Required Equipment

No FPGA hardware is required for this lab. Simulation only.

Traffic Light Controller

In this project, you will be implementing a traffic light controller that controls a main street, a side street, and walk lamps. You will be using a finite state machine to implement this controller. This project should look familiar if you took CS 152A. However, instead of using a graphical design, you will be coding the state machine.

The traffic light controller is for an intersection between a Main Street and a Side Street. Both streets have a red, yellow, and green signal light. Pedestrians have the option of pressing a walk button to turn all the traffic lights red and cause a single walk light to illuminate. Moreover, there is a sensor on the Side Street which tells the controller if there are cars still on the Side Street. This is summarized below:

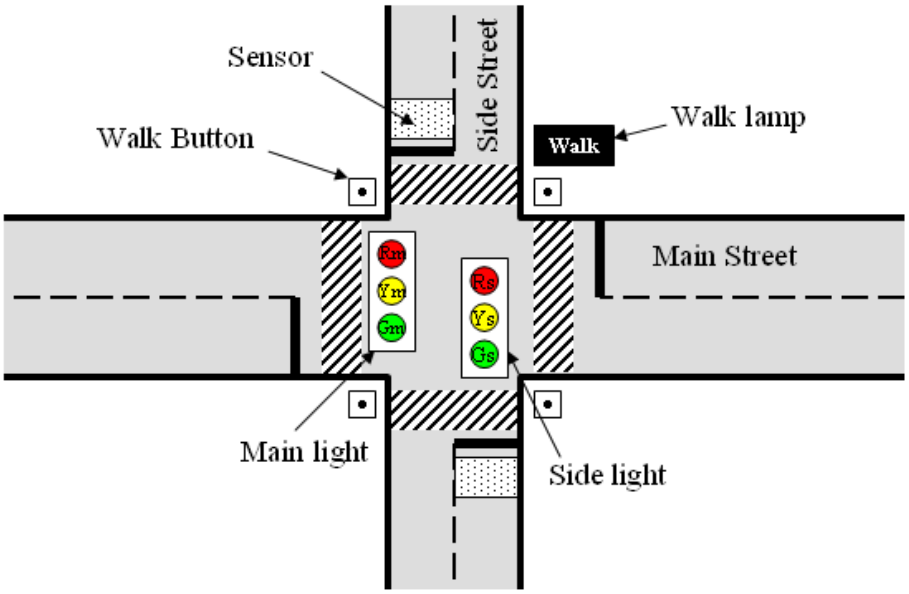

Figure 1: Diagram for intersection with corresponding lights.

The side street sensor is placed near the intersection to tell the controller when there are cars passing over the sensor. You may assume that the sensor will remain constantly high if several cars pass over the sensor, rather than quick pulses, provided the cars are close enough together. You do not need to implement this specific functionality. This input is named *Sensor*.

The traffic lights are timed on three parameters (in seconds): the base interval (6 seconds), the extended interval (3 seconds), and the yellow light interval (2 seconds).

The operating sequence of this intersection begins with the Main Street having a green light for 12 seconds. Next, the Main lights turn to yellow for *2 seconds* and then turn red while simultaneously turning on the Side Street green light. The Side Street is green for 6 seconds, and yellow is held for 2 seconds. Whenever a stoplight is green or yellow, the other street's stoplight is red. Under normal circumstances, this cycle repeats continuously.

There are several ways the controller can deviate from the typical loop:

1) A walk button allows pedestrians to submit a walk request. The internal Walk Register should be set on a button press and the controller should service the request after the Main Street yellow light by turning all street lights to red and the walk light to on. After a walk of *3* seconds, the traffic lights should return to their usual routine by turning the Side Street green. The Walk Register should be cleared at the end of a walk cycle.

2) The second deviation is the traffic sensor. If the traffic sensor is high at the end of the first 6 second length of the Main street green, the light should remain green only for an additional 3 seconds, rather than the full 6 seconds. Additionally, if the traffic sensor is high during the end of the Side Street green, it should remain green for an additional *3* seconds.

You may be required to debounce your input signals (as you learned in CS152A) in order to get your buttons to work properly. To check off this lab shows simulation results.

State Machine Implementation in Verilog

Below, please find the example code for a Verilog state machine. You are welcome to use other designs and state machine architectures for this lab. I don't recommend copy/pasting this code- but it should be a helpful reference.

Figure 1– Verilog State Machine (asic-world.com)

```
module fsm_using_always
(clock, reset, req_0, req_1, gnt_0, gnt_1);

input   clock,reset,req_0,req_1;
output  gnt_0,gnt_1;

wire    clock,reset,req_0,req_1;
reg     gnt_0,gnt_1;

//---------Internal Constants------------------
parameter SIZE = 3           ;
parameter IDLE  = 3'b001,GNT0 = 3'b010,GNT1 = 3'b100 ;

reg   [SIZE-1:0] state;
reg   [SIZE-1:0] next_state;

always @ (state or req_0 or req_1)
begin : FSM_COMBO
 next_state = 3'b000;
 case(state)
   IDLE : if (req_0 == 1'b1) begin
            next_state <= GNT0;
          end else if (req_1 == 1'b1) begin
            next_state <= GNT1;
          end else begin
            next_state <= IDLE;
          end
   GNT0 : if (req_0 == 1'b1) begin
            next_state <= GNT0;
          end else begin
            next_state <= IDLE;
          end
   GNT1 : if (req_1 == 1'b1) begin
            next_state <= GNT1;
          end else begin
            next_state <= IDLE;
          end
   default : next_state <= IDLE;
  endcase
end
```

```
always @ (posedge clock)
begin : FSM_SEQ
  if (reset == 1'b1) begin
    state <= #1 IDLE;
  end else begin
    state <= #1 next_state;
  end
end

always @ (posedge clock)

begin : OUTPUT_LOGIC
if (reset == 1'b1) begin
  gnt_0 <= 1'b0;
  gnt_1 <= 1'b0;
end
else begin
  case(state)
    IDLE : begin
              gnt_0 <= 1'b0;
              gnt_1 <= 1'b0;
           end
   GNT0 : begin
              gnt_0 <= 1'b1;
              gnt_1 <= 1'b0;
           end
   GNT1 : begin
              gnt_0 <= 1'b0;
              gnt_1 <= 1'b1;
           end
   default : begin
              gnt_0 <= 1'b0;
              gnt_1 <= 1'b0;
             end
  endcase
end
end // End Of Block OUTPUT_LOGIC

endmodule // End of Module arbiter
```

Report Requirements

Please submit your project report on CCLE. Include relevant snippets of code into your report. More importantly, show your test cases, screen shots of waveforms, and explanations of the tests and results to validate "correctness" of your design.