ECEN2350-Project1

Chengming Li

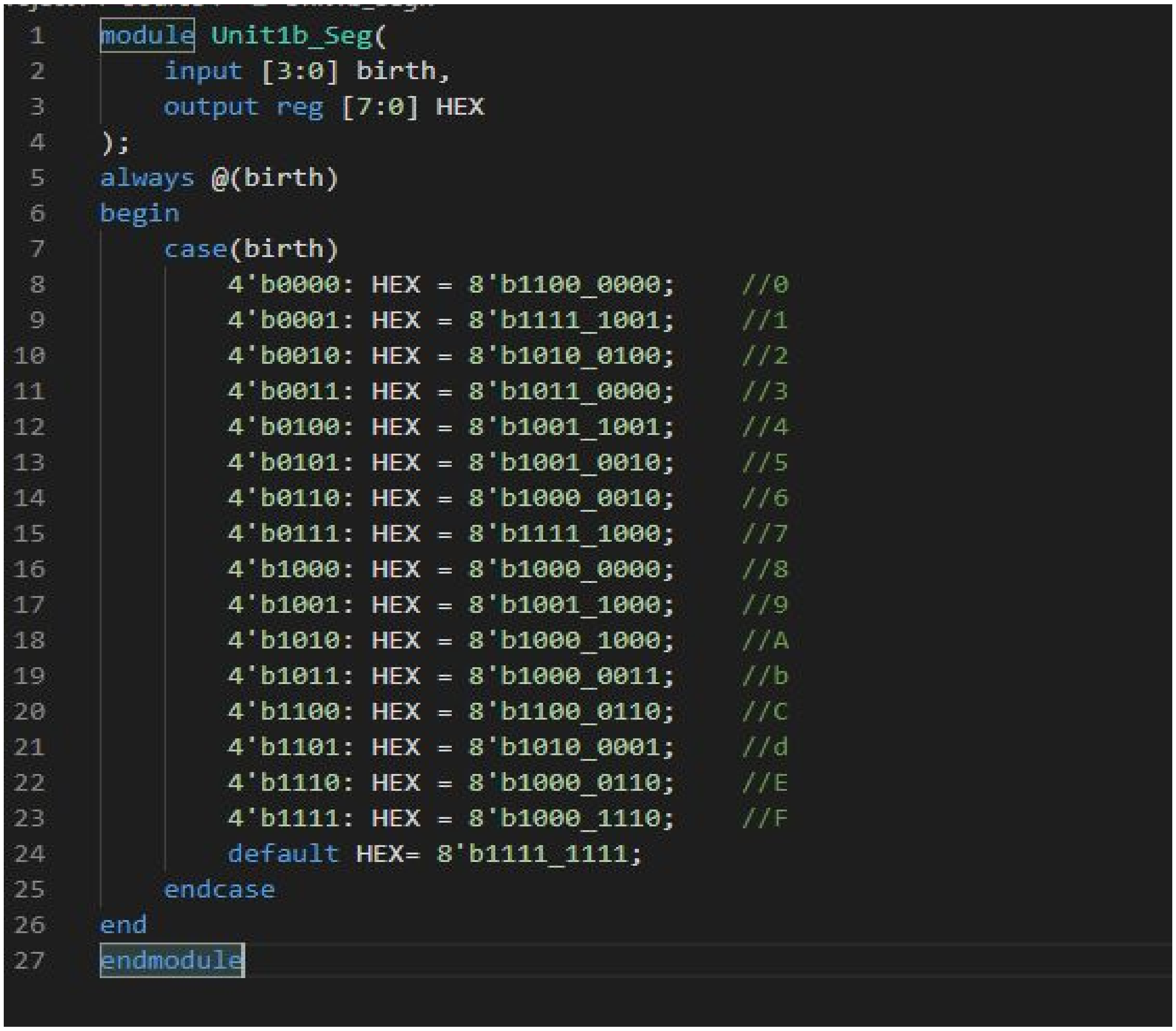
1.Project Statement

In this project, it consists of 2 separate units. In the first unit, there are also two parts controlled by KEY on the De10-lite board. The first part is connecting slide switches SW[7:0] to the corresponding LEDR[7:0]. And there are two operations controlled by KEY[0]. When KEY[0] is not pressed and SW is down, the LEDR should be on.When KEY[0] is pressed, the states of LEDR will invert. The second part is displaying two different birthdays by pressing KEY[1]. When KEY[1] is not pressed, HEX will display my birthdays. And when it is pressed, HEX will show my mother's birthdays. In the second unit, we are going to design a comparison by entering input via SW[7:4](input 1) and SW[3:0] (input 2). Then HEX5, HEX3 and HEX1 will display the input 1, results of comparison and input2. At the end, we create multiplexers controlled by SW[9] on the board to select the selected Design Unit output.

2. Theory of operation.

In this project, we should notice what input state will turn on the corresponding output. For SW[9:0], logic 1 means the SW is up. For LEDR[9:0], logic 1 will turn on the corresponding lights. And for KEY[1:0], logic 0 means key is pressed. For HEX[7:0], logic 0 will turn on the corresponding segment in each HEX display. Briefly, we could get our expected observations in LEDR and HEX by slipping the Switch and pressing the KEY to finish our project.

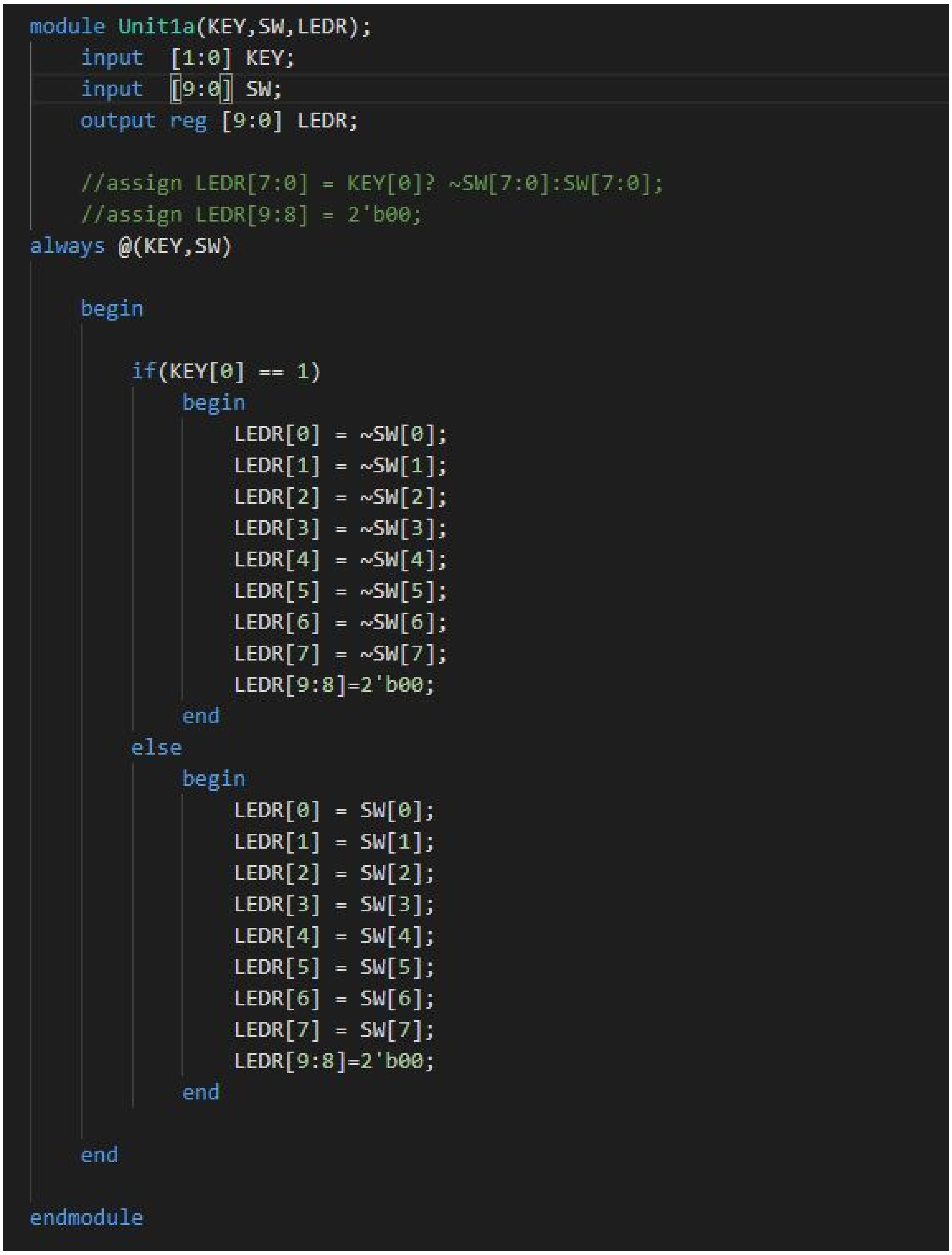
For this project, I also created a separate encoder module to light up the HEX. Generally, in this encoder, it takes 4 bits input to generate 8 bits output which could turn on the corresponding segments in each HEX. This encoder could interpret 0-F value by using case statements in the verilog. It is fairly useful in this project, especially for displaying the Birthdays and results of comparison. When I want to use this encoder in a specific situation, I could just instantiate it in that module which will automatically convert 4 bits input to 8 bits output.



```
module Unit1b_Seg(
    input [3:0] birth,
    output reg [7:0] HEX
);
always @(birth)
begin
    case(birth)
        4'b0000: HEX = 8'b1100_0000;    //0
        4'b0001: HEX = 8'b1111_1001;    //1
        4'b0010: HEX = 8'b1010_0100;    //2
        4'b0011: HEX = 8'b1011_0000;    //3
        4'b0100: HEX = 8'b1001_1001;    //4
        4'b0101: HEX = 8'b1001_0010;    //5
        4'b0110: HEX = 8'b1000_0010;    //6
        4'b0111: HEX = 8'b1111_1000;    //7
        4'b1000: HEX = 8'b1000_0000;    //8
        4'b1001: HEX = 8'b1001_1000;    //9
        4'b1010: HEX = 8'b1000_1000;    //A
        4'b1011: HEX = 8'b1000_0011;    //b
        4'b1100: HEX = 8'b1100_0110;    //C
        4'b1101: HEX = 8'b1010_0001;    //d
        4'b1110: HEX = 8'b1000_0110;    //E
        4'b1111: HEX = 8'b1000_1110;    //F
        default HEX= 8'b1111_1111;
    endcase
end
endmodule
```

Encoder for 0-F

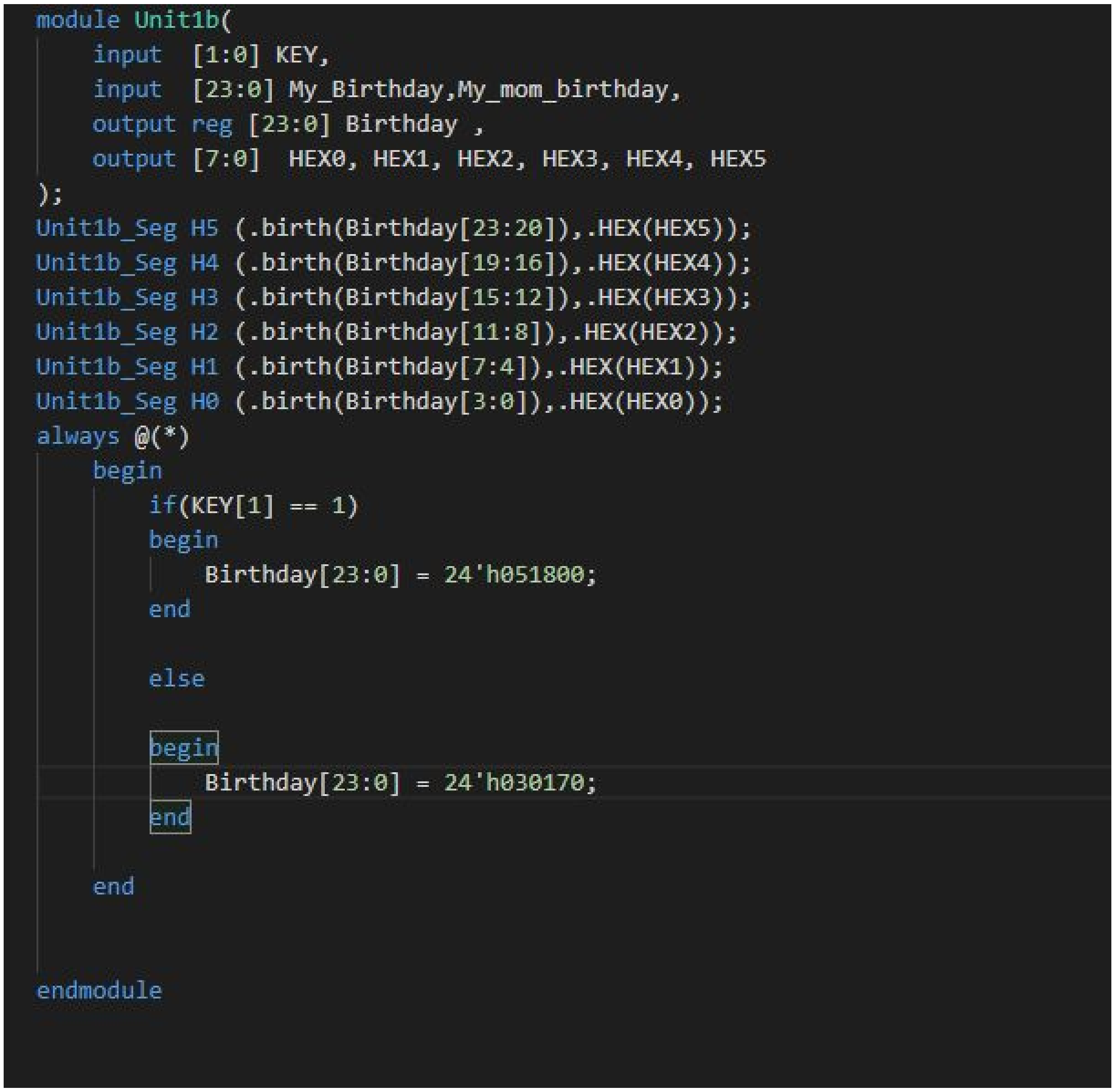
In the De10-lite board, we could use SW[9] to select the operations we want. If SW[9] is up, the board will behave as a Unit 2 Comparison. We could enter the input by slipping SW[7:4] and SW[3:0]. Then the results of comparison will display in the HEX5(Input 1), HEX3(E, or L, or Blank), and HEX1(Input 2). If SW[9] is down, the board will behave as Unit 1. If KEY[1:0] is not pressed, the LEDs are on when SW is down and HEX will display my birthdays. If KEY[1:0] is pressed, the output state of LEDs will invert and HEX will display my mom' birthday.



```verilog
module Unit1a(KEY,SW,LEDR);
    input  [1:0] KEY;
    input  [9:0] SW;
    output reg [9:0] LEDR;

    //assign LEDR[7:0] = KEY[0]? ~SW[7:0]:SW[7:0];
    //assign LEDR[9:8] = 2'b00;
always @(KEY,SW)

    begin

        if(KEY[0] == 1)
            begin
                LEDR[0] = ~SW[0];
                LEDR[1] = ~SW[1];
                LEDR[2] = ~SW[2];
                LEDR[3] = ~SW[3];
                LEDR[4] = ~SW[4];
                LEDR[5] = ~SW[5];
                LEDR[6] = ~SW[6];
                LEDR[7] = ~SW[7];
                LEDR[9:8]=2'b00;
            end
        else
            begin
                LEDR[0] = SW[0];
                LEDR[1] = SW[1];
                LEDR[2] = SW[2];
                LEDR[3] = SW[3];
                LEDR[4] = SW[4];
                LEDR[5] = SW[5];
                LEDR[6] = SW[6];
                LEDR[7] = SW[7];
                LEDR[9:8]=2'b00;
            end

    end

endmodule
```

Unit1a

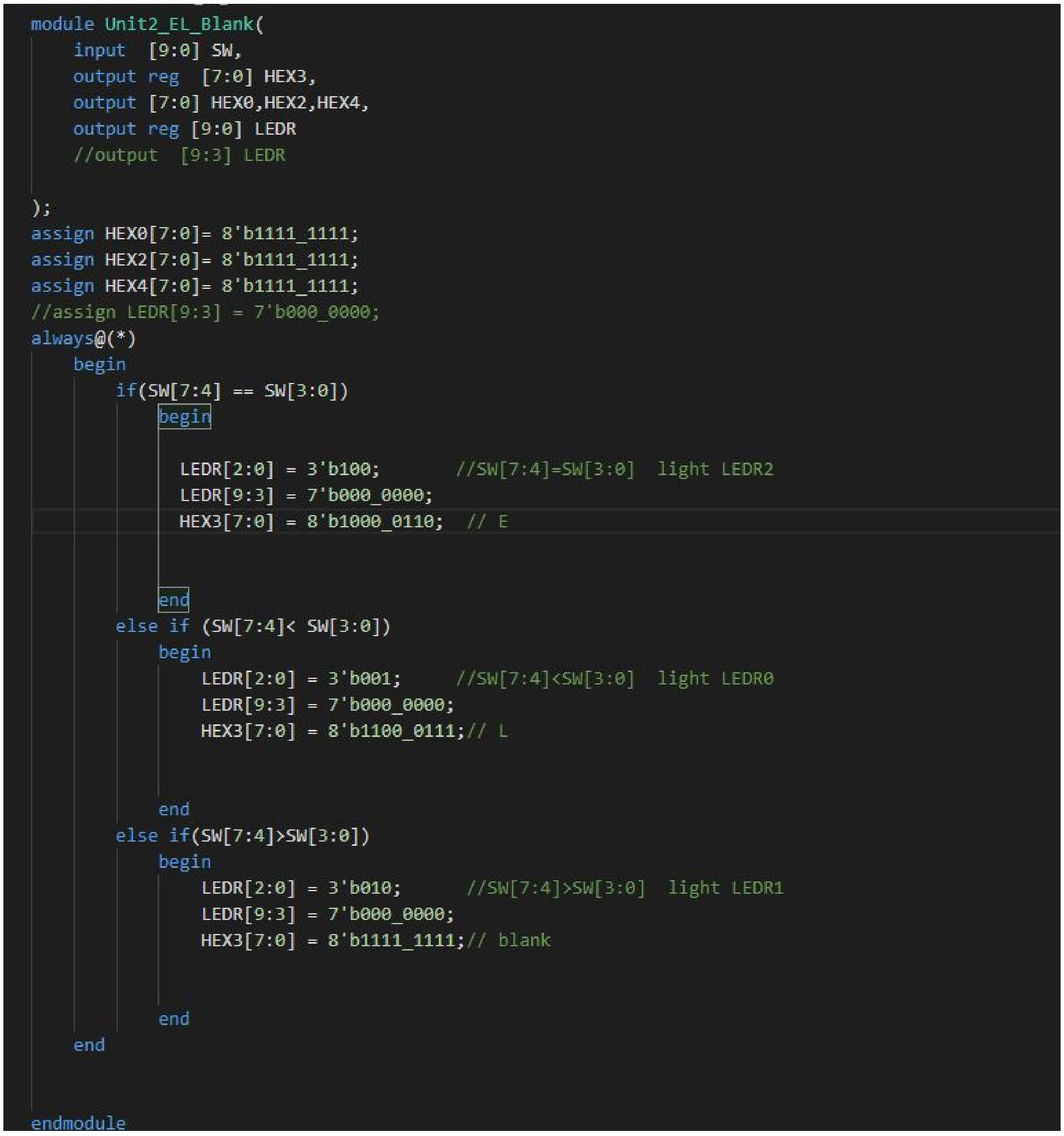
For unit1 a, I am using the states of each SW to control the states of LEDR. When KEY[0] is not pressed, KEY[0] has state of logic 1. And, from the requirement, I should turn on the LEDR by assigning logic 1 which is the opposite states of each SW. When KEY[0] is pressed, LEDR will work in the same states of corresponding SW.



```
module Unit1b(
    input  [1:0] KEY,
    input  [23:0] My_Birthday,My_mom_birthday,
    output reg [23:0] Birthday ,
    output [7:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5
);
Unit1b_Seg H5 (.birth(Birthday[23:20]),.HEX(HEX5));
Unit1b_Seg H4 (.birth(Birthday[19:16]),.HEX(HEX4));
Unit1b_Seg H3 (.birth(Birthday[15:12]),.HEX(HEX3));
Unit1b_Seg H2 (.birth(Birthday[11:8]),.HEX(HEX2));
Unit1b_Seg H1 (.birth(Birthday[7:4]),.HEX(HEX1));
Unit1b_Seg H0 (.birth(Birthday[3:0]),.HEX(HEX0));
always @(*)
    begin
        if(KEY[1] == 1)
        begin
            Birthday[23:0] = 24'h051800;
        end

        else

        begin
            Birthday[23:0] = 24'h030170;
        end

    end


endmodule
```

Unit 1b

In 1b, I instantiate the decoder file 5 times, which could display the days, months and years in corresponding HEX. Inside of the decoder file, it takes 4 bits input, then converting to 8 bits segment display. Furthermore, I set an if statement to determine whether the button is pressed or not so that De10-lite could display the correct birthdays.

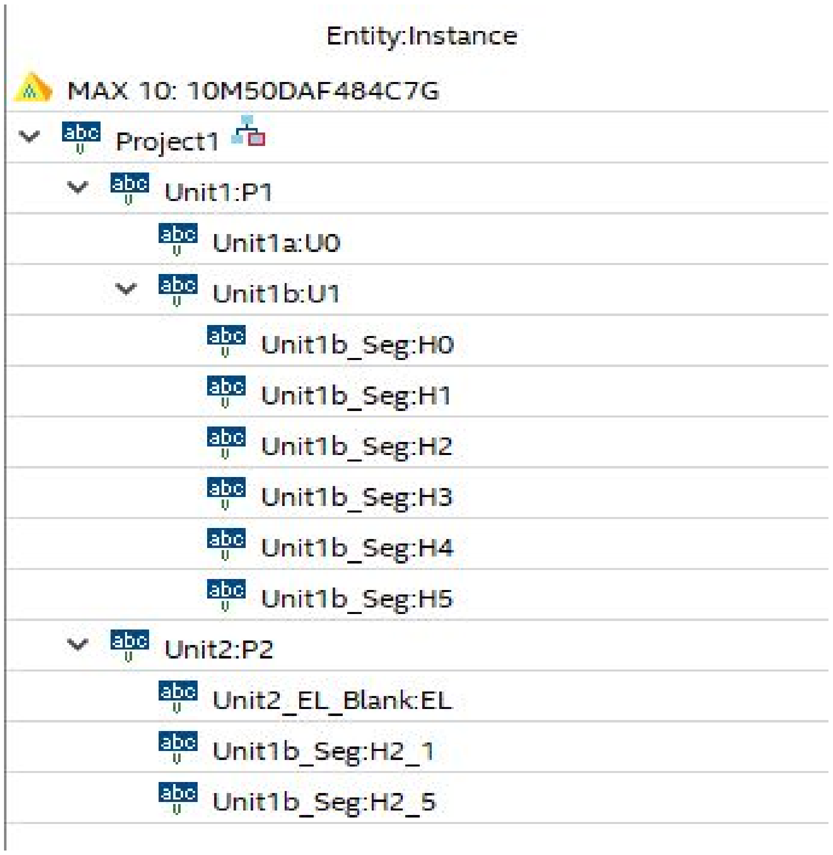


```
module Unit2_EL_Blank(
    input  [9:0] SW,
    output reg  [7:0] HEX3,
    output [7:0] HEX0,HEX2,HEX4,
    output reg [9:0] LEDR
    //output  [9:3] LEDR

);
assign HEX0[7:0]= 8'b1111_1111;
assign HEX2[7:0]= 8'b1111_1111;
assign HEX4[7:0]= 8'b1111_1111;
//assign LEDR[9:3] = 7'b000_0000;
always@(*)
    begin
        if(SW[7:4] == SW[3:0])
            begin

              LEDR[2:0] = 3'b100;       //SW[7:4]=SW[3:0]  light LEDR2
              LEDR[9:3] = 7'b000_0000;
              HEX3[7:0] = 8'b1000_0110;  // E


            end
        else if (SW[7:4]< SW[3:0])
            begin
                LEDR[2:0] = 3'b001;     //SW[7:4]<SW[3:0]  light LEDR0
                LEDR[9:3] = 7'b000_0000;
                HEX3[7:0] = 8'b1100_0111;// L


            end
        else if(SW[7:4]>SW[3:0])
            begin
                LEDR[2:0] = 3'b010;      //SW[7:4]>SW[3:0]  light LEDR1
                LEDR[9:3] = 7'b000_0000;
                HEX3[7:0] = 8'b1111_1111;// blank


            end
    end


endmodule
```

Unit2

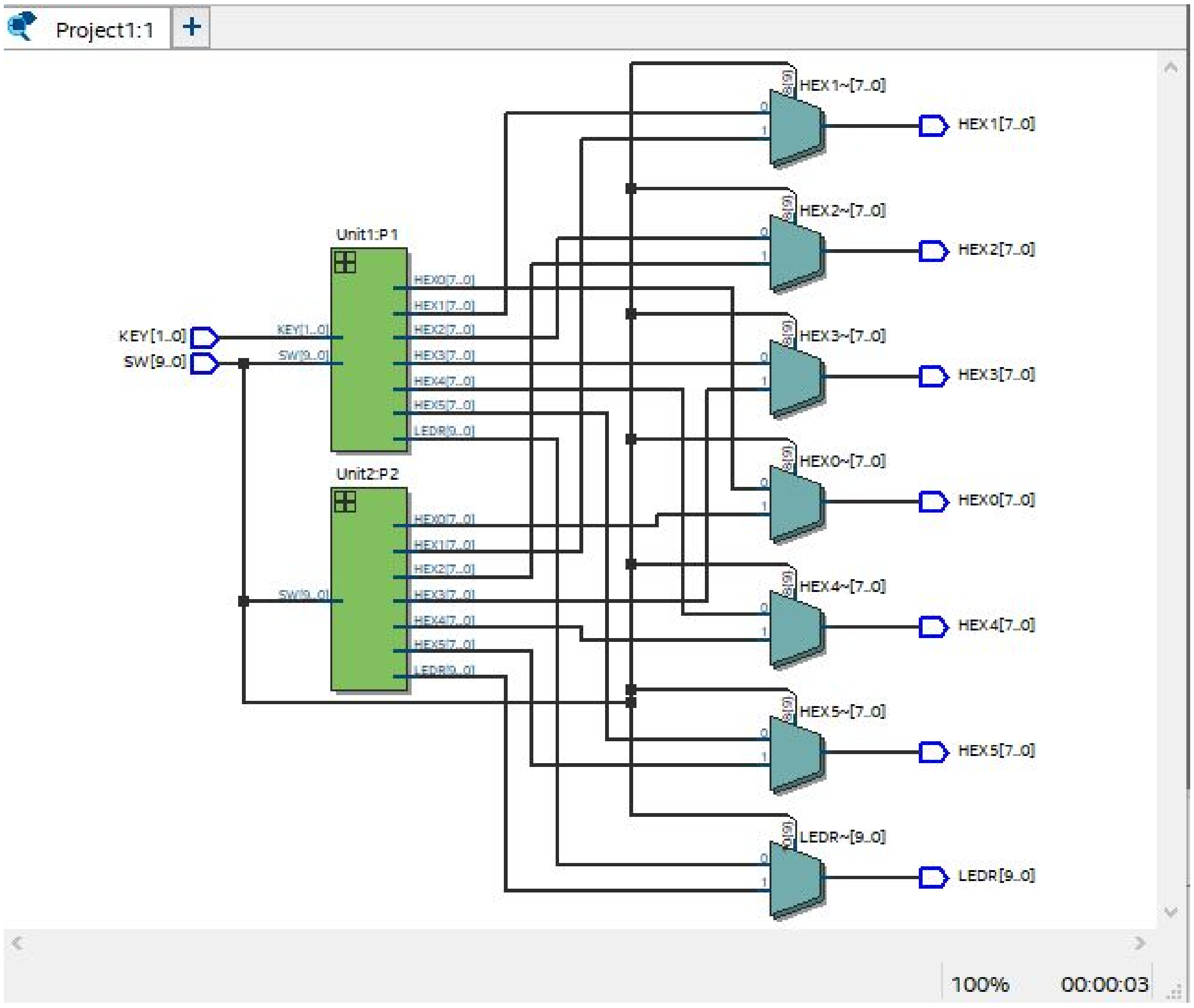
In unit 2, it takes 2 inputs. Each input is 4 bits wide. Generally, I compare these two inputs inside of 3 if statements which are working in 3 different situations. When they are equal to each other, LEDR[2] will turn on and HEX3 will display E (1000_0110). When input 1 is smaller than input 2, LED[0] will turn on and HEX3 will display L(1100_0111); When input 1 is greater than input 2, LED[1] will turn on and HEX3 will be blanked(1111_1111).

3. Hierarchy of My project source files

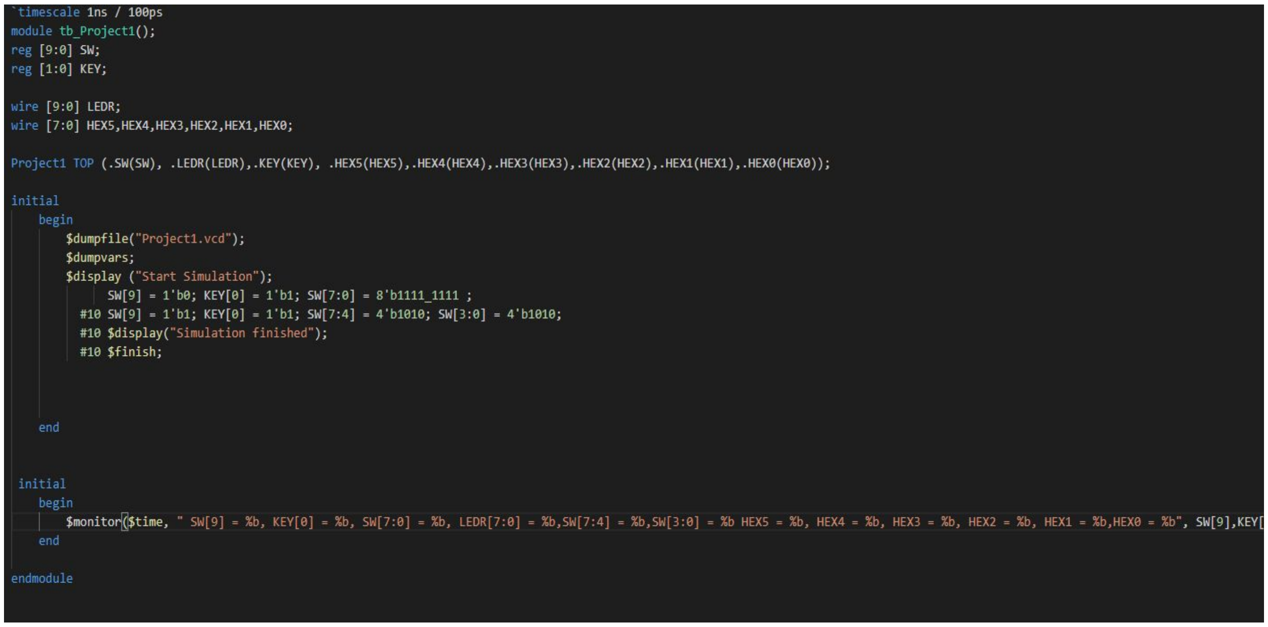


```
Entity:Instance
MAX 10: 10M50DAF484C7G
Project1
  Unit1:P1
    Unit1a:U0
    Unit1b:U1
      Unit1b_Seg:H0
      Unit1b_Seg:H1
      Unit1b_Seg:H2
      Unit1b_Seg:H3
      Unit1b_Seg:H4
      Unit1b_Seg:H5
  Unit2:P2
    Unit2_EL_Blank:EL
    Unit1b_Seg:H2_1
    Unit1b_Seg:H2_5
```

4.Block diagram of design



5. Description of Testbench operation



```
`timescale 1ns / 100ps
module tb_Project1();
reg [9:0] SW;
reg [1:0] KEY;

wire [9:0] LEDR;
wire [7:0] HEX5,HEX4,HEX3,HEX2,HEX1,HEX0;

Project1 TOP (.SW(SW), .LEDR(LEDR),.KEY(KEY), .HEX5(HEX5),.HEX4(HEX4),.HEX3(HEX3),.HEX2(HEX2),.HEX1(HEX1),.HEX0(HEX0));

initial
    begin
        $dumpfile("Project1.vcd");
        $dumpvars;
        $display ("Start Simulation");
                SW[9] = 1'b0; KEY[0] = 1'b1; SW[7:0] = 8'b1111_1111 ;
          #10 SW[9] = 1'b1; KEY[0] = 1'b1; SW[7:4] = 4'b1010; SW[3:0] = 4'b1010;
          #10 $display("Simulation finished");
          #10 $finish;



    end


 initial
    begin
        $monitor($time, " SW[9] = %b, KEY[0] = %b, SW[7:0] = %b, LEDR[7:0] = %b,SW[7:4] = %b,SW[3:0] = %b HEX5 = %b, HEX4 = %b, HEX3 = %b, HEX2 = %b, HEX1 = %b,HEX0 = %b", SW[9],KEY[
    end

endmodule
```

Figure 1 tb_top.v

In the testbench.v file, I only instantiate the Project1.v file as a top file since it consists of two lower unit design files. Inside of the initial block, $dumpfile will generate a .vcd file which is used for GTKwave. $dumpvars tells the simulator to capture all the signals in the .vcd file. Then we could begin our simulation with 10ns delay by assigning the values to input SW and KEY. At the end of the initial block, I use $finish to tell the simulation to halt and do not take any command from the terminal. In the next initial block, I use $monitor to display the value of signals listed in anytime when one of the signals changes. At the end , $display is used to display the simulation input and output in text format.

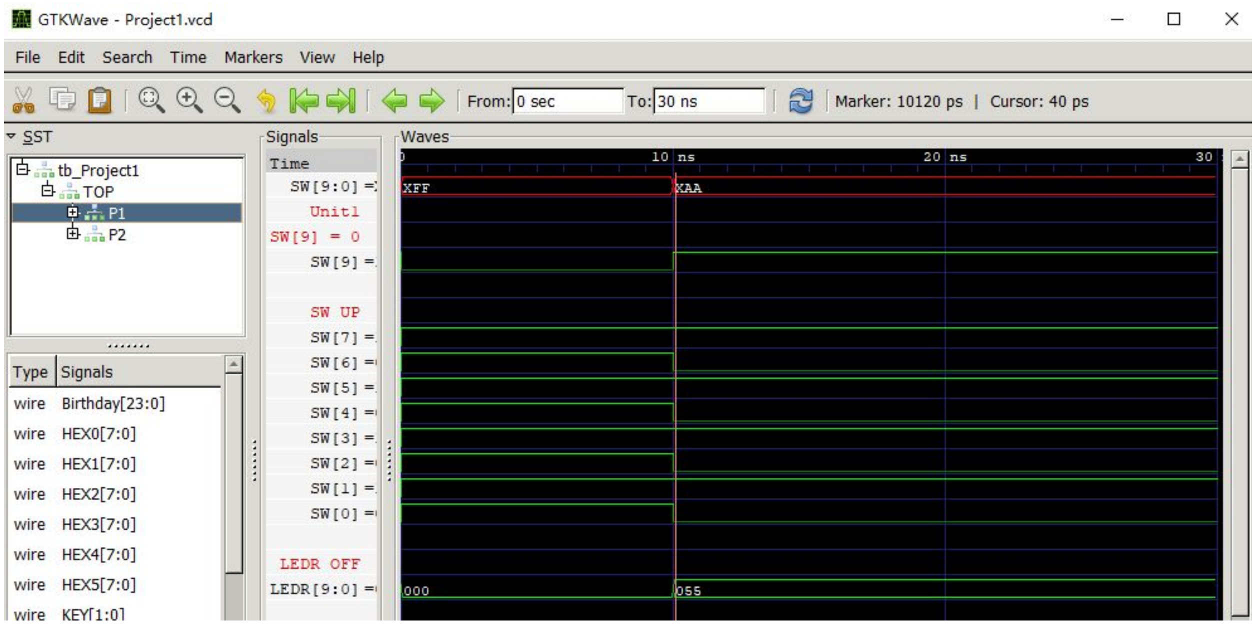


Figure 2. Unit 1 testbench(KEY[0] =1, SW[7:0] =1)

In the top testbench, I only simulate 2 situations to check the effects of slipping of SW[9] and corresponding results of each unit. At first 10ns, SW[9] is logic 0 (Down) and KEY[0] is not pressed which means the De10-lite board behaves as Unit 1. In this situation, LEDs will behave in the opposite way. During the first 10ns, I set SW[7:0] to up so that the LEDs will turn off(logic 0) like shown in the figure 2.

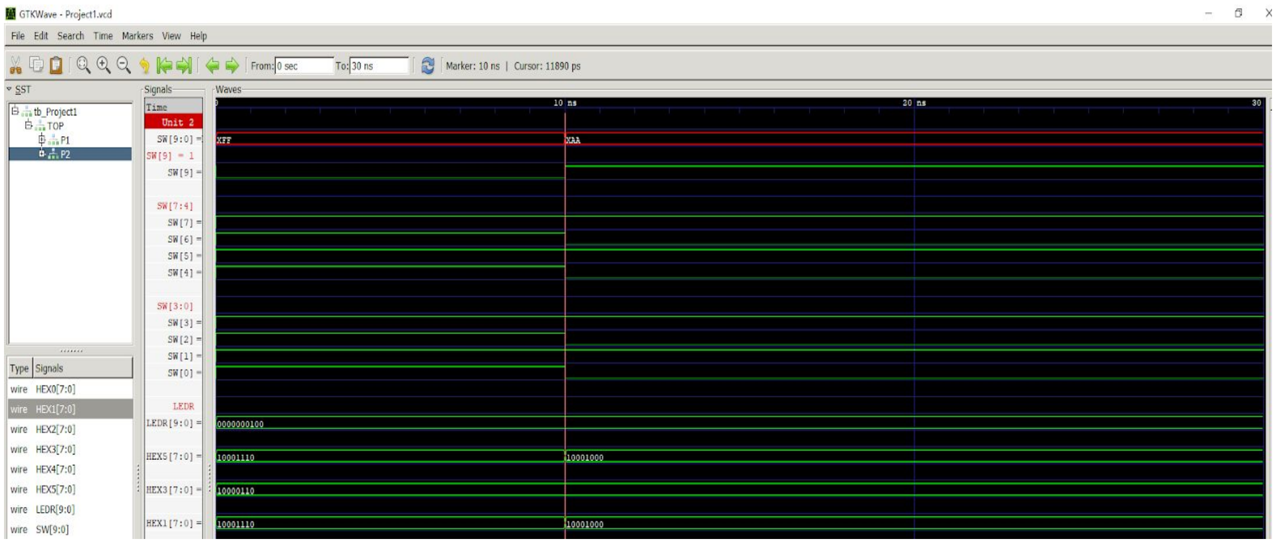
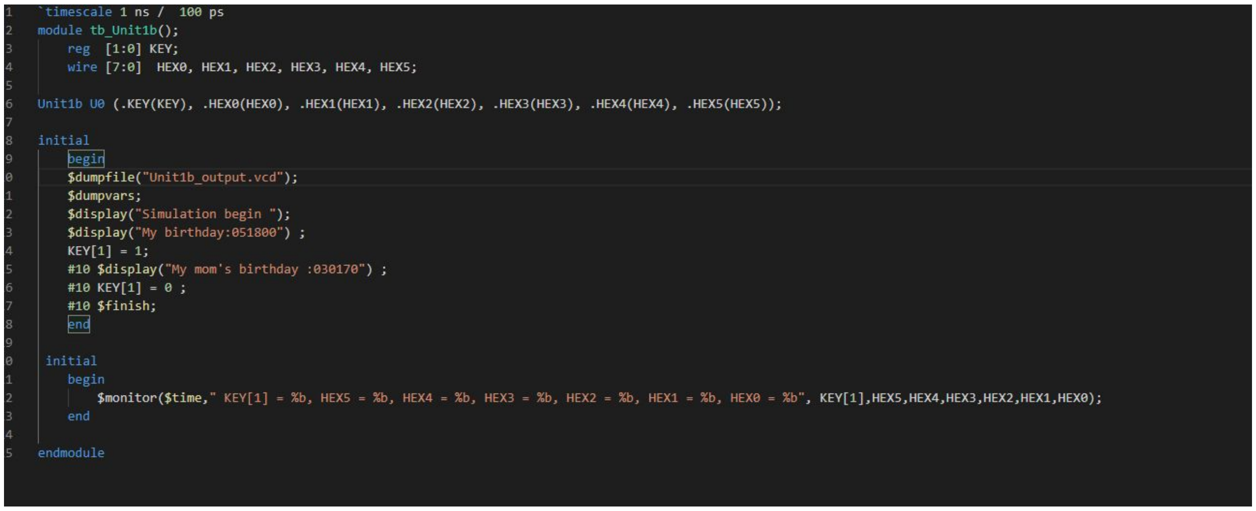


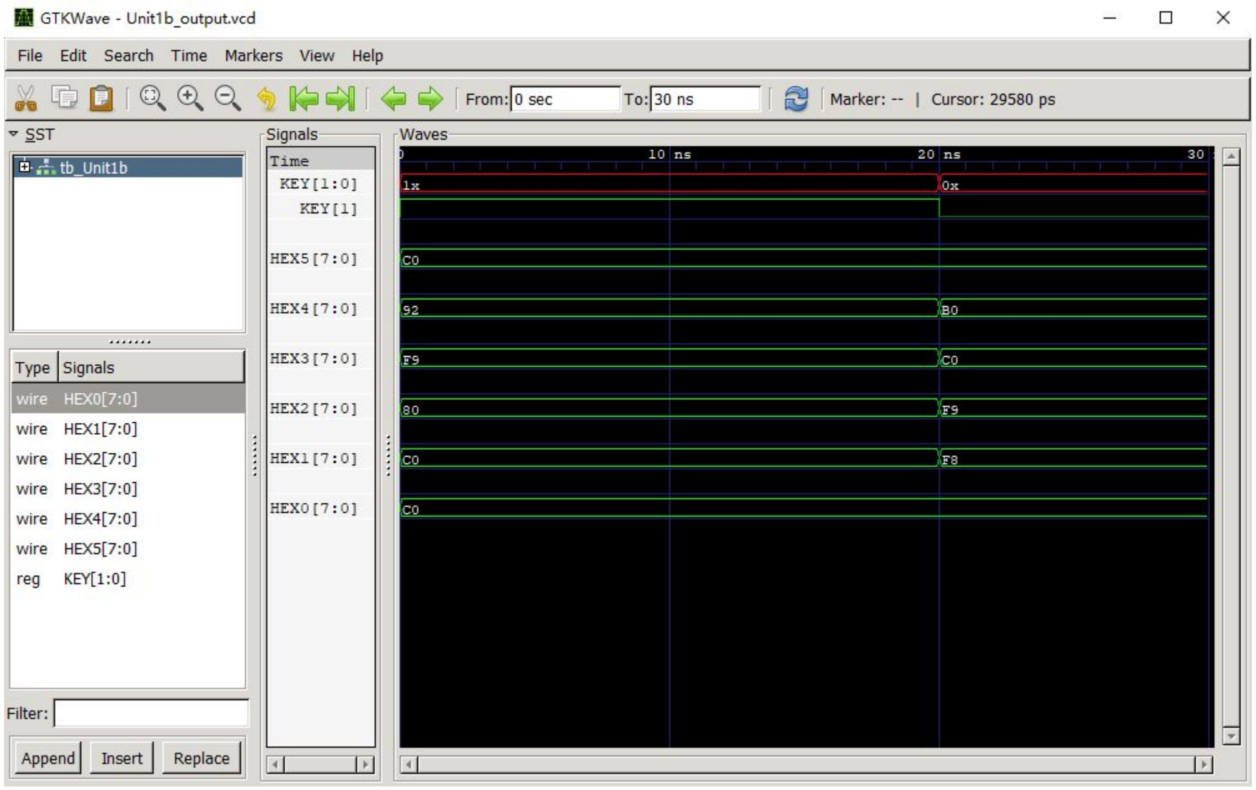
Figure 3. Unit2(SW[9]=1,SW[7:4]=4’b1010,SW[3:0] =4’b1010)

At second 10ns, I set the input state of SW[9] (logic 1) to up which means the De10-lite board behaves as Unit 2(comparison). Meanwhile, I set SW[7:4](Input 1) =4 ’b1010 (10 in decimal) and SW[3:0] (Input 2) = 4’b1010(10 in decimal). Thus, HEX5 and HEX 1 will display A (8’b1000_1000) as input value. Then, HEX3 will display letter E (8’b1000_0110) to tell that both inputs are equal to each other.



```
`timescale 1 ns /  100 ps
module tb_Unit1b();
    reg  [1:0] KEY;
    wire [7:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;

Unit1b U0 (.KEY(KEY), .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5));

initial
    begin
    $dumpfile("Unit1b_output.vcd");
    $dumpvars;
    $display("Simulation begin ");
    $display("My birthday:051800") ;
    KEY[1] = 1;
    #10 $display("My mom's birthday :030170") ;
    #10 KEY[1] = 0 ;
    #10 $finish;
    end

 initial
    begin
        $monitor($time," KEY[1] = %b, HEX5 = %b, HEX4 = %b, HEX3 = %b, HEX2 = %b, HEX1 = %b, HEX0 = %b", KEY[1],HEX5,HEX4,HEX3,HEX2,HEX1,HEX0);
    end

endmodule
```

Figure 4. Unit1b_testbench



From: 0 sec To: 30 ns Marker: -- | Cursor: 29580 ps

Figure 5 Unit1b_gtkwave

In my Unit1b testbench, it only has two situations. First situation is showing my birthday when KEY[1] is not pressed. Thus, in my gtkwave, the first 20ns shows my birthday in terms of HEX5->HEX0. After 20 ns, it simulates my mom’s birthday when KEY[1](state =0) is pressed.

6. Summary

This project helps me to have a better understanding of coding verilog and testbench, simulating design, using gtkwave, and loading projects through quartus. Meanwhile, this project gives me a general idea of the hierarchy of a project. Since some projects may consist of more than 3 design units, or even more, Hierarchy could help me to know where I should start and how to test each unit part by part. Generally, Hierarchy coding could be more efficient to debug the project and to check whether the project is working or not. In this project, I feel good so far because it is not complex and syntax is kind of similar to C language. And, everything is working as the description of this project. Also, encoders and multiplexers are getting more sense after I finish this project. Meanwhile, I feel good with the syntax of verilog, and using GTKwave and Quartus. But, I feel a bit difficult to draw the block diagram in the Quartus. I am not sure what contents and format we need to include.