Gabriel Emerson
ELEC 4200 – Lab 11
11/18/21

Lab 11: Lab Final Project – Digital Alarm Clock

Goals:

The goal for this lab is to finish designing and present your own Lab final Project. For the final project I decided to design a digital alarm clock on the Nexys DDR4 Artic 7 FPGA board. The clock should be able to be set by the user and then proceed to keep track of that time on the board to mimic real time. The board should also be able to set the alarm at any time, then alarm at that time if the Alarm_on switch is set on. Then when the alarm is set, pressing the reset alarm button should turn the alarm back off and wait for more instructions. The clock should continue to work until either the full_reset button is pressed or the time is set again. Both the clock time and alarm time will be shown on the 8 separate 7segment displays.

Design Process:

This project can be split into a few different parts. These parts being:

1. Getting the clock and alarm input
2. Operation of the clock
3. Comparing the clock and alarm to check for turning on alarm
4. Displaying the clock and alarm
5. Displaying alarm set high and resetting when turning alarm off

In some programs it is easier to read/debug if all of those processes are in different files or maybe even separate modules. However, I found it easier to keep all of these in the same module, so that there was no confusion with calling different variables from each file/module. The code is well commented to be able to find each section easily and is very easy to read. Since there was no overall project, there is no test or chart to follow to verify the design process.

Detailed Design:

Since this entire project is of our own idea and design, we have no exact design to follow. This means we can split things up however we seem fit, in order to solve this overall program. With this in mind, I decided to design this process in order of the process shown above going from 1 to 5. Then after finishing each section, would go back and comment heavily throughout that section so that I could easily debug later. The code used for the overall project is shown below in Figure 1.

**Note: In the creation of the clock, I have to use a decimal place on the 7 segment since the colon was not available from the Artix 7 constraint file.

**Note: There are two UUT's in this main file. These are only used by the IP Catalog and a divider file which are only used in the creation of a 1 second clock signal. Knowing this, we can assume these calls work since they are simple and only use one in and one out signal.

```
module aclock (
    input reset,  /* Active high reset pulse, to set the time to the input hour and minute (as defined by the H_in1, H_in0, M_in1, and M_in0 inputs) and the second to 00. It should also set
    input clk,  /* A 10Hz input clock. This should be used to generate each real-time second*/
    input [1:0] H_in1, /*A 2-bit input used to set the most significant hour digit of the clock (if LD_time=1),or the most significant hour digit of the alarm (if LD_alarm=1). Valid values
    input [3:0] H_in0, /* A 4-bit input used to set the least significant hour digit of the clock (if LD_time=1),or the least significant hour digit of the alarm (if LD_alarm=1). Valid valu
    input [3:0] M_in1, /*A 4-bit input used to set the most significant minute digit of the clock (if LD_time=1),or the most significant minute digit of the alarm (if LD_alarm=1). Valid val
    input [3:0] M_in0, /*A 4-bit input used to set the least significant minute digit of the clock (if LD_time=1),or the least significant minute digit of the alarm (if LD_alarm=1). Valid v
    input LD_time,  /* If LD_time=1, the time should be set to the values on the inputs H_in1, H_in0, M_in1, and M_in0. The second time should be set to 0.If LD_time=0, the clock should act
    input   LD_alarm,  /* If LD_alarm=1, the alarm time should be set to the values on the inputs H_in1, H_in0, M_in1, and M_in0.If LD_alarm=0, the clock should act normally.*/
    input   STOP_al,  /* If the Alarm (output) is high, then STOP_al=1 will bring the output back low. */
    input   AL_ON,  /* If high, the alarm is ON (and Alarm will go high if the alarm time equals the real time). If low the the alarm function is OFF. */
    output reg Alarm,  /* This will go high if the alarm time equals the current time, and AL_ON is high. This will remain high, until STOP_al goes high, which will bring Alarm back low.*/
    output reg Alarm2,
    //output [1:0]  H_out1,
    /* The most significant digit of the hour. Valid values are 0 to 2. */
    //output [3:0]  H_out0,
    /* The least significant digit of the hour. Valid values are 0 to 9. */
    //output [3:0]  M_out1,
    /* The most significant digit of the minute. Valid values are 0 to 5.*/
    //output [3:0]  M_out0, /* The least significant digit of the minute. Valid values are 0 to 9. */
    output [3:0]  S_out1, /* The most significant digit of the minute. Valid values are 0 to 5. */
    output [3:0]  S_out0,  /* The least significant digit of the minute. Valid values are 0 to 9. */
    /* Trying to get 7seg working*/
    output reg [7:0] an,
    //output DP,
    output clk_out2,
    output clk_out3,
    output reg test,
    //output reg aflag,
    //output reg rotate,
    output reg [7:0] sseg
    );

    wire [1:0]  H_out1;
    wire [3:0]  H_out0;
    wire [3:0]  M_out1;
    wire [3:0]  M_out0;
    //reg [3:0]  S_out1;
    //reg [3:0]  S_out0;
    wire [7:0] dp_in;
    assign dp_in = 8'b10111011;


    // internal signal
```

```
    reg clk_1s; // 1-s clock
    reg [3:0] tmp_1s; // count for creating 1-s clock

    // counter for clock hour, minute and second
    reg [5:0] tmp_hour, tmp_minute, tmp_second;
    /* The most significant hour digit of the temp clock and alarm. */
    reg [1:0] c_hour1,a_hour1;
    /* The least significant hour digit of the temp clock and alarm. */
    reg [3:0] c_hour0,a_hour0;
    /* The most significant minute digit of the temp clock and alarm.*/
    reg [3:0] c_min1,a_min1;
    /* The least significant minute digit of the temp clock and alarm.*/
    reg [3:0] c_min0,a_min0;
    /* The most significant second digit of the temp clock and alarm.*/
    reg [3:0] c_sec1,a_sec1;
    /* The least significant minute digit of the temp clock and alarm.*/
    reg [3:0] c_sec0,a_sec0;

    /************************************************/
    /*************Creating 10Hz clock****************/
    /************************************************/
    wire clk5;
    wire clk_out;
    clk_wiz_0 UUT1 ( .clk_in1(clk), .clk_out1(clk5));
    divider DUT1 (.clock_in(clk5), .clock_out(clk_out));

    assign clk_out2 = clk_out; //TESTING PURPOSES
    assign clk_out3 = clk;

    /************************************************/
    /*****************function*********************/
    /************************************************/
    function [3:0] mod_10;
    input [5:0] number;
    begin
    mod_10 = (number >=50) ? 5 : ((number >= 40)? 4 :((number >= 30)? 3 :((number >= 20)? 2 :((number >= 10)? 1 :0))));
    end
    endfunction

    /************************************************/
    /************ Clock operation***************/
    /************************************************/
    always @(posedge clk_1s or posedge reset )
    begin
        if(reset) begin // reset high => alarm time to 00.00.00, alarm to low, clock to H_in and M_in and S to 00
```

```
            a_hour1 <= 2'b00;
            a_hour0 <= 4'b0000;
            a_min1 <= 4'b0000;
            a_min0 <= 4'b0000;
            a_sec1 <= 4'b0000;
            a_sec0 <= 4'b0000;
            tmp_hour <= H_in1*10 + H_in0;
            tmp_minute <= M_in1*10 + M_in0;
            tmp_second <= 0;
        end
        else begin
            if(LD_alarm) begin // LD_alarm =1 => set alarm clock to H_in, M_in
                a_hour1 <= H_in1;
                a_hour0 <= H_in0;
                a_min1 <= M_in1;
                a_min0 <= M_in0;
                a_sec1 <= 4'b0000;
                a_sec0 <= 4'b0000;
            end
            if(LD_time) begin // LD_time =1 => set time to H_in, M_in
                tmp_hour <= H_in1*10 + H_in0;
                tmp_minute <= M_in1*10 + M_in0;
                tmp_second <= 0;
            end
            else begin  // LD_time =0 , clock operates normally
                tmp_second <= tmp_second + 1;
                if(tmp_second >=59) begin // second > 59 then minute increases
                    tmp_minute <= tmp_minute + 1;
                    tmp_second <= 0;
                    if(tmp_minute >=59) begin // minute > 59 then hour increases
                        tmp_minute <= 0;
                        tmp_hour <= tmp_hour + 1;
                        if(tmp_hour >= 24) begin // hour > 24 then set hour to 0
                            tmp_hour <= 0;
                        end
                    end
                end
            end
        end
    end

    /************************************************/
    /******** Create 1-second clock***************/
    /************************************************/
    always @(posedge clk_out or posedge reset)
```

```
    begin
        if(reset) begin
            tmp_ls <= 0;
            clk_ls <= 0;
        end
        else begin
            tmp_ls <= tmp_ls + 1;
            if(tmp_ls <= 5)
                clk_ls <= 0;
            else if (tmp_ls >= 10) begin
                clk_ls <= 1;
                tmp_ls <= 1;
            end
            else
                clk_ls <= 1;
        end
    end

    /************************************************/
    /***OUTPUT OF THE CLOCK*********************/
    /************************************************/
    always @(*) begin
        if(tmp_hour>=20) begin
            c_hour1 = 2;
        end
        else begin
            if(tmp_hour >=10)
                c_hour1  = 1;
            else
                c_hour1 = 0;
        end
        c_hour0 = tmp_hour - c_hour1*10;
        c_min1 = mod_10(tmp_minute);
        c_min0 = tmp_minute - c_min1*10;
        c_sec1 = mod_10(tmp_second);
        c_sec0 = tmp_second - c_sec1*10;
    end

    assign H_out1 = c_hour1; // the most significant hour digit of the clock
    assign H_out0 = c_hour0; // the least significant hour digit of the clock
    assign M_out1 = c_min1; // the most significant minute digit of the clock
    assign M_out0 = c_min0; // the least significant minute digit of the clock
    assign S_out1 = c_sec1; // the most significant second digit of the clock
    assign S_out0 = c_sec0; // the least significant second digit of the clock
```

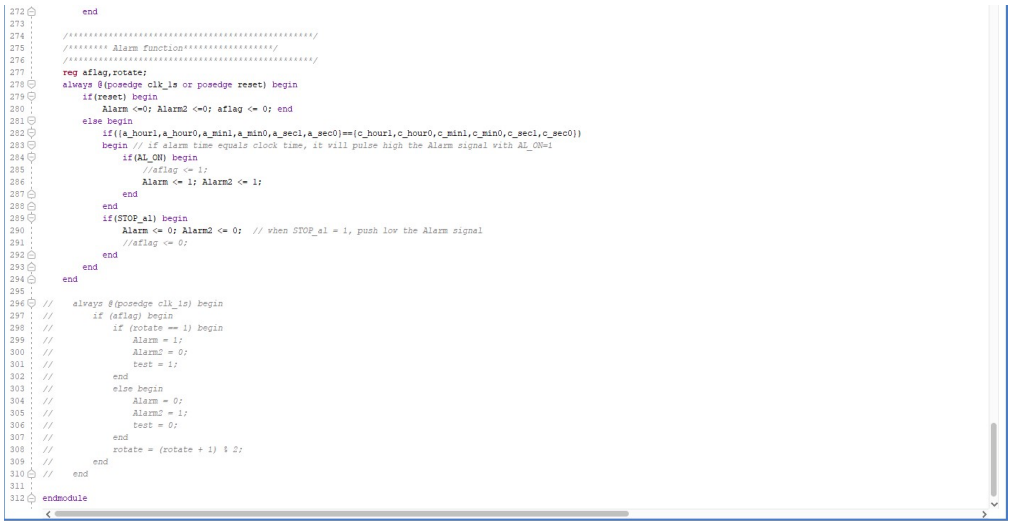
```
    /************************************************/
    /********** 7 Segment Decoder controller ********/
    /************************************************/
    localparam N = 18; //50 Mhz clock crossover using low 16 bits (50MHZ/2^16)
    reg [N-1:0] regN; //High two bits as control signal and low 16 bits as counter to divide clock frequency
    reg [3:0] hex_in; //Segment Selection Control Signal
    reg dp;

    always@(posedge clk, posedge reset)
    begin
        if(reset)
            regN <= 0;
        else
            regN <= regN + 1;
    end

    always@ (clk) begin
            case(regN[N-1:N-3])
            3'b000:begin
                an = 8'b11111110; //Select 1st Digital Tube
                hex_in = c_min0; //The number displayed by the tube is hex_in control, showing the number entered by hex0
                dp = dp_in[0];
            end
            3'b001:begin
                an = 8'b11111101; //Select the second tube
                hex_in = c_min1;
                dp = dp_in[1];
            end
            3'b010:begin
                an = 8'b11111011;
                hex_in = c_hour0;
                dp = dp_in[2];
            end
            3'b011:begin
                if (c_hour1 == 1 | c_hour1 == 2) begin
                    an = 8'b11110111;
                    hex_in = c_hour1;
                    dp = dp_in[3];
                end
                else
                    an = 8'b11111111;
            end
            3'b100:begin
                an = 8'b11101111;
                hex_in = a_min0;
```

```
                dp = dp_in[4];
            end
            3'b101:begin
                an = 8'b11011111;
                hex_in = a_min1;
                dp = dp_in[5];
            end
            3'b110:begin
                an = 8'b10111111;
                hex_in = a_hour0;
                dp = dp_in[6];
            end
            default:begin
                if (a_hour1 == 1 | a_hour1 == 2) begin
                    an = 8'b01111111;
                    hex_in = a_hour1;
                    dp = dp_in[7];
                end
                else
                    an = 8'b11111111;
            end
            endcase
        end

        always@(clk)
        begin
            case(hex_in)
                4'h0: sseg[6:0] = 7'b0000001; //Common anode digital tube
                4'h1: sseg[6:0] = 7'b1001111;
                4'h2: sseg[6:0] = 7'b0010010;
                4'h3: sseg[6:0] = 7'b0000110;
                4'h4: sseg[6:0] = 7'b1001100;
                4'h5: sseg[6:0] = 7'b0100100;
                4'h6: sseg[6:0] = 7'b0100000;
                4'h7: sseg[6:0] = 7'b0001111;
                4'h8: sseg[6:0] = 7'b0000000;
                4'h9: sseg[6:0] = 7'b0000100;
                4'ha: sseg[6:0] = 7'b0001000;
                4'hb: sseg[6:0] = 7'b1100000;
                4'hc: sseg[6:0] = 7'b0110001;
                4'hd: sseg[6:0] = 7'b1000010;
                4'he: sseg[6:0] = 7'b0110000;
                default: sseg[6:0] = 7'b0111000;
            endcase
            sseg[7] = dp;
```



```
        end

    /*************************************************/
    /********* Alarm function*********************/
    /*************************************************/
    reg aflag,rotate;
    always @(posedge clk_1s or posedge reset) begin
        if(reset) begin
            Alarm <=0; Alarm2 <=0; aflag <= 0; end
        else begin
            if({a_hour1,a_hour0,a_min1,a_min0,a_sec1,a_sec0}=={c_hour1,c_hour0,c_min1,c_min0,c_sec1,c_sec0})
            begin // if alarm time equals clock time, it will pulse high the Alarm signal with AL_ON=1
                if(AL_ON) begin
                    //aflag <= 1;
                    Alarm <= 1; Alarm2 <= 1;
                end
            end
            if(STOP_al) begin
                Alarm <= 0; Alarm2 <= 0;  // when STOP_al = 1, push low the Alarm signal
                //aflag <= 0;
            end
        end
    end

//    always @(posedge clk_1s) begin
//        if (aflag) begin
//            if (rotate == 1) begin
//                Alarm = 1;
//                Alarm2 = 0;
//                test = 1;
//            end
//            else begin
//                Alarm = 0;
//                Alarm2 = 1;
//                test = 0;
//            end
//            rotate = (rotate + 1) % 2;
//        end
//    end

endmodule
```

Figure 1

Verification:

Since this was a project of our own making, there was no testbench required in order to verify the functionality. However, I attempted to make my own testbench but since the main output I was testing had to be on a 7 segment decoder, it made this very difficult to test. This means the main way of testing this design is to upload the program to a board and visually verify all aspects of the clock. This means setting time, alarm, alarm going off (and being reset), clock functionality, correct display, and anything else that may need to be checked. After uploading and checking, I ran into just a couple problems before finally figuring it out and getting the digital alarm clock to work perfectly. I have no errors in my design.

Conclusion:

What I learned in this final project lab is how many ideas we have used throughout the semester can come together and create one large project that runs many of the smaller ideas, all at the same time. My biggest problem was trying to get the 7 segment controller to work correctly in order to refresh the displays correctly and fast enough to appear like the clock is running all at one time. After figuring this part out the rest of the design worked well besides a few small errors in the alarm function and running the clock. This final lab went really well since I was able to use a lot of previous ideas to get the basics working and combine that with new ideas that must be used to get the exact alarm clock that was designed.