Design of a Cricket Game Using FPGAs

Ronald J. Egyir and Ryan P. Devendorf

Department of Electrical and Computer Engineering Class of '20 & '21

Under the Guidance of Dr. Maqsood Mughal

June 2020

Acknowledgements

We would like to thank Worcester Polytechnic Institute for giving us the opportunity to conduct our Major Qualifying Project research on the "Design of a Cricket Game Using FPGAs".

The Completion of this project could not have been made possible without the guidance and support we have received throughout our time at WPI. A special thanks is owed to Dr. Maqsood Mughal for overseeing our efforts throughout this project.

Abstract

Within this project, we have broken down our approach to implementing a realistic cricket game onto a Basys 3 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.

Table of Figures

Appendix Figures

Table of Contents

1. Introduction

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. It is no wonder that many of the leading technology fields such as industrial, medical, aerospace, and defense have all been finding more and more ways to utilize them. Whether you are programming a rocket or simulating the wonderful game of cricket each of these complex tasks can be broken down into their many simpler parts. The goal of this project is to combine basic digital circuits such as flip-flops, decoders, comparators, counters etc. in order to create a functioning simulation of the sport cricket. 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.

The board used throughout this project is the Digilent Basys 3 prototyping board which contains the Xilinx Artix-7 FPGA. All programming has been completed using Verilog Hardware Description Language (HDL) through Vivado Design Suite. A Digital Schematic Tool (DSCH) was also used in order to create timing waveforms to help us validate the architecture of the logic circuit prior to moving into design.

To provide a breakdown of the project and understand the steps we have taken, in the following sections you will find a brief overview of the important aspects of the game of cricket, various block diagrams and RTL (Register Transfer Level) schematics representing the Verilog modules to help you understand how the logic is interconnected, a flowchart to explain the full functionality of the game and what to expect, all of our code is included in the appendix with detailed comments to further explain functionality, as well as DSCHs to visualize the game functioning properly.

Figure 1: Cricket Scoreboard Shows Runs(177), Wickets (6), and Balls Left (74)

2.The Game of Cricket

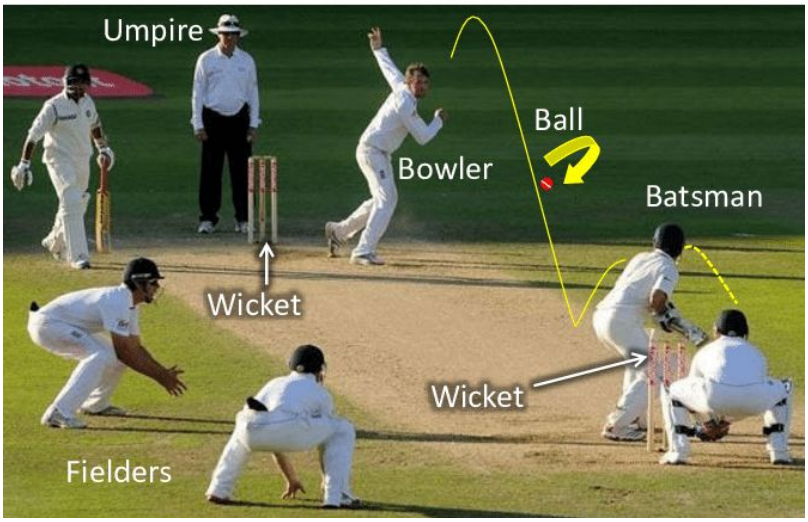

Figure 2: Key Elements in the Game of Cricket

2.1 Overview

The game of cricket is played with a bat and ball on a large field between two 11 player teams. The objective in cricket is for the batting team to score as many runs as they can and for the fielding team to prevent them from scoring. As a batsman there are various ways to score runs but in general when the game is finished the team with the most runs wins. There are many variations of the way cricket is played but for the purposes of this project we will stick to the basics in order to maintain a good grasp on the rules of the game while keeping the majority of the information adhered to the project's implementation. Below you will find further information on the rules of the game and how it is played.

2.2 Structure of the Game

Typically the game of cricket will be made up of two innings. Before the game starts there is a coin toss to determine which team will bat or field first. Each team has the opportunity to score as many runs as they can before one of two things occurs that ends the inning. The first way this can occur is that one team has been delivered all 120 balls by the opposing team. The other way is that the fielding team has managed to get 10 out of the 11 players on the batting team out. It is important to note that when a fielding team gets a batsman out it is referred to as a 'wicket'. Once both teams have had a chance to bat the score will be compared and the team with the higher score is the winner. If the score happens to be a tie then the game will then go for another two innings, however each team will only be delivered 6 balls each before the score is compared again [7].

2.3 Scoring Runs

The most common way of scoring runs in cricket is when the batsman hits the ball within the field, they must run to the other end (bowling end) to complete a run and can continue back to where they started to score more runs if one of the bowlers hasn't regained possession of the ball. In the later, failing to make it back to the batting position or falling short of the crease will result in one run only with batsmen ruled out if the bails are removed before the batsmen goes across the batting crease. It is possible for the batsmen to continue running and score up to three or more runs from one hit. More ways a batsman can score is if he hits the ball hard enough to reach the boundary. If the ball bounces on the field and then hops over the boundary the team is awarded 4 runs and if the ball makes it beyond the boundary without touching the field the team is awarded 6 runs. If either of these two cases are to occur then the batsmens holds the strike and doesn't need to run. More possible ways of a team scoring runs is from the extras. Extras results from a bowler violating the cricket rules while bowling. Examples of this include bowling above waist height which is considered a "No Ball", and an extra run is awarded to the opponent team. Similarly, If a bowler delivers the ball from the wrong position such that his foot lands outside of the crease the ball will also be called a 'No Ball' by the umpire. If the ball is declared as being too far away from the batsman it will be considered a 'Wide Ball'. A 'No Ball' and a 'Wide Ball' both fall into a category referred to as 'Extras' and when that occurs the batting team will be awarded 1 run, and additionally that bowl is not considered legal and will not count towards the 120 balls, instead being rebowled. To summarize the possibilities of scoring, for any particular delivery the batsman can score 0, 1, 2, 3, 4, or 6 runs and that's unless of course the fielding team gets the batsman out [7].

2.4 Wickets

Just like there is more than one way to score runs as a batsman, there are also multiple ways for the fielders to get the batsman out. To list a few common ways a fielder can get a batsman out there is 'Bowled', 'Caught', run out, and 'Leg Before Wicket'. If a batsman is bowled out it means that the ball has knocked the stumps behind the batsman shown in figure 2. If the batsman hits the ball and it is caught before touching the ground it is referred to as 'Caught'. 'Leg Before Wicket' refers to when a batsman is hit by the ball without it touching the bat but only if the ball was going to hit the wickets had it not been obstructed by the body of the player. As a reminder going forward, when a batsman gets out for whatever reason it is counted as a 'wicket' against their team. If the wicket count reaches 10 the inning ends and the other team gets to bat no matter how many balls have been delivered [7].

3. Implementation

Within this section, you will find the scope of the project where we have laid out what components on the Basys 3 board we will be using for the game. You will find detailed explanations with flowcharts and diagrams that express the overall game functionality as well as thorough breakdowns of the individual modules. All commented code for these modules can be found in the appendix section.

3.1 Scope

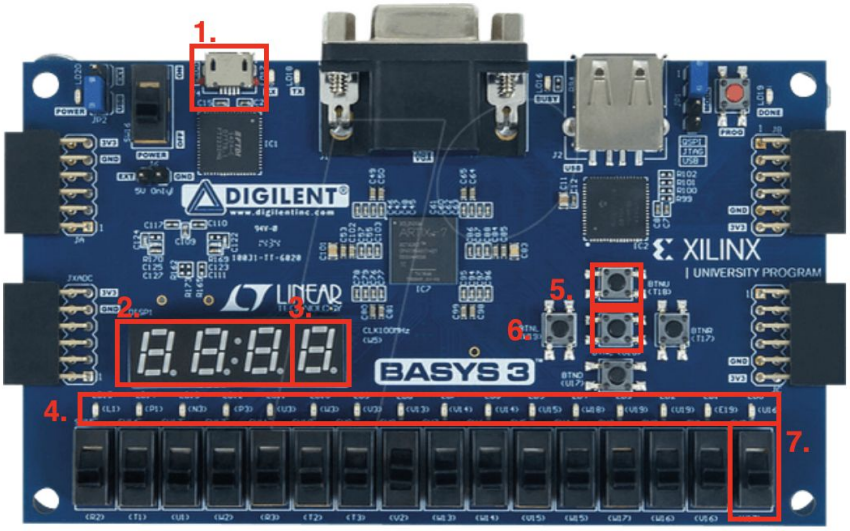

Figure 3: Basys 3 Board with Labeled Components in Use

1. We are using the micro-USB port for both programming and powering the board. The two blue jumpers you see along the top of the board will be set to ‘USB’ and ‘JTAG’ from left to right respectively. You should notice that the image shown in figure 3 does NOT have both jumpers in the proper position.

2. The first three digits on the seven segment display is used to display a team's current number of runs in decimal and can range from 0 runs to 255 runs.

3. The last digit on the seven segment display is used to display the number of wickets against the same team that has their runs displayed.

4. The row of sixteen leds will be used to display the current teams ball count in binary. Since the maximum ball count is 120 balls, we will only need the right most seven leds for this, however the remainder of the leds will be used for an end of game celebration.

5. The up pushbutton is used to simulate a delivery from the bowler. When the up button is pressed, depending on the outcome of the bowl the displays will change accordingly.

6. The middle button is used as a game reset. At any point in the game you can hit the reset button and the game will start over.

7. The rightmost switch is used as a 'team switch'. When the switch is in the up position it means that team 1 is batting and the displays will present team 1's current score, wicket and ball count. This is the same functionality in the bottom position for team 2.

More Useful Information:

- The team switch should be in the up position at the beginning of the game and should be switched to the bottom position when the display says ' IO ' which denotes 'inning over'. This means that team 1 has reached their maximum ball or wicket count and it is time for the teams to switch. Failure to do this will result in an unfair game and it is recommended that the reset switch be applied.

- Once both teams have gone the game will automatically compare the two scores and display the team with the higher score (the winner) on the three rightmost digits of the seven segment display in the form 't01' or 't02' and at the same time the leds will scroll in a celebratory fashion.

- If the game results in a tie game the same three digits will display 'tIE'. The switch will then be switched back to the team 1 position and the game will be replayed the exact same way, except this time each team will only receive 6 balls before the score is compared again.

3.2 Game Flow

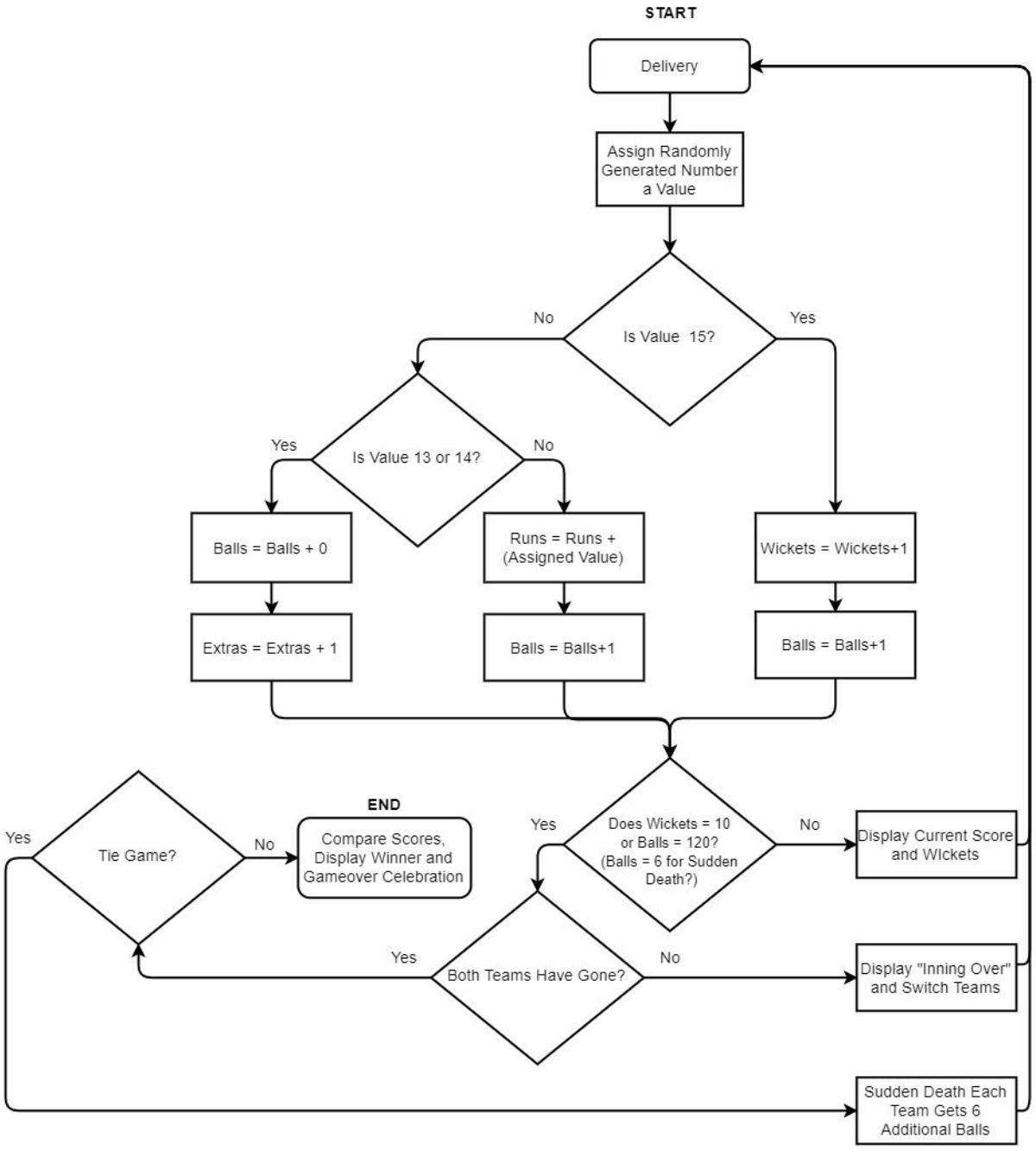

Figure 4: Entire Cricket Game Functionality

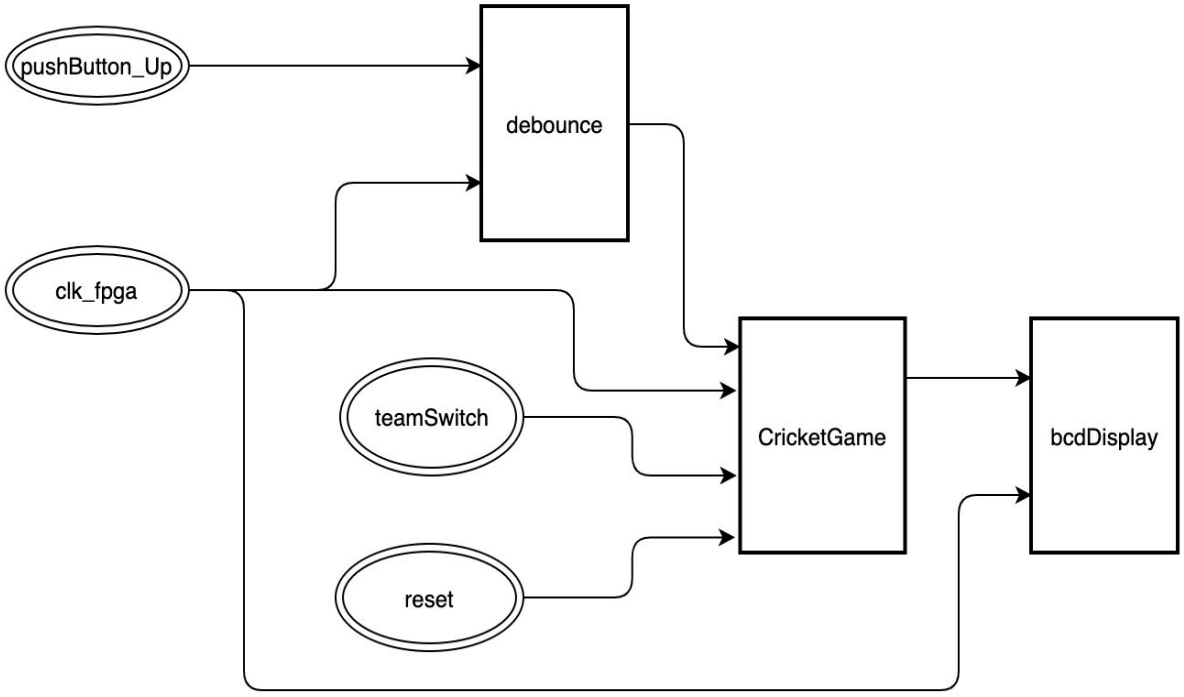

Figure 5: Functional Block Diagram of Cricket Top Module

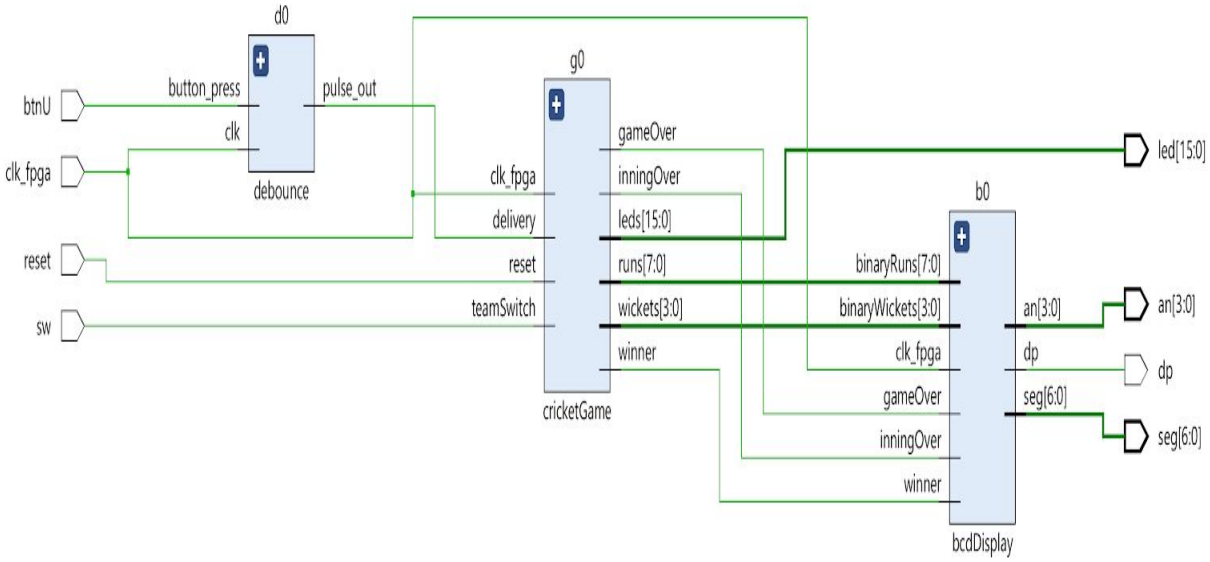

Figure 6: Verilog Schematic of Top Module

3.3 Modules

3.3.1 Slow Clock Module

The system clock for the Basys 3 runs at a frequency of 100 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. For instance, in the Basys 3 board reference manual [3] you will find that the seven segment display leds should run at a frequency between 60 Hz and 1,000 Hz. Anything higher than that can result in unstable transitions and anything lower will result in a noticeable flicker to the human eye. 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. Slow clock modules are shown in figures A-11 and A-14.

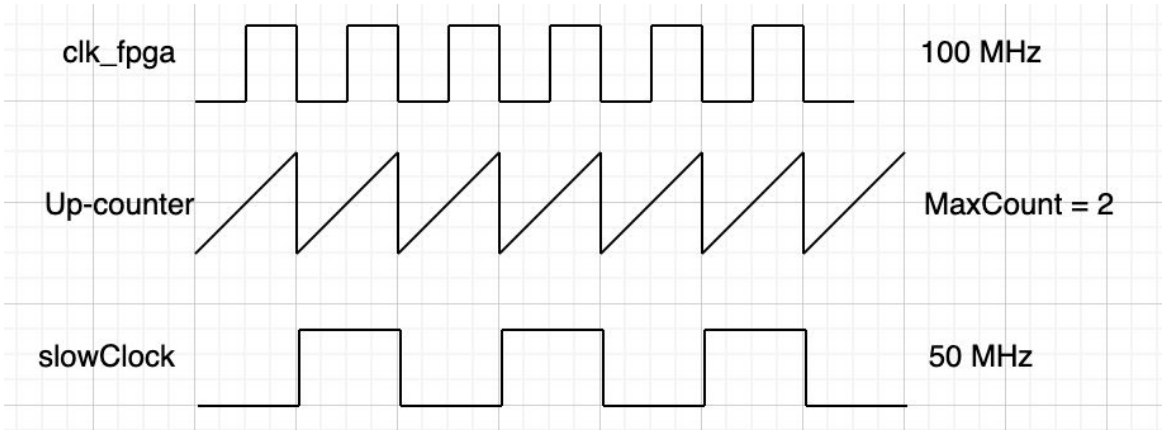

Figure 7: Timing Diagram to Generate Slow Clock from System Clock

3.3.2 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 Figures A-2 and A-3.

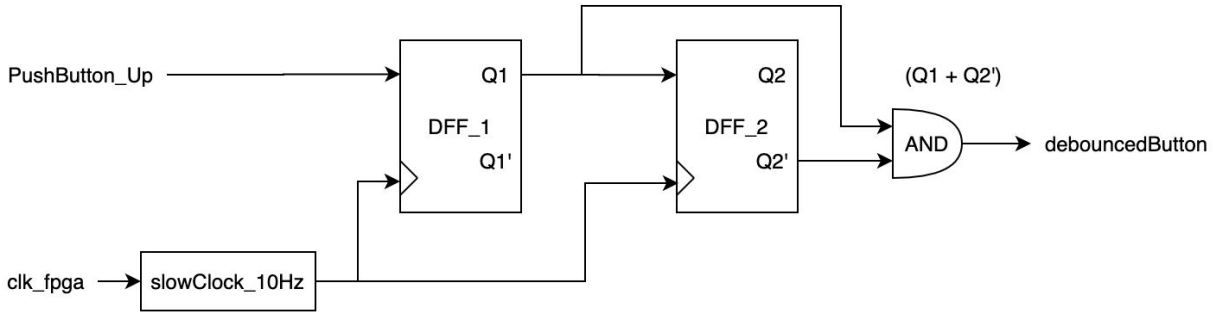

Figure 8: Block Diagram of Debounce Module

There are multiple ways of implementing this in a digital circuit but we chose to use a series of D-flip-flops that are fed with a slower clock frequency making it so the debouncedButton signal doesn't change states for multiple .1 second clock periods represented in figure 9.

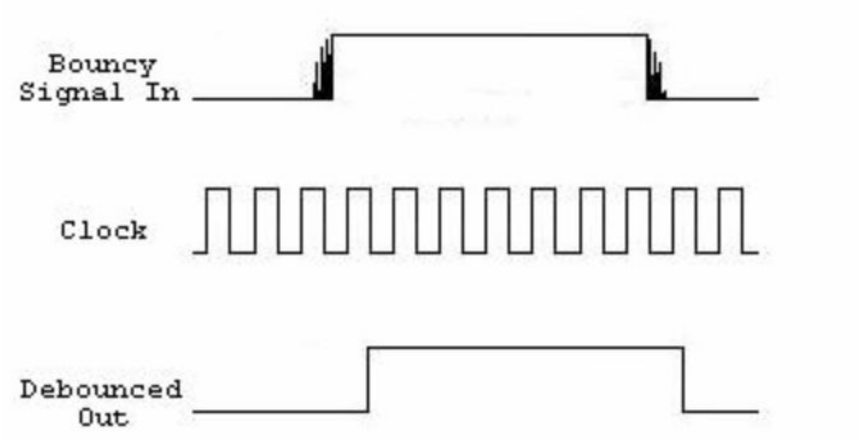

Figure 9: Timing Diagram for Generic Button Debouncing

3.3.3 Cricket Game Module

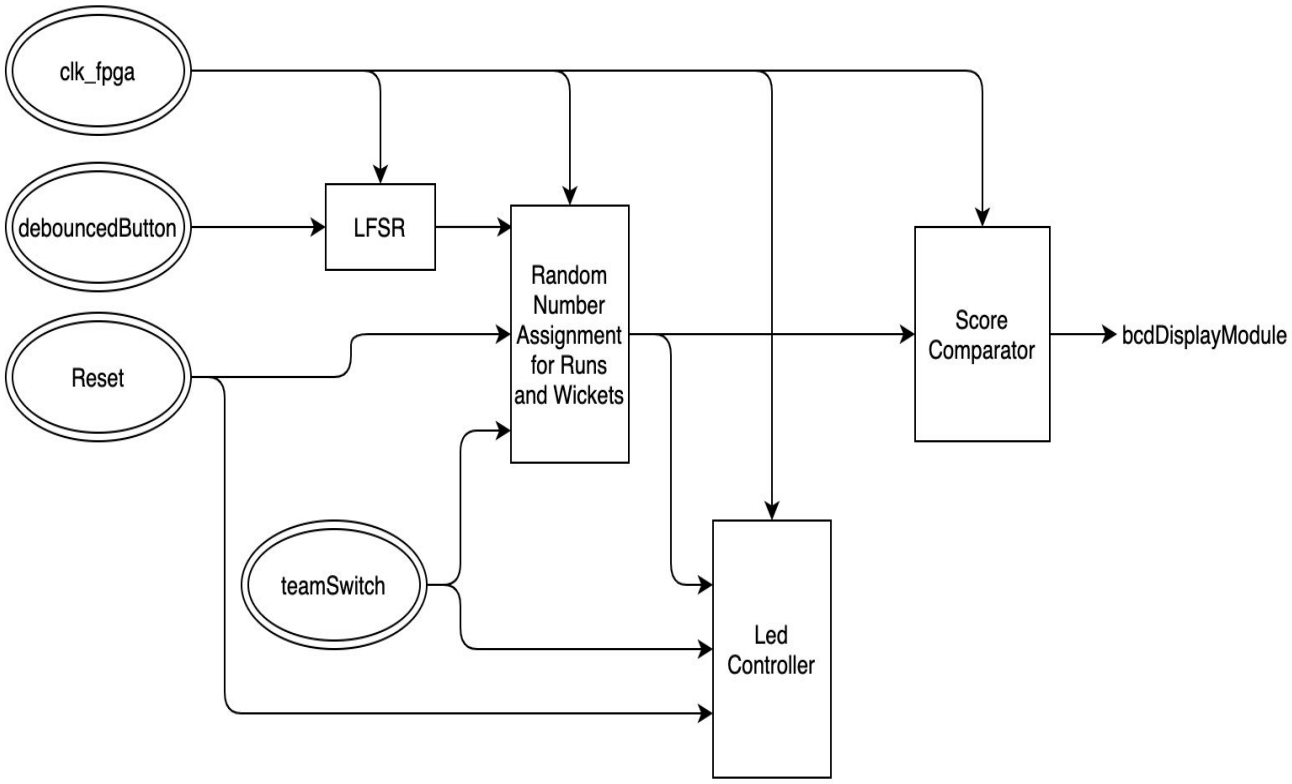

Figure 10: Block Diagram of Simplified Cricket Game Module

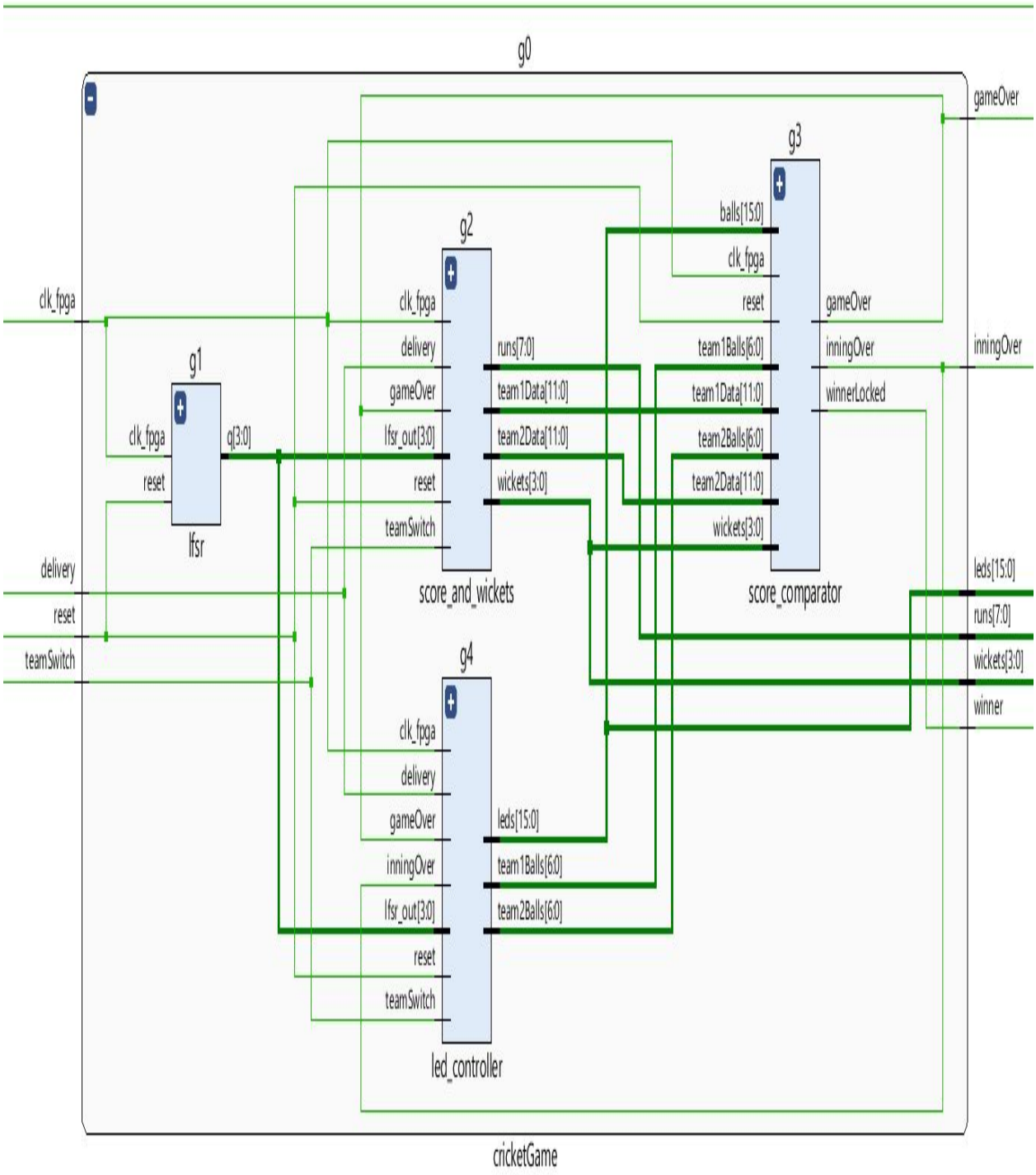

Figure 11: Verilog Schematic of Cricket Game Module

The *cricketGame* module consists of the four modules LFSR, score_and_wickets, score_comparator and led_controller shown in figures A-5, A-6, A-8, and A-9 respectively. As seen in Figure 11 above, the inputs are the Basys 3 master clock(clk_fpga), the debounced up pushbutton(delivery), the center pushbutton (reset) and the slider switch SW0 (teamSwitch) . The outputs of cricketGame are leds, runs, wickets, *inningOver*, *gameOver* and *winner.*

3.3.4 LFSR Module

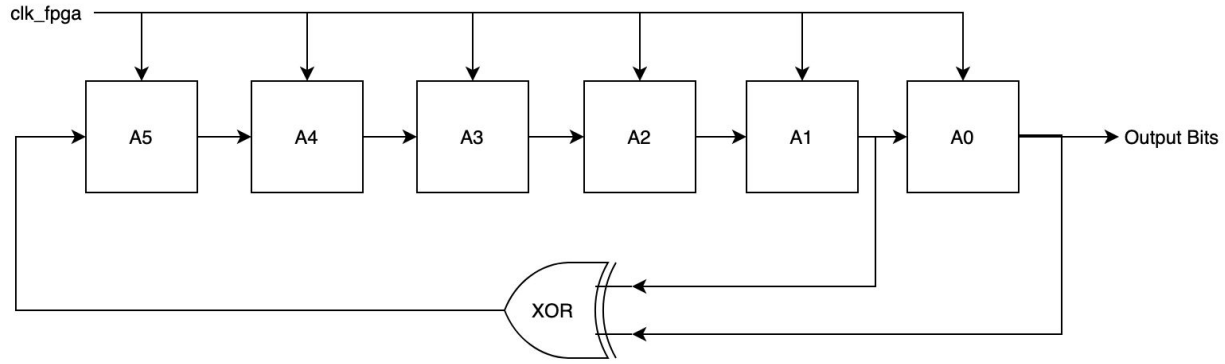

Figure 12: Block Diagram of LFSR Module for Random Number Generation [9]

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 12 above, 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 $2^n - 1$ [8]. Figure 13 shows the result of a linear feedback shift register that uses an XOR sum and starts with a seed of 6'b1111111(15d). The period of this sequence is 31 : (15,15,7,3,9,12,6,11,5,2,9,4,2,1,0,8,4,10,5,10,13,14,7,11,13,6,3,1,8,12,14).

Figure 13: Pseudorandom Numbers Generated from an LFSR

3.3.5 Score, Wickets and Comparator

ESPN cricinfo Live scores Series Teams Features Videos Stats

RECORDS / TWENTY20 INTERNATIONALS / TEAM RECORDS / HIGHEST INNINGS TOTALS

Like 2 Tweet

Highest totals

Team	Score	Overs	RR	Inns	Opposition	Ground	Match Date	Scorecard
Afghanistan	**278/3**	20.0	13.90	1	v Ireland	Dehradun	23 Feb 2019	T20I # 746
Czech Rep.	**278/4**	20.0	13.90	1	v Turkey	Ilfov County	30 Aug 2019	T20I # 872
Australia	**263/3**	20.0	13.15	1	v Sri Lanka	Pallekele	6 Sep 2016	T20I # 565
Sri Lanka	**260/6**	20.0	13.00	1	v Kenya	Johannesburg	14 Sep 2007	T20I # 27
India	**260/5**	20.0	13.00	1	v Sri Lanka	Indore	22 Dec 2017	T20I # 634
Scotland	**252/3**	20.0	12.60	1	v Netherlands	Dublin (Malahide)	16 Sep 2019	T20I # 884
Australia	**248/6**	20.0	12.40	1	v England	Southampton	29 Aug 2013	T20I # 328
Australia	**245/5**	18.5	13.00	2	v New Zealand	Auckland	16 Feb 2018	T20I # 649
West Indies	**245/6**	20.0	12.25	1	v India	Lauderhill	27 Aug 2016	T20I # 562
India	**244/4**	20.0	12.20	2	v West Indies	Lauderhill	27 Aug 2016	T20I # 562
New Zealand	**243/5**	20.0	12.15	1	v West Indies	Mount Maunganui	3 Jan 2018	T20I # 638
New Zealand	**243/6**	20.0	12.15	1	v Australia	Auckland	16 Feb 2018	T20I # 649
South Africa	**241/6**	20.0	12.05	1	v England	Centurion	15 Nov 2009	T20I # 125
England	**241/3**	20.0	12.05	1	v New Zealand	Napier	8 Nov 2019	T20I # 1008
Namibia	**240/3**	20.0	12.00	1	v Botswana	Windhoek	20 Aug 2019	T20I # 856
India	**240/3**	20.0	12.00	1	v West Indies	Mumbai	11 Dec 2019	T20I # 1024
Austria	**239/3**	20.0	11.95	1	v Luxembourg	Ilfov County	31 Aug 2019	T20I # 873
Singapore	**239/3**	20.0	11.95	1	v Malaysia	Bangkok	3 Mar 2020	T20I # 1071
West Indies	**236/6**	19.2	12.20	2	v South Africa	Johannesburg	11 Jan 2015	T20I # 414
Nepal	**236/3**	20.0	11.80	1	v Bhutan	Kirtipur	5 Dec 2019	T20I # 1018

Figure 14: Top 20 Highest Inning Totals for Twenty20 Cricket

The pseudorandom number sequence generated by *lfsr.v* (figure A-5) is used to assign the score and wickets for each delivery of a ball(up push button). The scores for Twenty 20 cricket tend to rarely exceed 256, as seen in figure 14 [10]. Teams also usually end their innings because they reached 120 balls, rather than by losing 10 wickets. The random number assignment shown in figure 15 below was carefully chosen in order to achieve realistic scores [2].

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%

Figure 15: Score Results with Probability of Occuring from LFSR Output

As shown in figure A- 8, *score_comparator.v* takes the scores, wickets, and balls, accumulated by each team in *score_and_wickets.v* (figure A-7) and compares them to determine if an inning is over, if the game is over, and if so determine who is the winner.

3.3.6 BCD to Display Module

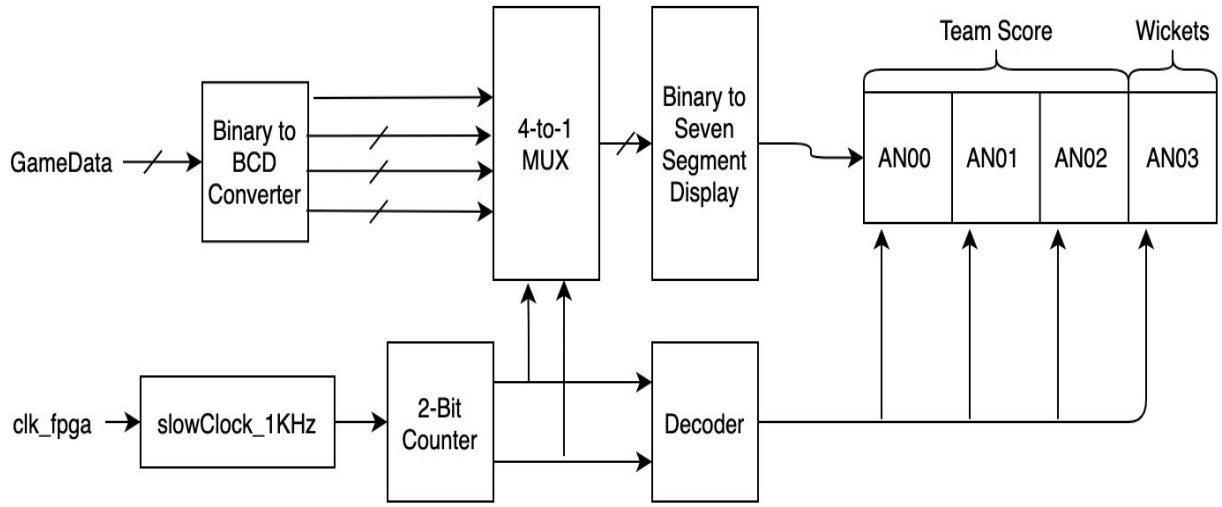

Figure 16: Block Diagram of Data from Game Module to Seven Segment Display [1]

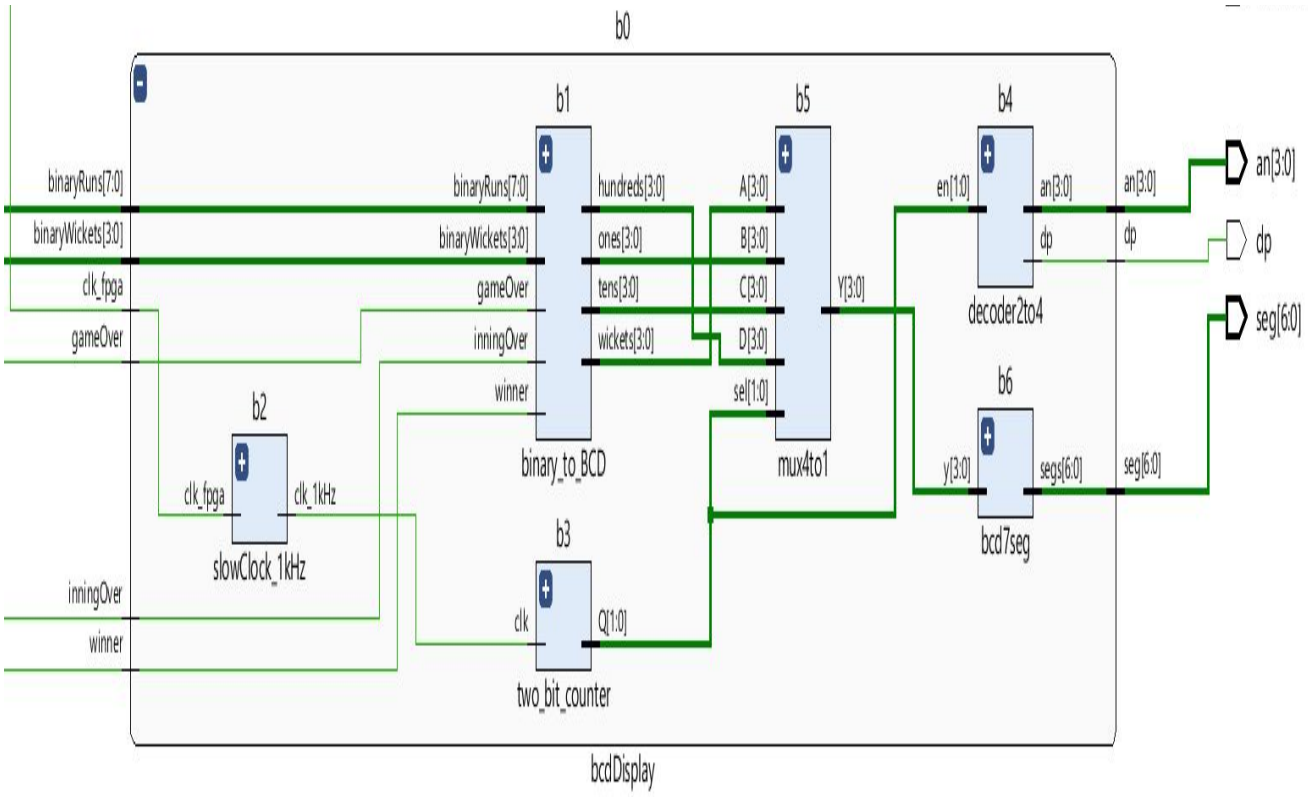

Figure 17: Verilog Schematic of Seven Segment Display

The *bcdDisplay.v* module shown in figure A-12 along with its submodules (figures A-13 to A-18) convert the binary runs and wickets from *cricketGame.v* (figure A- 4) to BCD (Binary Coded Decimal) and display them on the seven segment leds on the Basys 3 board [5]. The block diagram and verilog schematic of the bcd display system are shown in figures 16 and 17 above.

4. Conclusion and Results

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 Vivado Design Studio and photographs of the Basys 3 board while the game is in process.

4.1 Design Simulation

Figure 18 shows that on the rising edge of the delivery at 2510 ns, the lfsr output is 15 resulting in another wicket being taken. On the next free clock cycle at 2525 ns, the number of wickets goes from 9 to 10. Once the wickets are at 10 or the ball count reaches 120, 'inningOver' value goes high as seen at 2535 ns. Despite new delivery attempts, the runs, wickets, and balls of Team 1 are no longer allowed to be incremented in this state (inningOver = 1) [6].

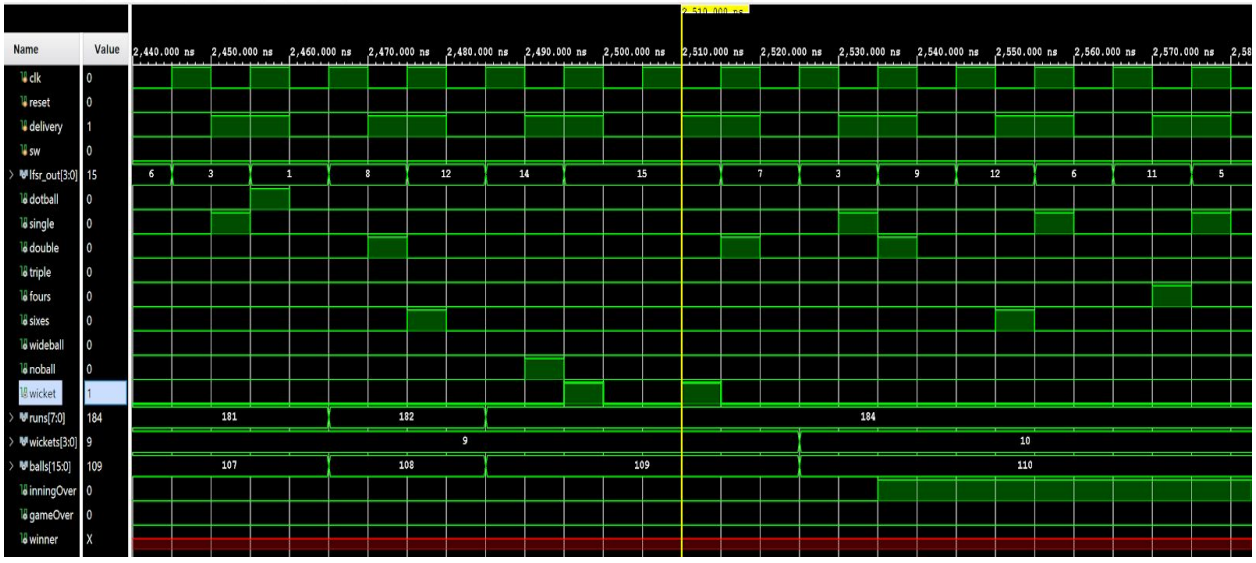

Figure 18: Timing Waveform - End of Team 1's Inning

Figure 19 below shows what happens when the teams are switched. The second team's runs, wickets and balls all start from zero.

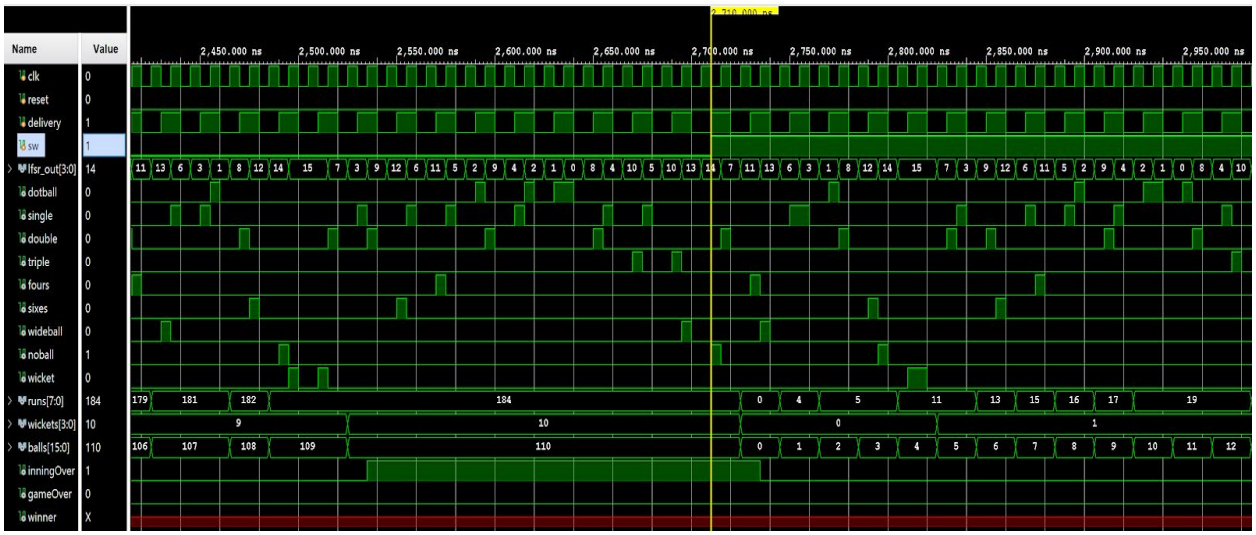

Figure 19: Timing Waveform - Switching Teams

Figure 20 below shows the second team completing their inning, which ends the game. Team 2 (sw = 1) delivers 120 balls and thus their inning is over (updated on next clock cycle in simulation). The game ends immediately because both teams' innings are over (team 1 took 10 wickets and team 2 delivered 120 balls). Gameover goes to 1 and since Team 2 had 203 runs while Team 1 had 184 runs, Team 2 is declared the winner('winner' changes from don't care(x) to 1, instead of going to 0) [6].

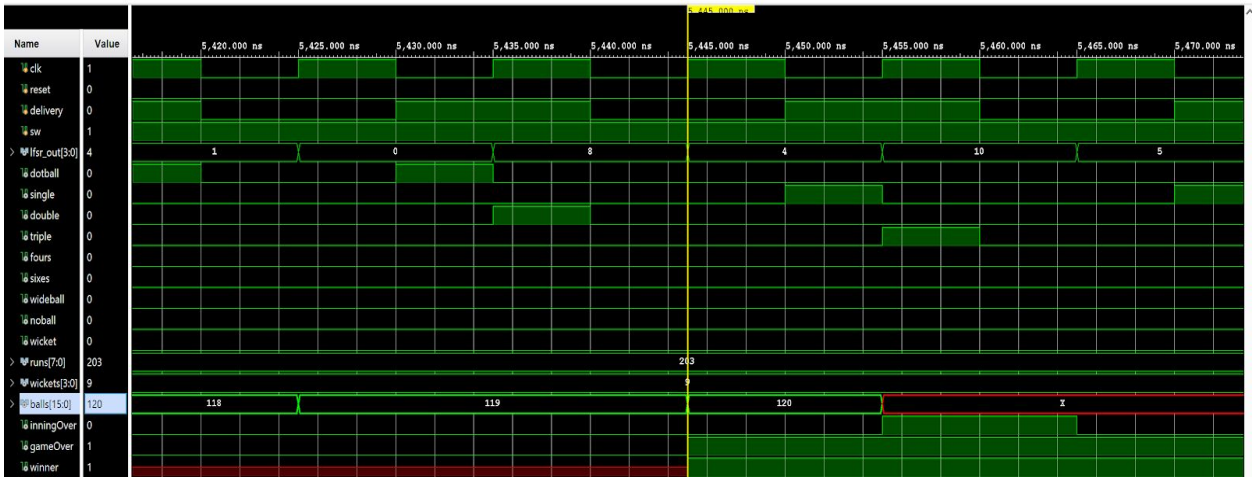

Figure 20: Timing Waveform - End of Team2's Inning and the End of the Game

4.2 Design Utilization

Figure 21 shows a summary of the post-implementation design utilization on a Basys 3 board.

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	177	20800	0.85
FF	155	41600	0.37
IO	32	106	30.19
BUFG	2	32	6.25

Figure 21: Design Utilization

4.3 Final Scoreboard Snapshots

Snapshots of the game in action are shown below:

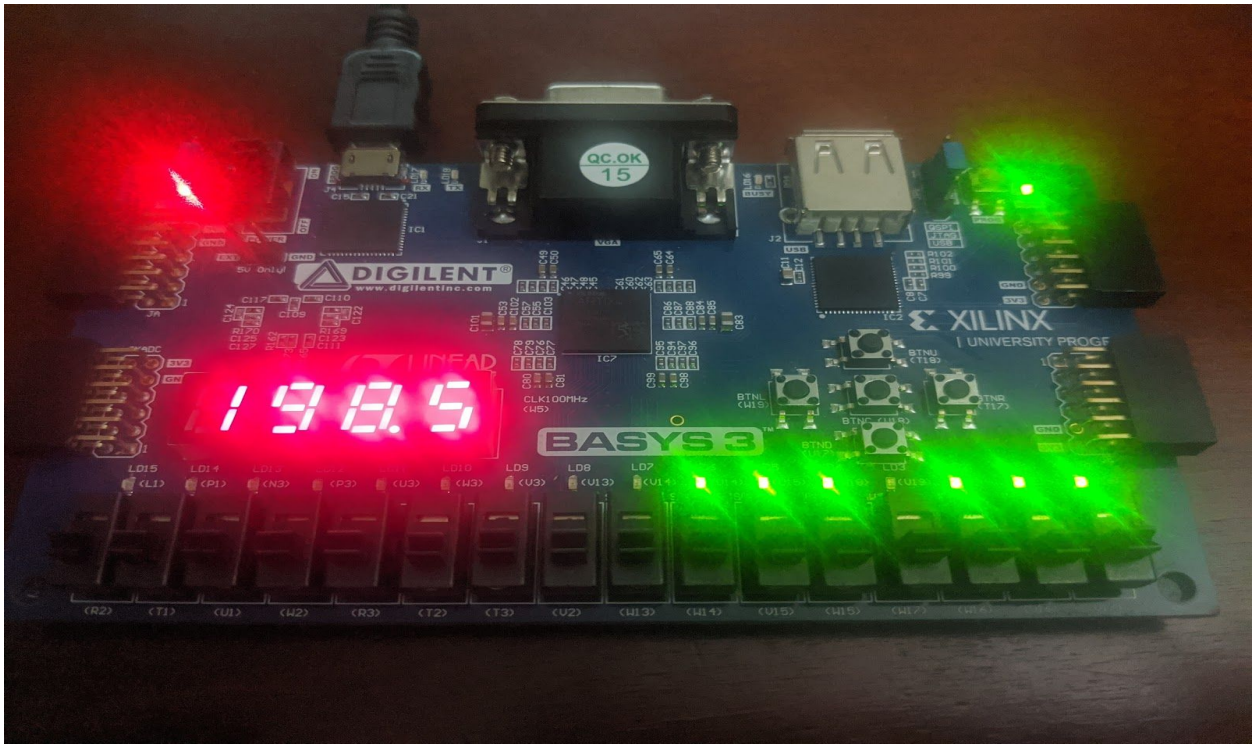

Figure 22: Team 1 with 198 runs, 5 Wickets and 119 Balls (leds = 7b1110111)

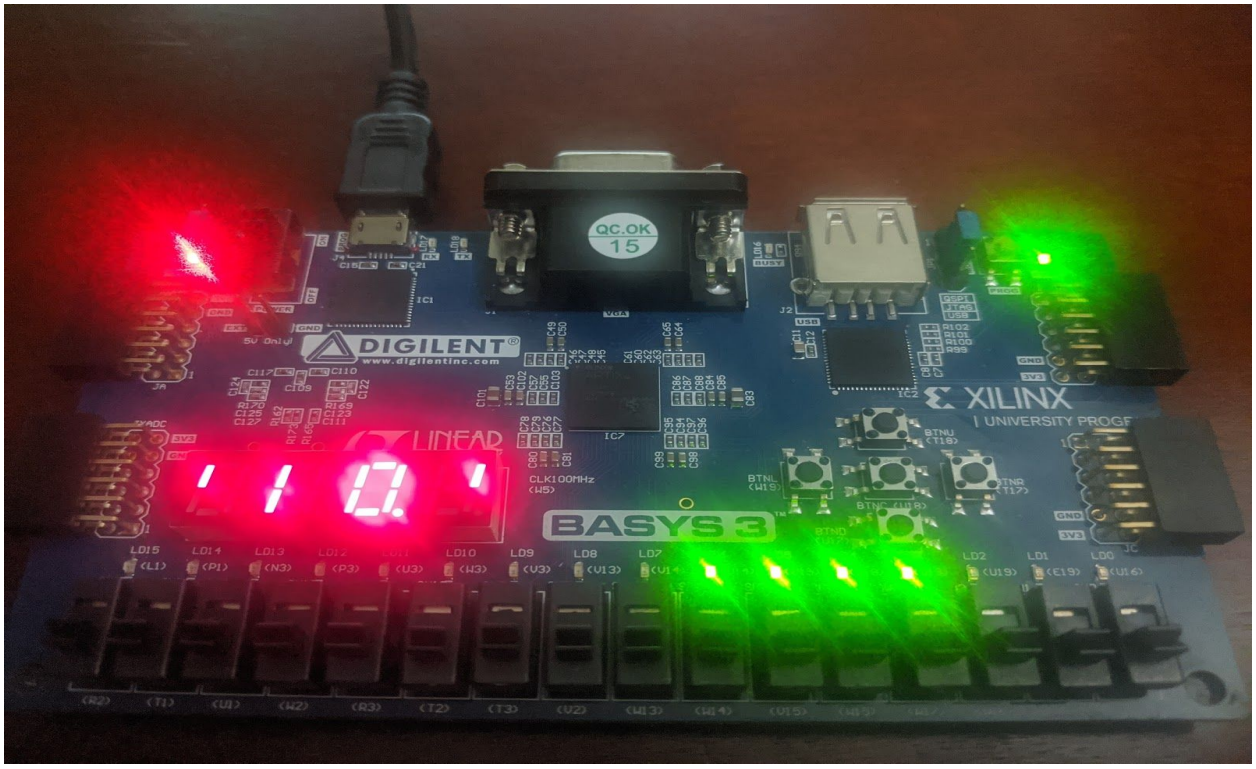

Figure 23: 'IO' Indicating Team 1's 'Inning Over' (notice ball count = 120)

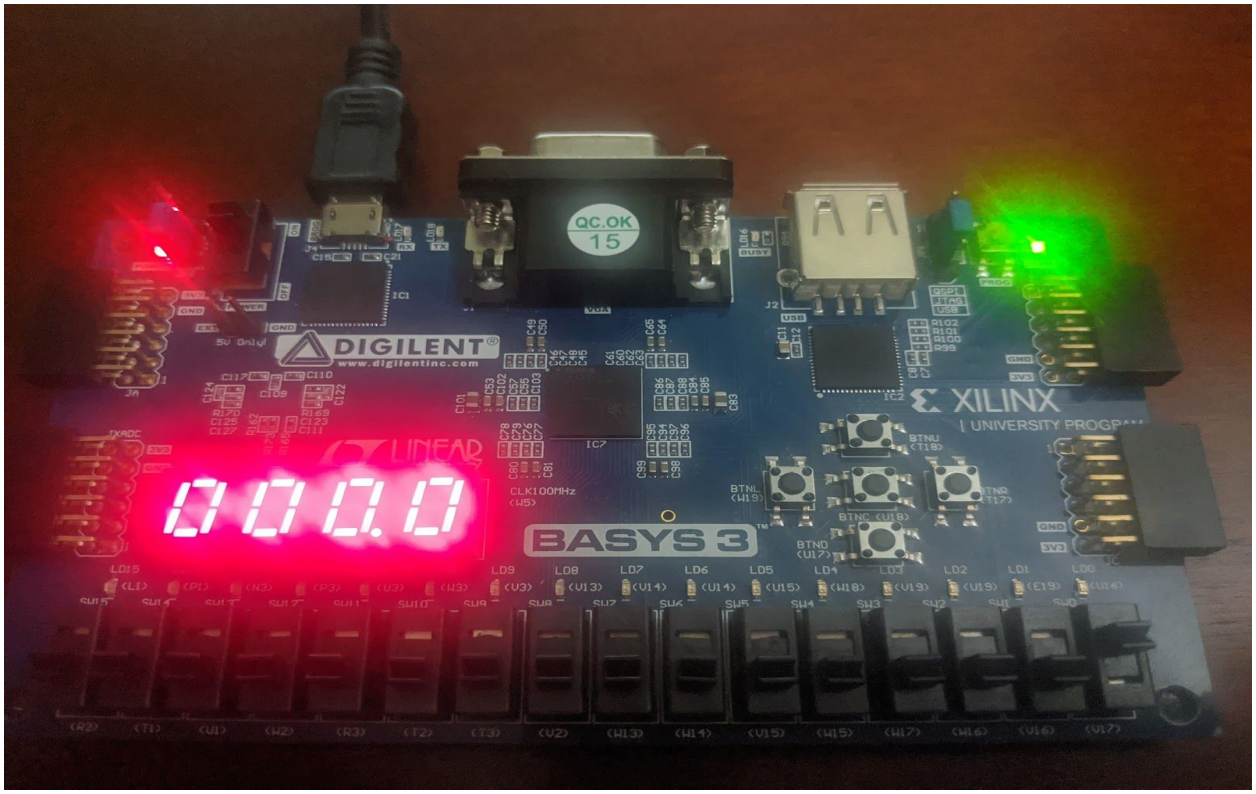

Figure 24: Switch to Team 2 (sw0 slider switch is pushed from 0 to 1)

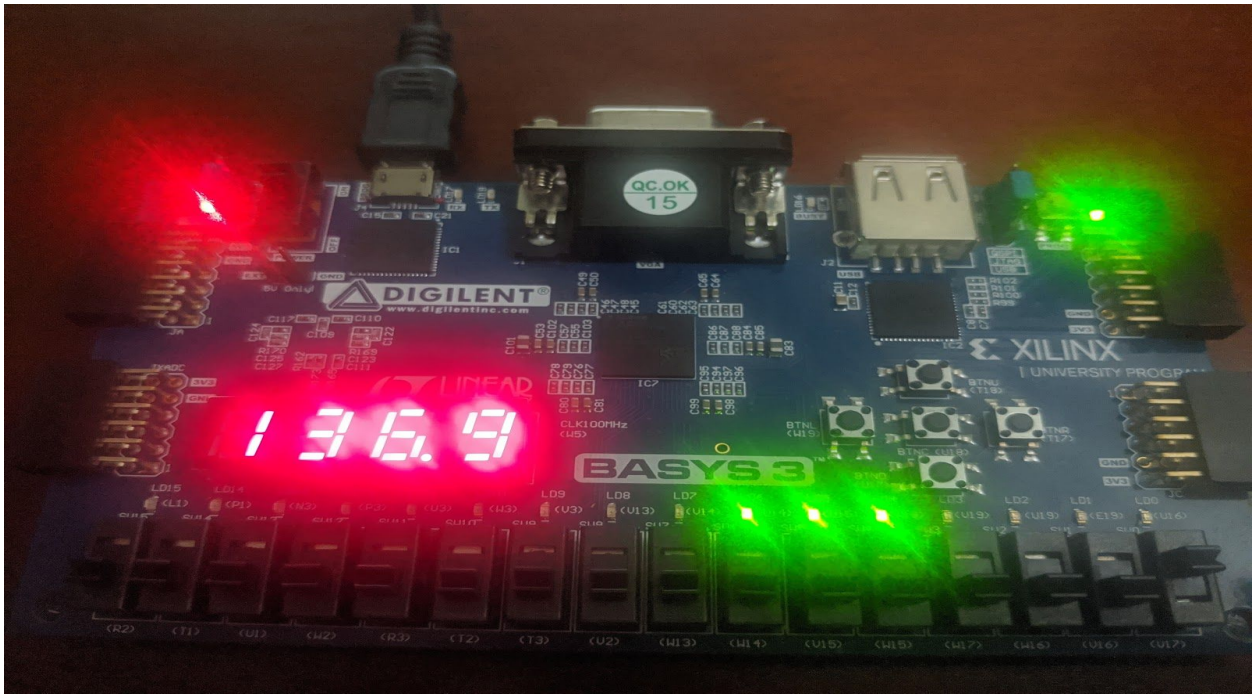

Figure 25: Team 2 with 136 Runs, 9 Wickets and 112 Balls

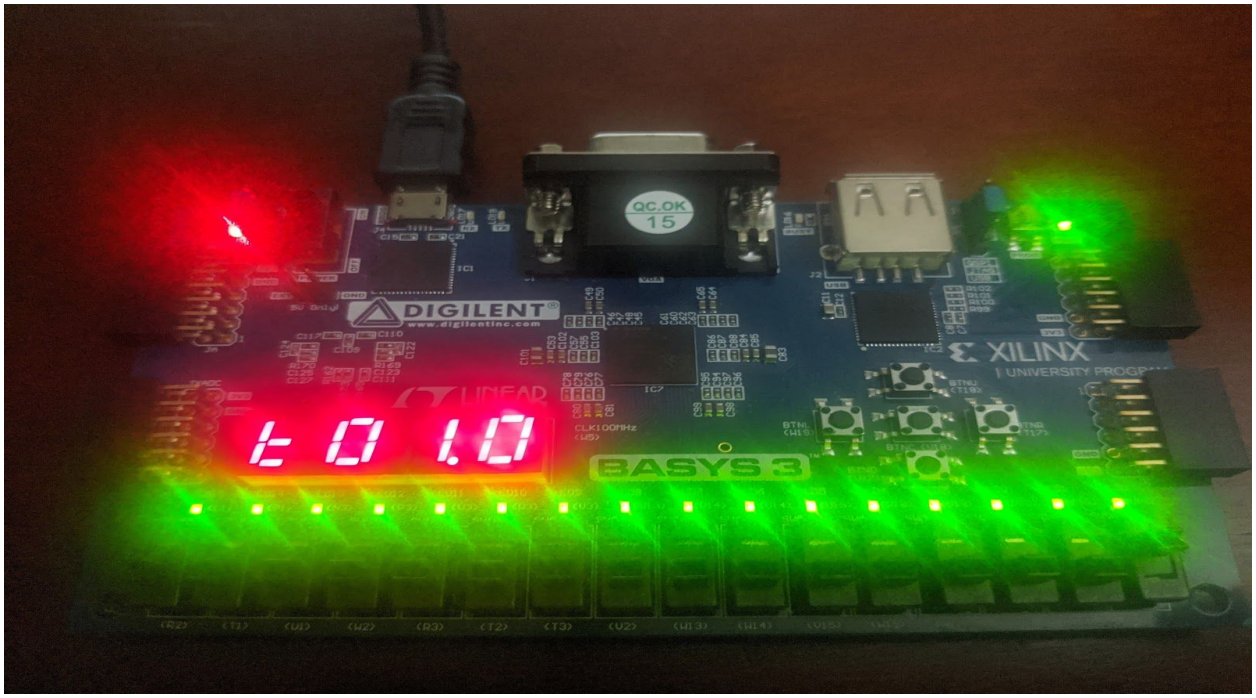

Figure 26: Team 1 Wins the Game

5. References

[1] Chakraborty Rajat Subhra, and Palchaudhuri Ayan. "ARCHITECTURE AND DESIGN AUTOMATION OF HIGH PERFORMANCE LARGE ADDERS AND COUNTERS ON FPGA THROUGH CONSTRAINED PLACEMENT." 18 May 2017: n. pag. Print.

[2] Fokhrulislam, Md., M. A. Mohd. Ali, and Burhanuddin Yeop Majlis. "FPGA Implementation of an LFSR Based Pseudorandom Pattern Generator for MEMS Testing." *International Journal of Computer Applications* 75.11 (2013): 30–34. Web.

[3] Digilent, "Basys3 Reference Manual," Basys3 Rev.c Datasheet, August 2014.

[4] Girard III Hilton W, and Christenson Keith A. "Debounce strategy for validating switch actuation." 15 Feb. 2012: n. pag. Print.

[5] Chu, Pong P. *FPGA Prototyping by Verilog Examples Xilinx Spartan -3 Version* . Hoboken, N.J: J. Wiley & Sons. Print.

[6] Salemi, Ray. *FPGA Simulation : A Complete Step-by-Step Guide* . S.l: Ray Salemi. Print.

[7] Orr, Graeme. "Test Cricket Versus One-Day Cricket: Regulations and Rhythms." *Alternative Law Journal* 31.1 (2006): n. pag. Print.

[8] Ghelani, Harsh H. et al. "FPGA Implementation of Configurable Linear Feedback Shift Register Using Verilog." *International Journal of Computer Sciences and Engineering* 6.4 (2018): 143–146. Web.

[9] Cerda, J. C et al. "An Efficient FPGA Random Number Generator Using LFSRs and Cellular Automata." *2012 IEEE 55th International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE, 2012. 912–915. Web.

[10] ESPN. "Records: Twenty20 Internationals: Team Records: Highest Innings Totals: ESPNcricinfo.com." *Cricinfo,* stats.espncricinfo.com/ci/content/records/283218.html. Web.

6. Appendix

```
module Cricket(
    input clk_fpga,     // 100MHz Basys3 master clock
    input reset,        // center push button used as reset
    input btnU,         // up push button
    input sw,           // switch between Team 1 and Team 2
    output dp,          // use the third decimal point as a separator
    output [6:0] seg,   // goes to anode ports defined in  constraints
    output [3:0] an,    // goes to seven seg cathodes defined in constraints
    output [15:0] 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 IO(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);

    /* converts and displays the runs on the three leftmost digits and the wickets on the last fourth digit,
       separated by a decimal point. Eg; 199.8 (199 runs , and 8 wickets)
       displays the match winner as t01.0 or t02.0 when the game is over */
    bcdDisplay b0(clk_fpga, binaryRuns, binaryWickets, inningOver, gameOver, winner, an, dp, seg);

endmodule
```

Figure A-1: *Cricket.v* (top module)

```
module debounce(
    input clk_10Hz,    // clock signal
    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 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
    */

    D_FF d1(clk_10Hz, button, Q1); // 1st 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
```

Figure A-2: *debounce.v*

```
module D_FF(
    input clk,
    input D,
    output reg Q
    );

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

endmodule
```

Figure A-3: *D_FF.v*

```
module cricketGame(
    input clk_fpga,
    input reset,
    input delivery,
    input teamSwitch,
    output [7:0] runs,
    output [3:0] wickets,
    output [15:0] leds,
    output inningOver,
    output gameOver,
    output winner
    );

    wire [11:0] team1Data; // stores and updates team1'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 team1'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

    // pseudo random number generator using a linear feedback shift register
    lfsr g1(clk_fpga, reset, lfsr_out);
    // for a more detailed simulation use:
   // lfsr g1(clk_fpga, reset, lfsr_out, delivery, dotball,single, double, triple, fours, sixes, wideball, noball, wicket);

    // assign score and wickets based on pseudo random number generated by lfsr
    score_and_wickets g2(clk_fpga, 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_fpga, 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_fpga, reset, teamSwitch, delivery, lfsr_out, inningOver, gameOver, leds, team1Balls, team2Balls );

endmodule
```

Figure A-4: *cricketGame.v*

```
module lfsr(
    input clk_fpga,
    input reset,
    output [3:0] q
    // for a more detailed simulation uncomment the following:
    // input delivery,
    // output dotball,single, double, triple, fours, sixes, wideball,noball,wicket,
    );

    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 q = shift[3:0]; // output of LFSR

    /*
    for a more detailed simulation, uncomment the following:
    assign dotball = delivery & ((shift[3:0] == 0) | (shift[3:0] == 1) | (shift[3:0] == 2) |);
    assign single = delivery & ((shift[3:0] == 3) |( shift[3:0] == 4) | shift[3:0] == 5) | shift[3:0] == 6));
    assign double = delivery & ((shift[3:0] == 7) | ( shift[3:0] == 8) | shift[3:0] == 9));
    assign triple = delivery & (shift[3:0] == 10);
    assign fours = delivery &  (shift[3:0] == 11);
    assign sixes = delivery & (shift[3:0] == 12);
    assign wideball = delivery & (shift[3:0] == 13);
    assign noball = delivery & (shift[3:0] == 14);
    assign wicket = delivery &  (shift[3:0] == 15);
    */

endmodule
```

Figure A-5: *lfsr.v*

```
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 team1's scores and runs when switch is 0
    output reg [11:0] team2Data // stores and updates team2's scores and runs when switch is 1
    );

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

    // update scores after each delivery(bowl) based on cricket rules
    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)
```

Figure A-6: ***score_and_wickets.v***

```
        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;
                    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 press for delivery
            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
```

Figure A-7: ***score_and_wickets.v cont'd***

```
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 [15:0] 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
```

Figure A-8: ***score_comparator.v***

```
module led_controller(
    input clk_fpga,
    input reset,
    input teamSwitch,
    input delivery,
    input [3:0] lfsr_out,
    input inningOver,
    input gameOver,
    output reg [15:0] leds,
    output reg [6:0] team1Balls,
    output reg [6:0] team2Balls
    );

    wire [15:0] 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  team1'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) // pseudorandom number from linear feedback shift register
                    13,14: team2Balls <= team2Balls ; //wide ball and no ball
                    default: team2Balls <= team2Balls + 1;   //ones,twos,threes,fours,sixes,dotballs
                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
```

Figure A-9: *led_controller.v*

```
module scroll_Leds(
    input clk_fpga,
    output reg [15:0] led
    );

    wire clk_10Hz; // 10Hz signal from slow clock

    //shift an led every rising edge
    always @ (posedge clk_10Hz) begin
        if (led == 16'hffff)
            led <= 16'hfffe; // reset to 16'b1111_1111_1111_1110
        else
            led <= {led[14:0], 1'b1}; //shift the rightmost unlit led in 16'hfffe to the left
    end

    slowClock_10Hz c0(clk_fpga, clk_10Hz);

endmodule
```

Figure A-10: *scroll_Leds.v*

```
module slowClock_10Hz(
    input clk_fpga, // 100MHz master clock
    output reg clk_10Hz //10Hz output clock
    );

    localparam clkdiv = 5_000_000 - 1; // clock divider
    reg [22:0] period_count = 0; // counts up to the clock divider

    // divide the 100MHz clock to 100Hz
    always@(posedge clk_fpga) begin
        if (period_count == clkdiv)
            begin
            period_count <= 0;
            clk_10Hz <= ~clk_10Hz;
            end
        else
            begin
            period_count <= period_count + 1'b1;
            clk_10Hz <= clk_10Hz;
            end
    end

endmodule
```

Figure A-11: *slowClock_10Hz.v*

```
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 [3:0] an,             // drives the anodes
    output dp,                   // decimal point on display
    output [6:0] seg             // drives the seven-segment cathodes
    );

    wire clk_1kHz;     //1kHz clock signal from slowClock_1kHz module
    wire [3:0] mux_out;  // sends output of mux to display a specific seven segment parameter
    wire [1:0] counter_out;  //sends output of the 2-bit counter to the mux and decoder
    wire [3:0] wickets,ones,tens,hundreds; // sends bcd output from converter to mux

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

    // generate a 1kHz clock from the 100MHz master clock
    slowClock_1kHz b2(clk_fpga, clk_1kHz);

    // two bit counter at 1kHz for refreshing anodes
    two_bit_counter b3(clk_1kHz, counter_out);

    //turn on one anode and turn off the other three on each 100 Hz tick of the 2-bit counter
    decoder2to4  b4(counter_out, dp, an);

    //select a bcd digit to display on the anode that is turned on
    mux4to1 b5(counter_out,wickets,ones,tens,hundreds,mux_out);

    /* display the digit selected by the mux by using an equivalent 7-bit constant
       to drive the seven segment cathodes
    */
    bcd7seg b6(mux_out,seg);

endmodule
```

Figure A-12: *bcdDisplay.v*

```
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, hundreds
    );

    reg [7:0] data;  //temporarily store binaryRuns for calculations

    always@(binaryRuns,binaryWickets,inningOver, gameOver,winner) begin
        if(~gameOver)   // still playing
            begin
            if(inningOver)
                begin
                hundreds <= 4'b1100; // ', see bcd7seg module in bcdDisplay
                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   //game is over : lock the winner on the screen till the reset button is hit
            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
```

Figure A-13: *binary_to_BCD.v*

```
module slowClock_1kHz(
    input clk_fpga, // 100MHz master clock
    output reg clk_1kHz //1kHz output clock
    );

    localparam clkdiv = 50_000 - 1; // clock divider
    reg [15:0] period_count = 0; // counts up to the clock divider

    // divide the 100MHz clock to 1kHz
    always@(posedge clk_fpga)   begin
        if (period_count == clkdiv)
            begin
            period_count <= 0;
            clk_1kHz <= ~clk_1kHz;
            end
        else
            begin
            period_count <= period_count + 1'b1;
            clk_1kHz <= clk_1kHz;
            end
        end

endmodule
```

Figure A-14: *slowClock_1kHz.v*

```
module two_bit_counter(
    input clk,  // input clock
    output reg [1:0] Q // 2-bit register
    );

    /* on each positive edge of the input clock,
       count up from 0 to 3, and wrap around from 3 back to 0,
       with the 2 bit sized register Q
    */
    always @(posedge clk) begin
        Q <= Q + 1'b1;
    end

endmodule
```

Figure A-15: *two_bit_counter.v*

```
module decoder2to4(
    input [1:0] en,
    output reg dp,
    output reg [3:0] an
    );

    // cycle through the four anodes using en(output of 2 bit counter)
    always@(en) begin
        case (en)
            0:  begin
                an = 4'b1110;
                dp = 1'b1;
                end
            1:  begin
                an = 4'b1101;
                dp = 1'b0; //turn on the active low decimal point after anode AN1
                end
            2:  begin
                an = 4'b1011;
                dp = 1'b1;
                end
            3:  begin
                an = 4'b0111;
                dp = 1'b1;
                end
        endcase
    end

endmodule
```

Figure A-16: *decoder2to4.v*

```
module mux4to1(
    input [1:0] sel,
    input [3:0] A,B,C,D,
    output [3:0] Y
    );

    //selector switches decide what data at the input datalines go through to the output (Y).
    assign Y = (sel==0)?A : (sel==1)?B : (sel==2)?C : D;

endmodule
```

Figure A-17: *mux4to1.v*

```
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; //I using I to help display 'IO' instead of D
            14: segs = 7'b111_1101; //' using a right apostrophe instead of E
            15: segs = 7'b000_0111; //t using t instead of f to show the winning team as t01 or t02
        endcase
   end

endmodule
```

Figure A-18: *bcd7seg.v*