Arman Khondker
EID: aak2464

Lab 6

1. Report

a)

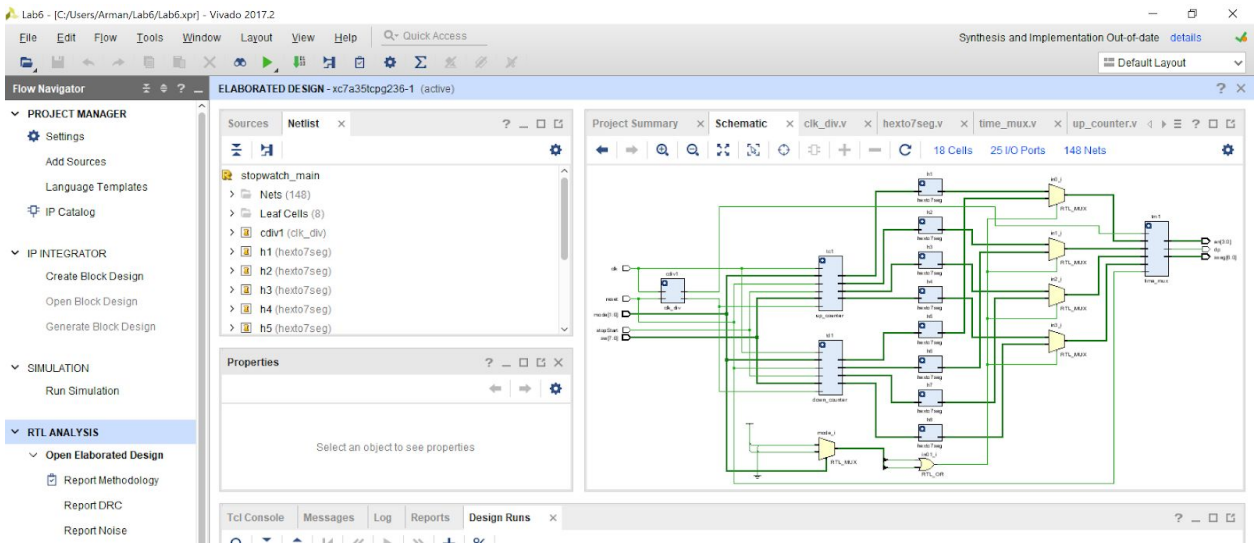
The hardest part of this lab was the initial stage of thinking about how to implement the design. My thought process for the stopwatch design was to make everything as simple as possible and while using as much as i could from past labs. We were given full freedom to implement this project with different designs and use implementations from previous labs. I began this project by deconstructing the four different modes that we had to cover. I knew a mux with two select lines would be needed to implement accessing these different states. I knew that there were going to be four different modes so I started with the first mode. First, I learned from lab four that four numbers can't be displayed at the same time; therefore, I had to make the clock cycle faster than the human eye can detect. I used the clock divider used a 19 bit register to implement a 10 millisecond time resolution would display. Next, I reused code from hexto7seg module to display digits for the segment display. Then I call either one of my up_counter or down_counter functions in the stopwatch main which determines which mode we are on.

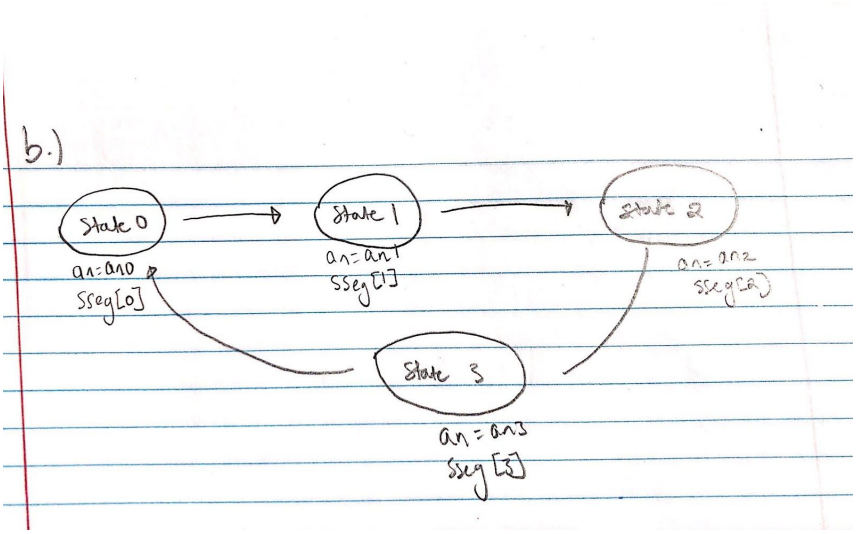
I continuously check the reset button first and if pressed, I reset the segment display depending on to the mode that was selected. Then, I check if any of the segment displays are at 9 and are about to be increased. If they are, then I set the correct displays to 0 and increase the left number to 1 with the carry over bit. If none of the segment displays need to be carried over, I add 1 or subtract 1 to the right most display depending on which mode I am in. Afterwards, I send my updated segment number to my hexto7seg module from lab 4. I select my segments to display based off the mode. Lastly, I call the time mux function that acts as the FSM controller.

Arman Khondker
EID: aak2464

A major problem I faced when trying to implement the design was bouncing on the inputs. Sometimes, when I pressed buttons there was a bouncing effect where it would count as multiple button presses. Additionally, I was concerned with figuring out how everything will function together in the final design. A design choice I made was separating the counter up and count down functionality into two separate modules. . In conclusion, the design of a StopWatch using verilog was extremely challenging and rewarding.

b) Processor Architecture





Arman Khondker
EID: aak2464

c) Verilog Codes and constraints

Stopwatch_main

```
module stopwatch_main(
    input clk,
    input reset,
    input [7:0] sw,
    input [1:0] mode,
    output [3:0] an,
    output [6:0] sseg,
    output dp,
    input stopStart
    );

    localparam [6:0] zero = 7'b0000001;

    wire [6:0] in0, in1, in2, in3;
    wire [6:0] iu0, iu1, iu2, iu3;
    wire [6:0] id0, id1, id2, id3;
    wire [3:0] su0, su1, su2, su3;
    wire [3:0] sd0, sd1, sd2, sd3;
    wire tick;
    wire slow_clk, faster_clk;
```

Arman Khondker
EID: aak2464

```
        clk_div cdiv1 (.clk(clk), .reset(reset), .slow_clk(slow_clk), .tick(tick),
.faster_clk(faster_clk));
        up_counter tc1 (.clk(clk), .reset(reset), .tick(tick), .stopStart(stopStart), .mode(mode),
.sw(sw), .seg0(su0), .seg1(su1), .seg2(su2), .seg3(su3));
        down_counter td1 (.clk(clk), .reset(reset), .tick(tick), .stopStart(stopStart),
.mode(mode), .sw(sw), .seg0(sd0), .seg1(sd1), .seg2(sd2), .seg3(sd3));

        //instantiate hexto7segment decoder
        hexto7seg h1 (.x(su0), .r(id0));
        hexto7seg h2 (.x(su1), .r(id1));
        hexto7seg h3 (.x(su2), .r(id2));
        hexto7seg h4 (.x(su3), .r(id3));

        hexto7seg h5 (.x(sd0), .r(iu0));
        hexto7seg h6 (.x(sd1), .r(iu1));
        hexto7seg h7 (.x(sd2), .r(iu2));
        hexto7seg h8 (.x(sd3), .r(iu3));

        assign in0 = ((mode == 2'b00 || mode == 2'b01)?id0:iu0);
        assign in1 = ((mode == 2'b00 || mode == 2'b01)?id1:iu1);
        assign in2 = ((mode == 2'b00 || mode == 2'b01)?id2:iu2);
        assign in3 = ((mode == 2'b00 || mode == 2'b01)?id3:iu3);
```

Arman Khondker
EID: aak2464

```
time_mux tm1 (.clk(faster_clk), .reset(reset), .in0(in0), .in1(in1), .in2(in2), .in3(in3),
.an(an), .sseg(sseg), .dp(dp));

Endmodule
```

TIME MUX

```
module time_mux(
    input clk,
    input reset,
    input [6:0] in0,
    input [6:0] in1,
    input [6:0] in2,
    input [6:0] in3,
    output reg [6:0] sseg,
    output reg [3:0] an,
    output reg dp
    );

    reg [1:0] state;
    reg [1:0] next_state;

    localparam [6:0] zero = 7'b0000001;
    localparam [6:0] nine = 7'b0001100;
```

Arman Khondker
EID: aak2464

```
always @(*) begin
    case(state)  //State transition
        2'b00: next_state = 2'b01;
        2'b01: next_state = 2'b10;
        2'b10: next_state = 2'b11;
        2'b11: next_state = 2'b00;
    endcase
end

always @(*) begin
    case(state)  //Multiplexer
        2'b00: sseg = in0;
        2'b01: sseg = in1;
        2'b10: sseg = in2;
        2'b11: sseg = in3;
    endcase

    case(state)  //Decoder
        2'b00:
            begin
            an = 4'b1110;
            dp = 1'b1;
            end
        2'b01:
```

```
            begin
            an = 4'b1101;
            dp = 1'b1;
            end
        2'b10:
            begin
            an = 4'b1011;
            dp = 1'b0;
            end
        2'b11:
            begin
            an = 4'b0111;
            dp = 1'b1;
            end
    endcase
end

always @(posedge clk or posedge reset) begin
    if(reset)
        begin
            state <= 2'b00;
        end
    else
        state <= next_state;
```

Arman Khondker
EID: aak2464

```
    end


Endmodule
```

CLK DIV

```
module clk_div(


    input clk,
    input reset,
    output slow_clk,
    output faster_clk,
    output tick
    );


    reg [19:0] COUNT;  //20 bits for 10ms


    assign slow_clk = COUNT[19]; //get top bit
    assign faster_clk = COUNT[14];



    always @(posedge clk or posedge reset)
      begin
        if (reset)
        begin
```

Arman Khondker
EID: aak2464

```
            COUNT <= 0;
        end
        else
            COUNT <= COUNT + 1;
    end

    assign tick = ((COUNT == 1000000)?1'b1:1'b0);

Endmodule
```

HEXTO7SEG

```
module hexto7seg(
    input [3:0] x,
    output reg [6:0] r
    );

    always @(*)
        case (x)
            4'b0000 : r = 7'b0000001;
            4'b0001 : r = 7'b1001111;
            4'b0010 : r = 7'b0010010;
            4'b0011 : r = 7'b0000110;
            4'b0100 : r = 7'b1001100;
            4'b0101 : r = 7'b0100100;
```

Arman Khondker
EID: aak2464

```
            4'b0110 : r = 7'b0100000;
            4'b0111 : r = 7'b0001111;
            4'b1000 : r = 7'b0000000;
            4'b1001 : r = 7'b0001100;
            4'b1010 : r = 7'b0001100;
            4'b1011 : r = 7'b0001100;
            4'b1100 : r = 7'b0001100;
            4'b1101 : r = 7'b0001100;
            4'b1110 : r = 7'b0001100;
            4'b1111 : r = 7'b0001100;
        endcase

    Endmodule
```

UP_COUNTER

```
    module up_counter(
  input clk,
  input reset,
  input tick,
  input stopStart,
  input [1:0] mode,
  input [7:0] sw,
  output reg [3:0] seg0,
  output reg [3:0] seg1,
```

Arman Khondker
EID: aak2464

```
output reg [3:0] seg2,
output reg [3:0] seg3
);

reg stop = 0;

always @ (posedge clk or posedge reset)
    begin
    if (reset)
        begin
        case(mode)
            2'b00:begin     //Mode 1 count up from 0
            seg0 <= 0;
            seg1 <= 0;
            seg2 <= 0;
            seg3 <= 0;
            stop <= 0;
            end

            2'b01: begin    //Mode 2 count up from a given value
            seg0 <= 0;
            seg1 <= 0;
            seg2 <= sw[3:0];
            seg3 <= sw[7:4];
```

```
        stop <= 0;
        end

        default: stop <= 0;

    endcase
    end

else if (stopStart)
    begin
    if(stop == 0)
        stop <= 1;
    else if(stop == 1)
        stop <= 0;
    end

else if (tick && stop)
    begin
    if(seg0 == 9)
    begin
    if(seg3 == 9 && seg2 == 9 && seg1 == 9)
        seg0 <= 9;
    else
        seg0  <= 0;
```

```
    if(seg1 == 9)
    begin
    if(seg3 == 9 && seg2 == 9)
        seg1 <= 9;
    else
        seg1 <= 0;
        if(seg2 == 9 && seg3 != 9)
        begin
        seg2 <= 0;
            if(seg3 >= 9)
            seg3 = 9;
            else if(seg3 != 9)
            seg3 <= seg3 + 1;
            end
        else if(seg2 != 9)
        seg2 <= seg2 + 1;
        end
    else if(seg1 != 9)
    seg1 <= seg1 + 1;
    end
else if(seg0 != 9)
    seg0 <= seg0 + 1;


end
```

```
    end


endmodule
```

DOWN COUNTER

```
module down_counter(
    input clk,
    input reset,
    input tick,
    input stopStart,
    input [1:0] mode,
    input [7:0] sw,
    output reg [3:0] seg0,
    output reg [3:0] seg1,
    output reg [3:0] seg2,
    output reg [3:0] seg3
    );

    reg stop = 0;

        always @ (posedge clk or posedge reset)
            begin
            if (reset)
                begin
```

```
    case(mode)
        2'b10:begin     //Mode 3 count down from 99
        seg0 <= 9;
        seg1 <= 9;
        seg2 <= 9;
        seg3 <= 9;
        stop <= 0;
        end

        2'b11: begin    //Mode 4 count down from a value
        seg0 <= 0;
        seg1 <= 0;
        seg2 <= sw[3:0];
        seg3 <= sw[7:4];
        stop <= 0;
        end

        default: stop <= 0;

    endcase
    end

else if (stopStart)
    begin
```

```
    if(stop == 0)
        stop <= 1;
    else if(stop == 1)
        stop <= 0;
    end

else if (tick && stop)
    begin
    if(seg0 == 0)
    begin
    if(seg3 == 0 && seg2 == 0 && seg1 == 0)
        seg0 <= 0;
    else
        seg0  <= 9;
        if(seg1 == 0)
        begin
        if(seg3 == 0 && seg2 == 0)
            seg1 <= 0;
        else
            seg1 <= 9;
            if(seg2 == 0 && seg3 != 0)
            begin
            seg2 <= 9;
                if(seg3 <= 0)
```

```
                seg3 = 0;
                else if(seg3 != 0)
                seg3 <= seg3 - 1;
                end
            else if(seg2 != 0)
            seg2 <= seg2 - 1;
            end
        else if(seg1 != 0)
        seg1 <= seg1 - 1;
        end
    else if(seg0 != 0)
        seg0 <= seg0 - 1;


    end
end

Endmodule
```

D) Simulation Waveform

Arman Khondker
EID: aak2464