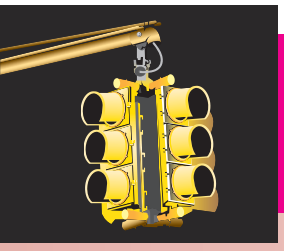


Applied Logic

Traffic Signal Controller: Part 1

The control logic is developed for a traffic signal at the intersection of a busy main street and a lightly used side street. The system requirements are established, and a general block diagram is developed. Also, a state diagram is introduced to define the sequence of operation. The combinational logic unit of the controller is developed in this chapter, and the remaining units are developed in Chapter 7.

Timing Requirements

The control logic establishes the sequencing of the lights for a traffic signal at the intersection of a busy main street and an occasionally used side street. The following are the timing requirements:

- The green light for the main street will stay on for a minimum of 25 s or as long as there is no vehicle on the side street.
- The green light for the side street will stay on until there is no vehicle on the side street up to a maximum of 25 s.
- The yellow caution light will stay on for 4 s between changes from green to red on both the main street and the side street.

The State Diagram

From the timing requirements, a state diagram can be developed to describe the complete operation. A state diagram graphically shows the sequence of states, the conditions for each state, and the requirements for transitions from one state to the next.

Defining the Variables The variables that determine how the system sequences through the various states are defined as follows:

- V_s A vehicle is present on the side street.
- T_L The 25 s timer (long timer) is *on.*
- T_S The 4 s timer (short timer) is *on.*

A complemented variable indicates the opposite condition.

State Descriptions A state diagram is shown in Figure 6–63. Each of the four states is assigned a 2-bit Gray code as indicated. A looping arrow means that the system remains in a state, and an arrow between states means that the system transitions to the next state. The Boolean expression or variable associated with each of the arrows in the state diagram indicate the condition under which the system remains in a state or transitions to the next state.

First State The Gray code is 00. In this state, the light is green on the main street and red on the side street for 25 s when the long timer is *on* or there is no vehicle on the side street. This condition is expressed as $T_L + \overline{V}_s$. The system transitions to the next state when the long timer goes *off* and there is a vehicle on the side street. This condition is expressed as $\overline{T}_L V_s$.

Second State The Gray code is 01. In this state, the light is yellow on the main street and red on the side street. The system remains in this state for 4 s when the short timer is *on.* This condition is expressed as T_S. The system transitions to the next state when the short timer goes *off*. This condition is expressed as $\overline{T}_S$.

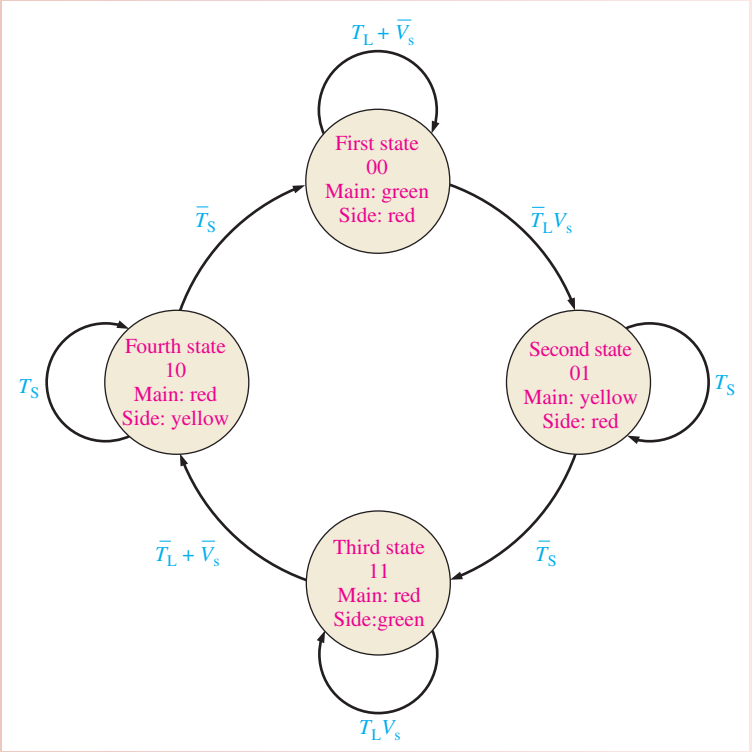


FIGURE 6–63 State diagram for the traffic signal control.

Third State The Gray code is 11. In this state, the light is red on the main street and green on the side street for 25 s when the long timer is *on* as long as there is a vehicle on the side street. This condition is expressed as $T_L V_s$. The system transitions to the next state when the long timer goes *off* or when there is no vehicle on the side street. This condition is expressed as $\overline{T}_L + \overline{V}_s$.

Fourth State The Gray code is 10. In this state, the light is red on the main street and yellow on the side street. The system remains in this state for 4 s when the short timer is *on.* This condition is expressed as T_S. The system transitions back to the first state when the short timer goes *off*. This condition is expressed as $\overline{T}_S$.

Exercise

1. How long can the system remain in the first state?
2. How long can the system remain in the fourth state?
3. Write the expression for the condition that produces a transition from the first state to the second state.
4. Write the expression for the condition that keeps the system in the second state.

Block Diagram

The traffic signal controller consists of three units: combinational logic, sequential logic, and timing circuits, as shown in Figure 6–64. The combinational logic unit provides outputs to turn the signal lights on and off. It also provides trigger outputs to start the long and short timers. The input sequence to this logic represents the four states described by the state diagram. The timing circuits unit provides the 25 s and the 4 s timing outputs. A frequency divider in the timing circuits unit divides the system clock down to a 1 Hz clock for use in producing the 25 s and 4 s signals. The sequential logic unit produces the sequence of 2-bit Gray codes representing the four states.

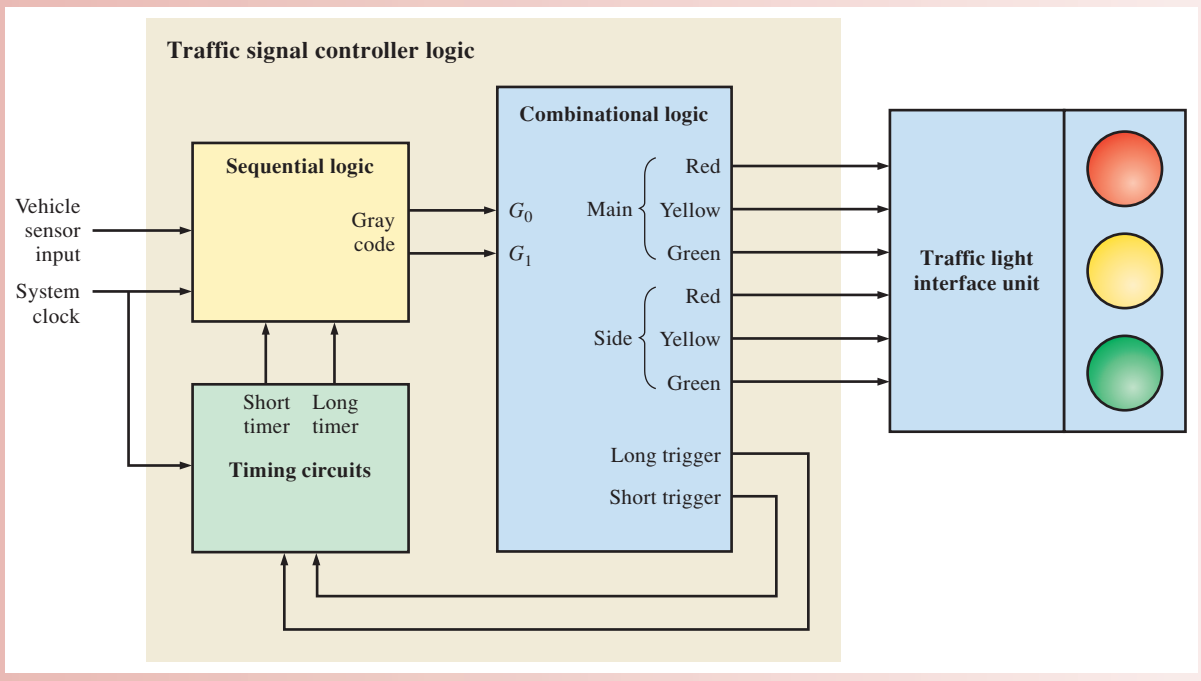


FIGURE 6–64 Block diagram of the traffic signal controller.

The Combinational Logic

The combinational logic consists of a state decoder, light output logic, and trigger logic, as shown in Figure 6–65.

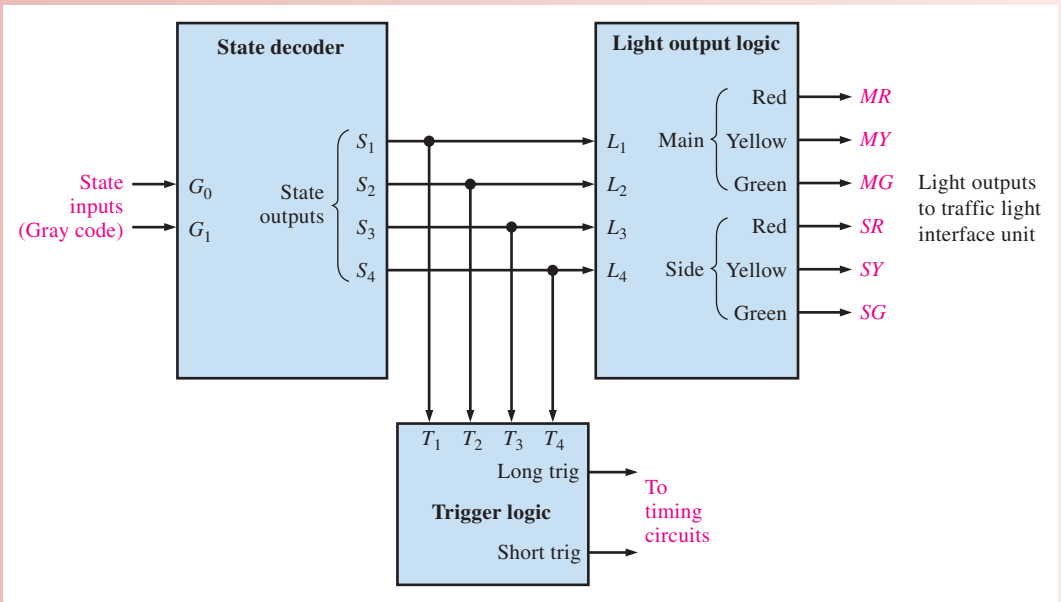


FIGURE 6–65 Block diagram of the combinational logic unit.

State Decoder This logic decodes the 2-bit Gray code from the sequential logic to determine which of the four states the system is in. The inputs to the state decoder are the two Gray code bits G_1 and G_0. There are four state outputs S_1, S_2, S_3, and S_4. For each of the

four input codes, one and only one of the outputs is activated. The Boolean expressions for the state outputs in terms of the inputs are

$$S_1 = \overline{G}_1\overline{G}_0$$
$$S_2 = \overline{G}_1 G_0$$
$$S_3 = G_1 G_0$$
$$S_4 = G_1\overline{G}_0$$

The truth table for the state decoder logic is shown in Table 6–11, and the logic diagram is shown in Figure 6–66.

TABLE 6–11

Truth table for the state decoder.

State Inputs (Gray Code)		State Outputs			
G_1	G_0	S_1	S_2	S_3	S_4
0	0	1	0	0	0
0	1	0	1	0	0
1	1	0	0	1	0
1	0	0	0	0	1

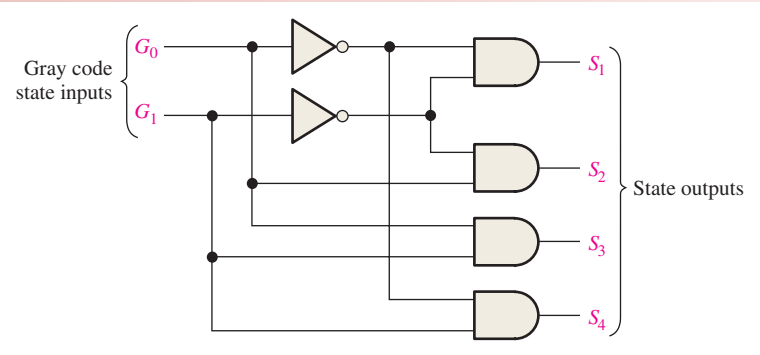


FIGURE 6–66 State decoder logic.

Light Output Logic This logic has the four state outputs (S_1–S_4) of the state decoder as its inputs (L_1–L_4) and produces six outputs to turn the traffic lights on and off. These outputs are designated *MR*, *MY*, *MG* (main red, main yellow, main green) and *SR*, *SY*, *SG* (side red, side yellow, side green).

The state diagram shows that the main red is *on* in the third state (L_3) or in the fourth state (L_4), so the Boolean expression is

$$MR = L_3 + L_4$$

The main yellow is *on* in the second state (L_2), so the expression is

$$MY = L_2$$

The main green is *on* in the first state (L_1), so the expression is

$$MG = L_1$$

Similarly, the state diagram is used to obtain the following expressions for the side street:

$$SR = L_1 + L_2$$
$$SY = L_4$$
$$SG = L_3$$

The logic circuit is shown in Figure 6–67.

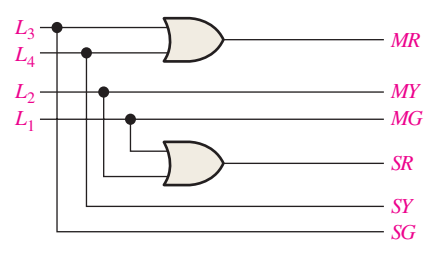


FIGURE 6–67 Light output logic.

Exercise

5. Show the logic diagram for the light output logic using specific IC devices with pin numbers.
6. Develop a truth table for the light output logic.

Trigger Logic The trigger logic produces two outputs, the long trigger output and the short trigger output. The long trigger output initiates the 25 s timer on a LOW-to-HIGH transition at the beginning of the first or third states. The short trigger output initiates the 4 s timer on a LOW-to-HIGH transition at the beginning of the second or fourth states. The Boolean expressions for this logic are

$$LongTrig = T_1 + T_3$$
$$ShortTrig = T_2 + T_4$$

Equivalently,

$$LongTrig = T_1 + T_3$$
$$ShortTrig = \overline{T_1 + T_3}$$

The logic circuit is shown in Figure 6–68.

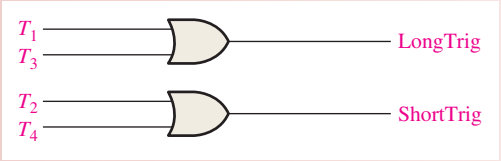


FIGURE 6–68 Trigger logic.

Exercise

7. Show the logic diagram for the trigger logic using specific IC devices with pin numbers.
8. Develop a truth table for the trigger logic.
9. Show the complete combinational logic by combining the state decoder, light output logic, and trigger logic. Include specific IC devices and pin numbers.

VHDL Descriptions

The VHDL program for the combinational logic unit of the traffic signal controller can be written using the data flow approach to describe each of the three functional blocks of the combinational logic unit. These functional blocks are the state decoder, the light output logic, and the trigger logic, as shown in Figure 6–65.

- The VHDL program code for the state decoder is as follows:

```
entity StateDecoder is
   port (G0, G1: in bit; S1, S2, S3, S4: out bit);
end entity StateDecoder;

architecture LogicOperation of StateDecoder is
begin
   S1 <= not G0 and not G1;
   S2 <= G0 and not G1;
   S3 <= G0 and G1;
   S4 <= not G0 and G1;
end architecture LogicOperation;
```

G0, G1: Gray code inputs
S1–S4: State outputs

Boolean expressions for state decoder outputs

- The VHDL program code for the light output logic is as follows:

```
entity LightOutputLogic is
   port (L1, L2, L3, L4: in bit; MR, MY, MG, SR, SY, SG: out bit);
end entity LightOutputLogic;

architecture LogicOperation of LightOutputLogic is
begin
   MR <= L3 or L4;
   MY <= L2;
   MG <= L1;
   SR <= L1 or L2;
   SY <= L4;
   SG <= L3;
end architecture LogicOperation;
```

Inputs and outputs declared

Boolean expressions for light output logic outputs

- The VHDL program code for the trigger logic is as follows:

```
entity TriggerLogic is
   port (T1, T2, T3, T4: in bit; LongTrig, ShortTrig: out bit);
end entity TriggerLogic;

architecture LogicOperation of TriggerLogic is
begin
   LongTrig <= T1 or T3;
   ShortTrig <= T2 or T4;
end architecture LogicOperation;
```

Inputs and outputs declared

Boolean expressions for trigger logic outputs

Development of the traffic signal controller will continue in the Applied Logic in Chapter 7.

Simulation

MultiSim

Open Multisim file AL06 in the Applied Logic folder on the website. Run the simulation for the combinational logic unit of the traffic signal controller and observe the operation for each of the four states in the light sequence.

Putting Your Knowledge to Work

There is a requirement for a pedestrian push button that would activate the yellow caution light for 4 s and the red light for 15 s on both the main street and the side street. (a) Modify the state diagram for this additional feature. (b) Develop the additional logic required.

SUMMARY

- Half-adder and full-adder operations are summarized in truth Tables 6–12 and 6–13.

TABLE 6–12

Inputs		Carry Out	Sum
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

TABLE 6–13

Inputs		Carry In	Carry Out	Sum
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Combination logic functions include comparators, decoders, encoders, code converters, multiplexers, demultiplexers, and parity generators/checkers.
- Software versions of standard logic functions from the 74XX series are available for use in a programmable logic design.

KEY TERMS

Key terms and other bold terms in the chapter are defined in the end-of-book glossary.

Cascading Connecting two or more similar devices in a manner that expands the capability of one device.

Comparator A digital circuit that compares the magnitudes of two quantities and produces an output indicating the relationship of the quantities.

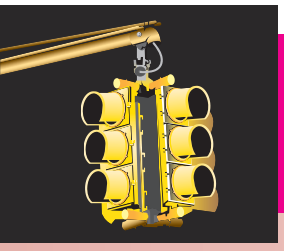
Decoder A digital circuit that converts coded information into a familiar or noncoded form.

Demultiplexer (DEMUX) A circuit that switches digital data from one input line to several output lines in a specified time sequence.

Encoder A digital circuit that converts information to a coded form.

Full-adder A digital circuit that adds two bits and an input carry to produce a sum and an output carry.

Glitch A voltage or current spike of short duration, usually unintentionally produced and unwanted.



Applied Logic

Traffic Signal Controller: Part 2

The combinational logic unit of the traffic signal controller was completed in Chapter 6. Now, the timing circuits and sequential logic are developed. Recall that the timing circuits produce a 25 s time interval for the red and green lights and a 4 s interval for the yellow caution light. These outputs will be used by the sequential logic. The block diagram of the complete traffic signal controller is shown in Figure 7–64.

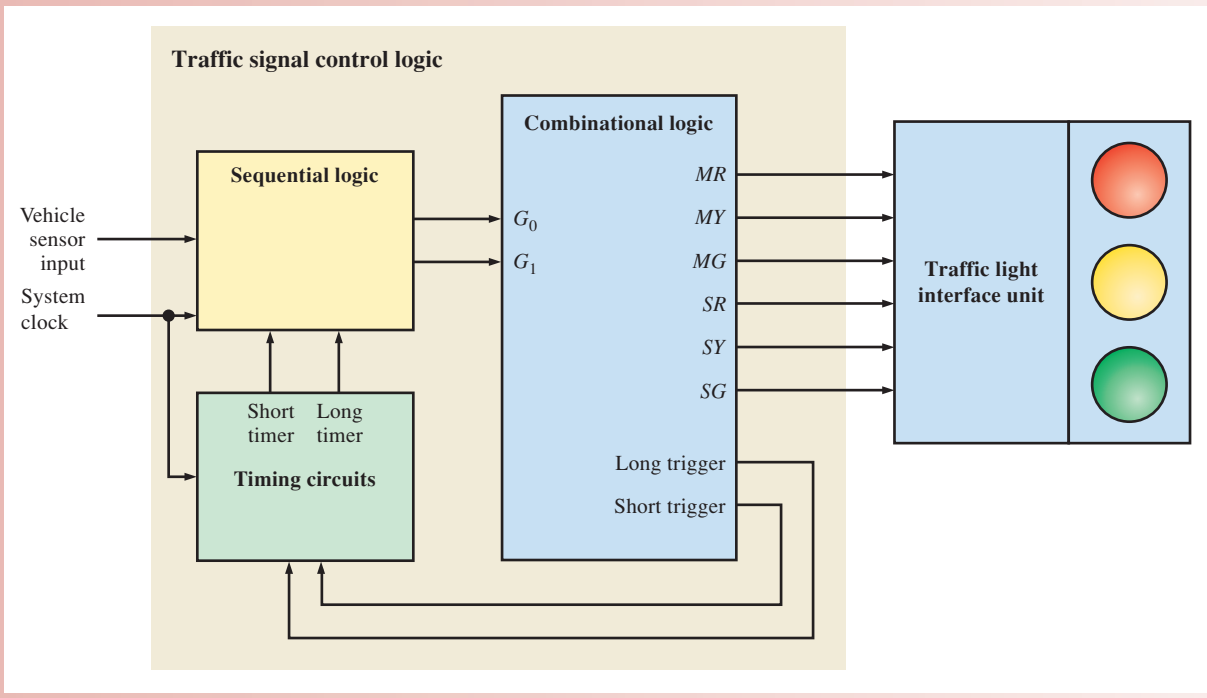


FIGURE 7–64 Block diagram of the traffic signal controller.

Timing Circuits

The timing circuits unit of the traffic signal controller consists of a 25 s timer and a 4 s timer and a clock generator. One way to implement this unit is with two 555 timers configured as one-shots and one 555 timer configured as an astable multivibrator (oscillator), as discussed earlier in this chapter. Component values are calculated based on the formulas given.

Another way to implement the timing circuits is shown in Figure 7–65. An external 24 MHz system clock (arbitrary value) is divided down to an accurate 1 Hz clock by the frequency divider. The 1 Hz clock is then used to establish the 25 s and the 4 s intervals by counting the 1 Hz pulses. This approach lends itself better to a VHDL description.

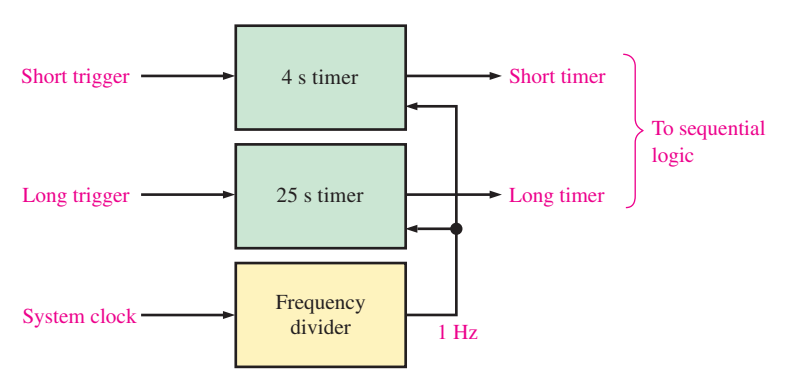


FIGURE 7–65 Block diagram of the timing circuits unit.

Exercise

1. Determine the values for the resistor and capacitor in a 25 s 555 timer.
2. Determine the values for the resistor and capacitor in a 4 s 555 timer.
3. What is the purpose of the frequency divider?

Controller Programming with VHDL

A programming model for the traffic signal controller is shown in Figure 7–66, where all the input and output labels are given. Notice that the Timing circuits block is split into two parts; the Frequency divider and the Timer circuits; and the Combinational logic block is divided into the State decoder and two logic sections (Light output logic and Trigger logic). This model will be used to develop the VHDL program codes.

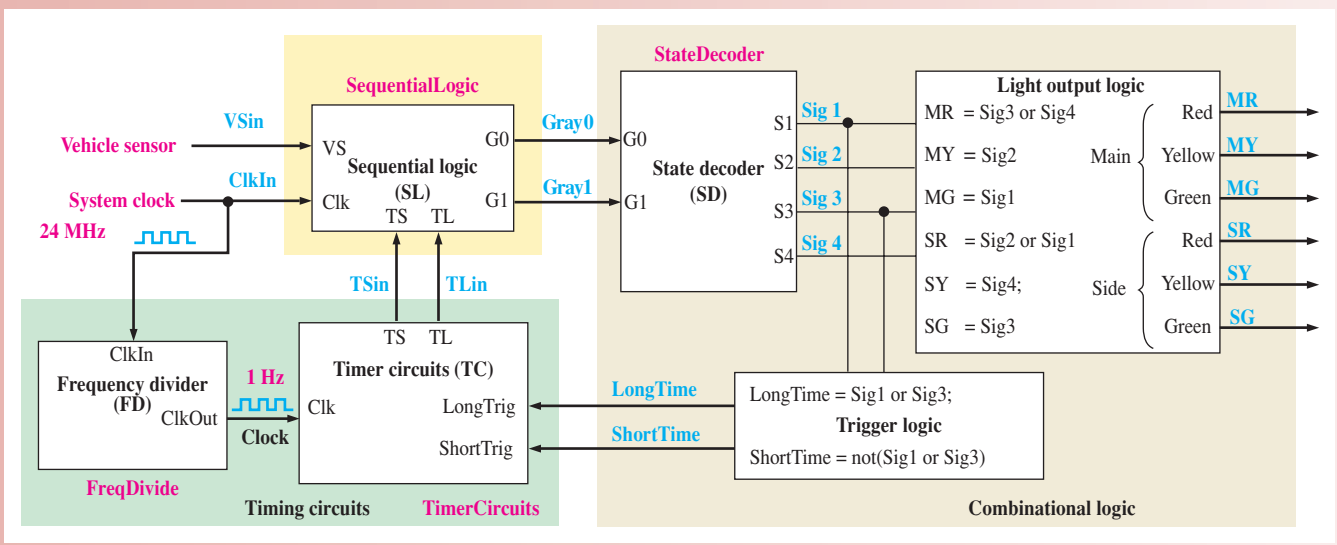


FIGURE 7–66 Programming model for the traffic signal controller.

Frequency Divider The purpose of the frequency divider is to produce a 1 Hz clock for the timer circuits. The input ClkIn in this application is a 24.00 MHz oscillator that drives the program code. SetCount is used to initialize the count for a 1 Hz interval. The program

FreqDivide counts up from zero to the value assigned to SetCount (one-half the oscillator speed) and inverts the output identifier ClkOut.

The integer value Cnt is set to zero prior to operation. The clock pulses are counted and compared to the value assigned to SetCount. When the number of pulses counted reaches the value in SetCount, the output ClkOut is checked to see if it is currently set to a 1 or 0. If ClkOut is currently 0, ClkOut is assigned a 1; otherwise, ClkIn is set to 1. Cnt is assigned a value of 0 and the process repeats. Toggling the output ClkOut each time the value of SetCount is reached creates a 1 Hz clock output with a 50% duty cycle.

The VHDL program code for the frequency divider is as follows:

```
library ieee;                                          Clkln: 24.00 MHz clock driver
use ieee.std_logic_1164.all;                           ClkOut: Output at 1 Hz

entity FreqDivide is
port(Clkln, in std_logic;                              Cnt: Counts up to value in SetCount
     ClkOut: buffer std_logic);                        SetCount: Holds 1/2 timer interval value
end entity FreqDivide;

architecture FreqDivide Behavior of FreqDivide is
begin
  FreqDivide: process(Clkln)
  variable Cnt: integer := 0;
  variable SetCount: integer;

begin
  SetCount := 12000000; -- 1/2 duty cycle              SetCount is assigned a value equal to half the system clock to produce a 1 Hz output. In this case, a 24 MHz system clock is used.
  if (ClkIn'EVENT and ClkIn = '1') then  <-- The if statement causes program to wait for a clock event and clock = 1 to start operation.
    if (Cnt = SetCount) then
      if ClkOut = '0' then                             Check that the terminal value in SetCount has been reached at which time ClkOut is toggled and Cnt is reset to 0.
        ClkOut <= '1'; --Output high 50%
      else
        ClkOut <= '0'; --Output Low 50%
      end if;
      Cnt := 0;
    else
      Cnt := Cnt + 1;  <-- If terminal value has not been reached, Cnt is incremented.
    end if;
  end if;
end process;
end architecture FreqDivideBehavior;
```

Timer Circuits The program TimerCircuits uses two one-shot instances consisting of a 25 s timer (TLong) and a 4 s timer (TShort). The 25 s and the 4 s timers are triggered by long trigger (LongTrig) and short trigger (ShortTrig). In the VHDL program, countdown timers driven by a 1 Hz clock input (Clk) replicate the one-shot components TLong and TShort. The values stored in SetCountLong and SetCountShort are assigned to the Duration inputs of one-shot components TLong and TShort, setting the 25-second and 4-second timeouts. When Enable is set LOW, the one-shot timer is initiated and output QOut is set HIGH. When the one-shot timers time out, QOut is set LOW. The output of one-shot component TLong is sent to TimerCircuits identifier TL. The output of one-shot component TShort is sent to TimerCircuits identifier TS.

The VHDL program code for the timing circuits is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity TimerCircuits is
   port(LongTrig, ShortTrig, Clk: in std_logic;
        TS, TL: buffer std_logic);
end entity TimerCircuits;

architecture TimerBehavior of TimerCircuits is
component OneShot is
   port(Enable, Clk: in std_logic;
        Duration :in integer range 0 to 25;
        QOut     :buffer std_logic);
end component OneShot;

signal SetCountLong, SetCountShort: integer range 0 to 25;
begin
   SetCountLong <= 25;
   SetCountShort <= 4;
   TLong:OneShot port map(Enable=>LongTrig, Clk=>Clk, Duration=>SetCountLong, QOut=>TL);
   TShort:OneShotport map(Enable=>ShortTrig, Clk=>Clk, Duration=>SetCountShort, QOut=>TS);
end architecture TimerBehavior;
```

LongTrig: Long timeout timer enable input
ShortTrig: Short timeout timer enable input
Clk: 1 Hz Clock input
TS: Short timer timeout signal
TL: Long timer timeout signal

Component declaration for OneShot.

SetCountLong: Holds long timer duration
SetCountShort: Holds short timer duration

Long and short count times are hard-coded to 25 and 4 based on a 1 Hz clock.

← Instantiation TLong

← Instantiation TShort

Sequential Logic

The sequential logic unit controls the sequencing of the traffic lights, based on inputs from the timing circuits and the side street vehicle sensor. The sequential logic produces a 2-bit Gray code sequence for each of the four states that were described in Chapter 6.

The Counter The sequential logic consists of a 2-bit Gray code counter and the associated input logic, as shown in Figure 7–67. The counter produces the four-state sequence on outputs G_0 and G_1. Transitions from one state to the next are determined by the short timer (T_S), the long timer (T_L), and vehicle sensor (V_s) inputs.

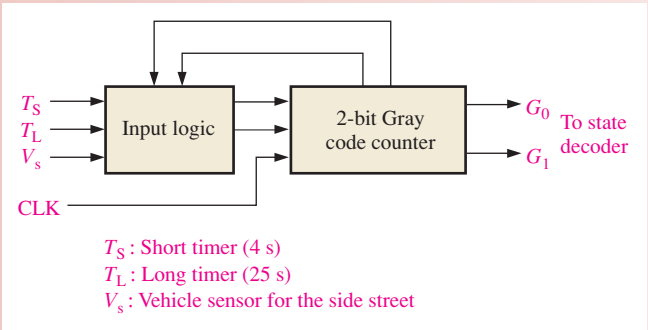


FIGURE 7–67 Block diagram of the sequential logic.

The diagram in Figure 7–68 shows how two D flip-flops can be used to implement the Gray code counter. Outputs from the input logic provide the *D* inputs to the flip-flops so they sequence through the proper states.

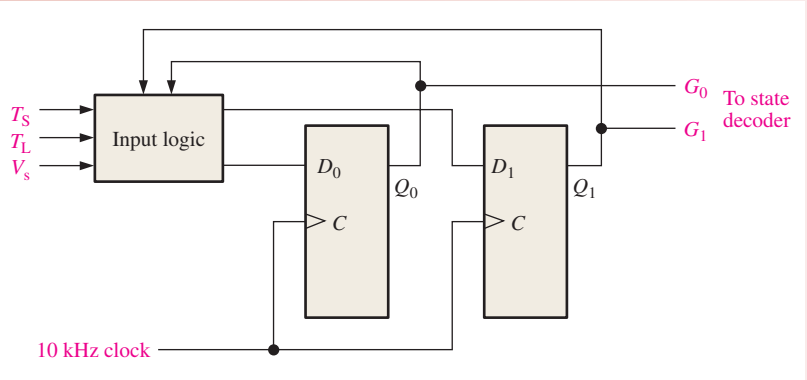


FIGURE 7–68 Sequential logic diagram with two *D* flip-flops used to implement the 2-bit Gray code counter.

The D flip-flop transition table is shown in Table 7–5. A next-state table developed from the state diagram in Chapter 6 Applied Logic is shown in Table 7–6. The subject of counter design is covered further in Chapter 8.

TABLE 7–5

D flip-flop transition table. Q_N is the output before clock pulse. Q_{N+1} is output after clock pulse.

Output Transitions			Flip-Flop Input
Q_N		Q_{N+1}	D
0	⟶	0	0
0	⟶	1	1
1	⟶	0	0
1	⟶	1	1

TABLE 7–6

Next-state table for the counter.

Present State		Next State			FF Inputs	
Q_1	Q_0	Q_1	Q_0	Input Conditions	D_1	D_0
0	0	0	0	$T_L + \overline{V}_s$	0	0
0	0	0	1	$\overline{T}_L V_s$	0	1
0	1	0	1	T_S	0	1
0	1	1	1	$\overline{T}_S$	1	1
1	1	1	1	$T_L V_s$	1	1
1	1	1	0	$\overline{T}_L + \overline{V}_s$	1	0
1	0	1	0	T_S	1	0
1	0	0	0	$\overline{T}_S$	0	0

The Input Logic Using Tables 7–5 and 7–6, the conditions required for each flip-flop to go to the 1 state can be determined. For example, G_0 goes from 0 to 1 when the present state is 00 and the condition on input D_0 is $\overline{T}_L V_s$, as indicated on the second row of Table 7–6. D_0 must be a 1 to make G_0 go to a 1 or to remain a 1 on the next clock pulse. A Boolean expression describing the conditions that make D_0 a 1 is derived from Table 7–6 as follows:

$$D_0 = \overline{G}_1\overline{G}_0\overline{T}_L V_s + \overline{G}_1 G_0 T_S + \overline{G}_1 G_0 \overline{T}_S + G_1 G_0 T_L V_s$$

In the two middle terms, the T_S and the $\overline{T}_S$ variables cancel, leaving the expression

$$D_0 = \overline{G}_1\overline{G}_0\overline{T}_L V_s + \overline{G}_1 G_0 + G_1 G_0 T_L V_s$$

Also, from Table 7–6, an expression for D_1 can be developed as follows:

$$D_1 = \overline{G}_1 G_0 \overline{T}_S + G_1 G_0 T_L V_s + G_1 G_0 \overline{T}_L + G_1 G_0 \overline{V}_s + G_1 \overline{G}_0 T_S$$

Based on the minimized expression for D_0 and D_1, the complete sequential logic diagram is shown in Figure 7–69.

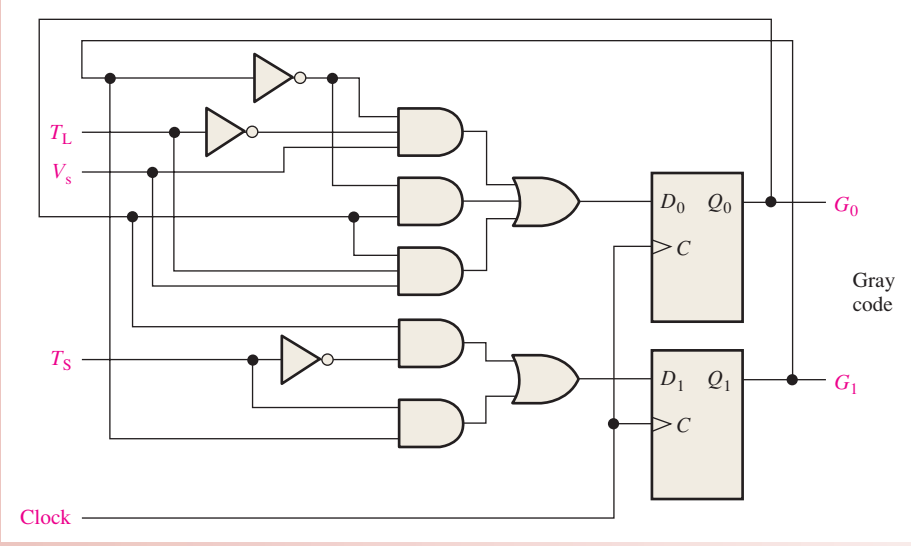


FIGURE 7–69 Complete diagram for the sequential logic.

Exercise

4. State the Boolean law and rule that permits the cancellation of T_S and $\overline{T}_S$ in the expression for D_0.
5. Use the Karnaugh map to reduce the D_0 expression further to a minimum form.
6. Use Boolean laws, rules, and/or the Karnaugh map to reduce the D_1 expression to a minimum form.
7. Do your minimized expressions for D_0 and D_1 agree with the logic shown in Figure 7–69?

The Sequential Logic with VHDL

The program SequentialLogic describes the Gray code logic needed to drive the traffic signal controller based on input from the timing circuits and the side street vehicle sensor. The sequential logic code produces a 2-bit Gray code sequence for each of the

four sequence states. The component definition dff is used to instantiate two D flip-flop instances DFF0 and DFF1. DFF0 and DFF1 produce the two-bit Gray code. The Gray code output sequences the traffic signal controller through each of four states. Internal variables D0 and D1 store the results of the D0 and D1 Boolean expressions developed in this chapter. The stored results in D0 and D1 are assigned to D flip-flops DFF0 and DFF1 along with the system clock to drive outputs G0 and G1 from the D flip-flop *Q* outputs.

The VHDL program code for the sequential logic is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity SequentialLogic is
port(VS, TL, TS, Clk: in std_logic; G0, G1: inout std_logic);
end entity SequentialLogic;

architecture SequenceBehavior of SequentialLogic is

component dff is
port (D, Clk: in std_logic; Q: out std_logic);
end component dff;

signal D0, D1: std_logic;
begin
D1 <= (G0 and not TS) or (G1 and TS);
D0 <= (not G1 and not TL and VS) or (not G1 and G0)
      or (G0 and TL and VS);

DFF0: dff port map(D=> D0, Clk => Clk, Q => G0);
DFF1: dff port map(D=> D1, Clk => Clk, Q => G1);

end architecture SequenceBehavior;
```

VS: Vehicle sensor input
TL: Long timer input
TS: Short timer input
Clk: System clock
G0: Gray code output bit 0
G1: Gray code output bit 1
D0: Logic for DFlipFlop DFF0
D1: Logic for DFlipFlop DFF1

Component declaration for D flip-flop (dff)

Logic definitions for D flip-flop inputs D0 and D1 derived from Boolean expressions developed in this chapter.

Component instantiations

The Complete Traffic Signal Controller

The program TrafficLights completes the traffic signal controller. Components FreqDivide, TimerCircuits, SequentialLogic, and StateDecoder are used to compose the completed system. Signal CLKin from the TrafficLights program source code is the clock input to the FreqDivide component. The frequency divided output ClkOut is stored as local variable Clock and is the divided clock input to the TimerCircuits and SequentialLogic components. TimerCircuits is controlled by local variables LongTime and ShortTime, which are controlled by the outputs Sig1 and Sig3 from component StateDecoder. StateDecoder also provides outputs Sig1 through Sig4 to control the traffic lights MG, SG, MY, SY, MR, and SR. TimerCircuit timeout signals TS and TL are stored in variables TLin (timer long in) and TSin (timer short in).

Signals TSin and TLin from TimerCircuits are used along with vehicle sensor VSin as inputs to the SequentialLogic component. The outputs from SequentialLogic G0 and G1 are stored in variables Gray0 and Gray1 as inputs to component StateDecoder. Component StateDecoder returns signals S1 through S4 which are in turn passed to variables Sig1 through Sig4. The light output logic and trigger logic developed in Chapter 6 are not used as components in this program, but are stated as logic expressions. The values stored in variables Sig1 through Sig4 provide the logic for outputs MG, SG, MY, SY, MR, SR; and local timer triggers LongTime and ShortTime are sent to TimerCircuits.

The VHDL program code for the traffic signal controller is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity TrafficLights is
port(VSin, ClkIn: in std_logic; MR, SR, MY, SY, MG, SG: out std_logic);
end entity TrafficLights;

architecture TrafficLightsBehavior of TrafficLights is

component StateDecoder is
port(G0, G1: in std_logic; S1, S2, S3, S4: out std_logic);
end component StateDecoder;

component SequentialLogic is
port(VS, TL, TS, Clk: in std_logic; G0, G1: inout std_logic);
end component SequentialLogic;

component TimerCircuits is
port(LongTrig, ShortTrig, Clk: In std_logic; TS, TL: buffer std_logic);
end component TimerCircuits;

component FreqDivide is
port(Clkin: in std_logic; ClkOut: buffer std_logic);
end component FreqDivide;

signal Sig1, Sig2, Sig3, Sig4, Gray0, Gray1: std_logic;
signal LongTime, ShortTime, TLin, TSin, Clock: std_logic;

begin
MR <= Sig3 or Sig4;
SR <= Sig2 or Sig1;
MY <= Sig2;
SY <= Sig4;
MG <= Sig1;
SG <= Sig3;

LongTime <= Sig1 or Sig3;
ShortTime <= not(Sig1 or Sig3);

SD: StateDecoder     port map (G0 => Gray0, G1 => Gray1, S1 => Sig1, S2 => Sig2, S3 => Sig3, S4 => Sig4);
SL: SequentialLogic  port map (VS => VSin, TL => TLin, TS => TSin, Clk => Fout, G0 => Gray0, G1 => Gray1);
TC: TimerCircuits    port map (LongTrig=>LongTime, ShortTrig=>ShortTime, Clk=>Clock, TS=>TSin, TL=>TLin);
FD: FreqDivide       port map (Clkln => CLKin, ClkOut =>-Clock);

end architecture TrafficLightsBehavior;
```

VSin : Vehicle sensor input
CLKin : System Clock
MR : Main red light output
SR : Side red light output
MY : Main yellow light output
SY : Side yellow light output
MG : Main green light output
SG : Side green light output

Component declaration for StateDecoder

Component declaration for SequentialLogic

Component declaration for TimerCircuits

Component declaration for FreqDivider

Logic definitions for the light output logic

Sig1-4 : Return values from StateDecoder
Gray0-1 : SequentialLogic Gray code return
LongTime : Trigger input to TimerCircuits
ShortTime : Trigger input to TimerCircuits
TLin : Store TimerCircuits long timeout
TSin : Store TimerCircuits Short timeout
Clock : Divided clock from FreqDivide

Logic definitions for the trigger logic

Component instantiations

Simulation

MultiSim

Open file AL07 in the Applied Logic folder on the website. Run the traffic signal controller simulation using your Multisim software and observe the operation. Lights will appear randomly when first turned on. Simulation times may vary.

Putting Your Knowledge to Work

Add your modification for the pedestrian input developed in Chapter 6 and run a simulation.

One-Shot with VHDL

An example of a VHDL program code for a one-shot is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity OneShot is
   port (Enable, Clk: in std_logic;
         Duration: in integer range 0 to 25;
         QOut: buffer std_logic);
end entity OneShot;

architecture OneShotBehavior of OneShot is
begin
   Counter: process (Enable, Clk, Duration)
   variable Flag        : boolean := true;
   variable Cnt         : integer range 0 to 25;
   variable SetCount : integer range 0 to 25;
   begin
        SetCount := Duration;
        if (Clk'EVENT and Clk = '1') then
        if Enable = '0' then
              Flag := true;
           end if;

           if Enable = '1' and Flag then
           Cnt := 1;
                Flag :=False;
        end if;

           if cnt = SetCount then
         Qout <= '0';
                Cnt := 0;
                     Flag := false;
        else
                if Cnt > 0 then
                Cnt := Cnt + 1;
                     Qout <= '1';
           end if;
              end if;
        end if;
   end process;
end architecture OneShotBehavior;
```

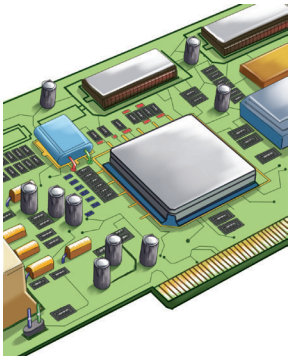
In normal operation, a one-shot produces only a single pulse, which can be difficult to measure on an oscilloscope because the pulse does not occur regularly. To obtain a stable display for test purposes, it is useful to trigger the one-shot from a pulse generator that is set to a longer period than the expected pulse width and trigger the oscilloscope from the same pulse. For very long pulses, either store the waveform using a digital storage oscilloscope or shorten the time constant by some known factor. For example, replace a 1000 μF capacitor with a 1 μF capacitor to shorten the time by a factor of 1000. A faster pulse is easier to see and measure with an oscilloscope.

Programmable Logic Device (PLD) The positive edge-triggered D flip-flop can be described using VHDL and implemented as hardware in a PLD. In this program, the behavioral approach will be used for the first time because it lends itself to describing sequential operations. A new VHDL statement, **wait until rising_edge**, is introduced. This statement allows the program to wait for the rising edge of a clock pulse to process the *D* input to create the desired results. Also the **if then else** statement is introduced. The keyword **process** is a block of code placed between the **begin** and **end** statements of the architecture to allow statements to be sequentially processed. The program code for a single D flip-flop is as follows:

```
library ieee;
use ieee.std_logic_1164.all;

entity dffl is                                          D: Flip-flop input
   port (D, Clock, Pre, Clr: in std_logic; Q: inout std_logic);   Clock: System clock
end entity dffl;                                        Pre: Preset input
                                                        Clr: Clear input
                                                        Q: Flip-flop output
architecture LogicOperation of dffl is
begin
process
   begin
      wait until rising_edge (Clock);
         if Clr = '1' then
            if Pre = '1' then                  } Check for Preset and Clear conditions
               if D = '1' then
                  Q <= '1';                    Q input follows D input when Clr and Pre inputs
               else                            are HIGH.
                  Q <= '0';
               end if;
            else
               Q <= '1';     Q is set HIGH when Pre input is LOW.
            end if;
         else
            Q <= '0';     Q is set LOW when Clr input is LOW.
      end if;
   end process;
end architecture LogicOperation;
```

IMPLEMENTATION: J-K FLIP-FLOP



Fixed-Function Device The 74HC112 dual J-K flip-flop has two identical flip-flops that are negative edge-triggered and have active-LOW asynchronous preset and clear inputs. The logic symbols are shown in Figure 7–29.

Programmable Logic Device (PLD) The negative edge-triggered J-K flip-flop can be described using VHDL and implemented as hardware in a PLD. In this program, the behavioral approach will be used. A new VHDL statement, **if falling edge then,** is introduced. This statement allows the program to wait for the falling edge of a clock pulse