FPGA Application Midterm Project

CYEE 10828241 Chen Da-Chuan

April 17, 2023

Contents

List of Figures

List of Tables

1 Project: Shot the Clock

1.1 Objective

This project aims to make a 24-hour clock that looks as if it can be destroyed by shooting it with a laser.

Table 1: Utility

No.	Utility
1.	24 hours clock
2.	Adjust hour, minute, second
3.	Fire the laser to destroy the clock

1.2 Operation

24-hour clock is displayed with 6 7-segment displays. User can manually adjust its hour, minute, and second numbers by switching switches 9 to 4. It can be reset to 00:00:00 by pressing switch 0. The clock will be automatically updated every second.

The gun is animated with 10 LEDs above the 10 switches. Pressing the button/key 0 will fire the laser. When the bullet reaches the 7-segment displays, the right-most displaying number will be dimmed. This can be repeated until all 6 numbers are dimmed. Then the user can manually turn all 6 numbers back to normal by pressing button/key 1. The clock won't be stopped during this process.

Table 2: Control

ID	Function
SW9	Increase hour
SW8	Decrease hour
SW7	Increase minute
SW6	Decrease minute
SW5	Increase second
SW4	Decrease second
SW0	Reset to 00:00:00
KEY0	Fire the laser
KEY1	Turn all numbers back to normal

1.3 Code

Only codes that are modified from provided sample code are shown here.

1. Fig 1 Line51-55: Implement switch signal debounce module.
2. Fig 1 Line57-62: Implement clock signal generation module.
3. Fig 1 Line64-82: Implement modified eclock module.
4. Fig 1 Line84-93: Implement 7-segment display module.
5. Fig 1 Line95-100: Implement custom meteor light module.
6. Fig 2 Line11-11: Define 10 digit register for light counter. Each digit is used to light or dimm individual LED.
7. Fig 2 Line13-13: Assign a variable to carry the remainder of the count by 10.
8. Fig 2 Line17-20: Reset "count" to 1 when triggered. (1 in "count" is used for lighting LED)
9. Fig 2 Line21-24: If "shot" is triggered, shift count digits left by 1. (Meteor only moves left when "shot" is triggered)
10. Fig 2 Line25-28: If "shot" is not triggered, reset "count" to 1.
11. Fig 3 Line 32-37: Assign 6 digits for clock, each digit is used to light or dimm individual 7-segment display.
12. Fig 3 Line39-49: If "shot" is triggered, the "count_shot" will increase by 1. If "reset", then "count_shot" will be reset to 0.
13. Fig 3 Line51-86: If "shotRst" is triggered, light all 6 digits back up. If not, dimm the corresponding digit according to "count_shot".
14. Fig 4 Line91-96: If "rst" is not triggered, reset 6 digits to 0.
15. Fig 4 Line98-105: If "hour_add" or "hour_sub" is triggered, increase or decrease the hour number by 1.
16. Fig 4 Line107-112: If the counter of hour, minute, or second reaches maximum value, reset all of them to 0.
17. Fig 4 Line114-121: If "min_add" or "min_sub" is triggered, increase or decrease the minute number by 1.
18. Fig 4 Line123-127: If minute counter reaches maximum value, reset it to 0 and increase hour counter by 1.
19. Fig 4 Line129-136: If "sec_add" or "sec_sub" is triggered, increase or decrease the second number by 1.
20. Fig 4 Line138-142: If second counter reaches maximum value, reset it to 0 and increase minute counter by 1.
21. Fig 4 Line144-145: Second counter addes 1 every second.

```verilog
//=========================================================
//  Structural coding
//=========================================================

SW_DEBOUNCE usw(
   .clk(clock_100ms),
   .iSW(SW),
   .oSW_d(SW_d)
);

clock_all uclock(
   .clk(MAX10_CLK1_50),
   .clock_100ms(clock_100ms),
   .clock_1s(clock_1s),
   .clock_10ms(clock_10ms)
);

eclock u_myclock(
   .clk(clock_1s),
   .rst(!SW_d[0]),
   .shot(KEY[0]),
   .shotClk(clock_100ms),
   .shotRst(KEY[1]),
   .hour_add(SW_d[9]),
   .hour_sub(SW_d[8]),
   .hour_tens(hour_tens),
   .hour_digits(hour_digits),
   .min_add(SW_d[7]),
   .min_sub(SW_d[6]),
   .min_tens(min_tens),
   .min_digits(min_digits),
   .sec_add(SW_d[5]),
   .sec_sub(SW_d[4]),
   .sec_tens(sec_tens),
   .sec_digits(sec_digits)
);

SEG7_LUT_6 u_seg(
   .oSEG0(HEX0),
   .oSEG1(HEX1),
   .oSEG2(HEX2),
   .oSEG3(HEX3),
   .oSEG4(HEX4),
   .oSEG5(HEX5),
   .iDIG ({hour_tens, hour_digits, min_tens, min_digits, sec_tens, sec_digits }),
   .iDot ({5'b0, clock_1s})
);

ledMeteor u_meteor(
   .clk(clock_10ms),
   .rst(!SW_d[0]),
   .shot(KEY[0]),
   .LEDR(LEDR)
);

endmodule
```

Figure 1: Top-level code

```verilog
module ledMeteor(
      clk,
      rst,
      shot,
      LEDR
);

input        clk, rst, shot;
output [9:0] LEDR;

reg [9:0] count = 1'b1;

assign LEDR     = count;

always @ (posedge clk or negedge rst)
begin
   if (!rst)
   begin
      count <= 1'b1;
   end
   else if (!shot)
   begin
      count <= count << 1;
   end
   else if (shot)
   begin
      count <= 1'b1;
   end
end

endmodule
```

Figure 2: LED meteor code

```verilog
// binary to decimal
assign hour_tens   = (count_shotLED6) ? 4'he : count_hour / 10;
assign hour_digits = (count_shotLED5) ? 4'he : count_hour % 10;
assign min_tens    = (count_shotLED4) ? 4'he : count_min / 10;
assign min_digits  = (count_shotLED3) ? 4'he : count_min % 10;
assign sec_tens    = (count_shotLED2) ? 4'he : count_sec / 10;
assign sec_digits  = (count_shotLED1) ? 4'he : count_sec % 10;

   always @ (posedge shot or negedge shotRst)
   begin
      if (!shotRst)
      begin
         count_shot <= 0;
      end
      else
      begin
         count_shot <= count_shot + 1;
      end
   end

   always @ (posedge shotClk or negedge shotRst)
   begin
      if (!shotRst)
      begin
         count_shotLED1 <= 0;
         count_shotLED2 <= 0;
         count_shotLED3 <= 0;
         count_shotLED4 <= 0;
         count_shotLED5 <= 0;
         count_shotLED6 <= 0;
      end
      else if (count_shot == 1)
      begin
         count_shotLED1 <= 1;
      end
      else if (count_shot == 2)
      begin
         count_shotLED2 <= 1;
      end
      else if (count_shot == 3)
      begin
         count_shotLED3 <= 1;
      end
      else if (count_shot == 4)
      begin
         count_shotLED4 <= 1;
      end
      else if (count_shot == 5)
      begin
         count_shotLED5 <= 1;
      end
      else if (count_shot == 6)
      begin
         count_shotLED6 <= 1;
      end
   end
```

Figure 3: eclock-1 code

```verilog
   always @ (posedge clk or negedge rst)
   begin
      // reset
      if (!rst)
      begin
         count_hour  <= 0;
         count_min <= 0;
         count_sec <= 0;
      end
      // hour operation
      else if (hour_add)
      begin
         count_hour <= count_hour + 1;
      end
      else if (hour_sub )
      begin
         count_hour <= count_hour - 1;
      end
      // hour maximum
      else if (count_hour >= 23 && count_min >= 59 && count_sec >= 59 )
      begin
         count_min <= 0;
         count_hour <= 0;
         count_sec <= 0;
      end
      // minute operation
      else if (min_add)
      begin
         count_min <= count_min + 1;
      end
      else if (min_sub)
      begin
         count_min <= count_min - 1;
      end
      // minute maximum
      else if (count_min >= 59)
      begin
         count_min <= 0;
         count_hour <= count_hour + 1;
      end
      // second operation
      else if (sec_add)
      begin
         count_sec <= count_sec + 2;
      end
      else if (sec_sub)
      begin
         count_sec <= count_sec - 2;
      end
      // second maximum
      else if (count_sec >= 59)
      begin
         count_sec <= 0;
         count_min <= count_min + 1;
      end
      // normal operation
      else
         count_sec <= count_sec + 1;
   end
```

Figure 4: eclock-2 code

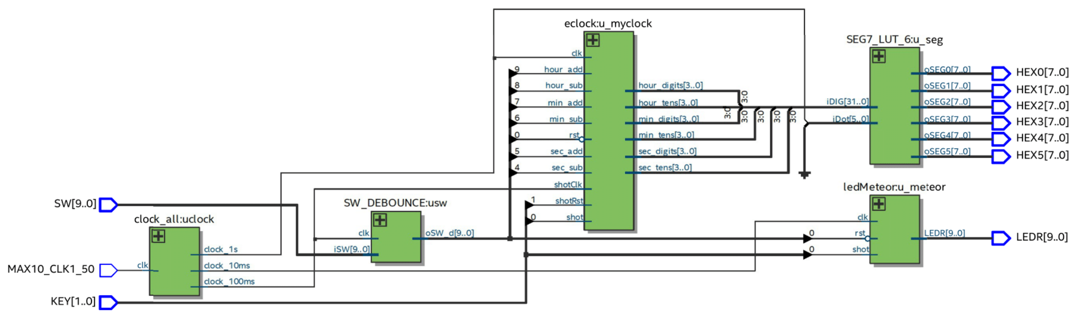

Figure 5: Top-level diagram

1.4 Execution Result

The resulting code can display clock correctly, modify clock time, and fire laser. Demonstration video is available at `https://youtu.be/DD0unr0UvBY`.