State Machine Design and Analysis

Due: Preliminary Questions By 6:00pm on Wednesday April 30.
Due: Final Lab Report By 6:00pm on Wednesday May 7.

The objective of this lab is to design and implement a clocked synchronous state machine using VHDL.

1 A Traffic Light

Our task is to implement a controller for a traffic light. The sequence of lights is to be as follows:

N-S Light	E-W Light
R	R
G	R
Y	R
R	R
R	G
R	Y

where the N-S light controls the flow of traffic arriving at the intersection from the north and the south and the E-W light controls traffic arriving at the intersection from the east and west. The red-red condition provides a bit of a safety margin.

However, traffic controllers also have timing requirements. For example we can specify that a Green light is required to stay on for 30 seconds and the Yellow light for 4 seconds. If we add the requirement that both lights are Red for only one second, the timing for the traffic light controller is completely specified.

Ordinarily, the above sequence is to be repeated without alteration, but to allow a traffic officer to manually direct traffic, we provide a HOLD control. If the HOLD signal is asserted, the normal sequence of lights proceeds until it arrives at either of the two points in the sequence where both lights are red, and then both lights are held red until the HOLD signal is released.

1.1 Implementing the Timing

There are a number of different ways to implement such a controller. Since the clock period for the controller is one second, one way to keep the green light on for 30 seconds would be to replace each of the two states that set the green light on with a linear sequence of 30 states, each of which has the same outputs as the state that was replaced. Similarly, we would replace each of the states that set a yellow light on with 4 equivalent states. Thus, with the two states that are responsible for Red-Red outputs, our controller would require a total of 70 states. While this only requires 7 flip-flops, it would be very difficult to design since it is next to impossible to use K-Map techniques to minimize functions of more than 6 variables.

Another way to implement the controller is to use a separate counter (incremented once each clock cycle) to keep track of the number of seconds that have elapsed. The counter is enabled when a state that produces a green or a yellow light is entered, and the machine stays in the state until the counter reaches the appropriate value. With this, our state diagram can be drawn as in Figure 1. Since the state in which a yellow light is turned on always follows a state in which the green light was on, we don't have to reset the counter between these two states. We merely stay in the state that produces a green light until the counter reaches 30, and then we move to the state that enables the yellow light and stay in this state until the counter reaches 34.

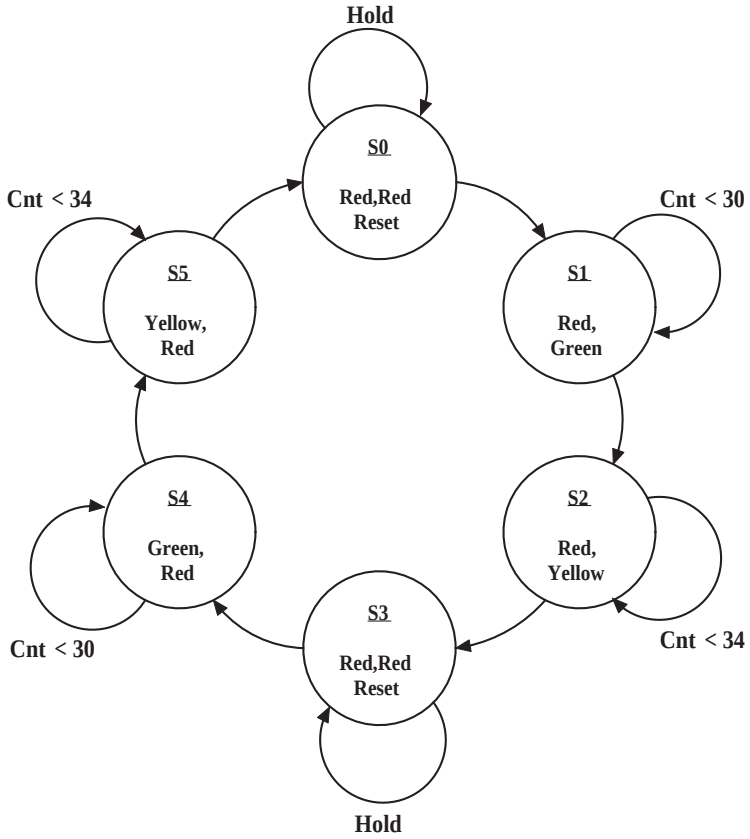


Figure 1: State Diagram for the Traffic Light Controller

We see that we will require three flip-flops to store the state variables, and that we also have three external inputs (CNT=30, CNT=34, HOLD).

As an alternative, this diagram could be drawn as an ASM chart as shown in Figure 2.

1.2 Using MSI Counters

The following circuit shows how to construct a counter circuit capable of generating the required CNT=30 and CNT=34 signals from two 7490 decimal counters. We note that each 7490 is actually two separate

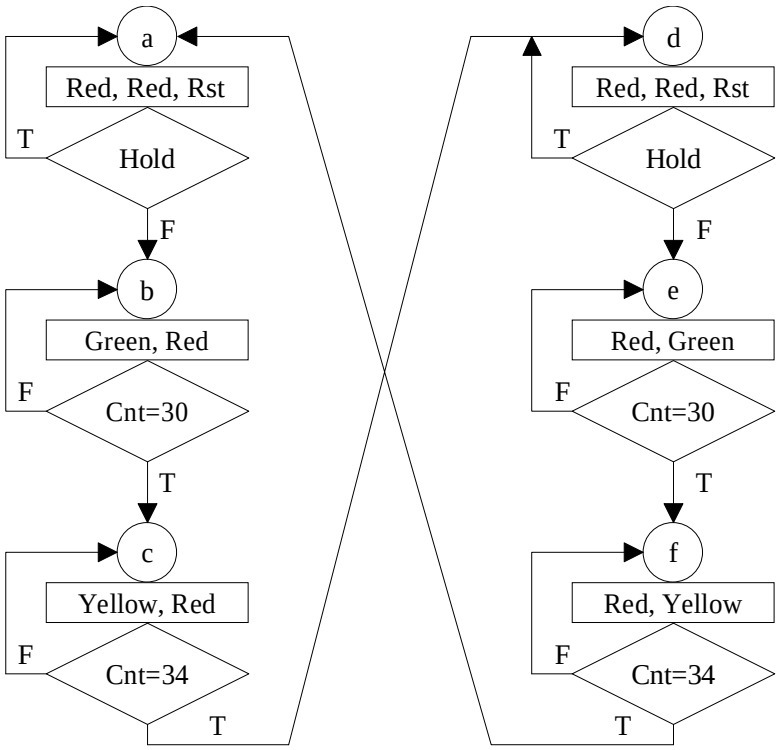


Figure 2: State Diagram for the Traffic Light Controller

counters. CKA is the input to a divide by two counter while CKB is the input to a divide by 5 counter. Using CKA as the circuit input and connecting Q_A to CKB creates a divide by 10 counter. The outputs Q_D Q_C Q_B and Q_A then give a binary count (0000_2 through 1001_2) of the number of negative going edge transitions presented to CKA. When the count is 1001_2, and CKA makes another negative transition the count proceeds to 0000_2. Thus, for each cycle of 10 clock pulses on CKA, Q_D produces 1 pulse, hence the designation of this circuit as a *divide by 10* counter. A two digit decimal counter can be produced by connecting the Q_D output to a CKB input on another divide by 10 counter as we do in our example. We note that since the yellow light always follows a green light, we don't have to actually test for the counter reaching 34. Testing the least significant decimal digit to see when it reaches 4 is equivalent to testing for 34, since we already reached 30 in the previous state.

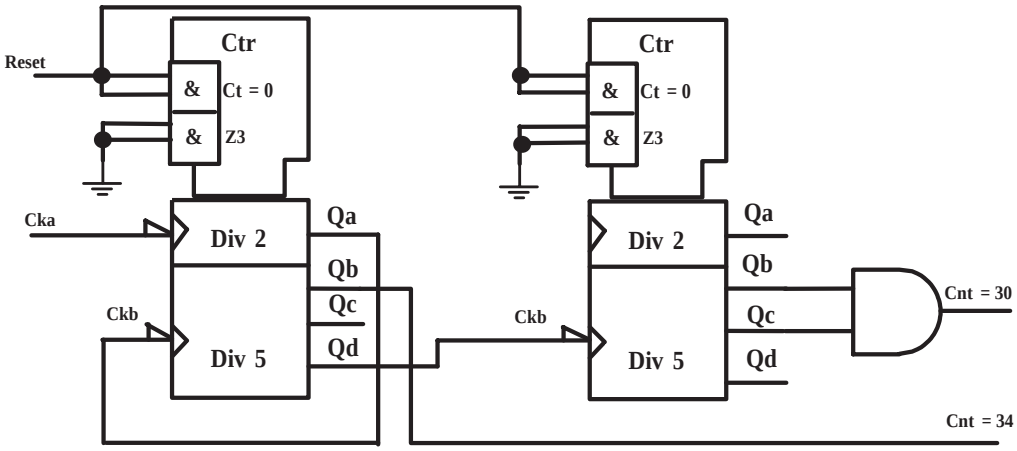


Figure 3: A Binary Counter

1.3 VHDL Implementation

The state machine described above, is actually easier to implement using VHDL that it is using hardware. This is because we can only need to describe the desired behavior. The source file is available on the course web page.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

library SYNOPSYS;
use SYNOPSYS.attributes.all;

entity  stop is
  port (CLK: in STD_LOGIC;
        Hold: in STD_LOGIC;
        RST: in STD_LOGIC;
        EW1: out STD_LOGIC;
        EW2: out STD_LOGIC;
        NS1: out STD_LOGIC;
        NS2: out STD_LOGIC);
end;

architecture  stop_arch of  stop is

component ibuf port(I: in std_logic;
```

```
  O: out std_logic);
end component;

component bufg port(I: in std_logic;
  O: out std_logic);
end component;


-- SYMBOLIC ENCODED state machine: Sreg0
type Sreg0_type is (S0, S1, S2, S3, S4, S5, S6);
signal Sreg0: Sreg0_type;

signal Clear: STD_LOGIC;
signal RESET: STD_LOGIC;
signal COUNT: INTEGER range 0 to 128;
signal in_c, buf_c: std_logic;

begin

-- clock needs to be routed through the clock buffer
u0: ibuf port map(I => clk, O => in_c);
u1: bufg port map(I => in_c, O => buf_c);

--concurrent signal assignments

RESET <= (Clear or RST);

--diagram ACTIONS;


-- 4-bit synchronous counter with count enable,
-- asynchronous reset and synchronous load
--  CLK: in STD_LOGIC;
--  RESET: in STD_LOGIC;
--  COUNT: inout INTEGER range 0 to 15;

process (CLK, RESET)
begin
   if RESET='1' then
      COUNT <= 0;
   elsif CLK='1' and CLK'event then
```

```
      COUNT <= COUNT + 1;
   end if;
end process;



Sreg0_machine: process (CLK)

begin

if CLK'event and CLK = '1' then
        if RST='1' then
                Sreg0 <= S0;
        else
        case Sreg0 is
                when S0 =>
                        Sreg0 <= S1;
                when S1 =>
                  if Hold='0' then
                        Sreg0 <= S2;
                  else
                         Sreg0 <= S1;
                  end if;
                when S2 =>
                  if COUNT=30 then
                         Sreg0 <= S3;
                  else
                         Sreg0 <= S2;
                  end if;
               when S3 =>
                  if COUNT=34 then
                         Sreg0 <= S4;
                  else
                         Sreg0 <= S3;
                  end if;
               when S4 =>
                  if Hold='0' then
                         Sreg0 <= S5;
                  else
                         Sreg0 <= S4;
```

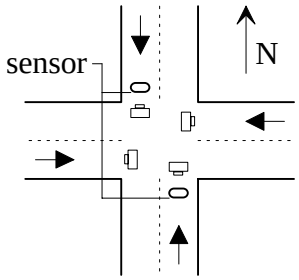
```
                end if;
             when S5 =>
                if COUNT=30 then
                       Sreg0 <= S6;
                else
                       Sreg0 <= S5;
                end if;
            when S6 =>
                if COUNT=34 then
                       Sreg0 <= S1;
                else
                       Sreg0 <= S6;
                end if;
            when others =>
                Sreg0 <= S0;
      end case;
      end if;
end if;
end process;

-- signal assignment statements for combinatorial outputs
NS1_assignment:
NS1 <= '1' when (Sreg0 = S1) else
       '0' when (Sreg0 = S2) else
       '0' when (Sreg0 = S3) else
       '1' when (Sreg0 = S4) else
       '1' when (Sreg0 = S5) else
       '1' when (Sreg0 = S6) else
       '0';

NS2_assignment:
NS2 <= '1' when (Sreg0 = S1) else
       '0' when (Sreg0 = S2) else
       '1' when (Sreg0 = S3) else
       '1' when (Sreg0 = S4) else
       '1' when (Sreg0 = S5) else
       '1' when (Sreg0 = S6) else
       '1';

EW1_assignment:
EW1 <= '1' when (Sreg0 = S1) else
```

```
       '1' when (Sreg0 = S2) else
       '1' when (Sreg0 = S3) else
       '1' when (Sreg0 = S4) else
       '0' when (Sreg0 = S5) else
       '0' when (Sreg0 = S6) else
       '0';

EW2_assignment:
EW2 <= '1' when (Sreg0 = S1) else
       '1' when (Sreg0 = S2) else
       '1' when (Sreg0 = S3) else
       '1' when (Sreg0 = S4) else
       '0' when (Sreg0 = S5) else
       '1' when (Sreg0 = S6) else
       '1';

Clear_assignment:
Clear <= '1' when (Sreg0 = S1) else
         '0' when (Sreg0 = S2) else
         '0' when (Sreg0 = S3) else
         '1' when (Sreg0 = S4) else
         '0' when (Sreg0 = S5) else
         '0' when (Sreg0 = S6) else
         '1';

end  stop_arch;
```

The text *Digital Design, Principles and Practices: 3rd Edition* by Wakerly has a slightly different traffic controller example in Chapter 9. The basic scheme is described in section 9.1.5 (page 808) and a partial VHDL implementation is given in section 9.2.4 (page 825).

2 Preliminary Questions

1. Create a state diagram for a controller for the traffic light that has magnetic loop detectors in the N-S traffic lanes as shown in the following figure.



Traffic in the eastbound and the westbound lanes will always see a green light unless a sensor detects a car in either the northbound or southbound lanes. Then the east-west traffic is stopped by the usual yellow to red sequence and the north-south traffic started. The north-south light should remain green at least long enough to allow one car to cross the intersection. As long as there are cars in either the northbound or southbound lanes (ie: the sensor is still asserted), the north-south light should remain green but for no longer than the maximum time given in the specifications below. After the north-south traffic is stopped, then the east-west traffic should be started again.

(a) The east-west green light is to be on for a minimum of 15 seconds. After this minimum period it is to remain on until the sensor is asserted.

(b) If the sensor is asserted, the north-south green light is to be on for a minimum of 10 seconds, and, following this, a maximum of 30 seconds.

(c) If the north-south light reaches the maximum time, it should cycle back red and allow the east-west traffic to move by. The east-west green light should be on for 15 seconds.

(d) At this point, if the sensor is still asserted, the east-west light should cycle to red and allow the north-south traffic to move as above.

(e) The yellow lights are to be on for 4 seconds.

(f) Both lights are to be red for 2 seconds following each yellow light.

(g) If the “hold” input is asserted, the traffic light sequence should proceed as usual but also as quickly as possible until both lights are red. In this process, the green light minimum times and the yellow light times are to be maintained. Once the lights are both red, the lights should remain red until the hold input is not asserted.

The times in the above specifications are themselves not terribly realistic, but it will be easier to debug the system with the shorter waiting periods.

For safety, if the machine gets into any unused states, the lights should both be set to yellow and the next state should be one of the two states in which both lights are red. Since the digilab boards

have two sets of red, yellow, and green LED's, assign the N-S traffic lights to one set of the red, green, and yellow LED's and the E-W traffic lights to the other set of red, green, and yellow LED's as follows:

N-S Light	N-S LED	E-W Light	E-W LED
R	LED1	R	LED4
G	LED3	R	LED4
Y	LED2	R	LED4
R	LED1	R	LED4
R	LED1	G	LED6
R	LED1	Y	LED5

Assign the sensor, hold, and reset inputs to either the buttons or the switches. Also, assign the sensor and hold inputs to the two extra red LED's 7 and 8 on the board as a visual indication of the state of the sensor and hold functions.

3 The Lab

1. Write a VDHL program to implement the traffic controller designed in the prelab.
2. Simulate your design using the Xilinx Foundation tools
3. Implement and download your design on the Digilab board. Have your design passed off by the TA.

Rather than using the clock on the Digilab board, implement the clock by connecting function generator at the lab station to your board as performed in lab 6. Set the function generator to 1 Hz.