ECEN2350
Project2-Report
Chengming Li

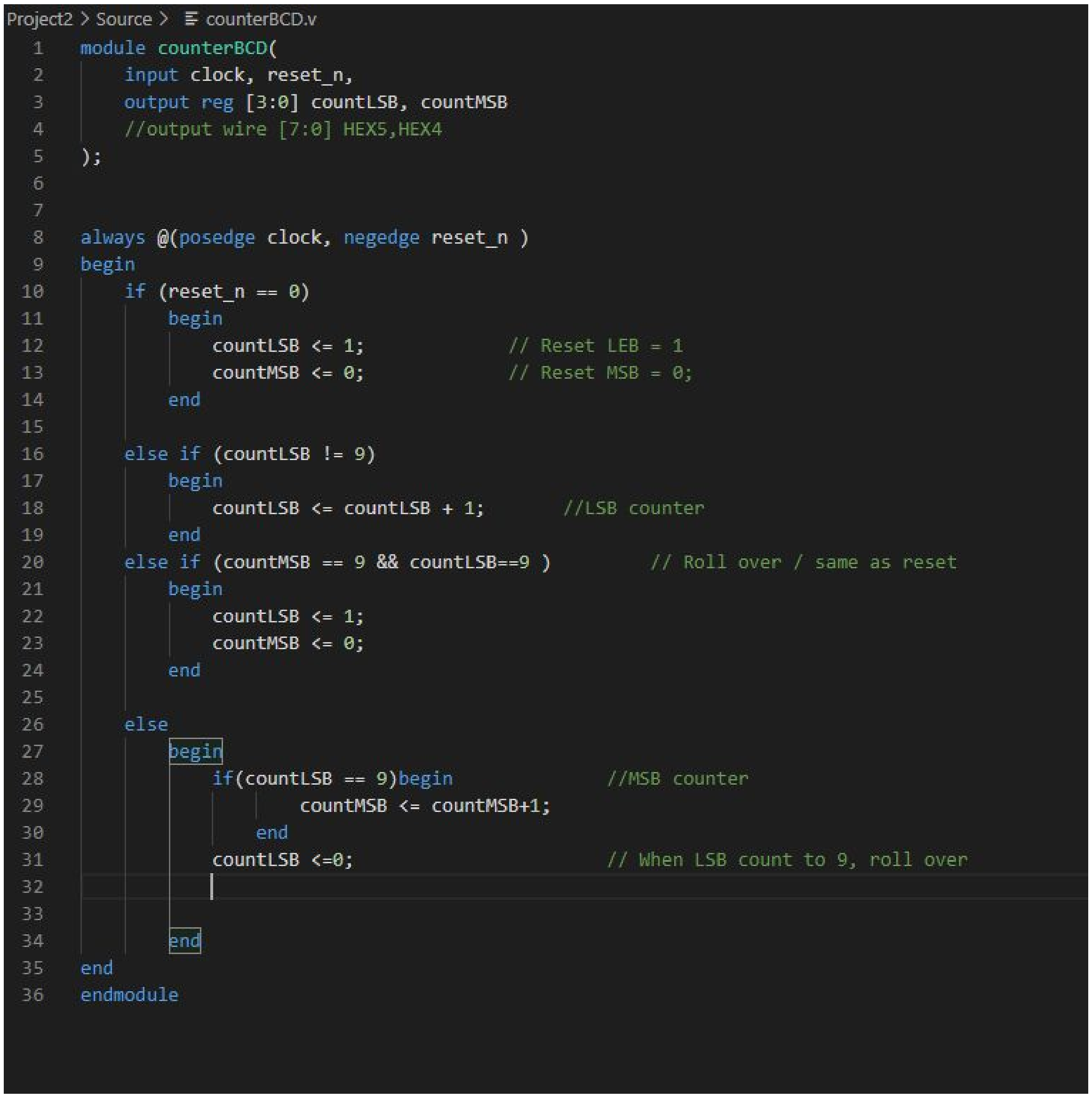
1. Project Statement

In this project, we are going to learn the synchronous design techniques,and to create testbenches that simulate this synchronous design. Generally, we are designing a calendar that displays the days of year, the month of years and the days of month on the De10-lite board. Also, for the calendar, we should be able to display the days of leap year, the month of leap years and the days of month. Moreover, we need to create a clock divider to slow down the frequency of updating. There are 2 frequencies we need to get. One is 2Hz and another one is 5Hz. And we need to Connect the display clock to LEDR[0] so we can see the LED blink at the display clock rate. At the end of the design, we should be able to switch the frequency of the clock by pressing the button KEY[1].

2. Theory of operation

In this project, we should know how to create a clock that starts the counting automatically. And the data changes at the positive edge of the tick. Also, for each clock, we must include the RESET signal, which makes effects at the negative edge of the tick, as the highest priority in the clock. In our design, the RESET signal is active low, which means it is logic true when it is in the 0 state. Moreover, we should notice that we are using non-blocking assignments (<=) at most times. A non-blocking assignment means that statements are evaluated using the variable values when the always or initial block was entered.

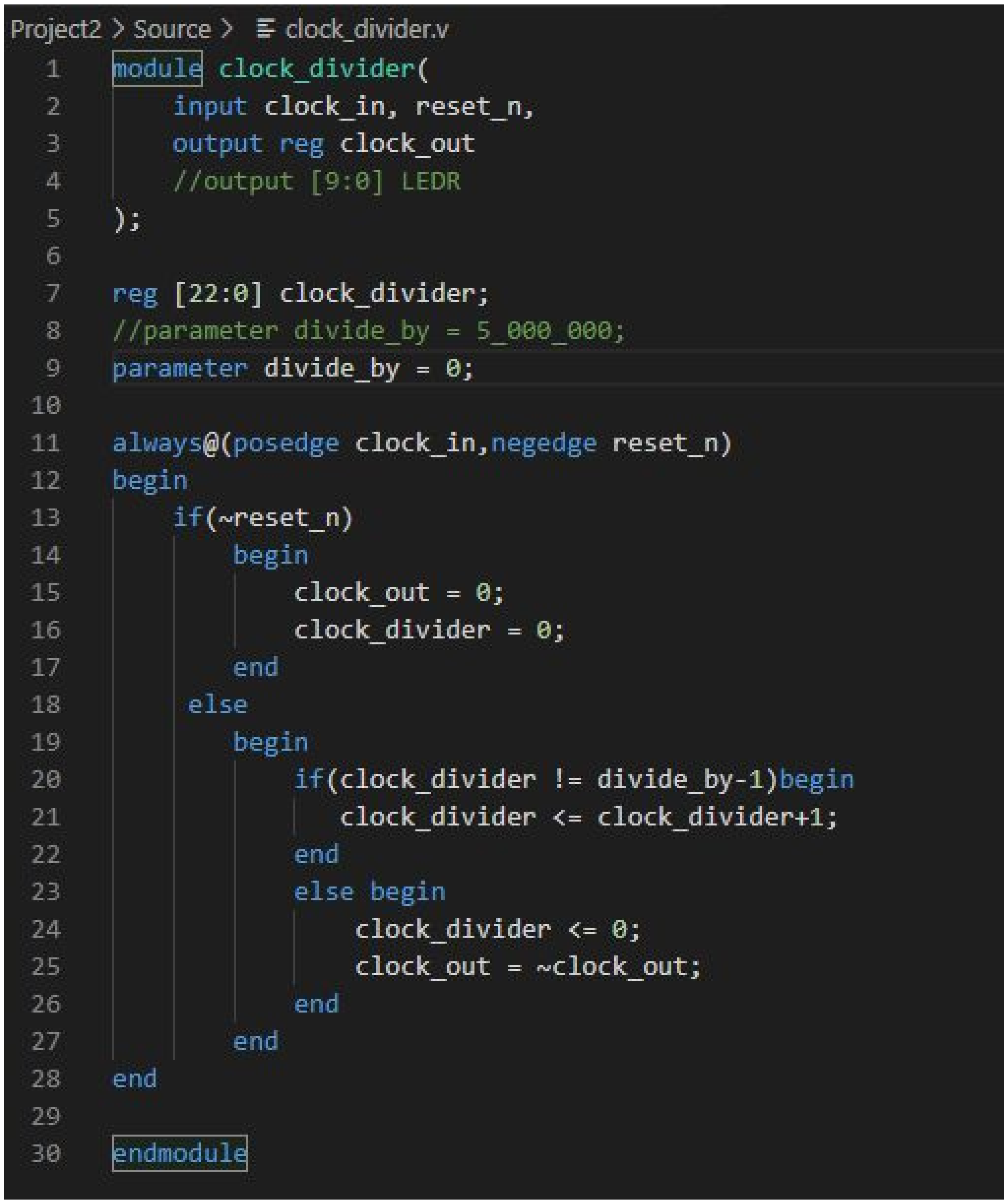
In the first requirement, we should design a counter that goes from 1 to 99 with a RESET signal, which represents the first 99 days of the year. For convenience, I create two counters, countMSBand countLSB. Each counter is 4 bit wide, so it only counts from 4'b0 to 4'b1001. And when countLSB counts to 9, countMSB will add 1 and countLSB roll back to 0. At the end, when they both count to 99, they will roll over. CountMSB will display at HEX5 and CountLSB will display at HEX4


Project2 > Source > counterBCD.v

```
module counterBCD(
    input clock, reset_n,
    output reg [3:0] countLSB, countMSB
    //output wire [7:0] HEX5,HEX4
);


always @(posedge clock, negedge reset_n )
begin
    if (reset_n == 0)
        begin
            countLSB <= 1;              // Reset LEB = 1
            countMSB <= 0;              // Reset MSB = 0;
        end

    else if (countLSB != 9)
        begin
            countLSB <= countLSB + 1;       //LSB counter
        end
    else if (countMSB == 9 && countLSB==9 )         // Roll over / same as reset
        begin
            countLSB <= 1;
            countMSB <= 0;
        end

    else
        begin
            if(countLSB == 9)begin              //MSB counter
                    countMSB <= countMSB+1;
                end
            countLSB <=0;                       // When LSB count to 9, roll over


        end
end
endmodule
```

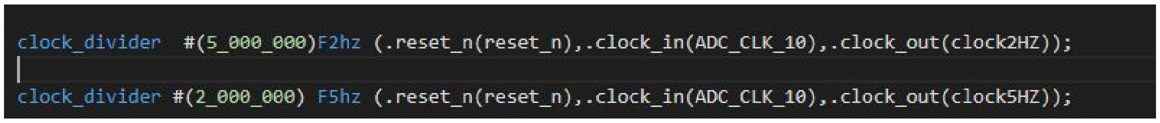
Figure1. 0-99 counter

The second major requirement, we should code a clock divider which could slow down the frequency of clock source so that we can see the updating of numbers in the HEX display. Since we are using 10M hz clock source, we should create a 23 bit wide clock divider to slow it down. So, we could get the expected clock frequency by this divider. In my design, I create a parameter divide_by which is used to slow down the clock to the expected frequency. Divide_by is 0 inside of my clock_divider file for convenience of instantiation. When I instantiate this clock_divider block, I could simply put #(expected value) in the line of my instantiation. #(expected value) will take the place of parameter divide_by inside of the clock_divider module.



```
Project2 > Source > clock_divider.v
module clock_divider(
    input clock_in, reset_n,
    output reg clock_out
    //output [9:0] LEDR
);

reg [22:0] clock_divider;
//parameter divide_by = 5_000_000;
parameter divide_by = 0;

always@(posedge clock_in,negedge reset_n)
begin
    if(~reset_n)
        begin
            clock_out = 0;
            clock_divider = 0;
        end
     else
        begin
            if(clock_divider != divide_by-1)begin
               clock_divider <= clock_divider+1;
            end
            else begin
                clock_divider <= 0;
                clock_out = ~clock_out;
            end
        end
end

endmodule
```

Figure 2. Clock_divider

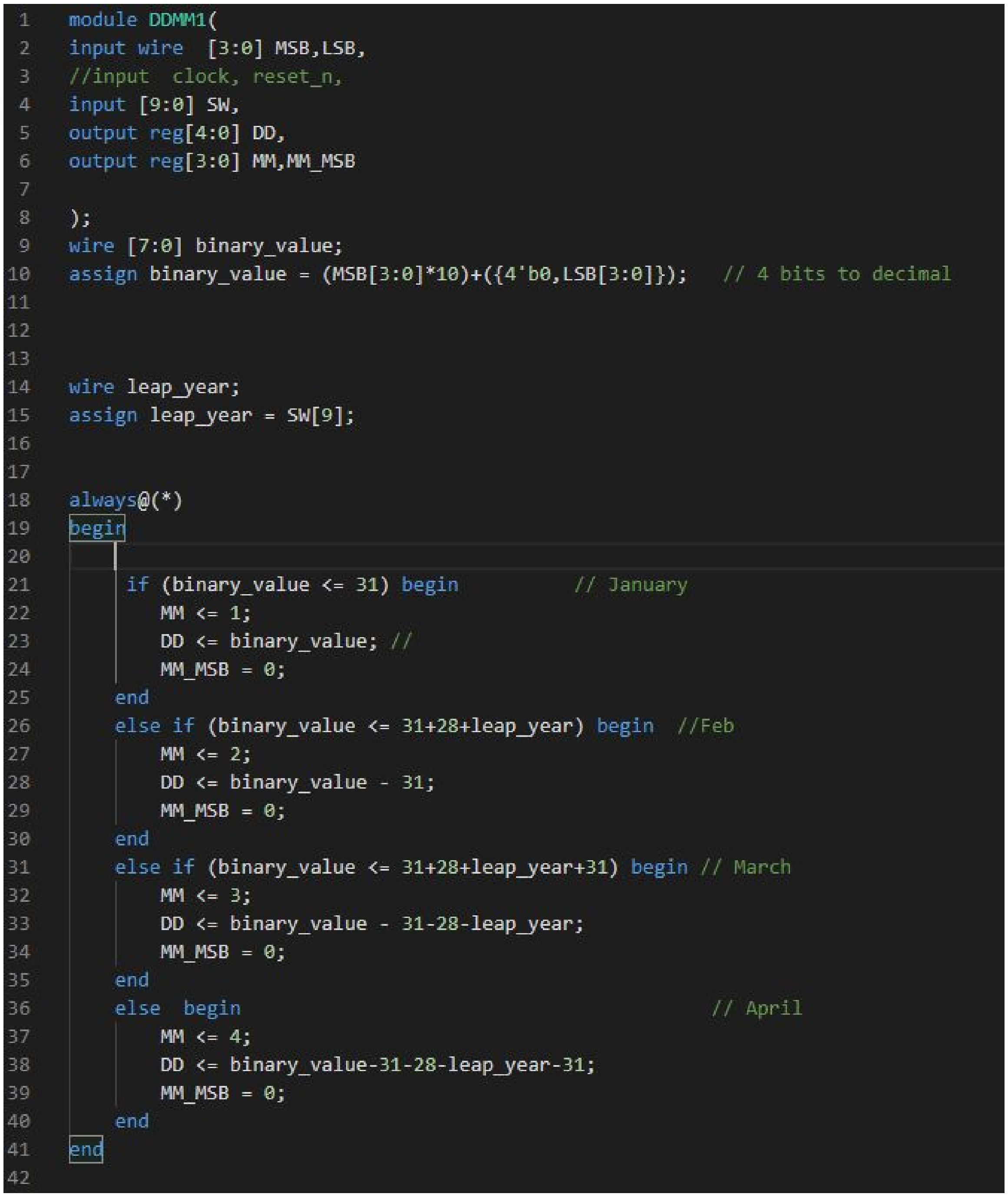


```
clock_divider  #(5_000_000)F2hz (.reset_n(reset_n),.clock_in(ADC_CLK_10),.clock_out(clock2HZ));

clock_divider #(2_000_000) F5hz (.reset_n(reset_n),.clock_in(ADC_CLK_10),.clock_out(clock5HZ));
```

Figure 3. Clock_divider_instantiation

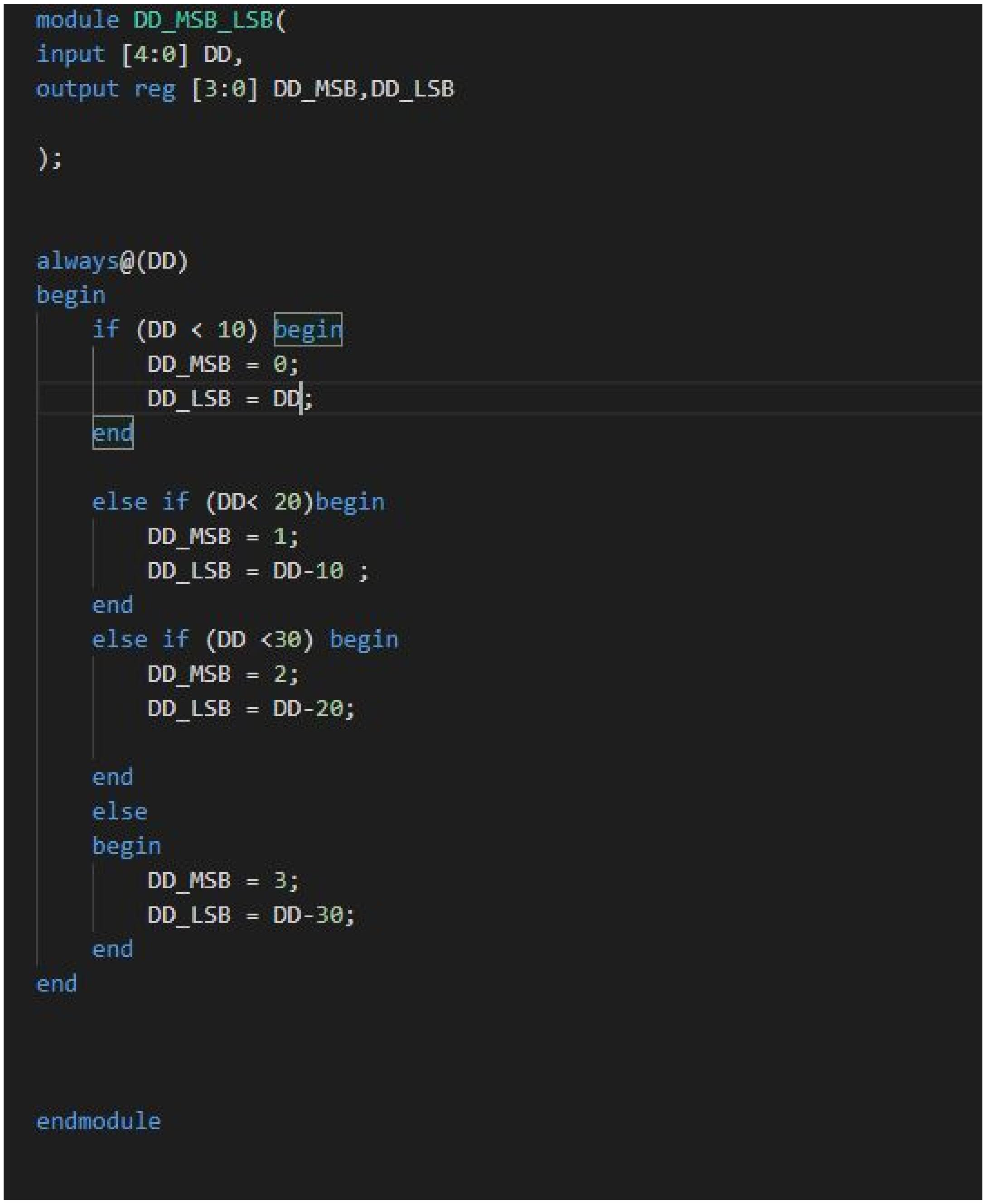
The last requirement is displaying the months of year and the days of month in HEX3, HEX2, HEX1 and HEX0. In this block, I make a binary to decimal decoder, which could concatenate countMSB and countLSB to decimal in binary. With this binary_value, we could make multiple if statements to determine which month are the counter in at that time. So we could assign the month number to month displays.

Moreover, in another block, I create a simple calculation to calculate the days of month by multiple if statements to determine the days_MSB and the days_LSB.



```
module DDMM1(
input wire  [3:0] MSB,LSB,
//input  clock, reset_n,
input [9:0] SW,
output reg[4:0] DD,
output reg[3:0] MM,MM_MSB

);
wire [7:0] binary_value;
assign binary_value = (MSB[3:0]*10)+({4'b0,LSB[3:0]});   // 4 bits to decimal



wire leap_year;
assign leap_year = SW[9];


always@(*)
begin

     if (binary_value <= 31) begin          // January
        MM <= 1;
        DD <= binary_value; //
        MM_MSB = 0;
    end
    else if (binary_value <= 31+28+leap_year) begin  //Feb
        MM <= 2;
        DD <= binary_value - 31;
        MM_MSB = 0;
    end
    else if (binary_value <= 31+28+leap_year+31) begin // March
        MM <= 3;
        DD <= binary_value - 31-28-leap_year;
        MM_MSB = 0;
    end
    else  begin                                  // April
        MM <= 4;
        DD <= binary_value-31-28-leap_year-31;
        MM_MSB = 0;
    end
end

```

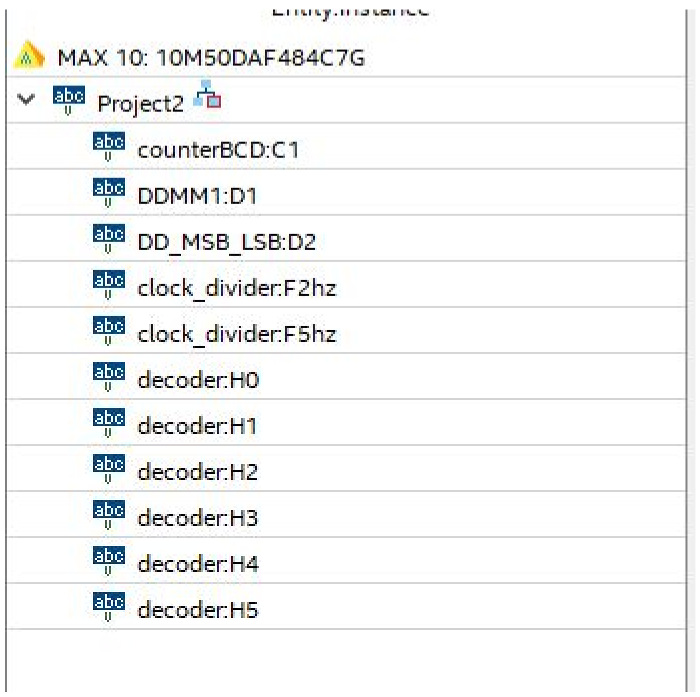
Figure 4. DDMM_month



```
module DD_MSB_LSB(
input [4:0] DD,
output reg [3:0] DD_MSB,DD_LSB

);


always@(DD)
begin
    if (DD < 10) begin
        DD_MSB = 0;
        DD_LSB = DD;
    end

    else if (DD< 20)begin
        DD_MSB = 1;
        DD_LSB = DD-10 ;
    end
    else if (DD <30) begin
        DD_MSB = 2;
        DD_LSB = DD-20;

    end
    else
    begin
        DD_MSB = 3;
        DD_LSB = DD-30;
    end
end



endmodule
```

Figure 5. DD_MSB_LSB

3. Hierarchy of My project source files



Entity:Instance

MAX 10: 10M50DAF484C7G

Project2

counterBCD:C1

DDMM1:D1

DD_MSB_LSB:D2

clock_divider:F2hz

clock_divider:F5hz

decoder:H0

decoder:H1

decoder:H2

decoder:H3

decoder:H4

decoder:H5

Figure 6. Hierarchy

4. Block diagram

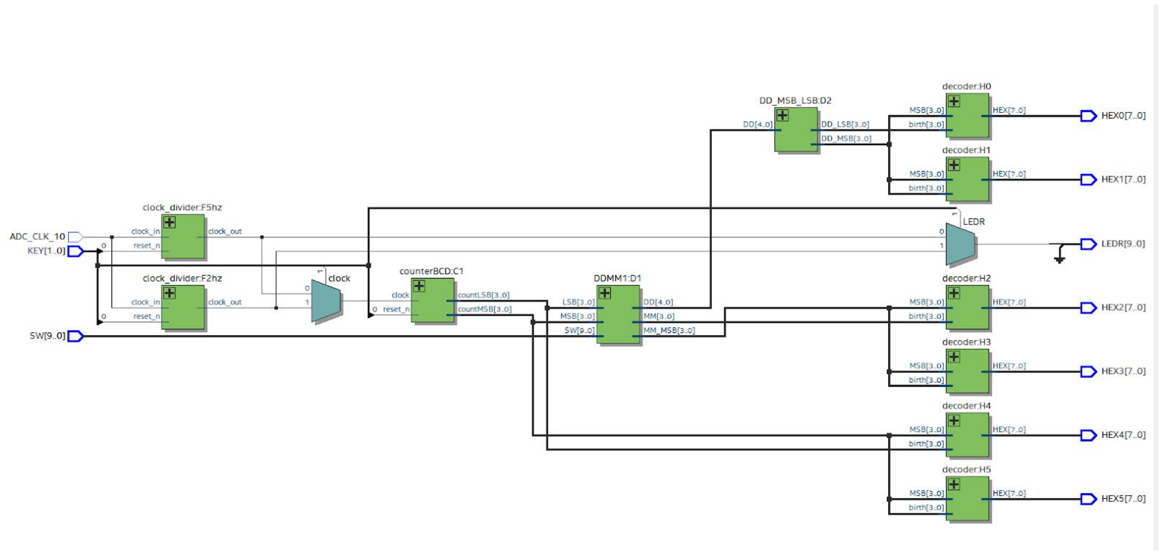
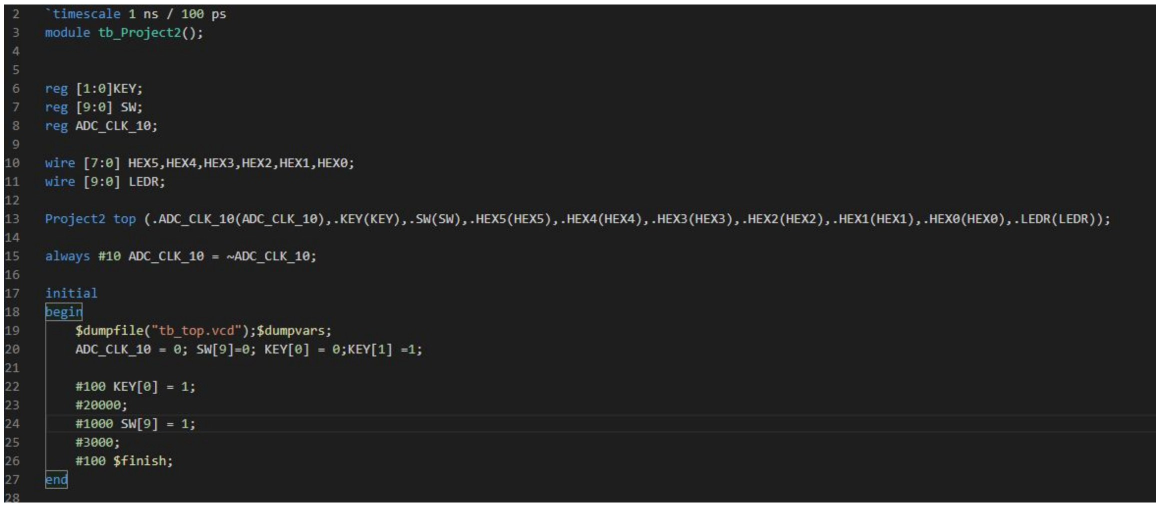


Figure 7. Block diagram

5. Description of Testbench operation



```
2   `timescale 1 ns / 100 ps
3   module tb_Project2();
4
5
6   reg [1:0]KEY;
7   reg [9:0] SW;
8   reg ADC_CLK_10;
9
10  wire [7:0] HEX5,HEX4,HEX3,HEX2,HEX1,HEX0;
11  wire [9:0] LEDR;
12
13  Project2 top (.ADC_CLK_10(ADC_CLK_10),.KEY(KEY),.SW(SW),.HEX5(HEX5),.HEX4(HEX4),.HEX3(HEX3),.HEX2(HEX2),.HEX1(HEX1),.HEX0(HEX0),.LEDR(LEDR));
14
15  always #10 ADC_CLK_10 = ~ADC_CLK_10;
16
17  initial
18  begin
19      $dumpfile("tb_top.vcd");$dumpvars;
20      ADC_CLK_10 = 0; SW[9]=0; KEY[0] = 0;KEY[1] =1;
21
22      #100 KEY[0] = 1;
23      #20000;
24      #1000 SW[9] = 1;
25      #3000;
26      #100 $finish;
27  end
28
```

Figure 8. top_tb.v

In the testbench of top.v module, I only instantiate the top module which is Project2 in line 13. Project 2.v includes all the counter, clock divider and decoder modules. For the simulation of a clock like this, we must create an always block to flip the state of clock source in each 10 times timescale delay, like in line 15, to start the counting. Then, within the initial clock, we must initialize everything that is related to our operation, such as clock source, SW and SW, which could make the simulation work properly. Moreover, we also need to reset the clock at the beginning of the counter in order to clean everything before the counting. For the counting, we simply put #(times) to start it. During some delay, we also could simulate some operations we want, like flipping the switch, pressing the KEY, etc.

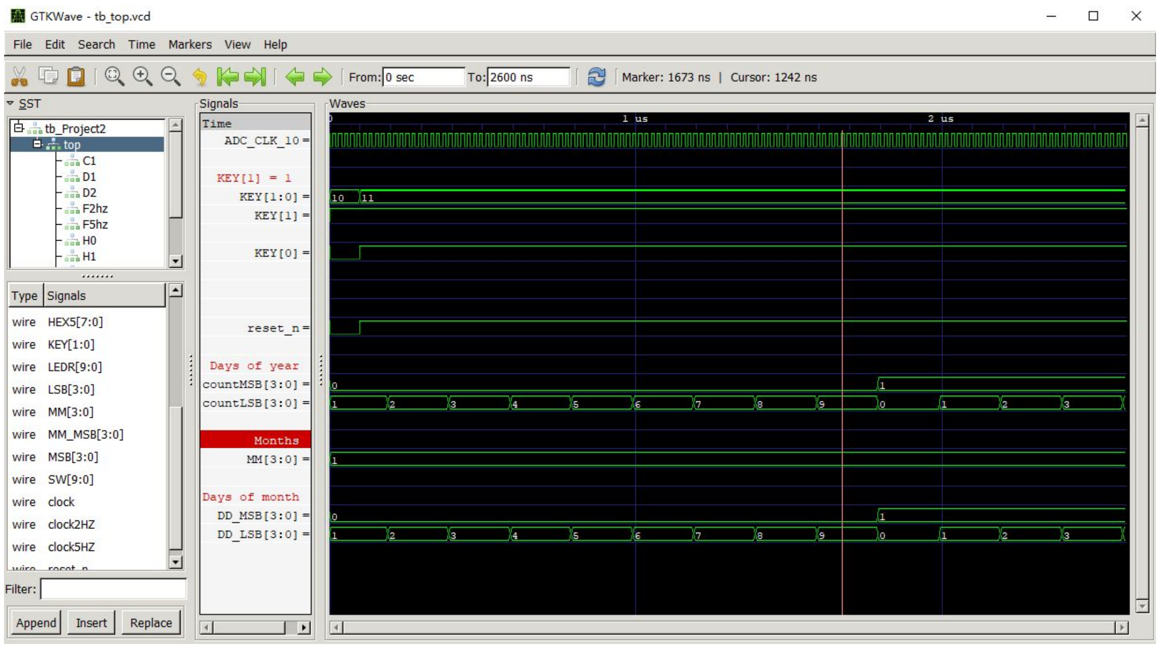
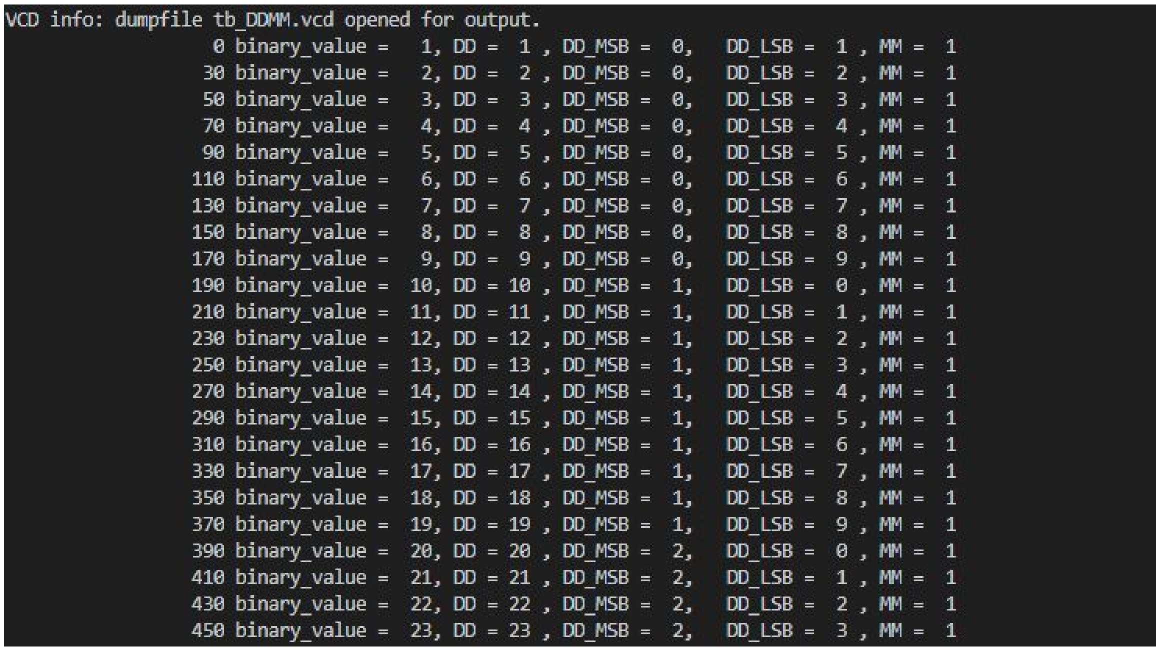


Figure 9. Gtkwave_top

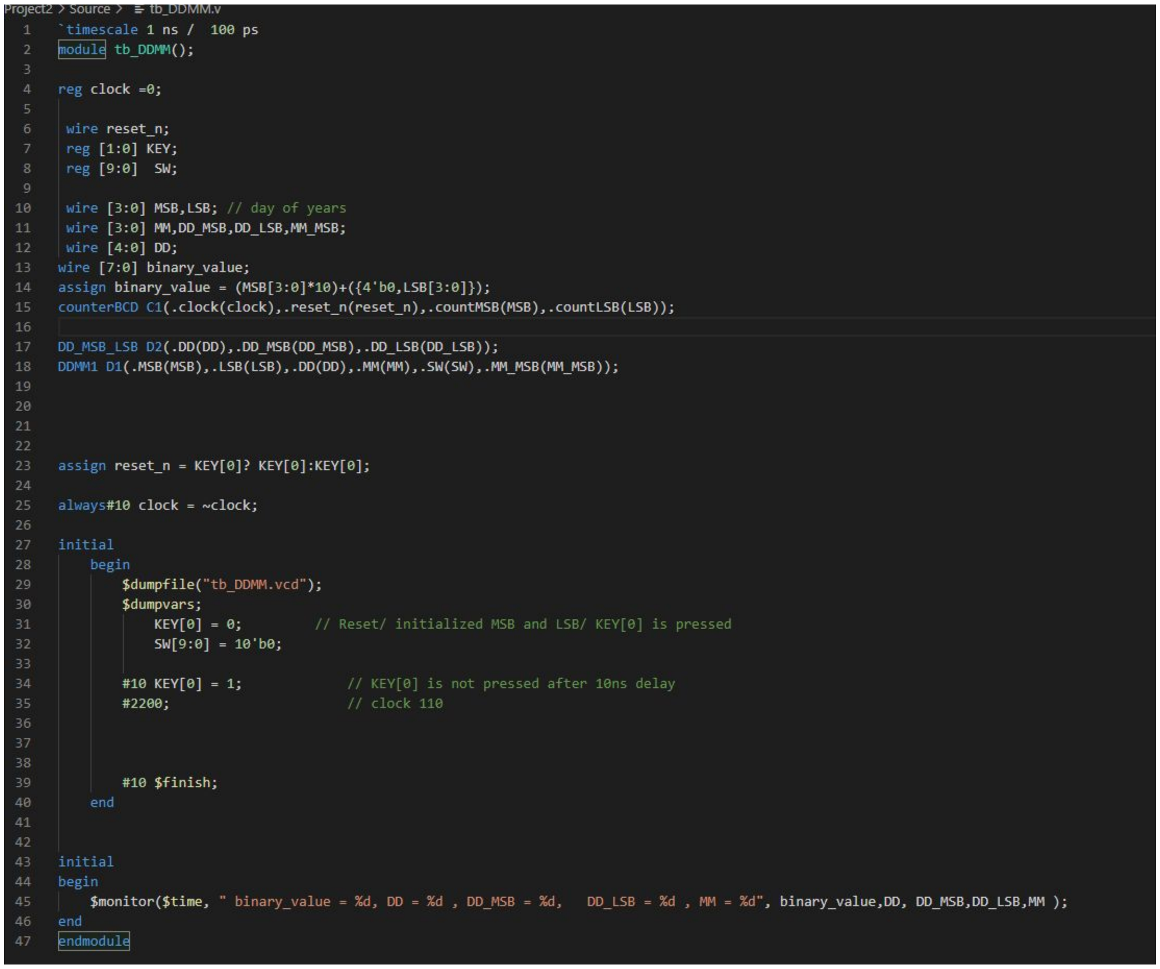
The figure above shows the beginning of the counter, which includes the days of year, months of year and the days of month with the RESET signal.



```
VCD info: dumpfile tb_DDMM.vcd opened for output.
                   0 binary_value =   1, DD =  1 , DD_MSB =  0,   DD_LSB =  1 , MM =  1
                  30 binary_value =   2, DD =  2 , DD_MSB =  0,   DD_LSB =  2 , MM =  1
                  50 binary_value =   3, DD =  3 , DD_MSB =  0,   DD_LSB =  3 , MM =  1
                  70 binary_value =   4, DD =  4 , DD_MSB =  0,   DD_LSB =  4 , MM =  1
                  90 binary_value =   5, DD =  5 , DD_MSB =  0,   DD_LSB =  5 , MM =  1
                 110 binary_value =   6, DD =  6 , DD_MSB =  0,   DD_LSB =  6 , MM =  1
                 130 binary_value =   7, DD =  7 , DD_MSB =  0,   DD_LSB =  7 , MM =  1
                 150 binary_value =   8, DD =  8 , DD_MSB =  0,   DD_LSB =  8 , MM =  1
                 170 binary_value =   9, DD =  9 , DD_MSB =  0,   DD_LSB =  9 , MM =  1
                 190 binary_value =  10, DD = 10 , DD_MSB =  1,   DD_LSB =  0 , MM =  1
                 210 binary_value =  11, DD = 11 , DD_MSB =  1,   DD_LSB =  1 , MM =  1
                 230 binary_value =  12, DD = 12 , DD_MSB =  1,   DD_LSB =  2 , MM =  1
                 250 binary_value =  13, DD = 13 , DD_MSB =  1,   DD_LSB =  3 , MM =  1
                 270 binary_value =  14, DD = 14 , DD_MSB =  1,   DD_LSB =  4 , MM =  1
                 290 binary_value =  15, DD = 15 , DD_MSB =  1,   DD_LSB =  5 , MM =  1
                 310 binary_value =  16, DD = 16 , DD_MSB =  1,   DD_LSB =  6 , MM =  1
                 330 binary_value =  17, DD = 17 , DD_MSB =  1,   DD_LSB =  7 , MM =  1
                 350 binary_value =  18, DD = 18 , DD_MSB =  1,   DD_LSB =  8 , MM =  1
                 370 binary_value =  19, DD = 19 , DD_MSB =  1,   DD_LSB =  9 , MM =  1
                 390 binary_value =  20, DD = 20 , DD_MSB =  2,   DD_LSB =  0 , MM =  1
                 410 binary_value =  21, DD = 21 , DD_MSB =  2,   DD_LSB =  1 , MM =  1
                 430 binary_value =  22, DD = 22 , DD_MSB =  2,   DD_LSB =  2 , MM =  1
                 450 binary_value =  23, DD = 23 , DD_MSB =  2,   DD_LSB =  3 , MM =  1
```

Figure 10. Txt_output

Figure 10 shows the text output in the terminal which shows the counting of days of year (DD or binary_value), months of year(MM) and the days of month (DD_MSB and DD_LSB). This simulation uses another testbench module shown in Figure 11.



Project2 > Source > tb_DDMM.v

```verilog
`timescale 1 ns /  100 ps
module tb_DDMM();

reg clock =0;

 wire reset_n;
 reg [1:0] KEY;
 reg [9:0]  SW;

 wire [3:0] MSB,LSB; // day of years
 wire [3:0] MM,DD_MSB,DD_LSB,MM_MSB;
 wire [4:0] DD;
wire [7:0] binary_value;
assign binary_value = (MSB[3:0]*10)+({4'b0,LSB[3:0]});
counterBCD C1(.clock(clock),.reset_n(reset_n),.countMSB(MSB),.countLSB(LSB));

DD_MSB_LSB D2(.DD(DD),.DD_MSB(DD_MSB),.DD_LSB(DD_LSB));
DDMM1 D1(.MSB(MSB),.LSB(LSB),.DD(DD),.MM(MM),.SW(SW),.MM_MSB(MM_MSB));




assign reset_n = KEY[0]? KEY[0]:KEY[0];

always#10 clock = ~clock;

initial
    begin
        $dumpfile("tb_DDMM.vcd");
        $dumpvars;
            KEY[0] = 0;         // Reset/ initialized MSB and LSB/ KEY[0] is pressed
            SW[9:0] = 10'b0;

        #10 KEY[0] = 1;             // KEY[0] is not pressed after 10ns delay
        #2200;                      // clock 110



        #10 $finish;
    end


initial
begin
    $monitor($time, " binary_value = %d, DD = %d , DD_MSB = %d,   DD_LSB = %d , MM = %d", binary_value,DD, DD_MSB,DD_LSB,MM );
end
endmodule
```

Figure 11. tb_DDMM.v

6. Summary

This project gives me a better understanding of clock and simulation. Like, I understand the syntax of creating the clock testbench and clock. Also, the gtkwave simulation helps me to better visualize the process of data transmitting, such as the positive edge and negative edge. Meanwhile, this project also teaches me how to create a clock divider to slow down the clock.

I'm feeling ok with this project because everything is working properly on my board. Especially, the leap year counting is working whenever I flip the SW[9] during the counting. There is 1 day difference whenever the SW[9] is flipped. And, the KEY[1] is also working properly whenever I want to accelerate the counting of the clock. Last but not least, this project really gives me a better understanding of all the concepts about timer or clock, and the ability to read the gtkwave output.