Lab 3 Report

Graded

2 Days, 13 Hours Late

Group

EMANUEL ZAVALZA
KAI WANG
View or edit group

Total Points

40 / 40 pts

Question 1

Introduction and requirement (10%). Summarize background information about the lab and the detailed design requirements. It's very important to make sure you are designing the right thing before starting. **10** / 10 pts

✔ **– 0 pts** Correct

Question 2

Design description (15%). Document the design aspects including the basic description of the design, modular architecture, interactions among the modules, and interface of each major module. You should include schematics for the system architecture. You can also include figures for state machines and Verilog code when needed. **15** / 15 pts

✔ **– 0 pts** Correct

Question 3

Simulation documentation (10%). Document all the simulation efforts (what requirements are tested and what the test cases are), document bugs found during simulation, and provide simulation waveforms. **10** / 10 pts

✔ **– 0 pts** Correct

– 2 pts Needs to be more detailed. i.e. need some detailed documentation of anything you encountered, or anything you tested, anything you tested, thought process, etc.

Question 4

Conclusion (5%). Summary of the design. Difficulties you encountered, and how you dealt with them. General suggestions for improving the lab, if any. **5** / 5 pts

✔ **– 0 pts** Correct

Questions assigned to the following page: 1 and 2

Introduction and Requirement

In this lab, we are implementing a basic stopwatch that can be controlled using the FPGA buttons and switches to display the desired time on the 7 segment display. The stopwatch counts up 1 seconds at a time by default, with pause and reset button functionality to stop the stopwatch and reset it to zero. The stopwatch counts up from 0-59 seconds and 0-99 in the minutes place. Additionally, the time shown on the stopwatch can be adjusted using an adjustment and select switch. Select decides which place will be incremented (minutes or seconds) while the adjust switch triggers adjustment mode, where the stopwatch is paused and the selected place is incremented at a rate of 2 seconds per seconds (2 Hz) while blinking to indicate it is being adjusted. Digit rollover is also implemented during adjustment mode

Design Description

For this lab we utilize several modules. Multiple clock signals are necessary for display multiplexing and clock counter logic, so first the clock_divider module takes in the FPGA 100MHz clk signal and outputs a 1Hz, 2Hz, and 500Hz signal.

```
clock_divider divider(
    .clk(clk),
    .rst_1(rst_1),
    .rst_2(rst_2),
    .rst_500(rst_500),
    .clk_1hz(clk_1hz),
    .clk_2hz(clk_2hz),
    .clk_500hz(clk_500hz)
);
```

Question assigned to the following page: 2

```
//divide 100 Mhz clk
module clock_divider(clk, rst_1, rst_2, rst_500, clk_1hz, clk_2hz, clk_500hz);
    reg[26:0] ticks_1;
    reg[27:0] ticks_2;
    reg[20:0] ticks_500;
    input clk, rst_1, rst_2, rst_500;
    output reg clk_1hz, clk_2hz, clk_500hz;
    always @ (posedge clk)
    begin
        ticks_1 <= ticks_1 + 1;
        ticks_2 <= ticks_2 + 1;
        ticks_500 <= ticks_500 + 1;
        //reset clk if necessary
        if (ticks_1 >= (100000000 - 1) || rst_1) begin
            ticks_1 <= 0;
        end

        if (ticks_2 >= (50000000 - 1) || rst_2) begin
            ticks_2 <= 0;
        end

        if (ticks_500 >= (200000 - 1) || rst_500) begin
            ticks_500 <= 0;
         end
     //set output clks
     clk_1hz <= (ticks_1 > (100000000 / 2))? 1:0;

     clk_2hz <= (ticks_2 > (50000000 / 2))? 1:0;

     clk_500hz <= (ticks_500 > (200000/ 2))? 1:0;
     end
endmodule
```

A debouncer module takes in the reset and pause button hardware inputs and debounces them so the inputs are consistent.

Question assigned to the following page: 2

```
debouncer db_rst(
    .clk(clk_500hz),
    .button(btnRst),
    .stable(clk_rst)
);

debouncer db_pause(
    .clk(clk_500hz),
    .button(btnPause),
    .stable(pause)
);
```

```
//debounce button and switch inputs
module debouncer(
    input wire clk,         // High-speed clock for sampling (clk_500hz)
    input wire button,      // Noisy button input
    output reg stable       // Debounced stable output
);

    parameter stable_threshold = 2; //arbitrary threshold

    reg [8:0] counter;
    reg button_last;

    initial begin
        stable = 0;
        counter = 0;
        button_last = 0;
    end

    always @(posedge clk) begin
        if (button != button_last) begin
            counter <= 0;
        end else begin
            counter <= counter + 1;

            if (counter >= stable_threshold) begin
                stable <= button_last;
            end
         end
         button_last <= button;
    end
endmodule
```

Next, the clock_counter module handles all logic for the stopwatch, incrementing 1 second per second by default, rolling over as necessary, and implementing the pause, reset, adjust, and select inputs. The final output for the clock_counter module is the min_cnt and sec_cnt counters.

Question assigned to the following page: 2

```
clock_counter counter(
    .clk_1hz(clk_1hz),
    .clk_2hz(clk_2hz),
    .min_cnt(min_cnt),
    .sec_cnt(sec_cnt),
    .rst(clk_rst),
    .pause(pause),
    .sel(sel),
    .adj(adj)
);
```

```
//handle all clock display logic
module clock_counter(
    input clk_1hz,
    input clk_2hz,
    input clk_500hz,
    input rst,
    input pause,
    input sel,
    input adj,
    output reg [6:0] sec_cnt,  // Seconds counter (0-59)
    output reg [7:0] min_cnt   // Minutes counter (0-99)
//    output reg pause_state,
//    output reg reset_state
);

    reg pause_state; //store togglable state of pause
    reg pause_last;
    reg reset_state;
    reg reset_last;

    initial begin
        pause_state <= 0;
        reset_state <= 0;
    end

    //button flip flops
    always @ (posedge clk_2hz) begin
        if (rst != reset_last) begin
            reset_state <= ~reset_state;
        end
        if (pause == 1 && pause_last == 0) begin
            pause_state <= ~pause_state;  // Toggle the pause_state on 'pause' signal
        end
        pause_last <= pause;
        reset_last <= rst;
    end
```

Question assigned to the following page: 2

```
    always @ (posedge clk_2hz) begin
        if (reset_state) begin
            sec_cnt <= 0;
            min_cnt <= 0;
        end else begin
            if (adj) begin
                // Adjustment mode (triggered by clk_2hz)
                if (sel == 0) begin
                    // Adjust minutes
                    if (clk_2hz) begin
                        min_cnt <= min_cnt + 1;
                        if (min_cnt >= 99) begin
                            min_cnt <= 0;
                        end
                    end
                end else if (sel == 1) begin
                    // Adjust seconds
                    if (clk_2hz) begin
                        sec_cnt <= sec_cnt + 1;
                        if (sec_cnt >= 59) begin
                            sec_cnt <= 0;
                            min_cnt <= min_cnt + 1;
                        end
                    end
                end
             end
            else if (~pause_state && ~adj) begin
                // Normal counting mode (triggered by clk_1hz)
                if (clk_1hz) begin
                    sec_cnt <= sec_cnt + 1;
                    if (sec_cnt >= 59) begin
                        sec_cnt <= 0;
                        min_cnt <= min_cnt + 1;
                    end

                    if (min_cnt >= 99) begin
                        min_cnt <= 0;
                    end
                end
            end
        end
    end

endmodule
```

Question assigned to the following page: 2

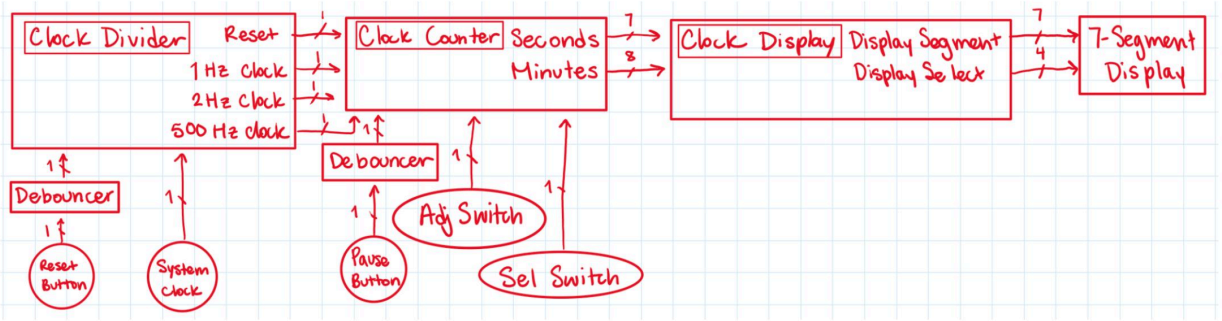
Finally, the clock_display module takes in various speed clock signals, adjust and select switches, and the minute and second counters and performs the multiplexing logic to write out the appropriate digits and blink as necessary.

```
clock_display display (
    .clk_2hz(clk_2hz),
    .clk_500hz(clk_500hz),
    .min_cnt(min_cnt),
    .sec_cnt(sec_cnt),
    .display_seg(seg),
    .display_sel(an),
    .adj(adj),
    .sel(sel)
);
```

```
//display time onto clock
module clock_display(clk_2hz, clk_500hz, min_cnt, sec_cnt, display_seg, display_sel, a
    input clk_2hz;
    input clk_500hz;
    input [6:0] sec_cnt;
    input [7:0] min_cnt;
    input adj;
    input sel;
    output reg [6:0] display_seg;
    output reg[3:0] display_sel;
    reg [1:0] digit;
    reg [3:0] digit_to_display;

    always @(posedge clk_500hz) begin
        //counter for select
        digit <= digit + 1;
        if (digit >= 4) begin
            digit <= 0;
        end

        //output selects
        case (digit)
            2'b10: begin
                display_sel <= 4'b1011; //seconds tens place
                digit_to_display <= (sec_cnt) / 10;
            end
            2'b11: begin
                display_sel <= 4'b1101; //seconds ones place
                digit_to_display <= (sec_cnt) % 10;
            end
            2'b00: begin
                display_sel <= 4'b1110; //minutes tens place
                digit_to_display <= (min_cnt) / 10;
            end
            2'b01: begin
                display_sel <= 4'b0111; //minutes ones place
                digit_to_display <= (min_cnt) % 10;
            end
        endcase
```

```
        //if adjust is on and clk_2hz is low, turn off everything
        if (adj && ~clk_2hz) begin
            if (sel == 0) begin //minutes
                display_sel <= 4'b1001;
            end
            else begin //seconds
                display_sel <= 4'b0110;
            end
        end

        //output display
        case(digit_to_display)
        4'b0000: display_seg <= 7'b1000000; // "0"
        4'b0001: display_seg <= 7'b1111001; // "1"
        4'b0010: display_seg <= 7'b0100100; // "2"
        4'b0011: display_seg <= 7'b0110000; // "3"
        4'b0100: display_seg <= 7'b0011001; // "4"
        4'b0101: display_seg <= 7'b0010010; // "5"
        4'b0110: display_seg <= 7'b0000010; // "6"
        4'b0111: display_seg <= 7'b1111000; // "7"
        4'b1000: display_seg <= 7'b0000000; // "8"
        4'b1001: display_seg <= 7'b0010000; // "9"
        default: display_seg <= 7'b1000000; // "0"
        endcase
    end

endmodule
```

Questions assigned to the following page: 2, 3, and 4



Simulation Documentation

For simulation we first tested the clock_divider module and manually verified the output frequency was correct using the waveform display. Then, we wrote some basic test code for the clock_counter module to validate that our logic worked. We found some minor bugs due to timing issues with the clk_1hz and clk_2hz signals, but after refactoring our always blocks everything successfully worked. Afterwards, we moved directly to testing on the board module by module since the remaining modules rely on hardware inputs or outputs that are difficult to simulate, such as physical buttons bouncing and the output display showing the correct numbers.

Conclusion

To sum it up, we implemented our stopwatch design with different modules to handle each of the respective components: Clock timing, display, and debouncing. We had some difficulties with the digit multiplexing, since the reference documentation we could find online for the digits had a minor error that we had to manually fix for each digit, and the order of the digits didn't seem to correspond with the physical location of each of the 4 digits. As a result, we had to spend some time debugging why our simulation outputs looked great but the FPGA was displaying garbled data. Eventually, we successfully got the digit display working for all 4 digits, but our blinking was implemented incorrectly and could only blink all 4 digits at a time. Everything except for blinking is implemented per the specification, including digit rollover during adjustment mode for both the minutes and seconds places. In addition, we forgot to have the seconds rollover from 59 -> 0 while in the adj phase, but we were able to fix this with a quick if-statement. In all, it was a fun lab finally interacting a lot more with the board.