# ECE3829: Light Sensor
# A-Term 2020

Prudence Lam

October 2, 2020

## 1 Introduction

The purpose of this lab was to create a light sensor using the Basys3 Board. An SPI interface was designed for an Ambient Light Sensor module, and its value was displayed on two of the board's seven segment displays. The other two displays were dedicated to displaying the last two digits of my WPI ID, or nine and six.

From the PmodALS module provided, a total of six signals are required, three of which were focused on: Chip Select (CS), SDO (Master-In-Slave-Out), and SCLK (Series Clock). For PmodALS to work, SCLK was designed to run at 1 MHz. The program was designed to provide a light sensor reading every 500ms, or when CS was low. To read the data from the sensor, 16 cycles of SCLK are needed, with the bits placed on the falling edge of the clock and valid on the subsequent rising edge. Additionally, the data bits are loaded such that there are three leading zeros and four trailing zeros. The middle eight bits provide the light sensor data.

To effectively implement the program, multiple sequential circuits, including three counters, a shift register, and a Finite State Machine were used. The FSM was used as a control for the CS signal, and was driven by counters to keep track of the rate of each signal. Once 16 cycles of SCLK has been reached, the eight important bits of data were outputted to the seven segment display, and CS is high until the next cycle.

## 2 Discussion and Results

A 10MHz internal clock was first derived from the FPGA's 100 MHz clock using the MMCM. This was provided to an SPI interface that utilizes a state machine to control when the light sensor is read. To display the data, the seven segment module from the previous labs was instantiated. The following block diagram summarizes the inputs and outputs from a top level.

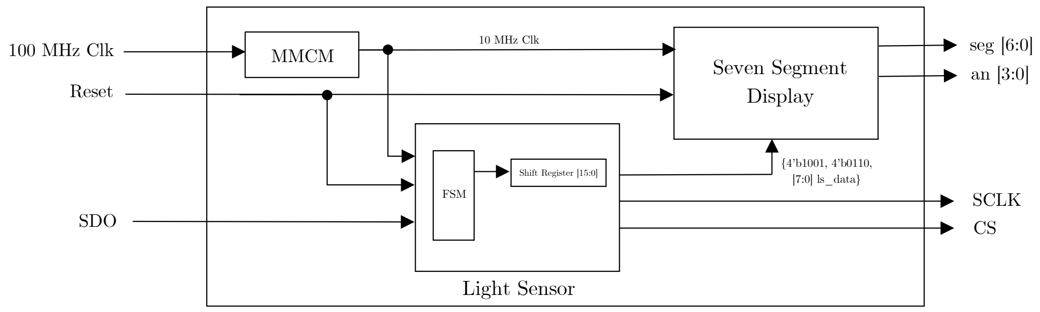


Figure 1: Block diagram of final implementation consisting of the MMCM, the light sensor logic, and the seven segment display.

Each module is explained in further detail below.

### 2.1 System Requirements

The light sensor module is mainly composed of two components: a Finite State Machine (FSM) and a 15-bit shift register. The FSM consists of three states, s0 (read), s1 (display), and s2 (wait 500ms), and can be depicted using the following diagram.

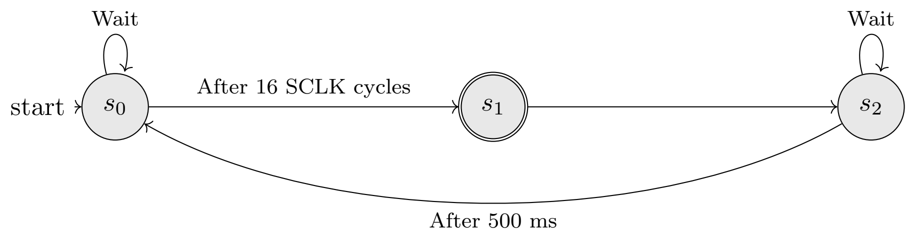


Figure 2: Diagram of the FSM acting as the "control" for the program. The three states are depicted along with their triggers.

The main purpose of the FSM is to act as a control for the program. It is driven by counters and is used to generate the signals to start the shift register and for CS.

1. **Counters**

   A total of three counters were implemented in this module. To derive the 1 MHz SCLK from the 10 MHz clock, a clock divider with a count of ten was used. For SCLK to appear as a square wave, it was set to one from zero to four, and zero for the rest. Another counter with a count of $5 * 10^5$ was created based on SCLK, effectively generating a signal every 2 Hz, or 500 ms. A final counter was used to count 16 cycles of SCLK while CS is low to determine when to output data to the seven segment display.

2. **S0: Reset State**

   When the program is first launched, SCLK is high. This state is used to generate CS, such that when it is entered, CS is low. As a result, the counter to 16 is incremented to count 16 cycles of SCLK. Additionally, a separate signal, `start_shift`, is generated at the falling edge of SCLK to inform the shift register to begin receiving data. By the time the counter reaches its maximum count, the shift register should have received all the data bits. As a result, a signal is sent back to the state machine to enter the next state.

3. **S1: Display State**

   This state is only entered for a short period of time, approximately one clock cycle of the 10 MHz clock. A flip flop was created to extract the important 8 bits of data from the data bus to output to the seven segment display. This allows the data to hold its value, only updating whenever this state is entered. The state machine simply moves on to the next state.

4. **S2: Wait State**

   For the light sensor data to be read every 2 Hz, or 500 ms, a state was dedicated to waiting. Once the 2 Hz clock divider reaches its maximum count, the state machine resets.

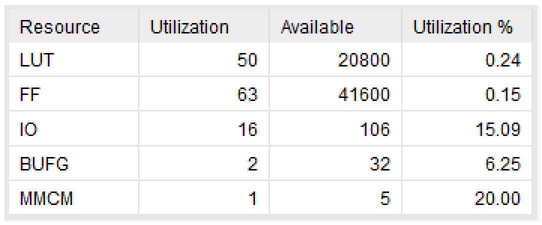
5. **Seven Segment Display**

   This module was taken from the previous lab. To display all four digits on the board, the 10 MHz clock was used to cycle through the digits. A clock divider was used to create a slower clock in order to properly display digits using the Basys3's LEDs.

   Two displays are used to show the last two digits of my WPI ID, or nine and six. The other two displays show the reading of the light sensor. To accomplish this, a 16-bit bus was created that combined the values of nine and six in binary (4'b1001 and 4'b0110) with the 8-bit light sensor data.

## 3 Testing and Synthesis

A total of 63 flip flops were used in this design.



| Resource | Utilization | Available | Utilization % |
|---|---|---|---|
| LUT | 50 | 20800 | 0.24 |
| FF | 63 | 41600 | 0.15 |
| IO | 16 | 106 | 15.09 |
| BUFG | 2 | 32 | 6.25 |
| MMCM | 1 | 5 | 20.00 |

Figure 3: Overall resource usage of the program reported during synthesis

The light sensor module utilizes 47 flip flops: 4 for the 1 MHz counter, 19 for the 2 Hz counter, 15 for the shift register, 2 for both the FSM's current and next state registers (4 in total), and 5 for the 16 SCLK cycles counter.

The seven segment module uses 16 flip flops: 4 for the digit displayed on a seven segment, 2 for the selection switch, and 10 for the 10k Hz counter. This counter is used to down clock the 10 MHz clock, as the LEDs are unable to properly read and display the seven segment data at such a fast rate.

In addition, the following warnings were provided upon Synthesis:

```
[Vivado 12-584] No ports matched ''.

[Common 17-576] 'use_project_ipc' is deprecated. This option is deprecated and no longer used.

[Constraints 18-5210] No constraints selected for write
```

The first two are both warnings with regards to IP. To solve this, all IPs can be checked and re-updated. The last warning has to do with the target constraint file for the clocking wizard. Given that the MMCM module is provided by Xilinx, I chose to ignore this warning. None of the warnings impacted the desired outcome of the program.

A test bench was written to verify all signals and to debug any program errors. The test bench, named `lab3_tf`, can be found in **Appendix B**. The inputs and outputs of the light sensor were implemented as registers and wires respectively, and the module itself was instantiated as the UUT. Afterwards, a clock was configured to change every 50ns. To test the logic of the module, a reset was given by setting `reset = 1`, and changing its value to 0 after 10ns. SDO is set to high, meaning that the desired outcome of the simulation is to have FF, or eight 1's, represent the light sensor data.

To speed up the simulation, the 500ms counter had its `MAX_COUNT` changed to 999 from 499,999. This allowed for the data bits to be read every 1ms instead of every 500ms.

As I was unable to verify with an oscilloscope, the SCLK and CS signals were first checked.

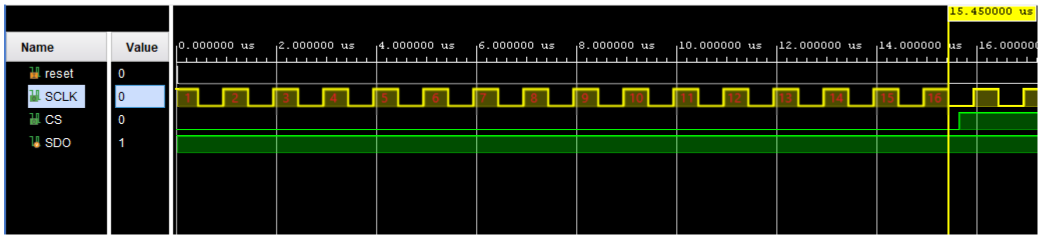


Figure 4: SCLK and CS signals. For 16 cycles of SCLK, CS is low. Note that there is a slight delay of 100ns between the rising edge of CS and the falling edge of the 16th SCLK cycle.

The simulation shows that CS is low for 16 cycles of SCLK. On the next rising edge of SCLK, CS is high. This is as desired, meaning that the FSM is producing a correct CS signal.

The shift register was checked to verify if the bits of data are being loaded on the falling edge of SCLK. It should then be valid on the next rising edge.

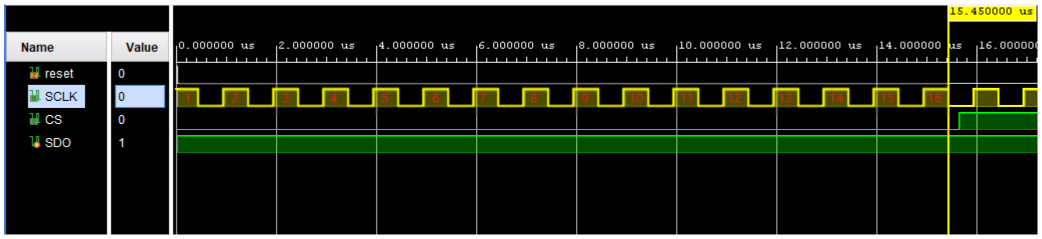


Figure 5: Data bits are loaded into the shift register on the falling edge of SCLK. From the test bench, a slight delay of 100ns can be observed. This is expected as the program is simulated in real time. However, the data access time ($t_{acc}$)$specifiedbythedatasheetgivesamaximumof40ns.BythenextrisingedgeofSCLK,however,thebitisfullyloaded.$

The signals for the FSM was included in the waveform diagram.

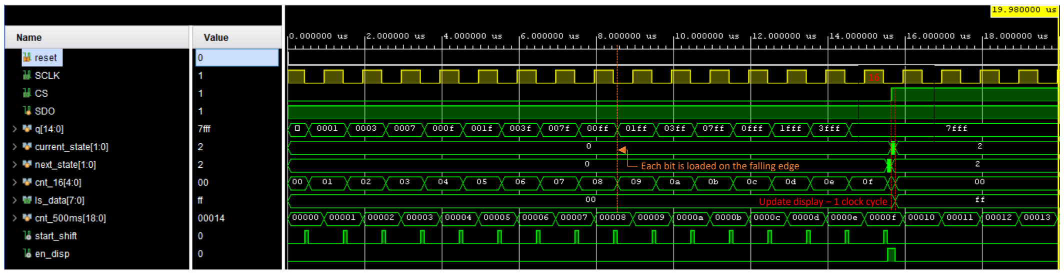


Figure 6: Waveform diagram including the state rotation and the signals generated by the counters and as a result of the FSM.

Since the FSM acted as the control for CS, it was expected to see a switch in state result in a high CS signal. The S1, or display state, took one 10 MHz clock cycle. Here, an enable signal, `en_disp` is delivered to update `ls_data` accordingly. As a result, the value of `ls_data` changes by the beginning of S2. It remains the same until 500ms (or 1ms on the test bench) has passed.

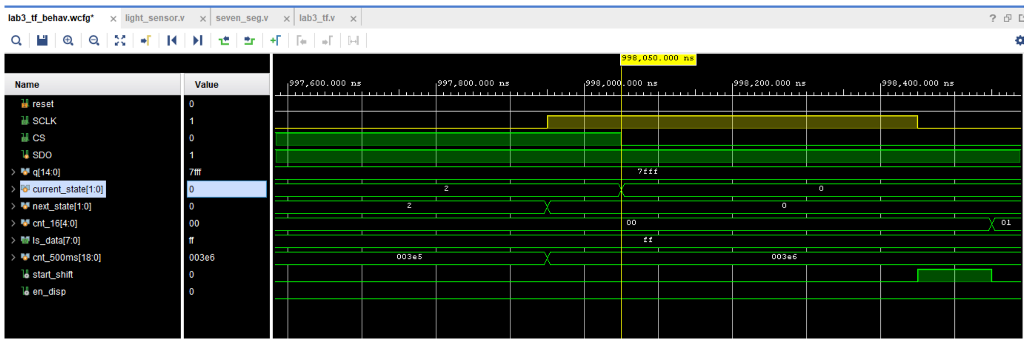


Figure 7: The next change between S2 and S0 occurs at 998,050 ns. The difference in time, 998050 - 15750 = 982300, or 0.982300 milliseconds, shows a slight delay. Although the counter that waits for 500ms is set to count to 999 instead of 499,999, this indicates that there are real time constraints on signal changes.

## 4 Summary and Conclusion

This lab accomplished several things, namely designing and connecting multiple sequential circuits. The greatest difficulty faced in the design process was in implementing the Finite State Machine, and ensuring all the signals lined up. The test bench was extremely helpful in troubleshooting errors such as not changing state or not having the seven segment display update accordingly. Additionally, it took some time to understand and configure the State Machine as a control for signals, namely chip select. Despite its many challenges, the lab taught me how to debug effectively with a test bench, and to lay out my design thoroughly before implementing it. As a result, I was able to implement a simple State Machine with only three states whose operation fulfilled all the system requirements.

For the future, I would like to see how my signals look on the oscilloscope. I would also like to see if I can be more precise with the timing in my program, such as between SCLK and CS, to match the ones listed in the PmodALs data sheet.

# 5 Appendices

## 5.1 Appendix A

Top Module:

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer: Prudence Lam
//
// Create Date: 09/29/2020 03:58:33 PM
// Design Name:
// Module Name: lab3_top
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Top module for lab 3. Includes the SPI interface for the Ambient Light Sensor, and outputs the received
// data, along with the last two digits of my WPI ID, to the four seven segment displays.
// The seven segment module is derived from previous labs.
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lab3_top(
input clk_fpga,             //100 MHz clock
input reset,                //Global reset
input SDO,                  //Master-In-Slave-Out
output SCLK,                //Series clock
output CS,                  //Chip select
output [6:0] disp,       //Output to seven-segment
output [3:0] an          //Output to anodes
);

wire [7:0] ls_data;       //8 bits for light sensor data
wire [15:0] seg_data;     //16 bit data bus for seven segment displays

    //Last two digits of WPI ID
parameter nine = 4’b1001;
parameter six = 4’b0110;

clk_wiz_0 mmcm              //Create a 10MHz signal from 100MHz
    (
     // Clock out ports
     .clk_out1(clk_10M),  // output clk_out1
     // Status and control signals
     .reset(reset), // input reset
     .locked(locked),     // output locked
    // Clock in ports
     .clk_in1(clk_fpga)); // input clk_in1

light_sensor ls1
     (
      .clk_10M(clk_10M),
      .SDO(SDO),
      .reset(reset),
      .SCLK(SCLK),
      .CS(CS),
      .ls_data(ls_data));

    //Create a 16-bit bus with all the data to display
assign seg_data = {nine, six, ls_data};

seven_seg ss1
     (
     .in(seg_data),
     .clk(clk_10M),
     .reset(reset),
     .seg(disp),
     .an(an));

endmodule
```

## Seven Segment Display:

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer: Prudence Lam
//
// Create Date: 09/02/2020 05:36:03 PM
// Design Name:
// Module Name: seven_seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Module to display 0000 to FFFF on four seven-segment displays.
// A clock is used to cycle through the digits for all four can seemingly appear
// An asynchronous reset signal depends on a pushbutton
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module seven_seg(
input[15:0] in,             // 16-bit input
input clk,                  // 25 MHz Clock
input reset,
output reg[6:0] seg,        // Output to seven-segment
output reg[3:0] an          // Output to anodes
);

parameter MAX_COUNT = 1000 - 1; // 10MHz to 10kHz
reg[3:0] sw;                   // 4 bit digit displayed
reg[1:0] SEL;               // Select
reg[9:0] cnt;                  // 10 bit register to store counter

wire[3:0] A, B, C, D;       // Group the input into 4 bits for each digit
assign A = in[3:0];
assign B = in[7:4];
assign C = in[11:8];
assign D = in[15:12];

always @(posedge clk, posedge reset)
     begin
         if (reset) begin                // Asynchronous reset for counter
             cnt <= 0;
             SEL <= 0;
             end
         else if (cnt == MAX_COUNT) begin // Signal for when MAX_COUNT is reached
             cnt <= 0;                   // Reset the counter
             SEL <= SEL + 1;             // Increment SEL to assign state of switches to display
             end
         else                            // If MAX_COUNT has not been reached
             cnt <= cnt + 1;             // Increment the counter
     end

always @(SEL,A,B,C,D)
     begin
         case(SEL)
             2’b00: begin
             sw = A;    // Display value of switch A
             an = 4’b1110; // Since the anodes are at low logic level, shows value on first display
             end
             2’b01: begin
             sw = B;    // Display the value of switch B
             an = 4’b1101; //Use the second display
             end
             2’b10: begin
             sw = C;    //Display value of switch C
             an = 4’b1011; //Use third display
             end
             2’b11: begin
```

```
            sw = D;    //Display value of switch D
            an = 4'b0111; //Use fourth display
            end
        endcase
    end

//Define constants for seven segment display
parameter zero = 7'b1000000;   //Display shows zero
parameter one = 7'b1111001; //Display shows one
parameter two = 7'b0100100; //Display shows two
parameter three = 7'b0110000;  //Display shows three
parameter four = 7'b0011001;   //Display shows four
parameter five = 7'b0010010;   //Display shows five
parameter six = 7'b0000010; //Display shows six
parameter seven = 7'b1111000;  //Display shows seven
parameter eight = 7'b0000000;  //Display shows eight
parameter nine = 7'b0010000;   //Display shows nine
parameter disp_A = 7'b0001000; //Display shows A
parameter disp_B = 7'b0000011; //Display shows B
parameter disp_C = 7'b1000110; //Display shows C
parameter disp_D = 7'b0100001; //Display shows D
parameter disp_E = 7'b0000110; //Display shows E
parameter disp_F = 7'b0001110; //Display shows F

always @(sw)
    begin
        case(sw) //Depending on slider switches, display corresponding number
            4'b0000: seg = zero;
            4'b0001: seg = one;
            4'b0010: seg = two;
            4'b0011: seg = three;
            4'b0100: seg = four;
            4'b0101: seg = five;
            4'b0110: seg = six;
            4'b0111: seg = seven;
            4'b1000: seg = eight;
            4'b1001: seg = nine;
            4'b1010: seg = disp_A;
            4'b1011: seg = disp_B;
            4'b1100: seg = disp_C;
            4'b1101: seg = disp_D;
            4'b1110: seg = disp_E;
            4'b1111: seg = disp_F;
            default: seg = zero;
        endcase
    end

endmodule
```

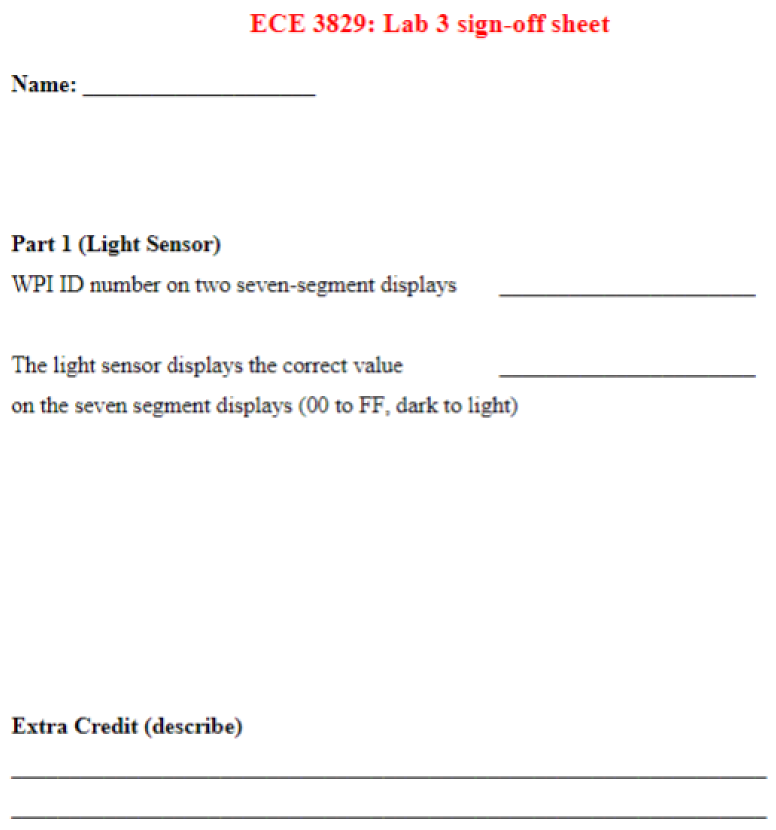
## Light Sensor SPI Interface:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer: Prudence Lam
//
// Create Date: 10/03/2020 12:36:34 PM
// Design Name:
// Module Name: light_sensor
// Project Name:
// Target Devices:
// Tool Versions:
// Description: SPI interface for the Ambient Light Sensor. A reading
// is taken every 500 ms. Multiple counters were used to keep track of the signals.
// A Finite State Machine acted as a control for CS and the shift register.
// At the end of 16 SCLK cycles, the important 8 bits are extracted and outputted to display.
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module light_sensor(
    input clk_10M,                          //10MHz clock derived using mmcm
    input SDO,                              //Master-In-Slave-Out
    input reset,                            //Global reset
    output SCLK,                            //SCLK
    output CS,                              //Chip select
    output reg [7:0] ls_data                //8-bits for light sensor data
    );

    reg [3:0] cnt_1M;                       //Counter to divide 10MHz to 1MHz
    reg [18:0] cnt_500ms;                   //Counter to divide 1MHz to 500ms
    reg [14:0] q;                           //15-bit shift register for light sensor data
    reg [1:0] current_state, next_state;    //Finite state machine states
    reg[4:0] cnt_16;                        //Counter to count 16 SCLK cycles

    wire en_disp;                           //Signal to output light sensor data to display
    wire start_shift;                       //Enable signal for shift register

    parameter MAX_COUNT_1M = 10 - 1;
    parameter MAX_COUNT_500ms = 999 - 1;
    parameter MAX_COUNT_16 = 16;
    parameter [1:0] s0 = 2'b00,             //Read light sensor data
                    s1 = 2'b01,             //Write to display
                    s2 = 2'b10;             //Wait 500ms


    //SCLK is at 1MHz
    always @(posedge clk_10M, posedge reset)
        if (reset)
            cnt_1M <= 0;
        else
            if (cnt_1M == MAX_COUNT_1M)
                cnt_1M <= 0;
            else
                cnt_1M <= cnt_1M + 1;

    assign SCLK = (cnt_1M <= 4);
    wire SCLK_en = cnt_1M ==0;

    //Counter to wait 500ms before reading from the sensor
    always @(posedge clk_10M, posedge reset)
        if (reset)
            cnt_500ms <= 0;
        else if (cnt_1M == MAX_COUNT_1M)
            if (cnt_500ms == MAX_COUNT_500ms)
                cnt_500ms <= 0;
            else
                cnt_500ms <= cnt_500ms + 1;
    //Finite state machine for light sensor reading
    always @(posedge clk_10M, posedge reset)
        if (reset)
            current_state <= s0;
        else
            current_state <= next_state;

    always @(current_state, cnt_16, cnt_500ms)
        case (current_state)
            s0: if (en_disp)                        //Wait for control
                    next_state = s1;
                else
                    next_state = s0;
            s1: next_state=s2;
            s2: if(cnt_500ms == MAX_COUNT_500ms)    //Wait 500ms
                    next_state=s0;
                else
                    next_state=s2;

        endcase

    //Begin shifting into the shift register
    assign start_shift = (current_state == s0 && cnt_1M == 5);

    //Generate CS signal
    assign CS = (current_state != s0);

    //Counter to count 16 cycles of SCLK
    always @(posedge clk_10M, posedge reset)
        if (reset) begin
            cnt_16 <= 0;
            q <= 15'b0;
            end
        else
            if(current_state!=s0)
                cnt_16 <=0;
            else if (start_shift) begin
                q <= {q[13:0], SDO};            //Load data from sensor
                cnt_16 <= cnt_16 + 1'b1;
            end

    assign en_disp = (cnt_16 == MAX_COUNT_16);

    always @(posedge clk_10M, posedge reset)
        if (reset)
            ls_data <= 8'b0;
        else if (current_state==s1)
            ls_data <= q[11:4];                 //Extract the important eight bits

endmodule
```

## 5.2 Appendix B

Test Fixture:

```
    ‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer: Prudence Lm
//
// Create Date: 09/30/2020 01:10:07 PM
// Design Name:
// Module Name: lab3_tf
// Project Name:
// Target Devices:
// Tool Versions:
// Description: Text fixture for lab 3
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lab3_tf(
);
   //Clock
reg clk;

//Inputs
reg SDO;
reg reset;

//Outputs
wire SCLK;
wire CS;
wire [7:0] data;

    reg i;

//Instantiate the UUT
light_sensor ls_test (
     .clk_10M(clk),
     .SDO(SDO),
     .reset(reset),
     .SCLK(SCLK),
     .CS(CS),
     .ls_data(data));

always
     begin
     clk = 0;
     #50;
     clk = 1;
     #50;
     end

initial begin
   //Initialize inputs
     SDO = 1;
     reset = 1;
     #10;

     //Print header statement in console
     $display("Testing Light Sensor Logic");

     //Begin testing
     reset = 0;

    end
endmodule
```



**ECE 3829: Lab 3 sign-off sheet**

**Name:** ____________________

**Part 1 (Light Sensor)**

|  |  |
|---|---|
| WPI ID number on two seven-segment displays | ______________________ |
| The light sensor displays the correct value on the seven segment displays (00 to FF, dark to light) | ______________________ |

**Extra Credit (describe)**

_________________________________________________________________

_________________________________________________________________