Lab 6 - Final Project

Report By - Apurv Goel

(Total Hours Spent: 26 hours)

Procedure:

TASK: Design Problem – Frogger

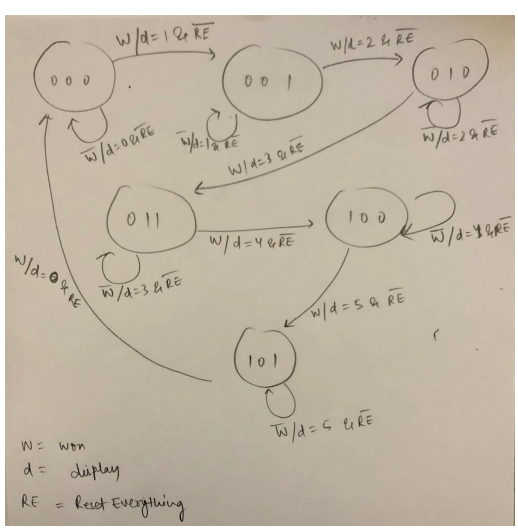
1. First, I started with seeing videos about frogger, to understand what the game was and how it was played.

2. When I was acquainted with how a frogger works, I tried breaking the frogger into parts. I successfully broke the problem into 3 parts. First, making a frog. Second, making obstacles in the way of frog. Third, to keep count of the wins and after a particular number of wins, reset the game.

3. After breaking the problem down, the most challenging part was connecting the LED Matrix onto the breadboard. After connecting the LED matrix, I started to understand how did the LED matrix worked so as to design my frog and obstacles module.

4. Then, I made the below frog and obstacle modules according to the behavior I desired. Also, I included a testbench to check if I was on a right path or not.

5. Now, I made the following state-diagram for the count module, similar to the ones we made earlier. Here, I decided to use a simple FSM, rather that using counters as number of wins were limited and it allowed me to display the output directly to HEX display.



6. I used the userInput and meta modules to handle all the user inputs effectively, like we did in Tug-of-War problem in Lab 4.

7. After making all these modules, I also thought about how the frog would lose. I realized if the frog and obstacle were in the same row then they would collide and player would lose. Through this knowledge, I made a lose module which kept a check on frog and obstacle colliding and resets the game at that point.

8. After, I had a working frogger, I also decided to make levels for increasing the user difficulty. I made a module named levels which upgraded the level as per user's wish. I used the bit overflow concept taught in class to implement this.

9. After I had all the modules working, I then connected all the modules with DE1-SoC to initialize the hardware according to my logic made. I also made a testbench here to check if this was working properly or not.

10. On next pages, is the code for all the modules I made and testbench for some of them that I thought were worth doing.

Code for handling UserInput

```
// Apurv Goel
// EE 271
// NetId :- ag2304
// Date:- 05/10/2019

// Manages the user input
module userInput(clk, w, out, reset);

   input logic clk, w, reset;
   output logic out;

   // State variables
   enum logic {A = 1'b0, B = 1'b1} ps, ns;

   // Next state logic and output logic
   always_comb begin
      case(ps)
         A: if (w) begin
                          ns = B;
                         out = 1;
            end
            else begin    ns = A;
                         out = 0;
            end
         B: if (w) begin
                          ns = B;
                         out = 0;
            end
            else begin    ns = A;
                         out = 0;
            end
      endcase
   end

   // DFF's
   always_ff @(posedge clk) begin
      if(reset) begin
         ps <= A;
      end
      else begin
         ps <= ns;
      end
   end
endmodule
```

TestBench for UserInput

```
// Tests the functionality of userInput
module userInput_testbench();
   logic  clk, reset, w;
   logic  out;

   userInput dut (.clk, .w, .out, .reset);

   // Set up the clock.
   parameter CLOCK_PERIOD=100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   // Set up the inputs to the design.  Each line is a clock cycle.
   initial begin
                                          @(posedge clk);
               reset <= 1;                @(posedge clk);
               reset <= 0; w <= 0;        @(posedge clk);
                                          @(posedge clk);
                                          @(posedge clk);
                                          @(posedge clk);
                           w <= 1;        @(posedge clk);
                                          @(posedge clk);
                                          @(posedge clk);
                           w <= 0;        @(posedge clk);
                                          @(posedge clk);
                                          @(posedge clk);
                           w <= 1;        @(posedge clk);
                                          @(posedge clk);
                                          @(posedge clk);
            $stop; // End the simulation.
      end
endmodule
```

Code for Handling Metastability

```
// Apurv Goel
// EE 271
// NetId:- ag2304
// 05/10/2019

// Handles metstability of input
module meta (clk, reset, in, out);

   input logic in, clk, reset;
   output logic out;

   logic temp;

   // DFF's
   always_ff @(posedge clk) begin
      if (reset)
         out <= 0;
      else begin
         temp <= in;
         out <= temp;
      end
   end
endmodule
```

Code for Count

```
module count (clk, reset, up, finalRow, display, resetEverything);
    input logic clk, reset, up;
    input logic [7:0] finalRow;
    output logic [6:0] display;
    output logic resetEverything;
    logic won;

    enum logic [2:0] {zero = 3'b000, one = 3'b001, two = 3'b010,
                      three = 3'b011, four = 3'b100, five = 3'b101} ps, ns;

    assign won = up & (finalRow != 8'b00000000);

    always_comb begin
        case(ps)
            zero: if (won) begin
                    ns = one;
                    display = 7'b1111001;
                    resetEverything = 1'b0;
                  end
                  else begin
                    ns = ps;
                    display = 7'b1000000;
                    resetEverything = 1'b0;
                  end
            one: if (won) begin
                    ns = two;
                    display = 7'b0100100;
                    resetEverything = 1'b0;
                  end
                  else begin
                    ns = ps;
                    display = 7'b1111001;
                    resetEverything = 1'b0;
                  end
            two: if (won) begin
                    ns = three;
                    display = 7'b0110000;
                    resetEverything = 1'b0;
                  end
                  else begin
                    ns = ps;
                    display = 7'b0100100;
                    resetEverything = 1'b0;
                  end
            three: if (won) begin
                    ns = four;
                    display = 7'b0011001;
                    resetEverything = 1'b0;
                  end
                  else begin
                    ns = ps;
                    display = 7'b0110000;
                    resetEverything = 1'b0;
                  end
            four: if (won) begin
                    ns = five;
                    display = 7'b0010010;
                    resetEverything = 1'b0;
                  end
                  else begin
                    ns = ps;
                    display = 7'b0011001;
                    resetEverything = 1'b0;
                  end
            five: if (won) begin
                    ns = ps;
                    display = 7'b1111111;
                    resetEverything = 1'b1;
                  end
                  else begin
                    ns = ps;
                    display = 7'b0010010;
                    resetEverything = 1'b0;
                  end

        endcase
    end
    always_ff @(posedge clk) begin
        if (reset) begin
            ps <= zero;
        end
        else begin
            ps <= ns;
        end
    end

endmodule
```

Testbench for count

```
module count_testbench ();
   logic clk, reset, up, resetEverything;
   logic [7:0] finalRow;
   logic [6:0] display;

   count dut (.clk, .reset, .up, .finalRow, .display, .resetEverything);


   // Set up the clock
   parameter CLOCK_PERIOD = 100;
   initial clk = 1;
   always begin
      #(CLOCK_PERIOD / 2);
      clk = ~clk;
   end

   // Set up the inputs to the design. Each line is a clock cycle.
   initial begin
      reset <= 1; up <= 0; finalRow <= 8'b00000000;          @(posedge clk);
      reset <= 0;;                                           @(posedge clk);
                                                             @(posedge clk);
      up <= 1; finalRow <= 8'b10000000;                      @(posedge clk);
      up <= 0; finalRow <= 8'b01000000;                      @(posedge clk);
      up <= 1; finalRow <= 8'b00100000;                      @(posedge clk);
      up <= 1; finalRow <= 8'b00010000;                      @(posedge clk);
      up <= 1; finalRow <= 8'b00001000;                      @(posedge clk);
      up <= 0; finalRow <= 8'b00000100;                      @(posedge clk);
      up <= 1; finalRow <= 8'b00000010;                      @(posedge clk);
      up <= 1; finalRow <= 8'b00000010;                      @(posedge clk);
      up <= 1; finalRow <= 8'b00000010;                      @(posedge clk);
      reset <= 1;                                            @(posedge clk);
   $stop;
   end
endmodule
```

Code for lose

```
module lose (clk, obsMove, frogMove, resetGame);
    input logic clk;
    input logic [7:0][7:0] obsMove, frogMove;
    output logic resetGame;

    always_ff @(posedge clk) begin
        if (((obsMove[0] & frogMove[0]) != 8'b00000000) | ((obsMove[1] & frogMove[1]) != 8'b00000000) |
            ((obsMove[2] & frogMove[2]) != 8'b00000000) | ((obsMove[3] & frogMove[3]) != 8'b00000000) |
            ((obsMove[4] & frogMove[4]) != 8'b00000000) | ((obsMove[5] & frogMove[5]) != 8'b00000000) |
            ((obsMove[6] & frogMove[6]) != 8'b00000000) | ((obsMove[7] & frogMove[7]) != 8'b00000000))
        begin
            resetGame <= 1'b1;
        end
        else begin
            resetGame <= 1'b0;
        end
    end
endmodule
```

Testbench for Lose

```
module lose_testbench ();
   logic clk;
   logic [7:0][7:0] obsMove, frogMove;
   logic resetGame;

   lose dut (.clk, .obsMove, .frogMove, .resetGame);


   // Set up the clock
   parameter CLOCK_PERIOD = 100;
   initial clk = 1;
   always begin
      #(CLOCK_PERIOD / 2);
      clk = ~clk;
   end

   // Set up the inputs to the design. Each line is a clock cycle.
   initial begin
      obsMove <= {      {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000}
                       };
      frogMove <= {     {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000},
                        {8'b00000000}
                       };

      frogMove[0] <= 8'b01000000; obsMove[0] <= 8'b01000000;   @(posedge clk);  //collision
      frogMove[0] <= 8'b01000000; obsMove[0] <= 8'b00100000;   @(posedge clk);
      frogMove[1] <= 8'b00001000; obsMove[1] <= 8'b10001000;   @(posedge clk);  // collosion
      frogMove[1] <= 8'b01000000; obsMove[1] <= 8'b00000100;   @(posedge clk);
      frogMove[2] <= 8'b00100000; obsMove[2] <= 8'b00100000;   @(posedge clk);  //collision
      frogMove[2] <= 8'b01000000; obsMove[2] <= 8'b10000000;   @(posedge clk);
      frogMove[3] <= 8'b00010000; obsMove[3] <= 8'b00010000;   @(posedge clk);  // collision
      frogMove[3] <= 8'b01000000; obsMove[3] <= 8'b10000000;   @(posedge clk);
      frogMove[4] <= 8'b00010000; obsMove[4] <= 8'b00010000;   @(posedge clk);  //collision
      frogMove[4] <= 8'b01000000; obsMove[4] <= 8'b10000000;   @(posedge clk);
      frogMove[5] <= 8'b00000100; obsMove[5] <= 8'b00000100;   @(posedge clk);  // collision
      frogMove[5] <= 8'b01000000; obsMove[5] <= 8'b10000000;   @(posedge clk);
      frogMove[6] <= 8'b00000100; obsMove[6] <= 8'b00000100;   @(posedge clk);  // collision
      frogMove[6] <= 8'b01000000; obsMove[6] <= 8'b10000000;   @(posedge clk);
      frogMove[7] <= 8'b00000010; obsMove[7] <= 8'b00000010;   @(posedge clk);  // collision
      frogMove[7] <= 8'b01000000; obsMove[2] <= 8'b10000000;   @(posedge clk);
   $stop; // End the simulation
   end

endmodule
```

Code for Frog

```
module frog(clk, reset, resetGame, pause, left, right, up, down, frogPrev, frogNext);
    input logic clk, left, right, up, down, reset, resetGame, pause;
    input logic [7:0][7:0] frogPrev;
    output logic [7:0][7:0] frogNext;
    logic frogger;
    integer i;

    always_ff @(posedge clk) begin
        if (reset | resetGame | (up & (frogPrev[7] != 8'b00000000))) begin
            frogNext <= {{8'b00000000},
                         {8'b00000000},
                         {8'b00000000},
                         {8'b00000000},
                         {8'b00000000},
                         {8'b00000000},
                         {8'b00000000},
                         {8'b00001000}
                         };
        end
        else if (pause | (down & (frogPrev[0] != 8'b00000000))) begin
            frogNext <= frogPrev;
        end
        else if (down) begin
            frogNext[7] <= 8'b00000000;
            frogNext[6:0] <= frogPrev[7:1];
        end
        else if (up) begin
            frogNext[0] <= 8'b00000000;
            frogNext[7:1] <= frogPrev[6:0];
        end
        else if (left & (frogPrev[0][7] | frogPrev[1][7] | frogPrev[2][7] |
                         frogPrev[3][7] | frogPrev[4][7] | frogPrev[5][7] |
                         frogPrev[6][7] | frogPrev[7][7])) begin
            for (i = 0; i < 7; i++) begin
                frogNext[i] <= {frogPrev[i][6:0], frogPrev[i][7]};
            end
        end
        else if (left) begin
            for (i = 0; i < 7; i++) begin
                frogNext[i] <= {frogPrev[i][6:0], 1'b0};
            end
        end
        else if (right & (frogPrev[0][0] | frogPrev[1][0] | frogPrev[2][0] |
                          frogPrev[3][0] | frogPrev[4][0] | frogPrev[5][0] |
                          frogPrev[6][0] | frogPrev[7][0])) begin
            for (i = 0; i < 7; i++) begin
                frogNext[i] <= {frogPrev[i][0], frogPrev[i][7:1]};
            end
        end
        else if (right) begin
            for (i = 0; i < 7; i++) begin
                frogNext[i] <= {1'b0, frogPrev[i][7:1]};
            end
        end
        else begin
            frogNext <= frogPrev;
        end
    end
endmodule
```

Testbench for Frog

```
module frog_testbench ();
   logic clk, reset, resetGame, left, right, up, down, pause;
   logic [7:0][7:0] frogMove;

   frog dut (.clk, .reset, .resetGame, .pause, .left, .right, .up, .down , .frogPrev(frogMove), .frogNext(frogMove));

   // Set up the clock
   parameter CLOCK_PERIOD = 100;
   initial clk = 1;
   always begin
      #(CLOCK_PERIOD / 2);
      clk = ~clk;
   end

   initial begin

         reset <= 1; resetGame<= 0; left <= 0; right<=0; up <= 0; down <= 0;  pause<=0;   @(posedge clk);
                                                                                           @(posedge clk);
         reset <= 0;                                                                       @(posedge clk);
                                  left <= 1;                                               @(posedge clk);
                                  left <= 1;                                               @(posedge clk);
                                            right <= 1;                                    @(posedge clk);
                                            right <= 1;                                    @(posedge clk);
                                                                down <= 1;                 @(posedge clk); // Should remian same
                                                                down <= 0;                 @(posedge clk);
                                                      up <= 1;                             @(posedge clk);
                                                                                           @(posedge clk); // Should be two rows up at the mid position
                                                      up <= 0;                             @(posedge clk);
                                            right <= 1;                                    @(posedge clk);
                                                                                           @(posedge clk);
                                                                                           @(posedge clk);
                                                                                           @(posedge clk);
                                                                                           @(posedge clk);
                                                                                           @(posedge clk); // Should reach right and stay the same
                                            right <= 0;                                    @(posedge clk);
      $stop; // End the simulation
    end

endmodule
```

Code for Obstacle

```
module obstacle(clk, reset, resetGame, pause, obsPrev, obsNext, level);
    input logic clk, reset, resetGame, pause;
    input logic [7:0][7:0] obsPrev;
    output logic [7:0][7:0] obsNext;
    input logic [3:0] level;
    logic enable, level1, level2, level3, level4;

    integer i;

    always_comb begin
      if (level == 4'b0010) begin
         enable = level2;
      end else if (level == 4'b0100) begin
         enable = level3;
      end else if (level == 4'b1000) begin
         enable = level4;
      end else begin
         enable = level1;
      end
    end

   // Use this for simulation
   //assign enable = level1;

   enable1 e1(.clk(clk), .enable(level1));
   enable2 e2(.clk(clk), .enable(level2));
   enable3 e3(.clk(clk), .enable(level3));
   enable4 e4(.clk(clk), .enable(level4));

    always_ff @(posedge clk) begin
        if (reset | resetGame) begin
            obsNext <= {
                        {8'b00000000},
                        {8'b00110110},
                        {8'b01100110},
                        {8'b01100000},
                        {8'b11000011},
                        {8'b01100000},
                        {8'b01100011},
                        {8'b00000000}
                       };
        end
        else if (pause) begin
            obsNext <= obsPrev;
        end
        else if (enable) begin
            for (i = 2; i < 7; i=i+2) begin
                obsNext[i] <= {obsPrev[i][0], obsPrev[i][7:1]};
            end
            for (i = 1; i < 7; i=i+2) begin
                obsNext[i] <= {obsPrev[i][6:0], obsPrev[i][7]};
            end
        end
```

```
            else begin
                obsNext <= obsPrev;
            end
        end
    endmodule
```

Testbench for Obstacle

```
module obstacle_testbench ();
   logic clk, reset, resetGame, pause;
   logic [3:0] level; //level won't work in modelsim
   logic [7:0][7:0] obsMove;
   obstacle dut (.clk, .reset, .resetGame, .pause, .obsPrev(obsMove), .obsNext(obsMove), .level );

   // Set up the clock
   parameter CLOCK_PERIOD = 100;
   initial clk = 1;
   always begin
      #(CLOCK_PERIOD / 2);
      clk = ~clk;
   end

   initial begin
         obsMove <= { {8'b11111111},
                      {8'b11111111},
                      {8'b11111111},
                      {8'b11111111},
                      {8'b11111111},
                      {8'b11111111},
                      {8'b11111111},
                      {8'b11110111}
                    };
         reset <= 1; resetGame <= 0;   pause<=0;                          @(posedge clk);
                                                                          @(posedge clk);
         reset <= 0;                            level = 4'b1000;          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                     pause <=1;                           @(posedge clk);
                                                                          @(posedge clk); // Should stay same
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                  pause <=0;                              @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);
                                                                          @(posedge clk);

      $stop; // End the simulation
    end

endmodule
```

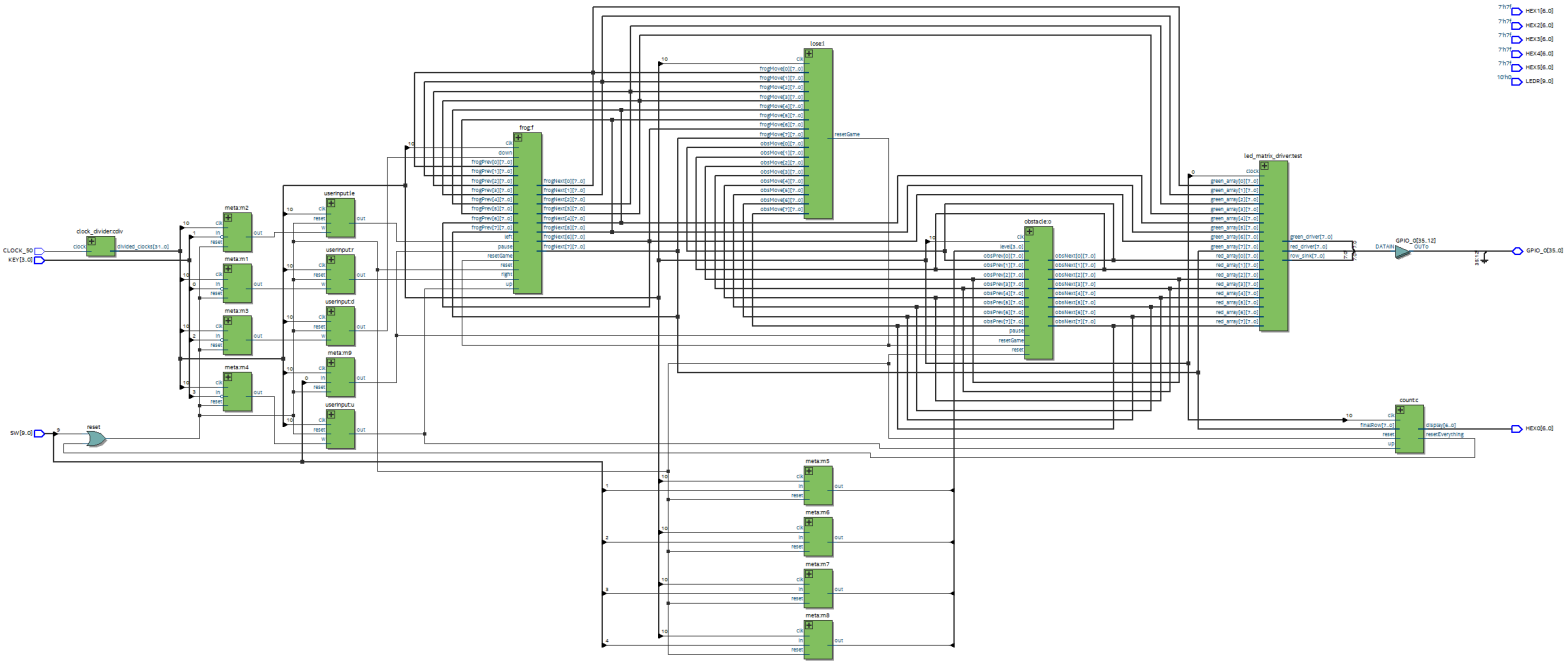
Code for Levels

```
module level_one (clk, enable);

   input logic clk;
   output logic enable;
   logic [16:0] num = 17'b00000000000000001;

   always @(posedge clk)
      if (num == 17'b00000000000000000) begin
         enable <= 1'b1;
         num <= 17'b00000000000000001;
      end
      else begin
         enable <= 1'b0;
         num <= num + 17'b00000000000000001;
      end

endmodule

module level_two (clk, enable);

   input logic clk;
   output logic enable;
   logic [15:0] num = 16'b0000000000000000;

   always @(posedge clk)
      if (num == 16'b0000000000000000) begin
         enable <= 1'b1;
         num <= 16'b0000000000000001;
      end
      else begin
         enable <= 1'b0;
         num <= num + 16'b0000000000000001;
      end

endmodule

module level_three (clk, enable);

   input logic clk;
   output logic enable;
   logic [14:0] num = 15'b000000000000000;

   always @(posedge clk)
      if (num == 15'b000000000000000) begin
         enable <= 1'b1;
         num <= 15'b00000000000001;
      end
      else begin
         enable <= 1'b0;
         num <= num + 15'b00000000000001;
      end

endmodule


module level_four (clk, enable);

   input logic clk;
   output logic enable;
   logic [11:0] num = 11'b000000000000;

   always @(posedge clk)
      if (num == 11'b000000000000) begin
         enable <= 1'b1;
         num <= 11'b000000000001;
      end
      else begin
         enable <= 1'b0;
         num <= num + 11'b000000000001;
      end

endmodule
```

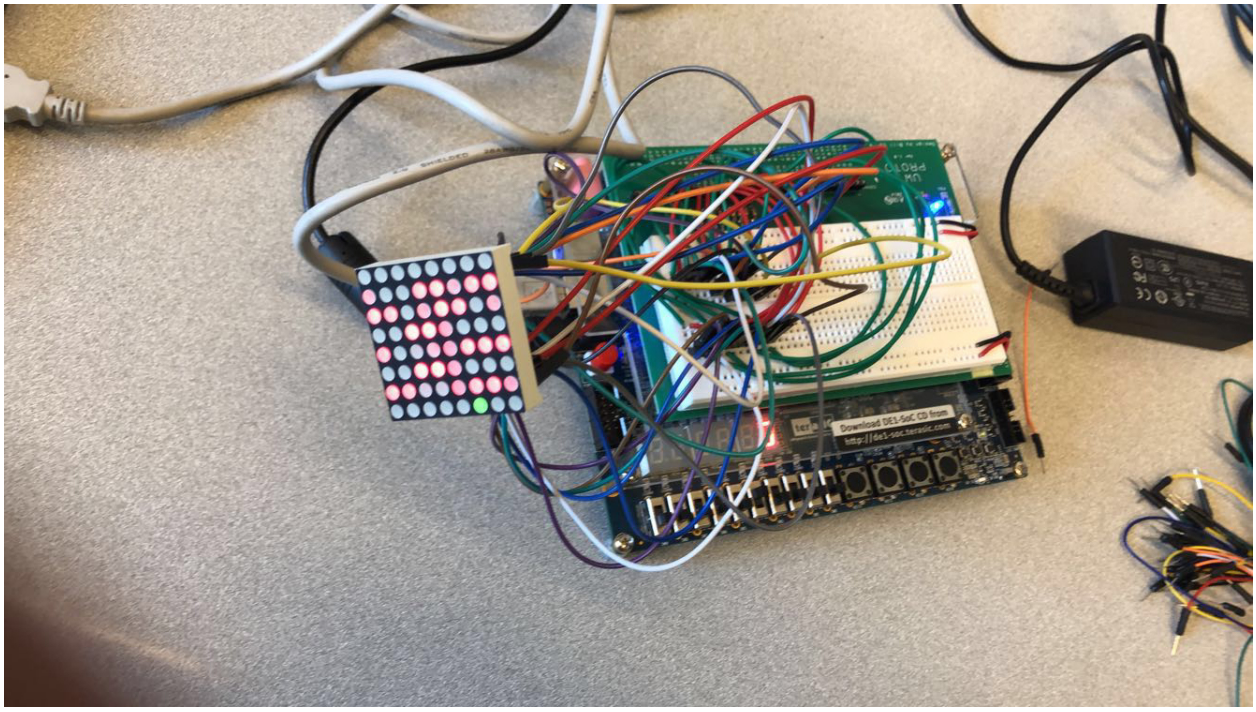
Code for DE1-SoC

```
module DE1_SoC (CLOCK_50, GPIO_0, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY,
                LEDR, SW);

   input logic CLOCK_50; // 50MHz clock.
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   input logic [3:0] KEY; // True when not pressed, False when pressed
   input logic [9:0] SW;
   inout logic [35:0] GPIO_0;
   logic [9:0] randomInput;

   logic resetLight, key;
   logic[2:0] LS, RS;

   logic left, right;

   logic [7:0] rowSink,orangeDriver, greenDriver;

   assign GPIO_0[11:0] = 12'b00000000000;

   assign GPIO_0[19:12] = rowSink;

   assign GPIO_0[27:20] = orangeDriver;

   assign GPIO_0[35:28] = greenDriver;

   logic keyl, keyr, keyd, keyu, reset, pause;
   logic L, R, up, down, resetGame;
   logic [3:0] level;
   logic [7:0][7:0] obsMove, frogMove;
      // Generate clk off of CLOCK_50, whichClock picks rate.
   logic [31:0] clk;
   parameter whichClock = 10;
   clock_divider cdiv (CLOCK_50, clk);


   // Hook up FSM inputs and outputs.
   // logic reset, w, out;
   assign reset = SW[9];   // Reset when KEY[0] is pressed.

   logic hardReset;

   assign HEX1 = 7'b1111111;
   assign HEX2 = 7'b1111111;
   assign HEX3 = 7'b1111111;
   assign HEX4 = 7'b1111111;
   assign HEX5 = 7'b1111111;

   meta m1(.clk(clk[whichClock]), .reset(reset | hardReset), .in(~KEY[0]), .out(keyr));
   meta m2(.clk(clk[whichClock]), .reset(reset | hardReset), .in(~KEY[1]), .out(keyl));
   meta m3(.clk(clk[whichClock]), .reset(reset | hardReset), .in(~KEY[2]), .out(keyd));
   meta m4(.clk(clk[whichClock]), .reset(reset | hardReset), .in(~KEY[3]), .out(keyu));
   meta m5(.clk(clk[whichClock]), .reset(reset | hardReset), .in(SW[1]), .out(level[0]));
   meta m6(.clk(clk[whichClock]), .reset(reset | hardReset), .in(SW[2]), .out(level[1]));
   meta m7(.clk(clk[whichClock]), .reset(reset | hardReset), .in(SW[3]), .out(level[2]));
   meta m8(.clk(clk[whichClock]), .reset(reset | hardReset), .in(SW[4]), .out(level[3]));
   meta m9(.clk(clk[whichClock]), .reset(reset | hardReset), .in(SW[0]), .out(pause));


   userInput u (.clk(clk[whichClock]), .reset(reset | hardReset), .w(keyu), .out(up));
   userInput d(.clk(clk[whichClock]), .reset(reset | hardReset), .w(keyd), .out(down));
   userInput r (.clk(clk[whichClock]), .reset(reset | hardReset), .w(keyr), .out(R));
   userInput le (.clk(clk[whichClock]), .reset(reset | hardReset), .w(keyl), .out(L));

   frog f(.clk(clk[whichClock]), .reset(reset | hardReset), .resetGame, .pause, .left(L), .right(R), .up, .down, .frogPrev(frogMove), .frogNext(frogMove));

   obstacle o(.clk(clk[whichClock]), .reset(reset | hardReset), .resetGame, .pause, .obsPrev(obsMove), .obsNext(obsMove), .level);

   count c(.clk(clk[whichClock]), .reset(reset | hardReset), .up, .finalRow(frogMove[7]), .display(HEX0), .resetEverything(hardReset));

   lose l(.clk(clk[whichClock]), .obsMove, .frogMove, .resetGame);

   led_matrix_driver test (.clock(clk[whichClock]), .red_array(obsMove), .green_array(frogMove), .red_driver(orangeDriver),
                           .green_driver(greenDriver), .row_sink(rowSink));

   //hazardlight h (.clk(clk[whichClock]), .reset, .w(SW), .out(LEDR));

endmodule


module clock_divider (clock, divided_clocks);
   input logic clock;
   output logic [31:0] divided_clocks;

   initial begin
      divided_clocks <= 0;
   end

   always_ff @(posedge clock) begin
      divided_clocks <= divided_clocks + 1;
    end
endmodule
```

11. Here is the block diagram for the entire frogger system.



12. Here is the picture of my final circuit presented to TA for demonstration:



13. <u>User's Manual</u> :

a. First start the game by pushing up the SWITCH 9 to reset the game. Then, push down the SWITCH 9 to start the game and set the level to 1 through SWITCH 1. After doing this you would see red lights passing from left to right and also right to left in columns 1 to 6 which are your obstacles. There would be one green light in the middle of the column 0 which is your frogger. There would be no light in the 7th column which is the place after passing which you win the game.

b. Then use KEY[3], KEY[2], KEY[1], KEY[0] to move the frogger in up, down, left and right directions to dodge the obstacles and reach the end point

c. If you don't want to hamper with your game in progress and want to go somewhere, there is a pause feature in the game where you can pause the current state. You can activate this feature through SWITCH 0.

d. If you have now started relishing this game, you can make it more challenging by pushing down SWITCH 1 and using SWITCH 2, 3, 4 for upgrading to levels 2, 3, and 4. Upgrading the levels would increase the speed of your obstacles in the way.

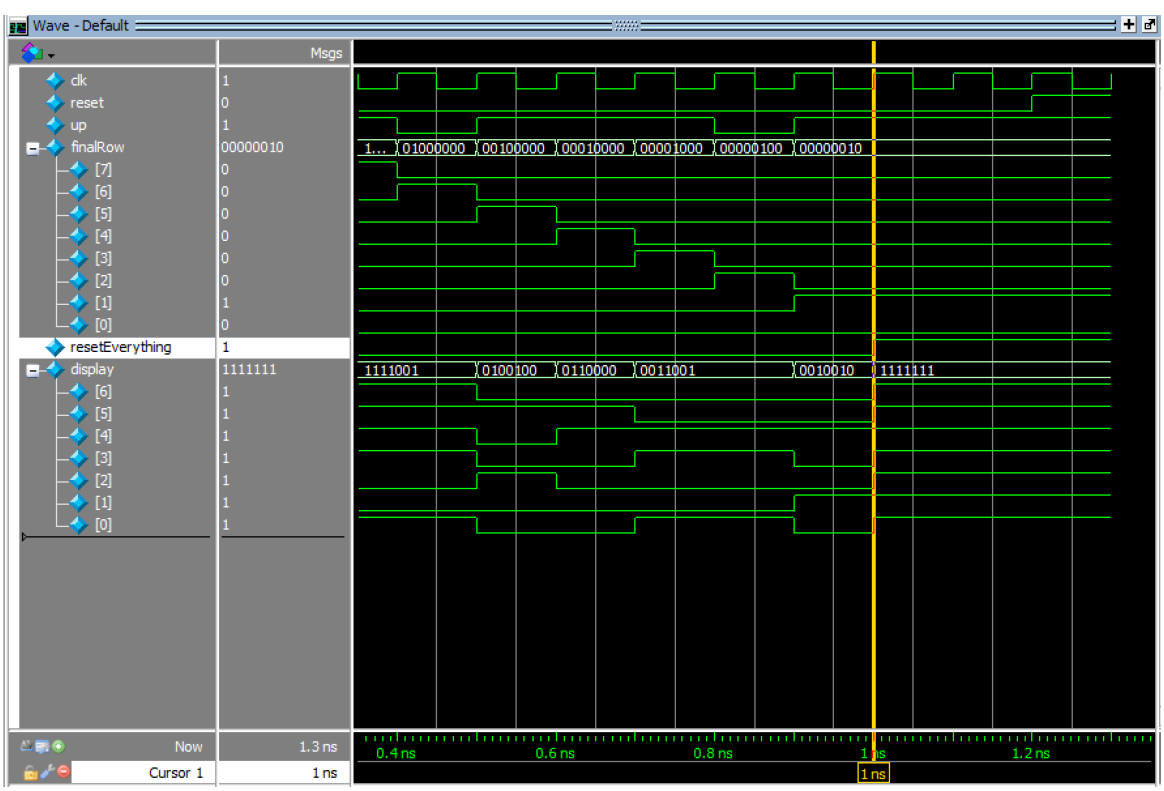
e. The game would automatically reset after you win 6 games irrespective of how many you lost in the middle.

Results

In the specification sheet, we were asked to make a frogger which took up, down, right and left as inputs and moves faster than the cars to reach from one point to the other. In the finished project, I connected the up, down, right and left to KEYS 0 to 3. We were asked to make a reset option available too which I did through SWITCH 9. In the end, to experiment more with SystemVerilog, I implemented a feature of pausing in the game with SWITCH 0. Also, I included a feature of speeding the cars to add difficulty with levels option available through pushing SWITCHES 1 to 4 up or down as per will. I used an LED matrix for the cars (red lights) and the frogger (green light) display.

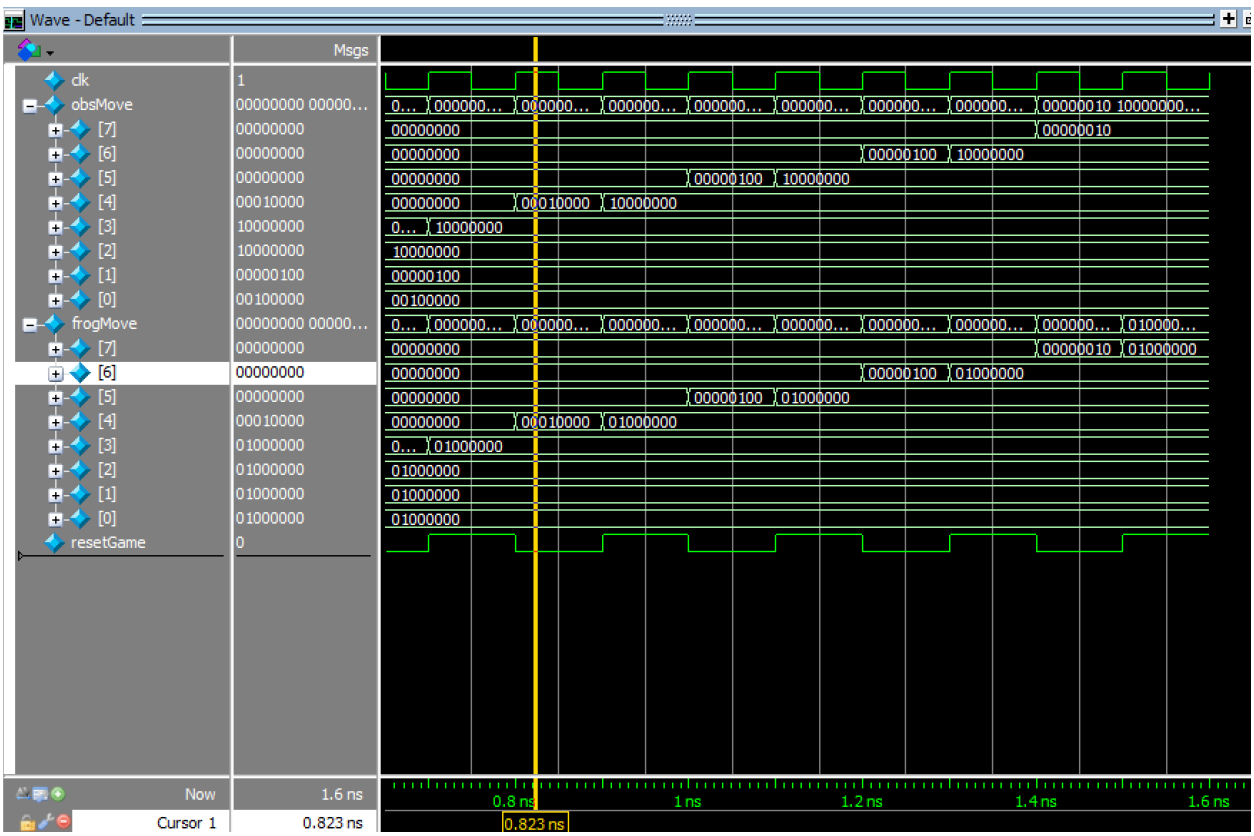
Below are the ModelSim simulation I ran on specific modules in the code.

ModelSim Simulation for Count



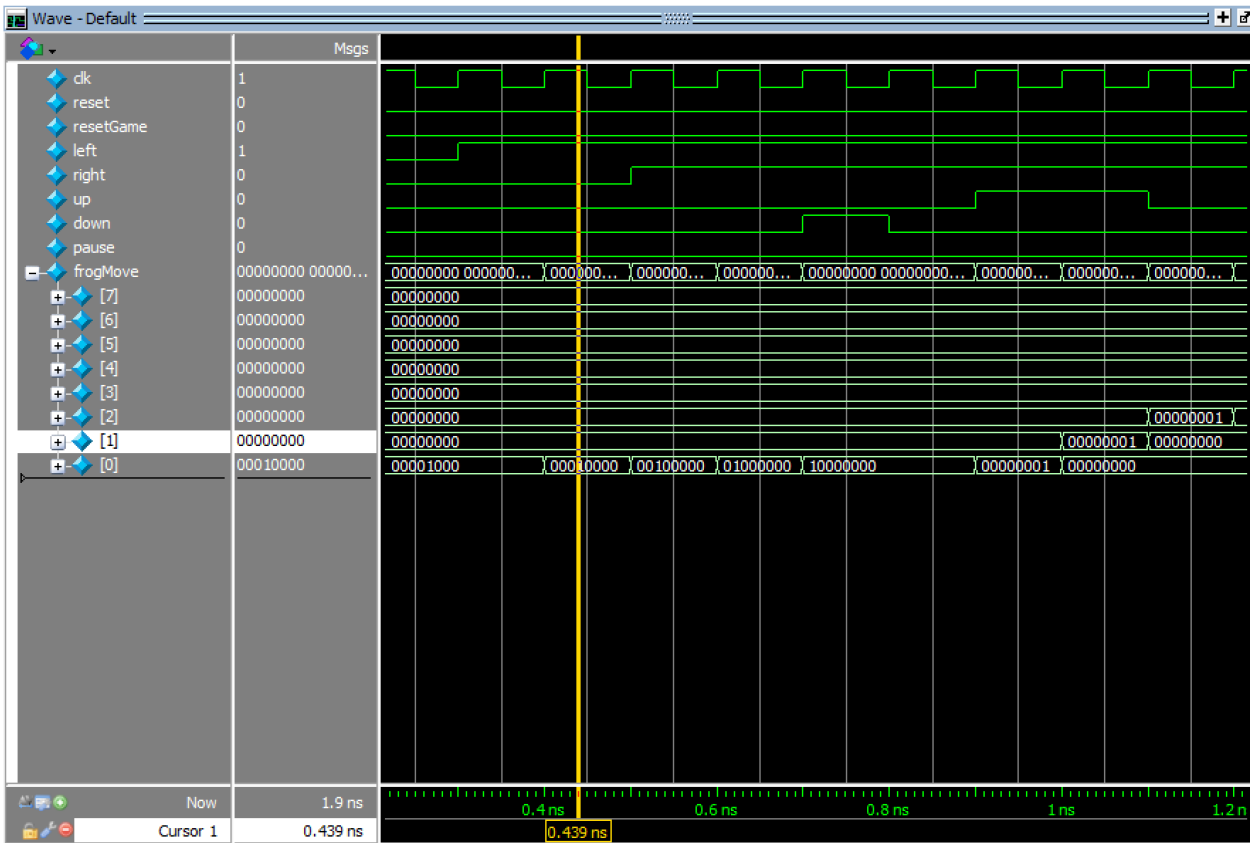
In the simulation above, I tried to see if the value on the display changes with a win or not. A win is defined by an up being High with finalRow not being 0. After 5 wins, the display sets to 0. In the figure above, we see the display is 7'b1111111 which is 0 on a HEX display. Before the change we can see the display showed 7'b0010010 which is 5 on HEX display. Therefore, the above simulation shows that it works according to my plan i.e. HEX display turning to all zeros and also, resetting everything on the board as shown by resetEverything being high in the picture above.

<u>ModelSim Simulation for Lose</u>



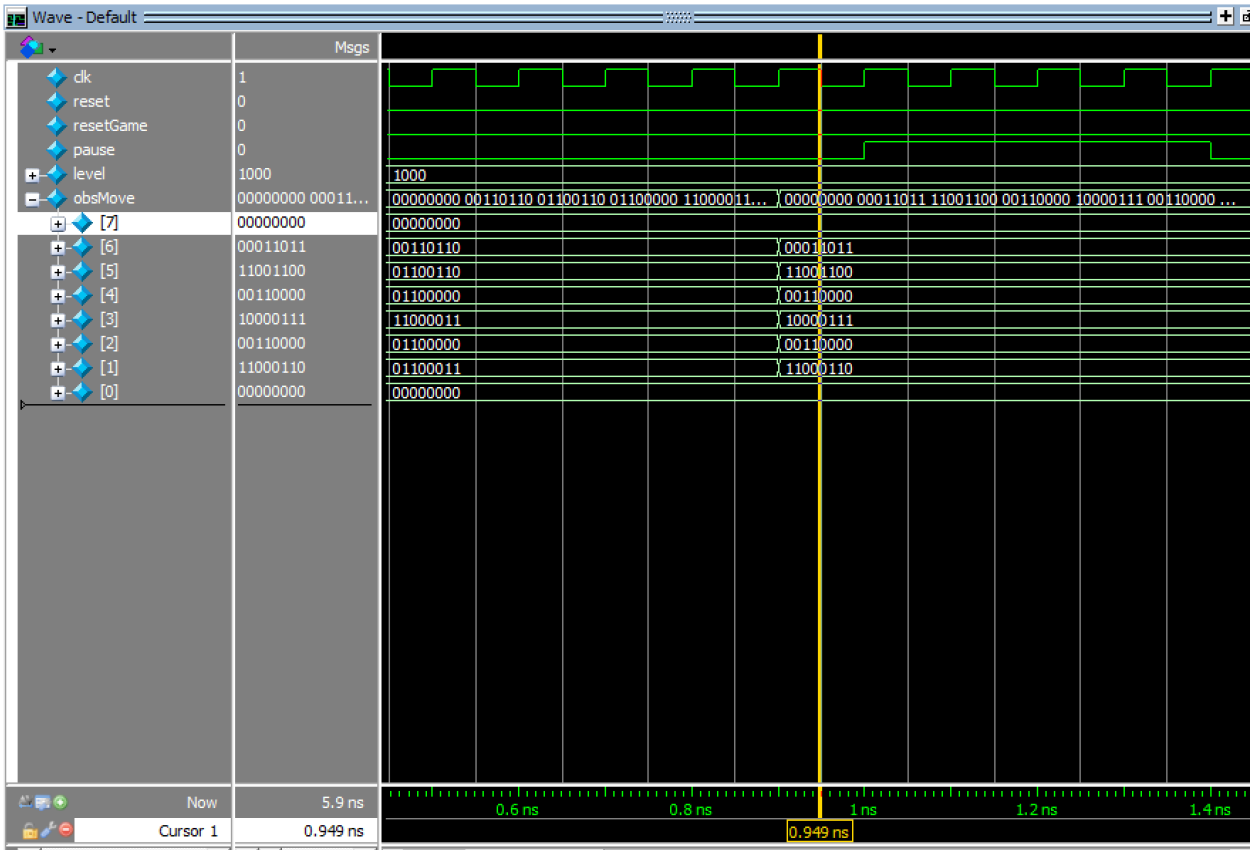
In the simulation above we try to see if the resetGame is turned to high if frogMove and obsMove are in same in a particular row and column. We do this to identify the collision between frog and obstacles. In the picture above, we see that the resetGame is turned to "one" when the frogMove[4] and obsMove[4] are equal. Therefore, the simulation works fine.

ModelSim Simulation for Frog



In the simulation above, I try to see whether the frog is moving as per the input provided or not. Here, in this simulation above, we see that earlier before the yellow line, the frog was in 0 row and the bit representation was 8’b00001000. On the yellow line, we see that the left is turned to high and our bit representation becomes 8’b00010000, i.e. shifted to left. Thus, we see that our situation works right and so does the frog module.

ModelSim simulations for obstacle



In the simulation, I try to see if there is a change in the values of obsMove when a level is shifted. Especially, I try to see if particular rows are shifted to right and some to left or not. In the simulation above, we see that the values in rows 6, 4 and 2 are shifted to right and the values in rows 1, 3 and 5 are shifted to left. Thus, the simulation works according to my will.

Resource Utiization By Entity for whole design

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name
1	|DE1_SoC	298 (1)	198 (0)	0	0	103	0	|DE1_SoC
1	|clock_divider:cdiv|	11 (11)	11 (11)	0	0	0	0	|DE1_SoC|clock_divider:cdiv
2	|count:c|	13 (13)	3 (3)	0	0	0	0	|DE1_SoC|count:c
3	|frog:f|	105 (105)	64 (64)	0	0	0	0	|DE1_SoC|frog:f
4	|led_matrix_driver:test|	49 (49)	3 (3)	0	0	0	0	|DE1_SoC|led_...x_driver:test
5	|lose:l|	16 (16)	1 (1)	0	0	0	0	|DE1_SoC|lose:l
6	|meta:m1|	1 (1)	2 (2)	0	0	0	0	|DE1_SoC|meta:m1
7	|meta:m2|	1 (1)	2 (2)	0	0	0	0	|DE1_SoC|meta:m2
8	|meta:m3|	1 (1)	2 (2)	0	0	0	0	|DE1_SoC|meta:m3
9	|meta:m4|	1 (1)	2 (2)	0	0	0	0	|DE1_SoC|meta:m4
10	|meta:m5|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|meta:m5
11	|meta:m6|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|meta:m6
12	|meta:m7|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|meta:m7
13	|meta:m8|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|meta:m8
14	|meta:m9|	0 (0)	2 (2)	0	0	0	0	|DE1_SoC|meta:m9
15	|obstacle:o|	95 (17)	94 (28)	0	0	0	0	|DE1_SoC|obstacle:o
1	|enable1:e1|	22 (22)	18 (18)	0	0	0	0	|DE1_SoC|obs...o|enable1:e1
2	|enable2:e2|	20 (20)	17 (17)	0	0	0	0	|DE1_SoC|obs...o|enable2:e2
3	|enable3:e3|	19 (19)	16 (16)	0	0	0	0	|DE1_SoC|obs...o|enable3:e3
4	|enable4:e4|	17 (17)	15 (15)	0	0	0	0	|DE1_SoC|obs...o|enable4:e4
16	|userInput:d|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|userInput:d
17	|userInput:le|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|userInput:le
18	|userInput:r|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|userInput:r
19	|userInput:u|	1 (1)	1 (1)	0	0	0	0	|DE1_SoC|userInput:u

Here, the computed size of the design is (298 + 198) - (11 + 11) = 474

Problems Faced and FeedBack

I had faced problem while wiring the LED Matrix onto breadboard. After I was done. Doing the wiring for first time, I realized that at first I did it in wrong way. Then, I had to rewire the circuit again. However, this time I saw that the wires I used were thick and they didn't allow my Matrix to go inside. After, consulting TA about this, I decided to use jumper wires to wire the matrix onto my current set up. Also, I faced problem while shifting the frog and obstacles in either direction. The got through this problem by understanding how to Matrix from TA.

I found the trick to use jumper wires really helpful and would use them instead of normal wire instead next time to save myself time.

In the end, I really found this project to be useful as it revived all the concepts in my mind that I learned throughout this course. I am really eager to use these concepts in any further project I do.