CS/EE 120A
(Section: 021)

Lab #4: Sequential Logic Design

Dan Murphy; SID: <u>862xxxxxx</u>
dmurp006@ucr.edu

Partner:
Jesse Garcia

Subject Overview

This lab consists of three parts which modifies a flight attendant call system through a clock-based system.

Part 1:

By creating a UCF file to implement the FPGA board to output signals, we were able to create a button that stays on after being pressed and turns off when the clear button is pressed. As an edge case, the LED will turn on if both buttons are pressed simultaneously (i.e. call and clear). An internal board clock was used to allow the LED to turn on when the call button is pressed. Through the provided template in Verilog, we were able to test for correct, simulated output.

Part 2:

Building off of part 1, we implemented a rising edge for the clock. In essence, the FPGA board should only take inputs during the rising edge of the clock and during all other times, input should not be read. An LED flickered every time we generated an input during the rising edge of the clock, showing the input changing from 0 to 1.

Part 3:

Using a seven-segment LED display from the previous lab, we were able to make an LED time multiplexing circuit. This particular system only enables one display at a time where each light segment was used with their own enable(s) respectively.

New Concepts

1. Describe how this lab built upon previous ones:

This lab was highly dependent on a successful previous lab. We essentially used the previous lab as a building block to create this lab. Fortunately, I was tinkering with the outputs of the previous lab and implemented HEX outputs onto the FPGA board which directly led into the requirements of this lab. Most of the extra implementation came from implementing a clock in addition to the previous lab's functionality.

2. Describe the most difficult part of this lab for you:

By far, the most difficult part was figuring out where errors were coming from. I am not accustomed to copy-pasting code from the lab manuals as they are mostly outdated. However, after asking the TA about the errors in my code, they were simple errors on our end due to not copying the templates provided within the lab manual. For example, I typed "wire" instead of "reg", which is a huge difference in circuit design.

3. Describe problems you faced and how you solved them:

After realizing that we had a modified lab manual which was updated for our particular board, we were able to use the templates provided (which got rid of the typo's I had made whilst completing the lab).

4. Do you verify that the code included with this report is yours and your partner's original work (yes/no)?

Yes. For the purpose of verifying it is our original code, yes; with the caveat of group collaboration outside of myself and Jesse Garcia. Most our code is original with the exception of partially shared code for the last portion of the lab. We had help from Micah Galos during this lab period.

Questions

1. What will happen if the "clock" signal is of very low frequency (1Hz)?

   The lower the clock's frequency is, the longer we would have to wait in between pushing button.

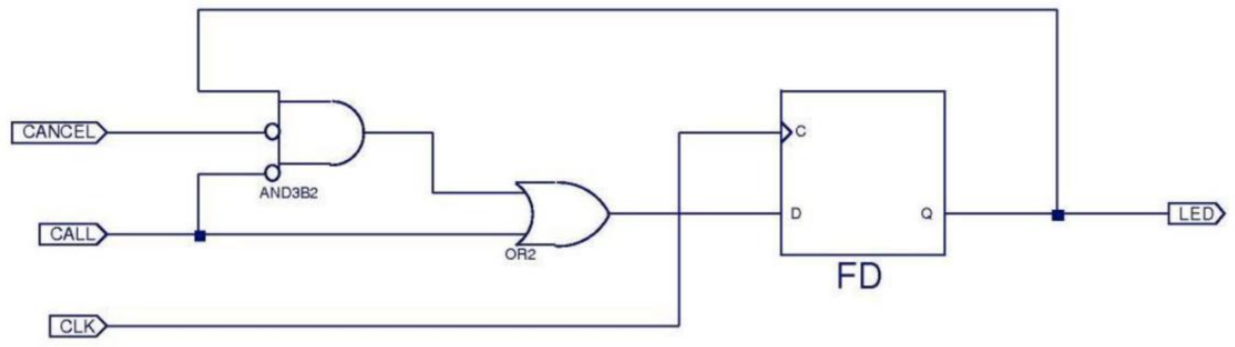
2. Design a testbench and verify the logic performance.

```
module facs;

     // Inputs -------------------------------------
     reg clock;         /* clock register         */
     reg reset;         /* reset register         */
     reg call_button;   /* call button register   */
     reg cancel_button; /* cancel button register */

     // Outputs ------------------------------------
     wire light_state; /* state of wire */

          // Instantiate the Unit Under Test (UUT)
          facs_bh uut (
               .clock(clock),
               .reset(reset),
               .call_button(call_button),
               .cancel_button(cancel_button),
               .light_state(light_state)
          ); /* End of UUT --------------------- */

          initial begin

```

```
    // -----------------------------------------
    // Initializing Inputs ---------------------
    // -----------------------------------------

    // =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

              clock = 0;
              reset = 0;
              call_button = 0;
              cancel_button = 0;
              #100; /* Wait 100 ns */

    // =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

              clock = 0;
              reset = 1;
              call_button = 0;
              cancel_button = 0;
              #100; /* Wait 100 ns */

    // =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

              clock = 0;
              reset = 0;
              call_button = 0;
              cancel_button = 0;
              #100; /* Wait 100 ns */

    // =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

              clock = 0;
              reset = 0;
              call_button = 1;
              cancel_button = 0;
              #100; /* Wait 100 ns */

    // =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

              clock = 0;
              reset = 0;
              call_button = 0;
```

```
          cancel_button = 0;
          #100; /* Wait 100 ns */

// =====================================

          clock = 0;
          reset = 0;
          call_button = 0;
          cancel_button = 1;
          #100; /* Wait 100 ns */

// =====================================

          clock = 0;
          reset = 0;
          call_button = 0;
          cancel_button = 0;
          #100; /* Wait 100 ns */

// =====================================

     end /* End of local scope */

always #5 clock =~clock;

endmodule /* END OF MODULE <<<<<<<<<<<<<<<< */
```

Schematic – Flight Attendant System:



UCF – Part 1:

```
NET clock LOC         = B8;
NET call_button LOC   = G12;
NET cancel_button LOC = C11;
NET light_state LOC   = M5;
```

UCF – Part 2:

```
NET clock LOC   = B8;
NET signal LOC  = P11;

NET outedge LOC = M5;
```

UCF – Part 3:

```
NET clock LOC = B8;
NET sw0 LOC   = P11;
NET sw1 LOC   = L3;
NET sw2 LOC   = K3;
NET sw3 LOC   = B4;

NET an[0] LOC = K14;
NET an[1] LOC = M13;
NET an[2] LOC = J12;
NET an[3] LOC = F12;

NET sseg[0] LOC = L14;
NET sseg[1] LOC = H12;
NET sseg[2] LOC = N14;
NET sseg[3] LOC = N11;
NET sseg[4] LOC = P12;
NET sseg[5] LOC = L13;
NET sseg[6] LOC = M12;
```

Code – Part 1:

```
module facs(
// Inputs ----------------
input wire clock,
input wire call_button,
input wire cancel_button,

// Outputs ---------------
output reg light_state );

```

```
reg c_state;

always @(*) begin

// Cases ---------------------------------------
case ({call_button, cancel_button, light_state})
3'b000 : c_state = 1'b0;
3'b001 : c_state = 1'b1;
3'b010 : c_state = 1'b0;
3'b011 : c_state = 1'b0;
3'b100 : c_state = 1'b1;
3'b101 : c_state = 1'b1;
3'b110 : c_state = 1'b1;
3'b111 : c_state = 1'b1;

default: c_state = 1'b0;
endcase /* End of case --- */

end /* End of local scope */

always @( posedge clock ) begin
light_state <= c_state;

end /* End of local scope */

endmodule /* End of module ------------------ */
```

<u>Code – Part 2:</u>

```
module edgedetector(
    input wire clock,
    input wire signal,
    output reg outedge
    );

wire slow_clock;

reg [1:0] c_state ;
reg [1:0] r_state ;

localparam ZERO = 'd0 ;
localparam CHANGE = 'd1 ;
localparam ONE = 'd2 ;

clockdiv c1(clock, slow_clock) ;
```

```
always @(*) begin
     case (r_state)

          ZERO : begin
               c_state = signal ? CHANGE : ZERO ;
               outedge = 'd0;
               end

          CHANGE : begin
               c_state = signal ? ONE : ZERO ;
               outedge = 'd1 ;
               end

          ONE :  begin
               c_state = signal ? ONE : ZERO ;
               outedge = 'd0 ;
               end

     default : begin

          c_state = ZERO ;
          outedge = 'd0 ;
          end

     endcase /* End of case --- */

end  /* End of local scope */

always @( posedge slow_clock ) begin
     r_state <= c_state ;

end /* End of local scope */

endmodule /* End of module */





/* ************* clockdiv ************** */

module clockdiv(clock, clock_out);
    input clock;
    output clock_out;

      reg [15:0] COUNT;
      assign clock_out = COUNT[15];

      always @(posedge clock) begin
          COUNT = COUNT + 1;
      end /* End of local scope */

endmodule /* End of module ---------------- */
```

Code – Part 3:

```
module dispux_main(
    input clock,
    input sw0,
    input sw1,
    input sw2,
    input sw3,
    output [3:0] an,
    output [7:0] sseg
    );

wire [7:0] in0;
wire [7:0] in1;
wire [7:0] in2;
wire [7:0] in3;

wire [3:0] sw;
wire slow_clock;

assign sw[0] = sw0;
assign sw[1] = sw1;
assign sw[2] = sw2;
assign sw[3] = sw3;

bcdto7led_bh c1(sw, in0);
bcdto7led_bh c2(sw, in1);
bcdto7led_bh c3(sw, in2);
bcdto7led_bh c4(sw, in3);


clockdiv clockk(clock, slow_clock );

disp_mux c5(
    .clock(clock) ,
      .in0 (in0) ,
      .in1 (in1) ,
      .in2 (in2) ,
      .in3 (in3) ,
      .an (an) ,
      .sseg(sseg));
endmodule /* End of module */




/* ***** bcdto7led ***** */

module bcdto7led(
     input [3:0] x,
     output wire [6:0] r
    );

      reg [6:0] z; /* Registers, not wires… */
      reg [6:0] zd; /* Registers, not wires… */

      always @(*) begin

          z[0] <= zd[6];
          z[1] <= zd[5];
          z[2] <= zd[4];
          z[3] <= zd[3];
          z[4] <= zd[2];
          z[5] <= zd[1];
          z[6] <= zd[0];

          case (x)
          4'b0000 : zd = 7'b1111110 ;
          4'b0001 : zd = 7'b0110000 ;
          4'b0010 : zd = 7'b1101101 ;
          4'b0011 : zd = 7'b1111001 ;
          4'b0100 : zd = 7'b0110011 ;
          4'b0101 : zd = 7'b1011011 ;
          4'b0110 : zd = 7'b1011111 ;
          4'b0111 : zd = 7'b1110000 ;
          4'b1000 : zd = 7'b1111111 ;
          4'b1001 : zd = 7'b1111011 ;
          4'b1010 : zd = 7'b1110111 ;
          4'b1011 : zd = 7'b0011111 ;
          4'b1100 : zd = 7'b1001110 ;
          4'b1101 : zd = 7'b0111101 ;
          4'b1110 : zd = 7'b1001111 ;
          4'b1111 : zd = 7'b1000111 ;
          endcase /* End of cases */
     end /* End of local scope */
     assign r = ~z ;

endmodule /* End of module ------------------ */




/* ***** clockdiv ***** */

module clockdiv(clock,clock_out);

  input clock;
  output clock_out;

  reg [15:0] COUNT;
  assign clock_out = COUNT[15];

  always @(posedge clock)   begin
  COUNT = COUNT + 1;

end /* End of local scope */

endmodule /* End of module ---------------- */




/* ***** dis_mux ***** */

module disp_mux(
     input clock,
     input wire [7:0] in0 ,
     input wire [7:0] in1 ,
     input wire [7:0] in2 ,
     input wire [7:0] in3 ,

     output reg [3:0] an ,
     output reg [7:0] sseg
     );

     reg [16:0] r_qreg ;
     reg [16:0] c_next ;

/* First case ----- */
     always@(*) begin
          case(r_qreg[1:0])
          2'b00 : sseg = in0 ;
          2'b01 : sseg = in1 ;
          2'b10 : sseg = in2 ;
          2'b11 : sseg = in3 ;
          endcase /* End of first case */
     end /* End of local scope */

/* Second case ----- */
     always @(*) begin
          case (r_qreg[1:0])
          2'b00 : an = ~(4'b0001) ;
          2'b01 : an = ~(4'b0010) ;
          2'b10 : an = ~(4'b0100) ;
          2'b11 : an = ~(4'b1000) ;
          endcase /* End of second case */
     end /* End of local scope */

     always @ (*) begin
     c_next = r_qreg +'d1;
     end /* End of local scope */

     always @( posedge clk) begin
     r_qreg <= c_next ;
     end /* End of local scope */

endmodule /* End of module ------------------------ */
```

Conclusion

The lab served its purpose of allowing us to become more familiarized with Verilog and implementing code which directly affects hardware. In particular, we learned about sequential logic design within our FPGA board through Verilog code. This was shown through the seven-segment LED's on our FPGA board via testbenches.

As a whole, we created a flight attendant call system, a rising edge detector and an LED display time multiplexing circuit. The circuit design and Boolean logic allowed us to see the implementation of the code within Verilog. Once the results expected were yielded, our LED output confirmed successful software to hardware communication.