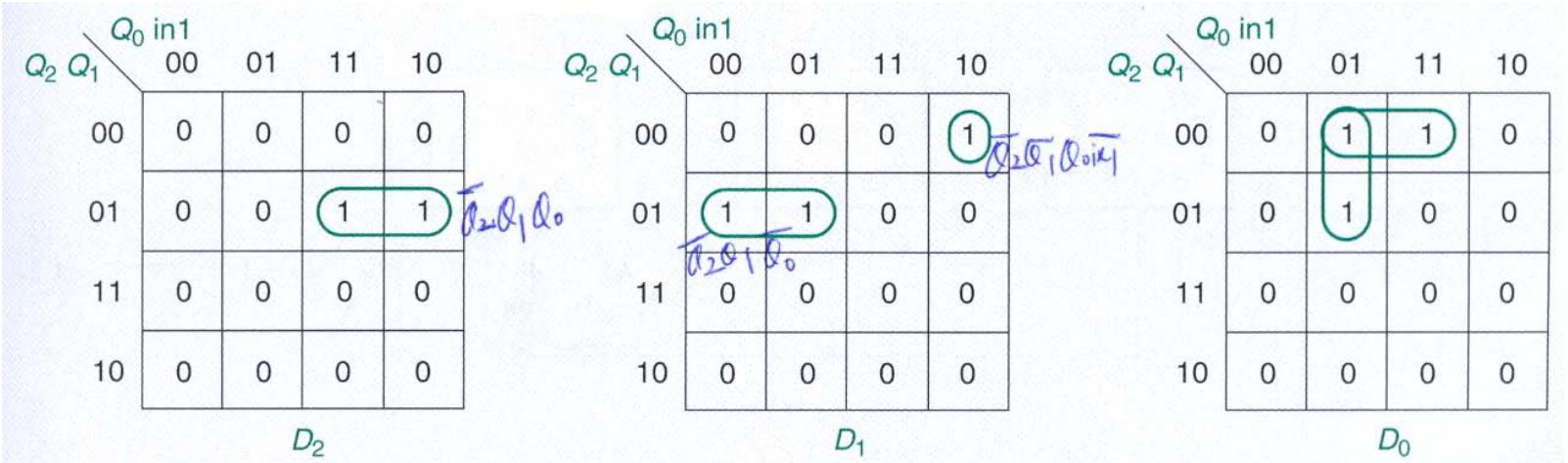


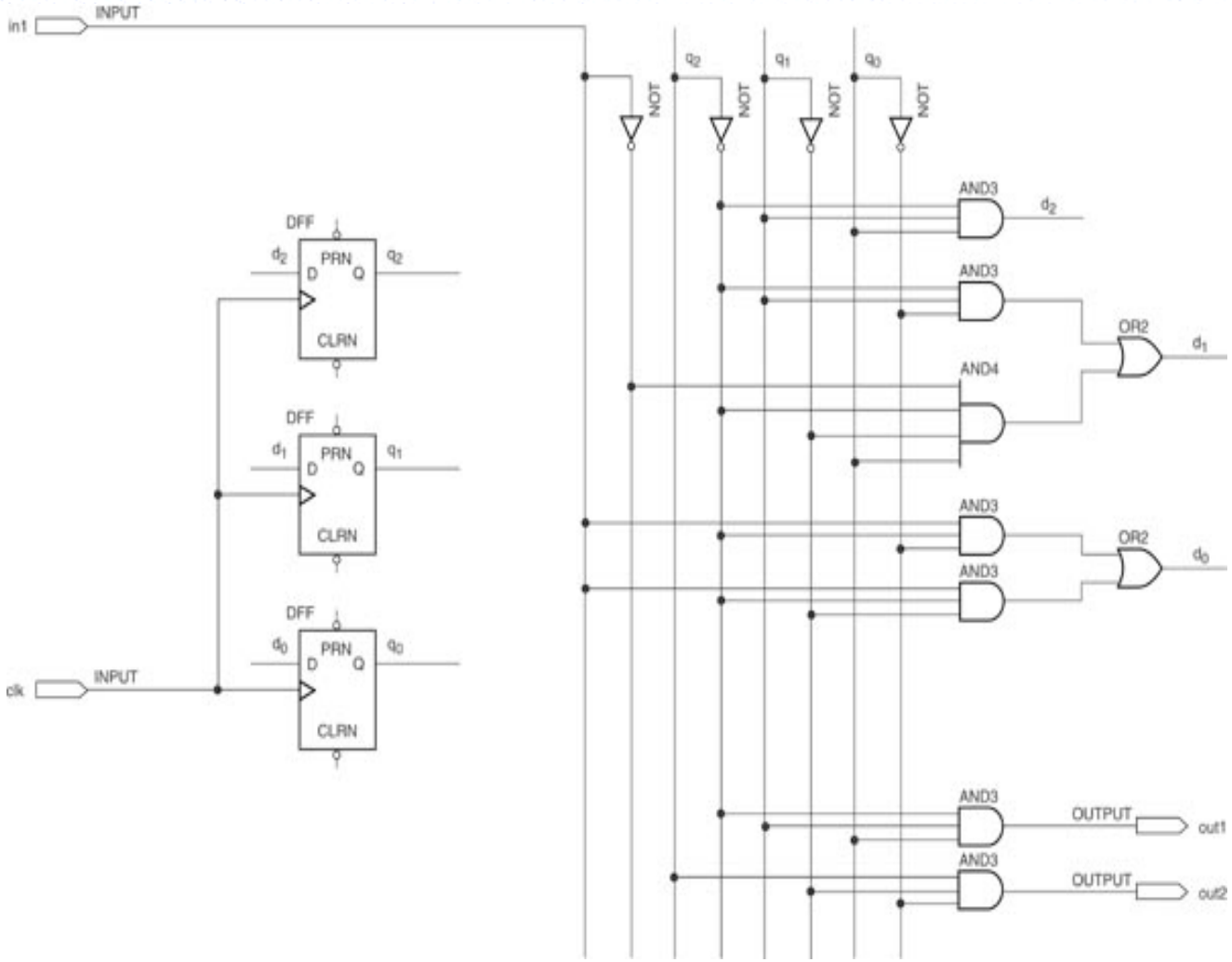
FIGURE 10.34 Example 10.5: K-Maps for Two-Pulse Generator

The next-state and output equations for the state machine are:

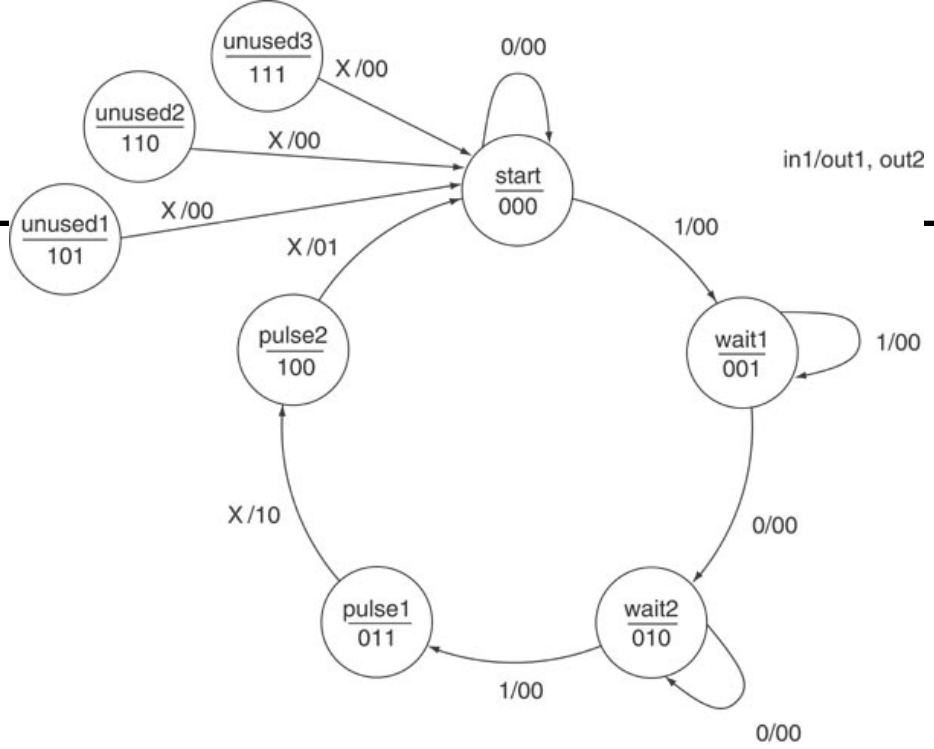
$$D_2 = \overline{Q}_2 Q_1 Q_0$$
$$D_1 = \overline{Q}_2 Q_1 \overline{Q}_0 + \overline{Q}_2 \overline{Q}_1 Q_0 \overline{in1}$$
$$D_0 = \overline{Q}_2 \overline{Q}_0 in1 + \overline{Q}_2 \overline{Q}_1 in1$$
$$out1 = \overline{Q}_2 Q_1 Q_0$$
$$out2 = Q_2 \overline{Q}_1 \overline{Q}_0$$

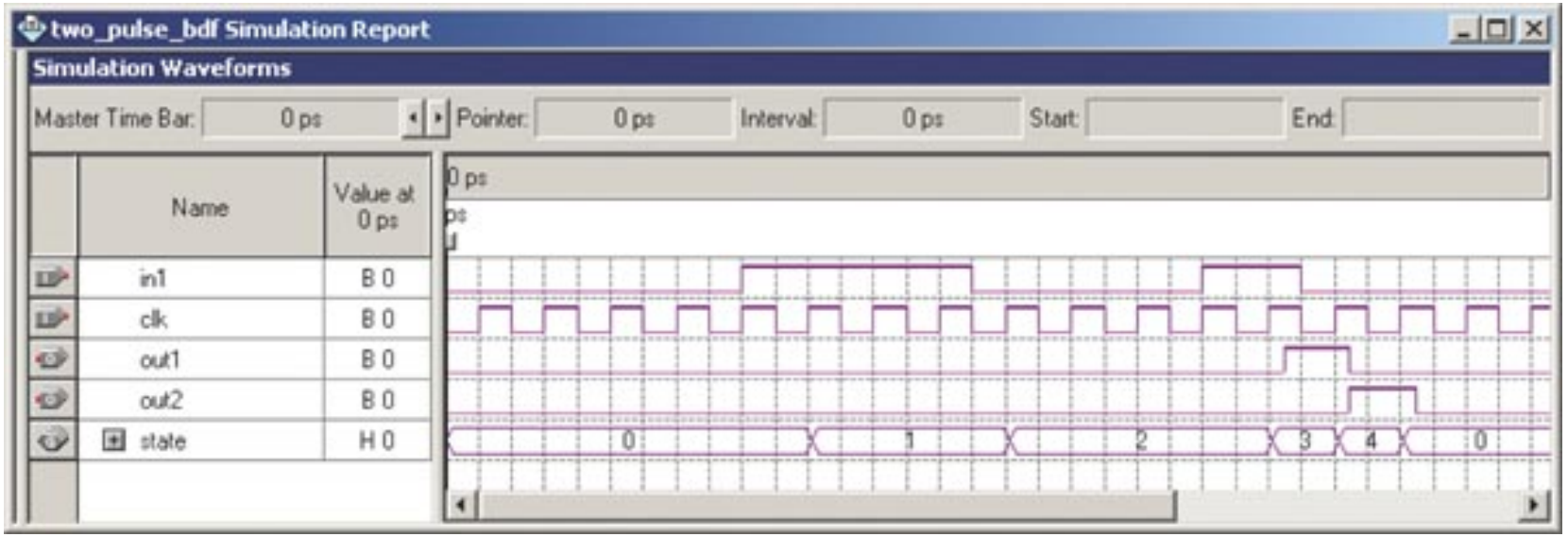
Example 10.5 -4

Figure 10.35 shows the Block Diagram File for the state machine. Figure 10.36 shows the Quartus II simulation waveforms.



Example 10.5 -5





10.6 Traffic Light Control -1

A simple traffic light controller can be implemented by a state machine with a state diagram such as the one shown in Figure 10.38.

The control scheme assumes control over a north-south road and an east-west road. The north-south lights are controlled by outputs called **nsr, nsy,** and **nsg** (north-south red, yellow, green). The east-west road is controlled by similar outputs called **ewr, ewy,** and **ewg.** A LOW controller output turns on a light. Thus an output 011110 corresponds to the north-south red and east-west green lights.

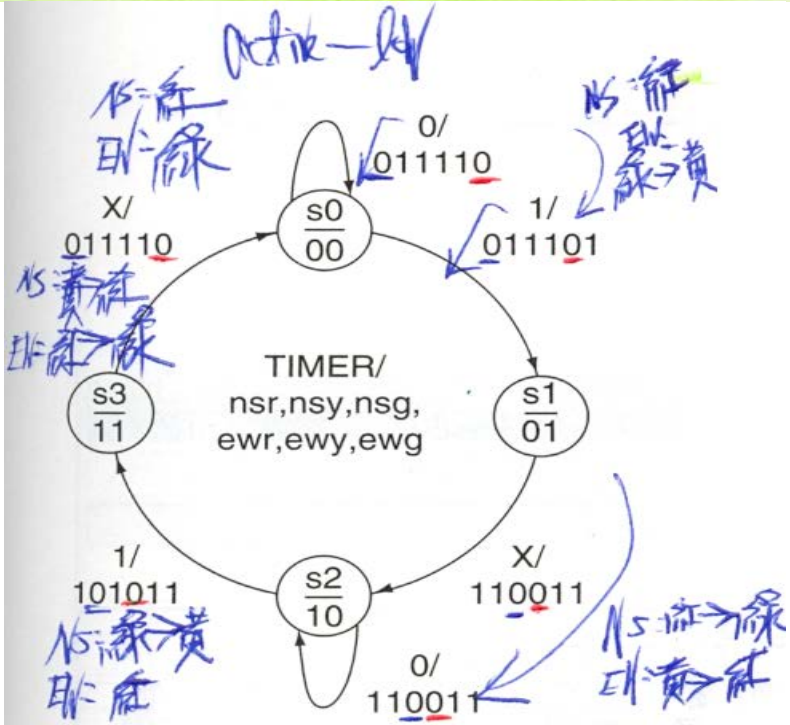


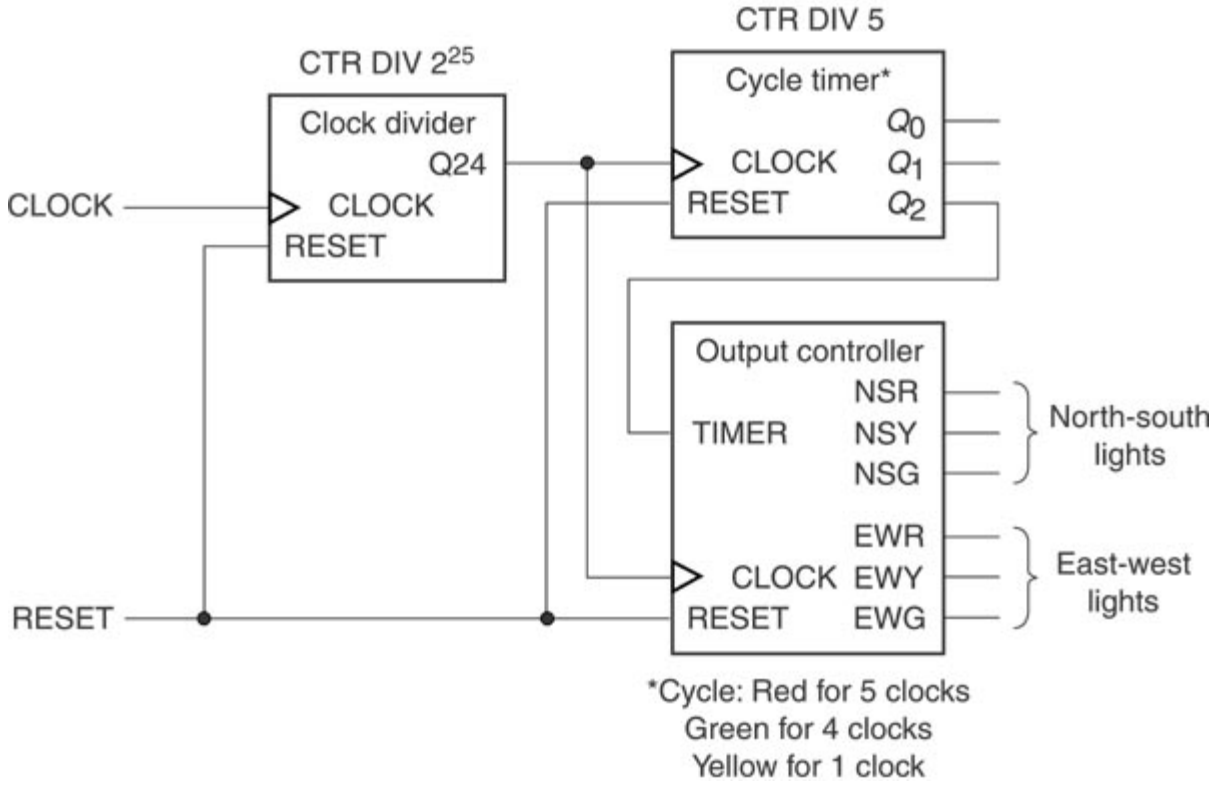
FIGURE 10.38 State Diagram of a Traffic Light Controller

10.6 Traffic Light Control -2

- An Input called *TIMER* controls the length of a light cycle (*TIMER* = 1 causes a S0 to S1 or a S2 to S3 transition).
- When one light is green (S0(EW) or S2(NS)), the other is red.
- There is an unconditional timed transfer from yellow to red or red to green.
- A normal cycle is 4 clocks GREEN, 1 clock YELLOW, 5 clocks RED.

10.6 Traffic Light Control -3

, we will use a cycle of ten clock pulses. For either direction, the cycle consists of 4 clocks GREEN, 1 clock YELLOW, and 5 clocks RED. This cycle can be generated by the MSB of a mod-5 counter, as shown in Figure 10.39.



10.6 Traffic Light Control -4

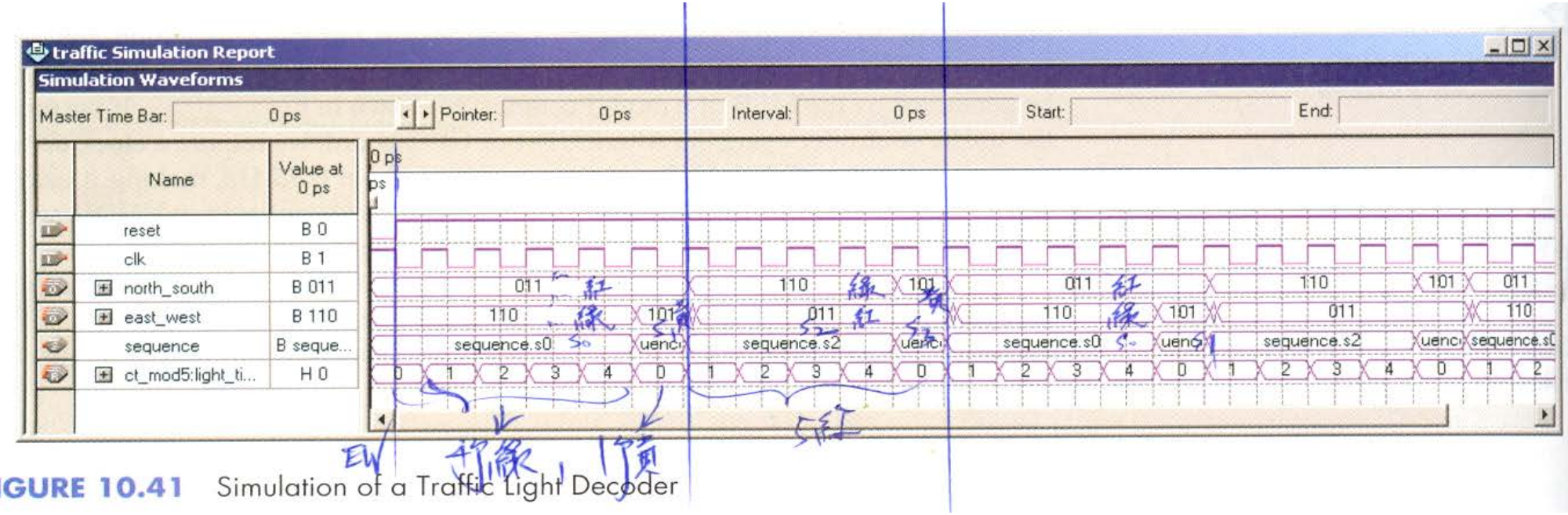


FIGURE 10.41 Simulation of a Traffic Light Decoder

Figure 10.41 shows a simulation of the mod-5 counter and output controller. The north-south lights are red for five clock pulses (shown by 011 in the **north_south** waveform). At the same time, the east-west lights are green for four clock pulses (**east_west** = 110), followed by yellow for one clock pulse (**east_west** = 101). The cycle continues with an east-west red and north-south green and yellow.