**Lab 2: 3TB4**

By:

Lazar Vlacic: (400236569)

Chris George (400312342)

**Code Description**

For Lab 2, our main objective was to program a reaction time-based game on the FPGA board. The board at first needed to flash the seven-segment display and then turn off for some random period. Once that random period has passed, the seven segment displays turn back on. The first player to press their button after the seven-segment display turns on wins the round.

To implement this game, we programmed an LSFR (linear feedback shift register) to generate a very long sequence of numbers for our random period (this sequence is so long that the numbers are essentially generated randomly). We also programmed the logic for our game through a finite state machine, where all the different states of the game were programmed to occur at their correct time as the game is played. Each state depended on various conditions for it to occur as the current state in the game. Our code for implementing the reaction time game is shown below:

module lab2(input CLOCK\_50, input [3:0] KEY, output [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5, output [9:0] LEDR);

parameter [2:0] RESET=3'b000, RESUME=3'b001, BLINKING=3'b010, OFF=3'b011, TIMER\_DISPLAY=3'b100, WINNER\_TIME\_DISPLAY=3'b101,

CHEAT=3'b110, WAIT=3'b111;

reg [2:0] state=RESET, next\_state=RESET;

wire clk\_ms;

wire [19:0] ms, display\_ms;

wire [3:0] w\_ms [5:0]; //wires after hex\_to\_bcd\_converter.v for displayed time

wire [3:0] w\_blink[5:0]; //wires afer blinking

wire [3:0] winner\_ms [5:0]; //wires afer hex\_to\_bcd\_converter.v for winner time

reg [1:0] cheater;

wire [23:0] digits;

wire [3:0] digit [5:0];

assign digit[0] = digits[3:0];

assign digit[1] = digits[7:4];

assign digit[2] = digits[11:8];

assign digit[3] = digits[15:12];

assign digit[4] = digits[19:16];

assign digit[5] = digits[23:20];

wire [13:0] random\_num;

reg [13:0] random\_wait\_time;

wire rnd\_ready;

reg [1:0] hex\_sel = 2'b00; //whether blinking or not

wire [1:0] w\_hex\_sel;

assign w\_hex\_sel = hex\_sel;

reg display\_counter\_start;

wire w\_display\_counter\_start;

reg player1\_win,player2\_win;

reg[4:0] win1,win2; // score for player 1 and 2.

reg [19:0]temp;

reg [19:0] winner\_time;

wire [19:0] w\_winner\_time;

wire conditioned\_key0, conditioned\_key3;

assign w\_winner\_time=winner\_time;

assign w\_display\_counter\_start=display\_counter\_start;

assign LEDR[4:0]=win1;

assign LEDR[9:5]={win2[0],win2[1],win2[2],win2[3],win2[4]} ;

clock\_divider #(.factor(50000)) (.Clock(CLOCK\_50), .Reset\_n(KEY[1]), .Pulse\_ms(clk\_ms));

counter c1(.clk(clk\_ms), .reset\_n(KEY[1]), .resume\_n(KEY[2]), .enable(1), .ms\_count(ms));

counter (.clk(clk\_ms), .reset\_n(KEY[1]), .resume\_n(KEY[2]), .enable(w\_display\_counter\_start), .ms\_count(display\_ms));

blinkHEX(.ms\_clk(clk\_ms),.Reset\_n(KEY[1]),.d0(w\_blink[0]),.d1(w\_blink[1]),.d2(w\_blink[2]),.d3(w\_blink[3]),.d4(w\_blink[4]),.d5(w\_blink[5]));

hex\_to\_bcd\_converter(.clk(CLOCK\_50),.reset(KEY[1]),.hex\_number(display\_ms),.bcd\_digit\_0(w\_ms[0]),.bcd\_digit\_1(w\_ms[1]),.bcd\_digit\_2(w\_ms[2]),.bcd\_digit\_3(w\_ms[3]),.bcd\_digit\_4(w\_ms[4]),.bcd\_digit\_5(w\_ms[5]));

hex\_to\_bcd\_converter(.clk(CLOCK\_50),.reset(KEY[1]),.hex\_number(w\_winner\_time),.bcd\_digit\_0(winner\_ms[0]),.bcd\_digit\_1(winner\_ms[1]),.bcd\_digit\_2(winner\_ms[2]),.bcd\_digit\_3(winner\_ms[3]),.bcd\_digit\_4(winner\_ms[4]),.bcd\_digit\_5(winner\_ms[5]));

mux m0(.a({w\_ms[5],w\_ms[4],w\_ms[3],w\_ms[2],w\_ms[1],w\_ms[0]}),.b({4'b1111,4'b1111,4'b1111,4'b1111,4'b1111,4'b1111}),.c({winner\_ms[5],winner\_ms[4],winner\_ms[3],winner\_ms[2],winner\_ms[1],winner\_ms[0]}),.d({w\_blink[5],w\_blink[4],w\_blink[3],w\_blink[2],w\_blink[1],w\_blink[0]}),.sel(w\_hex\_sel),.q(digits));

seven\_seg\_decoder decoder0(.x(digit[0]), .hex\_LEDs(HEX0));

seven\_seg\_decoder decoder1(.x(digit[1]), .hex\_LEDs(HEX1));

seven\_seg\_decoder decoder2(.x(digit[2]), .hex\_LEDs(HEX2));

seven\_seg\_decoder decoder3(.x(digit[3]), .hex\_LEDs(HEX3));

seven\_seg\_decoder decoder4(.x(digit[4]), .hex\_LEDs(HEX4));

seven\_seg\_decoder decoder5(.x(digit[5]), .hex\_LEDs(HEX5));

random(.clk(CLOCK\_50),.reset\_n(KEY[1]),.resume\_n(KEY[2]),.random(random\_num),.rnd\_ready(rnd\_ready));

always@(posedge CLOCK\_50) begin

if(rnd\_ready) begin

random\_wait\_time = random\_num;

end

end

always @ (posedge CLOCK\_50, negedge KEY[1], negedge KEY[2])

begin

if (!KEY[1]) // reset

begin

state<=RESET;

end

else if (!KEY[2]) //start/resume

begin

state<=RESUME;

end

else

begin

state<=next\_state;

end

end

always @ (posedge CLOCK\_50)

begin

//BLINKING, OFF, P1\_CHEAT, P2\_CHEAT, READY, P1\_WIN, P2\_WIN, WIN, RESET

case (state)

RESET:

begin

display\_counter\_start<=0;

winner\_time<=0;

hex\_sel <= 2'b01;

win1<= 5'b00000;

win2<= 5'b00000;

next\_state<=BLINKING;

player1\_win = 0;

player2\_win=0;

end

RESUME:

begin

//hex\_sel=2'b00;

display\_counter\_start<=0;

winner\_time<=0;

hex\_sel <= 2'b01;

player1\_win = 0;

player2\_win=0;

if (win1 == 5'b11111 || win2 == 5'b11111) begin

win1 = 5'b00000;

win2 = 5'b00000;

end

next\_state<=BLINKING;

end

BLINKING:

begin

hex\_sel<=2'b11;

display\_counter\_start<= 0;

//$monitor("[$monitor] time=%0t a-0x%0h",$time,ms);

if (ms>=3000) //blink for about 3 second

begin

//hex\_sel=2'b01;

next\_state<=OFF;

end

else begin

next\_state<=BLINKING;

end

end

OFF:

begin

hex\_sel<=2'b01;

if (ms>(5000+random\_wait\_time)) begin //(7-5) seconds + random seconds)

next\_state<=TIMER\_DISPLAY;

end else if (!KEY[0] && !KEY[3]) begin

cheater<=2'b00;

next\_state<=CHEAT;

end else if(!KEY[0] && KEY[3]) begin

cheater<=2'b01;

next\_state<=CHEAT;

end else if(!KEY[3] && KEY[0]) begin

cheater<=2'b10;

next\_state<=CHEAT;

end else begin

next\_state<=OFF;

end

end

TIMER\_DISPLAY:

begin

display\_counter\_start<=1;

hex\_sel<=2'b00;

if (!KEY[0] && !KEY[3])

begin

display\_counter\_start<=0;

next\_state<=RESUME;

end else if (!KEY[0] && KEY[3]) begin

display\_counter\_start<=0;

player1\_win <= 1;

next\_state<=WINNER\_TIME\_DISPLAY;

end else if (!KEY[3] && KEY[0]) begin

display\_counter\_start<=0;

player2\_win <= 1;

next\_state<=WINNER\_TIME\_DISPLAY;

end else begin

next\_state<=TIMER\_DISPLAY;

end

end

WINNER\_TIME\_DISPLAY:

begin

winner\_time<=display\_ms;

display\_counter\_start<=0;

hex\_sel<=2'b10;

if (player1\_win) begin

win1<=(win1<<1) | 5'b00001;

player1\_win <= 0;

next\_state<=WAIT;

end else if (player2\_win) begin

win2<=(win2<<1) | 5'b00001;

player2\_win <= 0;

next\_state<=WAIT;

end

end

CHEAT:

begin

if (cheater == 2'b00) begin

winner\_time <= 888888;

hex\_sel<=2'b10;

next\_state<=CHEAT;

end else if (cheater == 2'b01)begin

winner\_time <= 111111;

hex\_sel<=2'b10;

next\_state<=CHEAT;

end else if (cheater == 2'b10) begin

winner\_time <= 222222;

hex\_sel<=2'b10;

next\_state<=CHEAT;

end

end

WAIT:

begin

next\_state<=WAIT;

end

endcase

end

endmodule

**Questions:**

Question 1

Which type of LFSR have you used in your project? How many bits long was your LFSR? What taps have you used for your LFSR? Explain your choices.

For our project, we used a Fibonacci LFSR. The Fibonacci LFSR we used was 14 bits long, with taps at the 14th, 5th, 3rd, and 1st bit. There wasn’t a clear reason for choosing a Fibonacci LSFR over a Galois LSFR. A Galois LSFR runs more efficiently since the Fibonacci LSFR sequentially XORs bits while a Galois LSFR does not. However, our FPGA board runs at a very high frequency, so the higher efficiency of the Galois LSFR is barely noticeable. We used a 14-bit LSFR to get a very long sequence of generated values. The max possible number of numbers generated is 2^n – 1. With 14 bits, that would be 2^14 – 1 = 16 383 numbers. The taps at the 14th, 5th, 3rd, and 1st bit were used as based on the Xilnix Application Note, the taps need to be at bit 14,5,3, and 1 to get the longest possible sequence of values for a 14-bit LSFR.

Question 2

What are the advantages or disadvantages if the LFSR is too short, or too long?

Question 3

A screenshot of the compilation report is given below:

A screenshot of a computer

Description automatically generated

**Figure 1:** Screenshot of the project’s compilation report in Quartus Prime