Advanced Logic Design Laboratory

CPE 324-02

Final Project - Reaction Games

By: Nolan Anderson and David Thornton

Demonstration Due: 22 April 2021

Project Report Due: 25 April 2021

1. Introduction

1.1 Purpose

This report will discuss two modes of a reaction timer game running on the DE10-Lite board (10M50DAF484C7G). The purpose of this project is to show proficiency in the Verilog programming and Intel® Quartus® Prime. There will be two games in this project. Game one can be played by setting switch nine to off, and game two can be played by setting switch nine to on. There will be a high score value stored for each game.

2. Project Description

2.1 Theory Topics

2.1.1 Game One

Game one is a simple game to measure a player's reaction time. The player presses key zero, and after a short, random, delay, all the LEDs will light up, and the timer will start. The seven-segment display will show the timer in seconds. The player must press key zero again as quickly as possible. Once the player presses key zero a second time, the counter will stop, and if applicable, set a new high score. After one iteration of the game, the player can play again by simply pressing key zero. This clears the current timer, turns off the LEDs, and starts a new instance. Key one has no functionality in game one.

To play this game, the player needs to set switch nine to the off position. The positions of switches zero to seven do not matter. The player can technically play the game with switch eight, the high score switch, set to on, but this is not recommended. Setting switch eight to on and playing the game will not show your reaction time unless it sets a new high score.

2.1.2 Game Two

Game two is a variation of game one. The player presses key one, and after a short delay, the game randomly selects two switches in the range of switch zero to switch seven. The game then turns on the LEDs that are above the selected switches. This tells the player which switches to flip. The goal of game two is to flip the switches as quickly as possible.

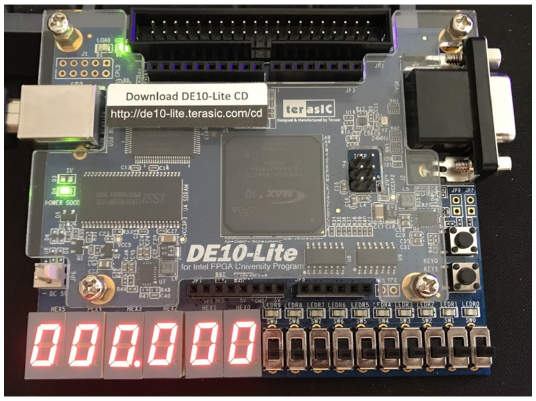
To play this game, the player needs to set switch nine to the on position. The positions of switches zero to seven do not matter, but it is highly recommended to set them off. This is because the game will stop the timer when only the two randomly selected switches are set to on. The player can technically play the game with switch eight, the high score switch, set to on, but this is not recommended. Setting switch eight to on and playing the game will not show your reaction time unless it sets a new high score.

2.1.3 High Scores

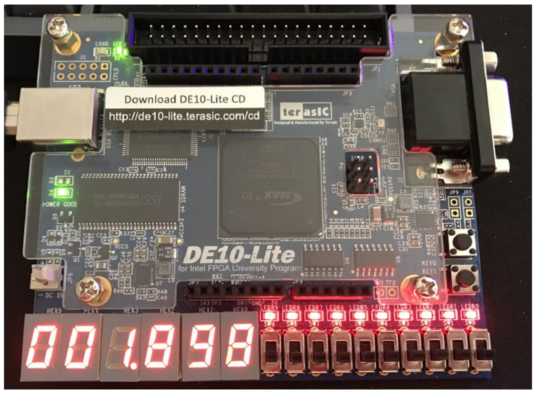
Each game stores the fastest reaction time as a high score. To view the high score for game one, set switch nine to the off position, and switch eight to the on position. The positions of switches zero to seven do not matter. To view the high score for game two, set switch nine to the on position, and switch eight to the on position. The positions of switches zero to seven do not matter.

3. Results

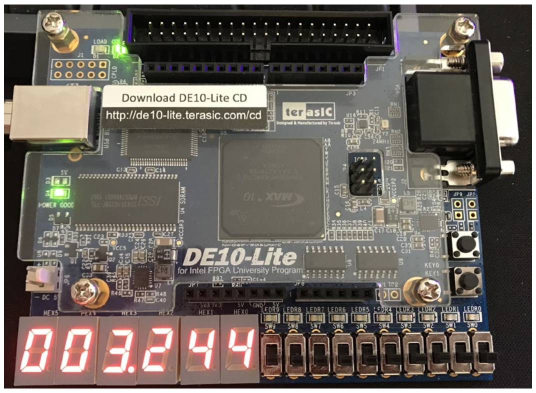
3.1 Observations



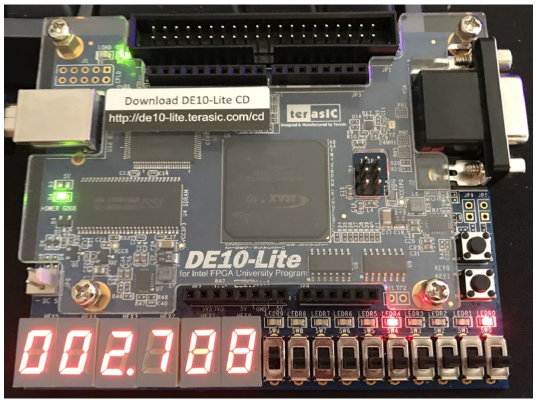
(Left) Figure 3.1: Initial Startup



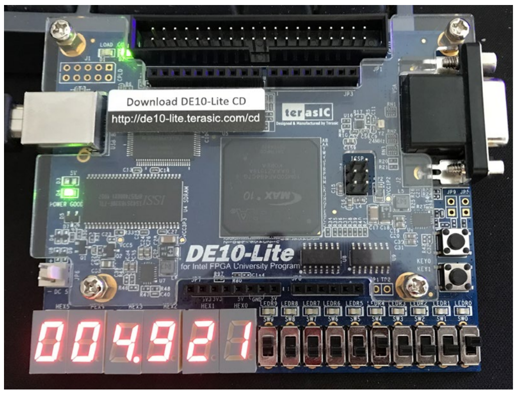
(Right) Figure 3.2: Game 1 Running

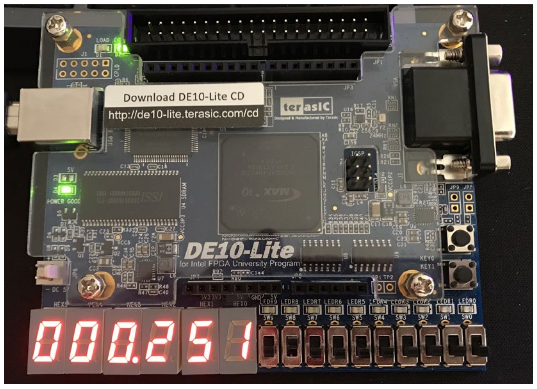


(Left) Figure 3.3 Game 1 Output



(Right) Figure 3.4 Game 2 running





(Left)Figure 3.5 Game 2 Output (Right) Figure 3.6 High Score

4. Conclusion

4.1 Results and Lessons Learned

Overall, this assignment was a good conclusion to the semester overall. It implemented many different components from the different labs. I genuinely feel as if I have made significant progress in my ability to code in verilog and implement code in a specific way. I have thoroughly enjoyed this lab and what it has taught me.

4.2 Demonstration

Nolan and David performed the demonstration live via Zoom on April 22, 2021, at 11:45 am for teaching assistant Preeti Kumari.
Here is also a short demo video of the code running.

https://drive.google.com/file/d/1jI2Aa4zontJaLdGdtXRc5AVfmM1crPjJ/view?usp=sharing

5. Appendix

THE CODE CAN BE FOUND AT THIS LINK:
https://drive.google.com/file/d/1TfNJLEmGNMX7jf_OHsap7Nm3ePXTTCzT/view?usp=sharing

Appendix 1: FinalProject.v

```
// ***********************************************
// CPE 324 Final Project
// Due: 04/25/21
// Authors: Nolan Anderson, David Thornton
// File: FinalProject.v
// Project: Reaction Timer Game
// ***********************************************

module FinalProject(key,sw,clock50M,HEX,d,led);
	input [1:0]key;
	input [9:0]sw;
	input clock50M;
	output [41:0]HEX;
	output d;
	output [9:0]led;

	wire clock1k, clock100, clock10, clock1, clock01, clock001,
clockn, A1, A2, A, b;
	wire [23:0]hs1;
	wire [23:0]hs2;
	wire [9:0]LED1;
	wire [9:0]LED2;
	wire [23:0]hs;
	wire [23:0]o;
	wire [23:0]C;
	wire R3, R4;
	reg R1, R2, R;

	clk_divisor U0(clock50M,clock1k);
	Game1
T1(~sw[9],key[0],o[23:0],clock50M,clock1k,hs1[23:0],A1,b,LED1[9:0]);
	Game2
T2(sw[9],~key[1],o[23:0],sw[7:0],clock50M,clock1k,hs2[23:0],A2,LED2[9
:0]);
	StateMachineR S2(sw[9],clock1k,R3);
	StateMachineR S3(~sw[9],clock1k,R4);
	CountTimer C0(R,A,clock1k,clock100,o[3:0]);
	CountTimer C1(R,A,clock100,clock10,o[7:4]);
	CountTimer C2(R,A,clock10,clock1,o[11:8]);
```

```
      CountTimer C3(R,A,clock1,clock01,o[15:12]);
      CountTimer C4(R,A,clock01,clock001,o[19:16]);
      CountTimer C5(R,A,clock001,clockn,o[23:20]);
      MUX M0(o[3:0],hs[3:0],sw[8],C[3:0]);
      MUX M1(o[7:4],hs[7:4],sw[8],C[7:4]);
      MUX M2(o[11:8],hs[11:8],sw[8],C[11:8]);
      MUX M3(o[15:12],hs[15:12],sw[8],C[15:12]);
      MUX M4(o[19:16],hs[19:16],sw[8],C[19:16]);
      MUX M5(o[23:20],hs[23:20],sw[8],C[23:20]);
      MUX24 M6(hs1[23:0],hs2[23:0],sw[9],hs[23:0]);
      HEXDecoder D0(C[3:0],HEX[6:0]);       // Hex 0
      HEXDecoder D1(C[7:4],HEX[13:7]); // Hex 1
      HEXDecoder D2(C[11:8],HEX[20:14]);    // Hex 2
      HEXDecoder D3(C[15:12],HEX[27:21]);   // Hex 3
      HEXDecoder D4(C[19:16],HEX[34:28]);   // Hex 4
      HEXDecoder D5(C[23:20],HEX[41:35]);   // Hex 5

      assign A = A1 | A2;
      assign led[9] = LED1[9] | LED2[9];
      assign led[8] = LED1[8] | LED2[8];
      assign led[7] = LED1[7] | LED2[7];
      assign led[6] = LED1[6] | LED2[6];
      assign led[5] = LED1[5] | LED2[5];
      assign led[4] = LED1[4] | LED2[4];
      assign led[3] = LED1[3] | LED2[3];
      assign led[2] = LED1[2] | LED2[2];
      assign led[1] = LED1[1] | LED2[1];
      assign led[0] = LED1[0] | LED2[0];

      always@(posedge clock50M) begin
            R1 = ~b & ~key[0] & ~sw[9];
            R2 = ~A2 & ~key[1] & sw[9];
            R = R1 | R2 | R3 | R4;
      end

      assign d = 0;

endmodule
```

```verilog
module Game1(on, w, C, clock50M, clock1k, hs1, A1, b, LED1);
    input on, w, clock50M, clock1k;
    input [23:0]C;
    output reg b;
    output reg A1;
    output reg [9:0] LED1;
    output [23:0]hs1;

    wire A;
    wire [11:0]r;

    LFSR12 L1(clock50M,r[11:0]);
    DownCountTimer DC1(r[11:0],~w,clock1k,A);
    HighScore H1(b,C[23:0],hs1[23:0]);

    always@(posedge w, negedge on) begin
        if (~on)
            b <= 1'b0;
        else
            b <= b ^ 1'b1;
    end

    always@(posedge clock50M) begin
        if (on) begin
            A1 = b & A;
            LED1[9] = b & A;
            LED1[8] = b & A;
            LED1[7] = b & A;
            LED1[6] = b & A;
            LED1[5] = b & A;
            LED1[4] = b & A;
            LED1[3] = b & A;
            LED1[2] = b & A;
            LED1[1] = b & A;
            LED1[0] = b & A;
        end
        else begin
            A1 = 1'b0;
            LED1[9:0] = 10'b0000000000;
            end
        end

endmodule

module Game2(on, w, C, sw, clock50M, clock1k, hs2, A2, LED2);
        input on, w, clock50M, clock1k;
        input [23:0]C;
        input [7:0]sw;
        output [23:0]hs2;
        output [9:0]LED2;
        output reg A2;

        wire match, Z, A;
        wire [7:0]c;
        reg W, Match;

        DownCountTimer DC2(12'b0010111011100,W,clock1k,A);
        Compare CMP1(clock50M,sw[7:0],LED2[7:0],match);
        StateMachine S1(W,clock50M,Match,Z);
        ShiftReg R1(clock50M,c[3:0]);
        ShiftReg R2(clock1k,c[7:4]);
        LED_Decoder LD1(c[3:0],c[7:4],A2,clock50M,LED2[9:0]);
        HighScore H2(A2,C[23:0],hs2[23:0]);

        always@(posedge clock50M) begin
            if(on) begin
                W <= w;
                Match <= match;
                if(A) begin
                    if(Z)
                        A2 = 1'b1;
                    else
                        A2 = 1'b0;
                end
                else
                    A2 = 1'b0;
            end
            else begin
                A2 = 1'b0;
                W <= 1'b0;
                Match <= 1'b1;
            end
        end

endmodule

module clk_divisor(clock50M, clock1k);
        input clock50M;
        output reg clock1k;

        reg [14:0]count;

        always@(posedge clock50M) begin
            count <= count + 1'b1;
            if (count == 25000) begin
                clock1k <= clock1k ^ 1'b1;
                count <= 1'b0;
            end
        end

endmodule

module Compare(clock50M, sw, LED2,Match);
        input clock50M;
        input [7:0]sw;
        input [7:0]LED2;
        output reg Match;
        always@(posedge clock50M) begin
            if (LED2 == 8'b00000000)
                Match = 1'b0;
            else begin
                if (sw[7:0] == LED2[7:0])
                    Match = 1'b1;

                else
                    Match = 1'b0;
            end
        end
endmodule

module CountTimer(reset, A, clock, clocko, o);
        input reset, A, clock;
        output reg clocko;
        output reg [3:0]o;

        always@(posedge clock, posedge reset) begin
            if(reset)
                o <= 4'b0000;

            else if (A) begin
                if (o == 4'b0100)
                    clocko <= 1'b0;
                if (o == 4'b1001) begin
                    clocko <= 1'b1;
                    o <= 1'b0000;
                end
                else
                    o <= o + 1'b1;
            end
            else
                o = o;
        end

endmodule

module DownCountTimer(c, start, clock1k, A);
        input [11:0]c;
        input start, clock1k;
        output reg A;

        reg [11:0]count;

        initial
            A = 0;
        always@(posedge clock1k, posedge start) begin
            if (start) begin
                count[11:0] <= c[11:0];
                A <= 1'b0;
            end

            else begin
                if (count == 12'b000000000000)
                    A <= 1'b1;

                else begin
                    count <= count - 1'b1;
                    A <= 1'b0;
                end
            end
        end

endmodule

module HEXDecoder(c, HEX);
        input [3:0]c;
        output reg [6:0]HEX;

        always@(c)
            case(c)
                4'b0000 : HEX[6:0] = 7'b1000000;
                4'b0001 : HEX[6:0] = 7'b1111001;
                4'b0010 : HEX[6:0] = 7'b0100100;
                4'b0011 : HEX[6:0] = 7'b0110000;
                4'b0100 : HEX[6:0] = 7'b0011001;
                4'b0101 : HEX[6:0] = 7'b0010010;
                4'b0110 : HEX[6:0] = 7'b0000010;
                4'b0111 : HEX[6:0] = 7'b1111000;
                4'b1000 : HEX[6:0] = 7'b0000000;
                4'b1001 : HEX[6:0] = 7'b0010000;
                default : HEX[6:0] = 7'b1000000;
            endcase
endmodule
```

```
module LED_Decoder(c, d, read, clock50M, LED2);
	input [3:0]c;
	input [3:0]d;
	input read, clock50M;
	output reg [9:0]LED2;

	initial
		LED2[9:0] = 10'b0000000000;

	always@(posedge clock50M) begin
		if (read) begin
			if (LED2[7:0] == 8'b00000000) begin
				LED2[3:0] <= c[3:0];
				LED2[7:4] <= d[3:0];
				LED2[9:8] <= 2'b00;
			end

			else
				LED2[9:0] <= LED2[9:0];
		end

		else
			LED2[9:0] <= 10'b0000000000;
	end

endmodule

module MUX(a, b, s, f);
	input [3:0]a;
	input [3:0]b;
	input s;
	output reg [3:0]f;

	always@(a, b, s)
		case(s)
			0 : f[3:0] <= a[3:0];
			1 : f[3:0] <= b[3:0];
		endcase
endmodule
```

```
module MUX24(a, b, s, f);
    input [23:0]a;
    input [23:0]b;
    input s;
    output reg [23:0]f;

    always@(a, b, s)
        case(s)
            0 : f[23:0] <= a[23:0];
            1 : f[23:0] <= b[23:0];
        endcase

endmodule

module HighScore(b, C, hs);
    input b;
    input [23:0]C;
    output reg [23:0]hs;

    always@(negedge b) begin
        if (hs == 0 && C != 0)
            hs = C;

        else begin
            if(C < hs && C != 0)
                hs[23:0] = C[23:0];

            else
                hs[23:0] = hs[23:0];
        end
    end
endmodule

module ShiftReg(clock, c);
    input clock;
    output reg [3:0]c;
    initial
        c = 4'b0001;
    always@(posedge clock) begin
            c[3] <= c[2];
            c[2] <= c[1];
            c[1] <= c[0];
            c[0] <= c[3];
    end
endmodule

module StateMachine(w, clock, Match, Z);
    input w, clock, Match;
    output reg Z;
    reg a, b;
    initial
        Z = 0;
    always@(posedge clock) begin
        a = w & ~Z;
        b = ~Match & Z;
        Z = a | b;
    end
endmodule

module LFSR12(clock50M, r);
    input clock50M;
    output reg [11:0]r;

    initial
        r = 12'b010010101101;

    always@(posedge clock50M) begin
        if (r == 12'b000000000000)
            r[11:0] <= 12'b010010101101;

        else begin
            r[11] <= r[10];
            r[10] <= r[9];
            r[9] <= r[8];
            r[8] <= r[7];
            r[7] <= r[6];
            r[6] <= r[5];
            r[5] <= r[4];
            r[4] <= r[3];
            r[3] <= r[2];
            r[2] <= r[1];
            r[1] <= r[0];
            r[0] <= r[6] ^ r[7];
        end
    end

endmodule

module StateMachineR(sw, clk1k, r);
    input sw, clk1k;
    output reg r;

    reg s[1:0];

    always@(posedge clk1k) begin
        s[1] <= s[0] & sw;
        s[0] <= sw;
        r    <= ~s[1] & s[0];
    end

endmodule
```