Table of Contents:

Chapter 1: Introduction

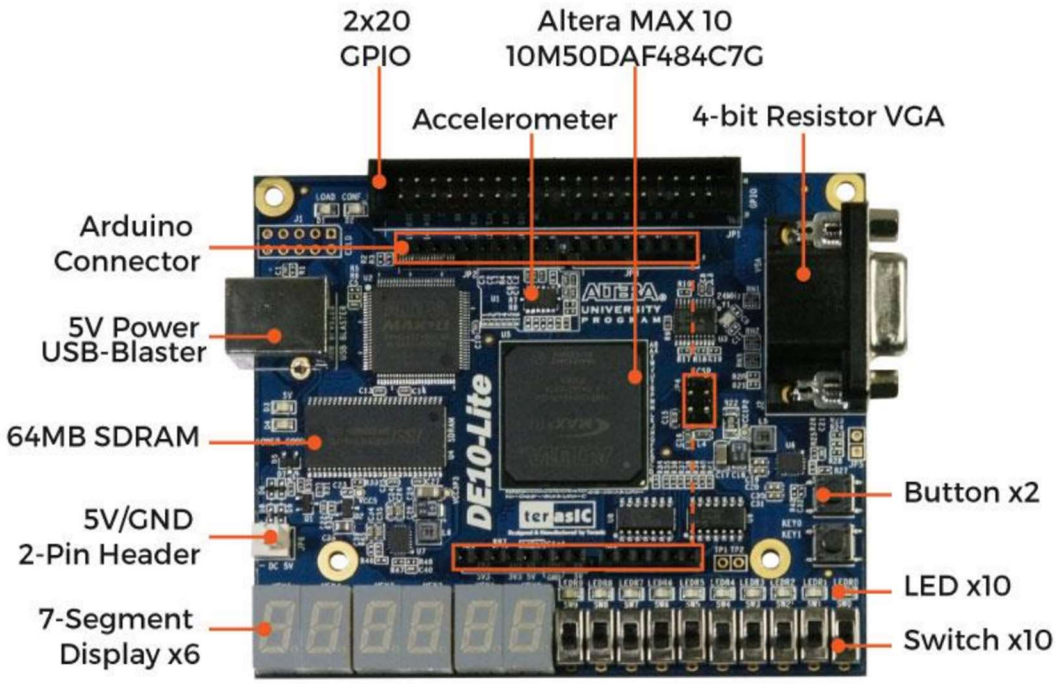
Within this project, we have broken down our approach to implementing a realistic cricket game onto a DE 10 LITE(FPGA board). Using Verilog, we created a pseudorandom number generator using an LFSR (Linear Feedback Shift Register) and assigned probabilities to the outcomes in order to closely reflect a real cricket game. Through various logical elements, we have utilized components such as push buttons to take in user input and have developed a lifelike cricket scoreboard to keep track of things like number of runs, current ball count etc. You will find explanations on various aspects of digital design such as button debouncing, using a pseudorandom number generator, slowing a clock using a counter, converting data for seven-segment led display and of course how to tie them all together in order to form a functioning circuit. All code written for this project is referenced in text and can be found in the appendix at the end of the paper.

With the ability to become any digital circuit, the FPGA (Field Programmable Gate Array) allows for tighter security, lower power consumption, higher data reliability as well as being the fastest way to integrate a specific digital design into a device.

This project is intended for anyone with basic knowledge of digital design and can aid as a detailed project for both students and individuals excited to jump into programming FPGAs.

APPARATUS REQUIRED :

1. DE 10 Lite



Modules:

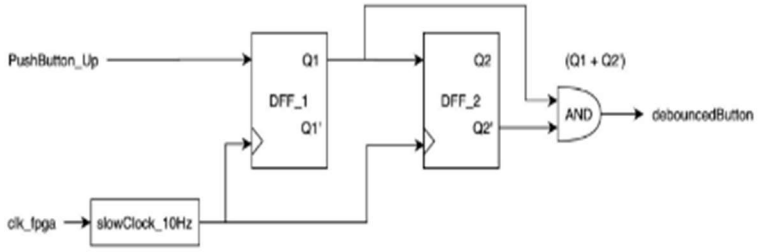
- Clock Module:

  The system clock for the DE 10 runs at a frequency of 50 MHz. Sometimes we are able to use this frequency for our applications but for others we need to slow this frequency down to produce the results we need.

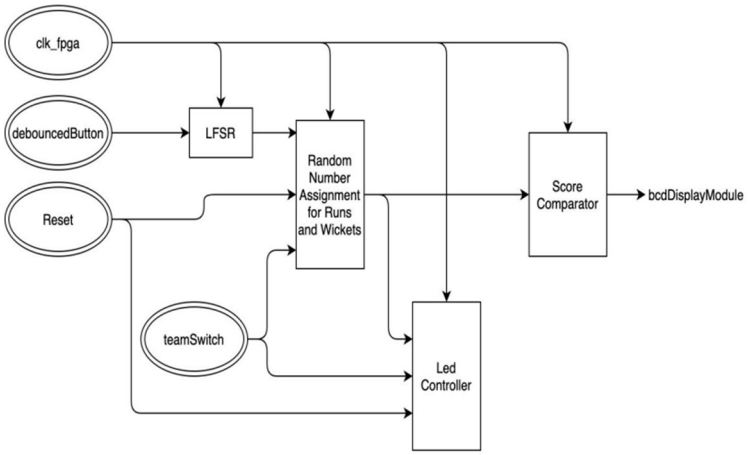
  In order to obtain the desired frequency of the clocks we can use a simple up-counter. When we give a counter a certain value to reset itself, we can then count the number of clock ticks of the system clock and generate a different clock pulse that transitions when we reach the maximum count. Depending how we set the max count value will determine the frequency of the slower

clock. For example, if we set the max count to a value of 2 it will create a slower clock at half the system clock frequency, as shown in figure 7.

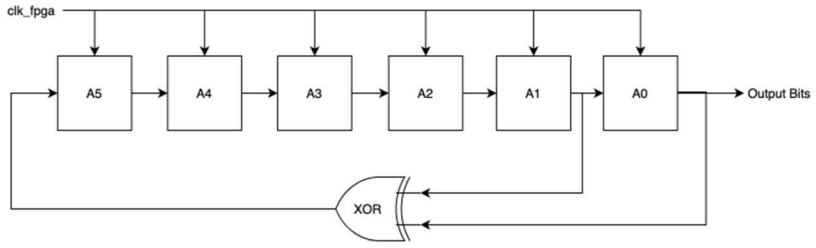
- Debounce Module:
  A common problem found in digital circuits is when a mechanical button is pressed generally it will make and lose contact multiple times before setting into a steady state. This is referred to as the button bouncing [4] and in order to not send multiple button press signals from a single press we have decided to implement a button debouncer shown in figure 8 and represented in Figure.



- Cricket Game Module:
  The Cricket Game module consists of the four modules LFSR, score and wickets, score comparator and led controller.



- LFSR Module:
  A linear feedback shift register is used in lfsr.v (figure A-5) to provide a pseudorandom 4 bit number for the assignment of scores and wickets. Shown in figure, a shift register of length 6-bits is used to increase the period, where the maximum period of an n-bit long lfsr is equal to 2 n – 1.



- Score Wicket and Comparator:

LFSR Random Number Output 0-15	**Assigned Value Based on LFSR Output**	**Probability of Receiving Score from any one Delivery**
0,1,2	Dotball: Runs + 0 ballCount + 1	18.75%
3,4,5,6	Single: Runs + 1 ballCount + 1	25%
7,8,9	Double: Runs + 2 ballCount + 1	18.75%
10	Triple: Runs + 3 ballCount + 1	6.25%
11	Four: Runs + 4 ballCount + 1	6.25%
12	Six: Runs + 6 ballCount + 1	6.25%
13	WideBall: Runs + 1 ballCount + 0	6.25%
14	NoBall: Runs + 1 ballCount + 0	6.25%
15	Wicket: wicketCount + 1 ballCount + 1	6.25%

Code:

```
module project(
   input clk_fpga,
   input reset,
   input btnU,
   input sw,    // switch between Team 1 and Team 2

   output [0:6] segW,segO, segT, segH,
   output [6:0] Y,
   output [7:0] led  // drive the leds
   );

         wire delivery;          // debounced up button press
         wire [7:0] binaryRuns;    // runs from game
         wire [3:0] binaryWickets; // wickets from game
         wire inningOver;         // signal from match to bcd display to
show 10 (inning over) on display
         wire gameOver;           // signal from match to bcd display to
lock in winner on display
         wire winner;            // signal from match to bcd display to select
winner to display

         // debounces the up push button
         debounce d0(clk_fpga, btnU, delivery);
         // A single game of cricket
         cricketGame g0(clk_fpga, reset, delivery, sw, binaryRuns,
binaryWickets, led, inningOver, gameOver, winner,Y);  // converts and
displays the runs on the three leftmost digits and the wickets on the last
fourth digit, separated by a decimal point.
         bcdDisplay b0(clk_fpga, binaryRuns, binaryWickets,
inningOver, gameOver, winner, segW, segO , segT, segH);

endmodule
```

```
module debounce(
   input clk_fpga,              // clock signal input button,
   input button,                        // input button
   output debounced_button      // debounced button
   );

        wire Q1;            // output of first D flip flop and input of
second D flip flop
        wire Q2;            // output of second D flip flop
        wire Q2_bar;        // inverted output of second D flip flop

        /*Most switches exhibit bounce rates at over 100Hz (under 10
ms).
        For seemingly instantaneous responses, it is reasonable to pick a
debounce period of 20ms (50Hz) - 50ms (20Hz).
        We debounce at 100ms (10Hz) to make use of a module made to
scroll leds at 10Hz
        */
        slowClock_10Hz u1(clk_fpga, clk_10Hz);      // 10Hz slow
clock
        D_FF d1(clk_10Hz, button, Q1);          // first flip flop
        D_FF d2(clk_10Hz, Q1, Q2);                  // second flip
flop

        assign Q2_bar = ~Q2;                        // invert output of
second D flip flop
        assign debounced_button = Q1 & Q2_bar;      // send out
debounced button

endmodule
```

```
module D_FF(
   input clk, // input clock, slow clock
   input D,   // pushbotton
   output reg Q,
output reg Qbar
   );

        always @ (posedge clk)     begin
                Q <= D;
                Qbar <= !Q;
        end
endmodule


module cricketGame(
   input clk_fpga,
   input reset,
   input delivery,
   input teamSwitch,
   output [7:0] runs,
   output [3:0] wickets,
   output [7:0] leds,
   output inningOver,
   output gameOver,
   output winner,
        output [6:0] y
   );
        wire clk_10;
        wire [11:0] team1Data; // stores and updates teaml's runs and
wickets when switch is 0
        wire [11:0] team2Data; // stores and updates team2's runs and
wickets when switch is 1
        wire [6:0] team1Balls; // the number of teaml's deliveries that are
legal balls, shown on LEDs
```

```
        wire [6:0] team2Balls; // the number of team2's deliveries that
are legal balls, shown on LEDs
        wire [3:0] lfsr_out;   // pseudorandom number from linear
feedback shift register

        slowClock_10Hz(clk_fpga,clk_10);
        // pseudo random number generator using a linear feedback shift
register
        lfsr g1(clk_10, reset, lfsr_out);

        lab6(y,lfsr_out);
        // assign score and wickets based on pseudo random number
generated by lfsr
        score_and_wickets g2(clk_10, reset, delivery, teamSwitch,
lfsr_out, gameOver, runs, wickets, team1Data, team2Data);
        // comparator that finds and locks in the winner when the game is
over
        score_comparator g3(clk_10, reset, team1Data, team2Data,
team1Balls, team2Balls, wickets, leds, inningOver, gameOver, winner);
        // assign Leds based on balls of team in play, or scroll leds when
the game is over
        led_controller g4(clk_10, reset, teamSwitch, delivery, lfsr_out,
inningOver, gameOver, leds, team1Balls, team2Balls);

endmodule


module lfsr(
   input clk_fpga,
        input reset,
   output [3:0] lfsr_out
   );

        reg [5:0] shift;
```

```
        wire xor_sum;

        assign xor_sum = shift[1] ^ shift[4]; // feedback taps
        always @ (posedge clk_fpga)
        begin
            if(reset)
            shift <= 6'b111111;


            else
            shift <= {xor_sum, shift[5:1]};  // shift right


        end
        assign lfsr_out = shift[3:0]; // output of LFSR

endmodule


module score_and_wickets(
    input clk_fpga,
    input reset,
    input delivery,
    input teamSwitch,
    input [3:0] lfsr_out,
    input gameOver,
    output reg [7:0] runs,
    output reg [3:0] wickets,
    output reg [11:0] team1Data, // stores and updates teaml's runs and
wickets when switch is 0
    output reg [11:0] team2Data  // stores and updates team2's runs and
wickets when switch is 1
    );
```

```
localparam single = 16;
localparam double = 32;
localparam triple = 48;
localparam four = 64;
localparam six = 96;

// update score after each delivery(bowl) based on cricket rule.
always @ (posedge clk_fpga, posedge reset)
begin
	if (reset)
		begin
		runs <= 0;
		wickets <= 0;
		team1Data <= 0;
		team2Data <= 0;
		end
	else if (gameOver)
		begin
		runs <= runs;
		wickets <= wickets;
		team1Data <= team1Data;
		team2Data <= team2Data;
		end
	else if(delivery)
		begin
			if((~teamSwitch) && (wickets < 10)) //
increment score of team 1
				begin
				case (lfsr_out) // pseudorandom
number from linear feedback shift register
					0,1,2:  team1Data <=
team1Data;		// dot balls
					3,4,5,6:	team1Data <=
team1Data + single;
```

```
                                        7,8,9:  team1Data <=
team1Data + double;
                                        10:             team1Data <=
team1Data + triple;
                                        11:             team1Data <=
team1Data + four;
                                        12:             team1Data <=
team1Data + six;
                                        13,14: team1Data <=
team1Data;          // wide ball and no balls
                                        15:             team1Data <=
team1Data + 1;          // wickets
                                endcase
                                runs <= team1Data[11:4];
                                wickets <= team1Data[3:0];
                                end
                        else if((teamSwitch) && (wickets < 10)) //
increment score of team 2
                                begin
                                case (lfsr_out) // pseudorandom
number from linear feedback shift register
                                        0,1,2:  team2Data <=
team2Data;                  //dot balls
                                        3,4,5,6:        team2Data <=
team2Data + single;
                                        7,8,9:  team2Data <=
team2Data + double;
                                        10:             team2Data <=
team2Data + triple;
                                        11:             team2Data <=
team2Data + four;
                                        12:             team2Data <=
team2Data + six;
```

```
                                        13,14: team2Data <=
team2Data;                  // wide ball and no balls
                                        15:             team2Data <=
team2Data + 1;              //wickets
                                endcase
                                runs <= team2Data[11:4];
                                wickets <= team2Data[3:0];
                                end
                            end
                        else // switching teams back and forth to check
scores without a up button
                            begin
                            case (teamSwitch)
                            0: begin
                                runs <= team1Data[11:4];
                                wickets <= team1Data[3:0];
                                end
                            1: begin
                                runs <= team2Data[11:4];
                                wickets <= team2Data[3:0];
                                end
                            endcase
                            end
        end
endmodule



module score_comparator(
   input clk_fpga,
   input reset,
        input [11:0] team1Data,
        input [11:0] team2Data,
        input [6:0] team1Balls,
```

```
        input [6:0] team2Balls,
        input [3:0] wickets,
        input [7:0] balls,
        output reg inningOver,
        output reg gameOver,
        output reg winnerLocked
    );

        // if the currently selected team has 120 balls or 10 wickets, their
inning is complete, so signal bcdDisplay to show IO on screen
        always @ (posedge clk_fpga)      begin
                if((wickets >= 10) || (balls >= 120))
                        inningOver <= 1;
                else
                        inningOver <= 0;
        end

        // if both teams either reach 120 balls or have lost 10 wickets,
end the game
        always @ (posedge clk_fpga, posedge reset)     begin
                if (reset)
                        gameOver <= 0;
                else if(((team1Data[3:0] >= 10) || (team1Balls >= 120))
&& ((team2Data[3:0] >= 10) || (team2Balls >= 120)))
                        gameOver <= 1;
                else
                        gameOver <= gameOver;
        end

        // on rising edge of gameover, lock in the winner
        always @ (posedge gameOver)    begin
                if (team1Data[11:4] > team2Data[11:4]) // most runs on
gameover wins
                        winnerLocked <= 0; // team 1 wins
```

```
                else
                        winnerLocked <= 1; // team 2 wins
        end

endmodule



module led_controller(
        input clk_fpga,
        input reset,
        input teamSwitch,
        input delivery,
        input [3:0] lfsr_out,
        input inningOver,
        input gameOver,
        output reg [7:0] leds,
        output reg [6:0] team1Balls,
        output reg [6:0] team2Balls
        );

        wire [7:0] scroll; // sends values for scrolling leds from
scrollLeds module when the game is over

        // count up the balls and update the leds
        always @ (posedge clk_fpga, posedge reset)     begin
                if (reset)
                        begin
                        leds <= 0;
                        team1Balls <= 0;
                        team2Balls <= 0;
                        end
                else if(gameOver)
```

```
                        leds <= scroll; // use scrolling Leds from scrollLeds
module when the game is over
                else if(delivery)
                        begin
                        if((teamSwitch == 0) && (inningOver == 0)) //
increment balls only if teaml's inning is not over
                                begin
                                case (lfsr_out)                         //
pseudorandom number from linear feedback shift register
                                        13,14: team1Balls <= team1Balls;
//wide ball and no ball
                                        default: team1Balls <= team1Balls +
1; //ones, twos, threes, fours, sixes, dotballs
                                endcase
                                leds <= team1Balls;
                                end
                        else if ((teamSwitch) && (inningOver == 0)) //
increment balls only if team2's inning is not over
                                begin
                                case (lfsr_out)
                                        13,14: team2Balls <= team2Balls;
                                default: team2Balls <= team2Balls +1;
                                endcase
                                leds <= team2Balls;
                                end
                        end
                else if(~teamSwitch)
                                leds <= team1Balls;
                else
                                leds <= team2Balls;
        end
        // supplies a signal of led values called 'scroll' to the block above.
for use when the game is over
        scroll_Leds g5 (clk_fpga, scroll);
```

```
endmodule


module scroll_Leds(
   input clk_fpga,
   output reg [7:0] led
   );

        wire clk_10Hz;
        always @ (posedge clk_10Hz)     begin
                if (led == 8'hff)
                        led <= 8'hfe; // reset to 8'b1111 1110
                else
                        led <= {led[6:0], 1'b1};
        end
        slowClock_10Hz c0 (clk_fpga, clk_10Hz);

endmodule


module slowClock_10Hz(
   input clk_fpga,     // 100MHz master clock
   output reg clk_10

        reg [22:0] period_count = 0;

        // divide the 100MHz clock to 10Hz
        always@ (posedge clk_fpga)
        begin
        period_count <= period_count + 1'b1;
        if (period_count == 2_500_000)
                begin
                period_count <= 0;  // count reset itself to zero
```

```
            clk_10Hz <= ~clk_10Hz;   // inverts the clock
            end
        end

endmodule

module bcdDisplay(
        input clk_fpga,                 // master clock 100Mhz
        input [7:0] binaryRuns,     // runs from cricket game
        input [3:0] binaryWickets, // wickets from cricket game
        input inningOver,           // show Io on the display
        input gameOver,                 // signals the end of the game
        input winner,                   // locked in winner of the game

        output [6:0] segW,segO,segT,seg;

        wire clk_1kHz;
        wire [3:0] mux_out;
        wire [1:0] counter_out;
        wire [3:0] wickets, ones, tens, hundreds;

        // binary to BCD converter
        binary_to_BCD b1 (binaryRuns, binaryWickets, inningOver,
gameOver, winner, wickets, ones, tens, hundreds);

        bcd7seg b2 (wickets, segW);
        bcd7seg b3 (ones, segO);
        bcd7seg b4 (tens, segT);
        bcd7seg b5 (hundreds, segH);

endmodule


module binary_to_BCD(
```

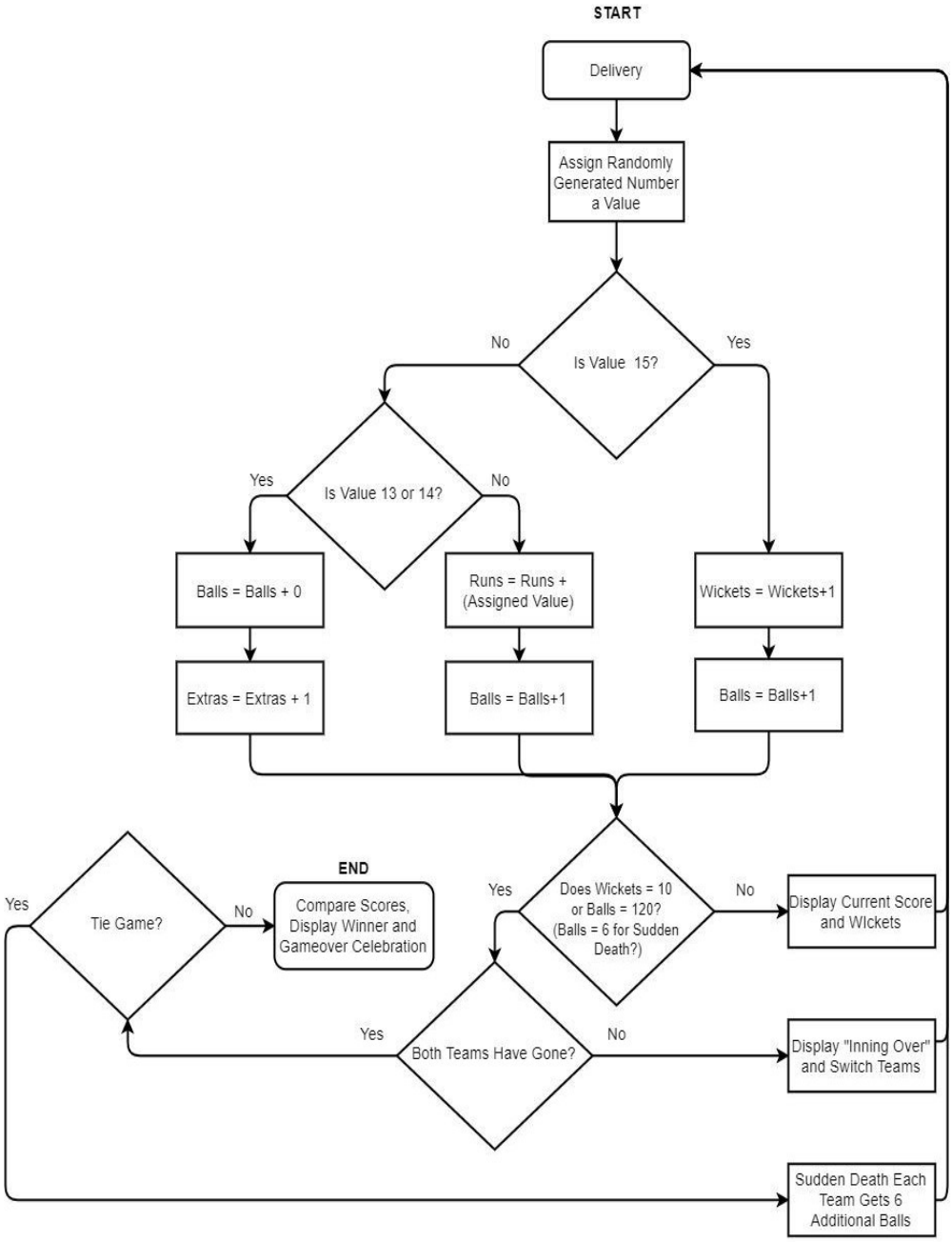
```
input [7:0] binaryRuns,
input [3:0] binaryWickets,
input inningOver,
input gameOver,
input winner,
output reg [3:0] wickets, ones, tens, hundred
      reg [7:0] data; //temporarily store binaryRuns for calculations
      always@
      begin
      if(~gameOver) // still playing
            begin
            if(inningOver)
                  begin
                  hundreds <= 4'b1100;
                  tens <= 4'b1101;     // I
                  ones <= 4'b0000;     // O
                  wickets <= 4'b1110;  // '
                  end
            else
                  begin
                  data = binaryRuns;
                  hundreds <= data / 100;
                  data = data % 100;
                  tens <= data / 10;
                  ones <= data % 10;
                  wickets <= (binaryWickets % 10);
                  end
            end
      else
            begin
            case (winner) //t010 or t020. f is swapped for t in bcd7seg
            0: begin      //t010
                  hundreds <= 4'b1111;
                  tens <= 4'b0000;
```

```
                ones <= 4'b0001;
                wickets <= 4'b0000;
                end
            1: begin     //t020
                hundreds <= 4'b1111;
                tens <= 4'b0000;
                ones <= 4'b0010;
                wickets <= 4'b0000;
                end
            endcase
            end
        end
endmodule
module bcd7seg(
        input [3:0] y,
        output reg [6:0] segs
    );

        //display 7-seg equivalent of 4-bit digit y
        always @ (y)
        begin
        case (y)
        0: segs = 7'b100_0000;  //0
        1: segs = 7'b111_1001;  //1
        2: segs = 7'b010_0100;  //2
        3: segs = 7'b011_0000;  //3
        4: segs = 7'b001_1001;  //4
        5: segs = 7'b001_0010;  //5
        6: segs = 7'b000_0010;  //6
        7: segs = 7'b111_1000;  //7
        8: segs = 7'b000_0000;  //8
        9: segs = 7'b001_0000;  //9
        10: segs = 7'b000_1000; //A
        11: segs = 7'b000_0011; //B
```

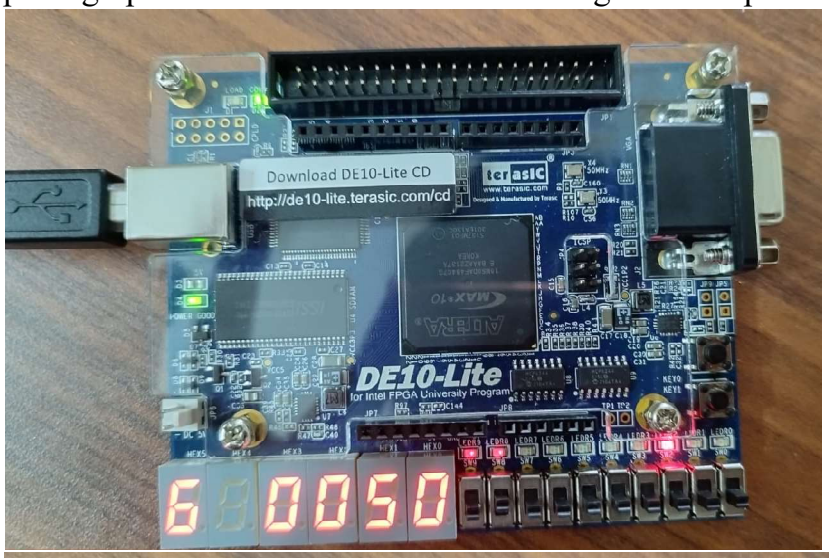
```
        12: segs = 7'b101_1111; //using a left apostrophe instead of C
        13: segs = 7'b100_1111;
        14: segs = 7'b111_1101;
        15: segs = 7'b000_0111;
        endcase
        end
endmodule
module lab6(se,s);
input [3:0]s;
output reg [0:6]se;
always@(s)
begin
case(s)
4'b0000:se=7'b0000001;
4'b0001:se=7'b1001111;
4'b0010:se=7'b0010010;
4'b0011:se=7'b0000110;
4'b0100:se=7'b1001100;
4'b0101:se=7'b0100100;
4'b0110:se=7'b0100000;
4'b0111:se=7'b0001111;
4'b1000:se=7'b0000000;
4'b1001:se=7'b0000100;
4'b1010:se=7'b0000010;
4'b1011:se=7'b1100000;
4'b1100:se=7'b0110001;
4'b1101:se=7'b1000010;
4'b1110:se=7'b0110000;
4'b1111:se=7'b0111000;
endcase
end
endmodule
```

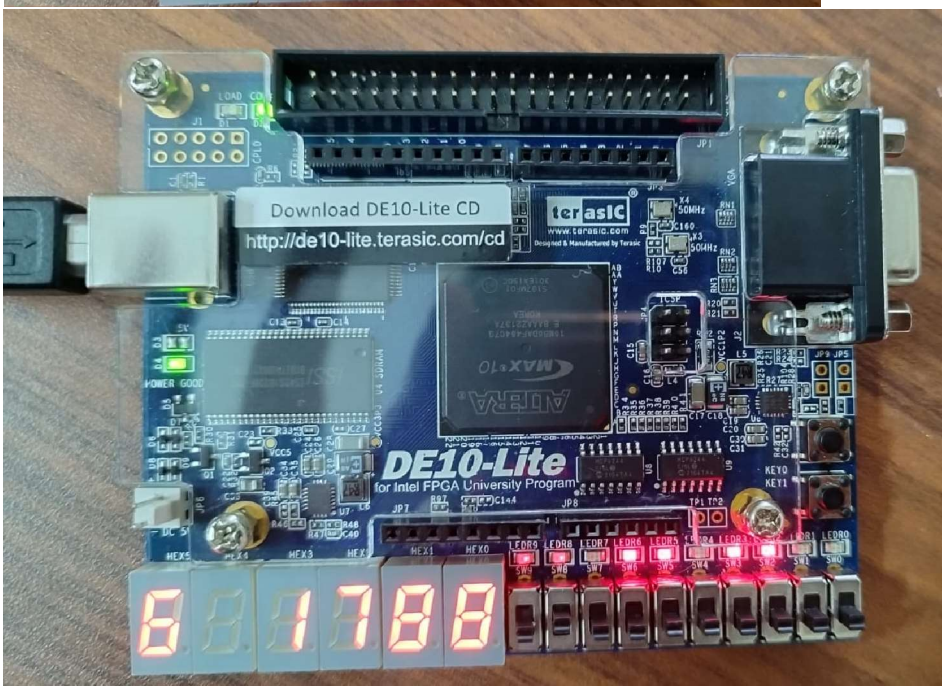
Game Flow:


START
Delivery
Assign Randomly Generated Number a Value
Is Value 15?
No
Yes
Is Value 13 or 14?
Yes
No
Balls = Balls + 0
Runs = Runs + (Assigned Value)
Wickets = Wickets+1
Extras = Extras + 1
Balls = Balls+1
Balls = Balls+1
Does Wickets = 10 or Balls = 120? (Balls = 6 for Sudden Death?)
Yes
No
Display Current Score and WIckets
Both Teams Have Gone?
Yes
No
Display "Inning Over" and Switch Teams
Tie Game?
Yes
No
END
Compare Scores, Display Winner and Gameover Celebration
Sudden Death Each Team Gets 6 Additional Balls

Conclusion and Result:

In the following section you will find our simulated digital schematics with explanations of what is being displayed. You can also find the design utilization report from our finished project in Quartus and photographs of the DE 10 board while the game is in process.





References:

https://youtu.be/OcZpzmVXPOQ
https://en.wikipedia.org/wiki/Field-programmable_gate_array
https://en.wikipedia.org/?title=Debounce&redirect=no
https://en.wikipedia.org/wiki/Linear-feedback_shift_register