EEET7063C
Gate Arrays

S3834951
Mitchell Reynolds

Student Project Report

Project outline

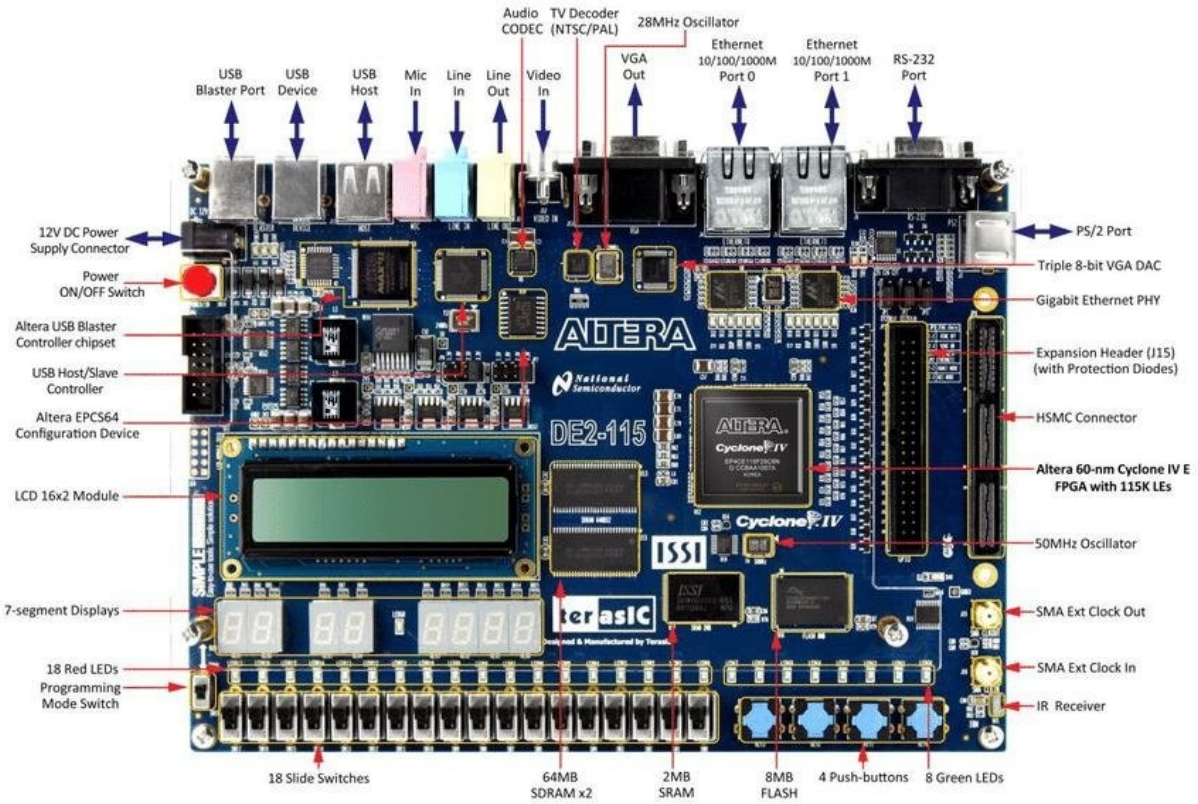
This project consists of a 24-hour clock, programmed in Quartus using Verilog language, and displayed on an Altera DE2-115 project board (which uses an EP4CE115F29C7 FPGA).

The 24-hour clock has outputs for seconds, minutes and hours which are each sent to two-digit seven-segment displays.

It also has a button to reset the clock (set everything back to 00), and buttons to increment the hours and minutes.

An important note is that the seven-segment displays are common anode, so they are already powered and need to go low for the segments to turn on.
The buttons also already have power running through them, and when pressed they go to zero.
This means that we need to use common anode logic, and we need to consider that a button press equals zero when we program functions to the buttons.

Image of Altera DE2-115 FPGA board:



Programming

Our Verilog program consists of multiple blocks and separate tasks, which I will explain below.

Here we are defining all of the inputs, outputs and registers:

```
module Projectv2(CLOCK_50,resetbutton, adjust, adjhour, adjmin, clk,sec,min,tens,units,hour,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,d1, d2,d3,d4,d5,d6);
   input CLOCK_50, resetbutton, adjust, adjhour, adjmin;
   output clk;
   output [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5;
   output [3:0] d1, d2,d3,d4,d5,d6;
   output [3:0] tens,units;

   output [5:0] sec, min,hour;

   reg [26:0]counter;
   reg [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5;
   reg [5:0] sec, min,hour;
```

It is important to note that we are assigning 7 bit outputs to HEX0, HEX1 etc because these are the 7-segment displays on the board and they are pre-defined with these names.
Also of note is that there are only 5 inputs, our four buttons as well as CLOCK_50 which is the FPGA boards own internal clock (which is being used in the counter section of our program).

The next section is the counter:

```
   initial
   begin
      hour=23;
      min=59;
      sec=50;
   end

   // Counter
   always @ (posedge CLOCK_50)
   begin
      if (counter == 25000000)
      begin
            clk=!clk;
            counter=0;
      end
      else
            counter=counter + 1;
   end
```

At the top here is an initial value that we are assigning to the 7-segment display upon programming.
Below is our counter program which increments our counter for every 25 million clock cycles of the FPGA board's internal clock. This works to count every second because the internal clock is 25 MHz.

Next is the 24 hour clock section of the program:

```
36    // 24 Hour Clock
37    always @ (posedge clk)
38    begin
39       if (resetbutton==0) // if resetbutton is pressed for 1second, set everything to 00
40       begin
41          hour=00;
42          min=00;
43          sec=00;
44       end
45
46       sec=sec+1; // count seconds
47       if (sec == 60)
48       begin
49          min = min+1; // if seconds is 60, add 1 to minute and set sec to 00
50          sec =0;
51       end
52       if (min == 60)
53       begin
54           hour = hour+1; // if min is 60, add 1 to hour & set min to 00
55           min =0;
56       end
57       if (hour == 24)
58       begin
59          hour = 0; // if hour = 24, set hour to 0
60       end
```

If the reset button is pressed, every value will revert back to 00 and the clock will start again counting from 00:00:00.
The seconds counter will always continue being incremented ("sec=sec+1").
But the minute counter will only increment when seconds equals 60, and then it will reset seconds to 00.
Similarly, the hour counter will only increment if minutes equal 60, and then it will reset the minutes to 00. The hour counter goes back to 00 once it reaches 24.

This section is calling each of the tasks that will appear later:

```
62        bcdconverter(sec,d2,d1);
63        bcdconverter(min,d4,d3);
64        bcdconverter(hour,d6,d5);
65        AdjustHours(adjust, adjhour);
66        AdjustMinutes(adjust, adjmin);
67
68
69        //bcdconverter(sec,tens,units);
70        Display_Digit(d2,HEX1);
71        Display_Digit(d1,HEX0);
72
73        //bcdconverter(min,tens,units);
74        Display_Digit(d4,HEX3);
75        Display_Digit(d3,HEX2);
76
77        //bcdconverter(hour,tens,units);
78        Display_Digit(d6,HEX5);
79        Display_Digit(d5,HEX4);
80        end
```

The below section is our Binary to BCD converter:

```
83      // BCD Converter Task
84      task bcdconverter;
85            integer i;
86            input [5:0] time_in;
87            output [3:0] MSD;
88            output [3:0]LSD;
89            reg [13:0] shift;
90            shift[5:0] = time_in[5:0];
91
92            for (i=0; i<5; i = i+1)
93            begin
94               shift = shift << 1;
95
96                  if(shift[9:6] >= 5)
97                     shift[9:6] = shift[9:6]+3;
98
99                  if(shift[13:10] >= 5)
100                    shift[13:10] = shift[13:10]+3;
101           end
102
103              shift = shift <<1;
104              MSD[3:0] = shift[13:10];
105              LSD[3:0] = shift[9:6];
106        endtask
```

This code creates a 14 bit shift register. At the first 6 bits of the shift register the time value is placed in binary (the biggest number used in this program is 60, so 6 bits are enough).

The core principle is that we need to convert from a 6 bit number into two four bit numbers, named Most Significant Digit (MSD) and Least Significant Digit (LSD) in the code.
We have defined sections of the shift register which correlate to units (6 bits), tens (3 bits) and hundreds (four bits).
To do this, we need to shift the input bits, adding a 1, until their value reaches 5. Then we add 3 to the number. We continue shifting and if each section of the shift register is greater than or equal to 5, we will add 3 to the number.
Then after 8 shifts and the appropriate amount of additions will see that the hundreds and tens sections will equate to the individual binary coded numbers (for two digits) that equal the original 6 bit input number.

Our Hex converter task simply uses the common anode 7-segment display logic that we are now familiar with:

```
//Hex Converter Task
task Display_Digit;
   input [3:0] bcd_in;
   output [6:0] seg_out;

   begin
      case(bcd_in)
         (0) : seg_out = 7'b1000000;
         (1) : seg_out = 7'b1111001;
         (2) : seg_out = 7'b0100100;
         (3) : seg_out = 7'b0110000;
         (4) : seg_out = 7'b0011001;
         (5) : seg_out = 7'b0010010;
         (6) : seg_out = 7'b0000010;
         (7) : seg_out = 7'b1111000;
         (8) : seg_out = 7'b0000000;
         (9) : seg_out = 7'b0010000;
        default : seg_out = 7'b1111111;
      endcase
   end
endtask
```

Note the 7 bit output.

Below are my adjust tasks for minutes and hours. These tasks use if statements to determine whether both the adjust button and the adjustminute or adjusthour buttons have been pressed (indicating a 0), and if so, it will increment the appropriate output:

```
// Adjust Hour Task
task AdjustHours;
input adjust, adjhour;
   if ((adjust==0) && (adjhour==0))// if both buttons are held for more than 1 sec hours will be incremented
      begin
         hour = hour+1;
      end
endtask

// Adjust Minutes Task
task AdjustMinutes;
input adjust, adjmin;
   if ((adjust==0) && (adjmin==0)) // if both buttons are held for more than 1 sec min will be incremented
      begin
         min = min+1;
      end
endtask
```

The buttons need to be held as they will only increment on each 1 second clock cycle, which can be seen in my video linked below.

Link to video of testing:

https://photos.app.goo.gl/U2P2hLs5thozmsKA6

Conclusion

This project was very challenging, but the immediacy of being able to program in Quartus and see the changes on the board made for a very rewarding and interesting experience when debugging.

I faced some challenges with my board, namely the display tasks and issues with the numbers not displaying on the 7-segment display even though they could be shown in binary on LEDs. Eventually these were overcome with persistence, and the debugging experience was very fruitful, however frustrating it was!

Using a development board like this is a great tool for the intuitive nature where you can make one alteration in the code and then check how this effects the FPGA's operation only a few seconds later after reprogramming the board.

I would like to thank Sukhvir Judge for his patience in helping us debug, as well as Constantinos and Dmitri who offered their advice and experience during the process.