Jost Luebbe jl64249 Touba

EE 316 Lab #6 Report

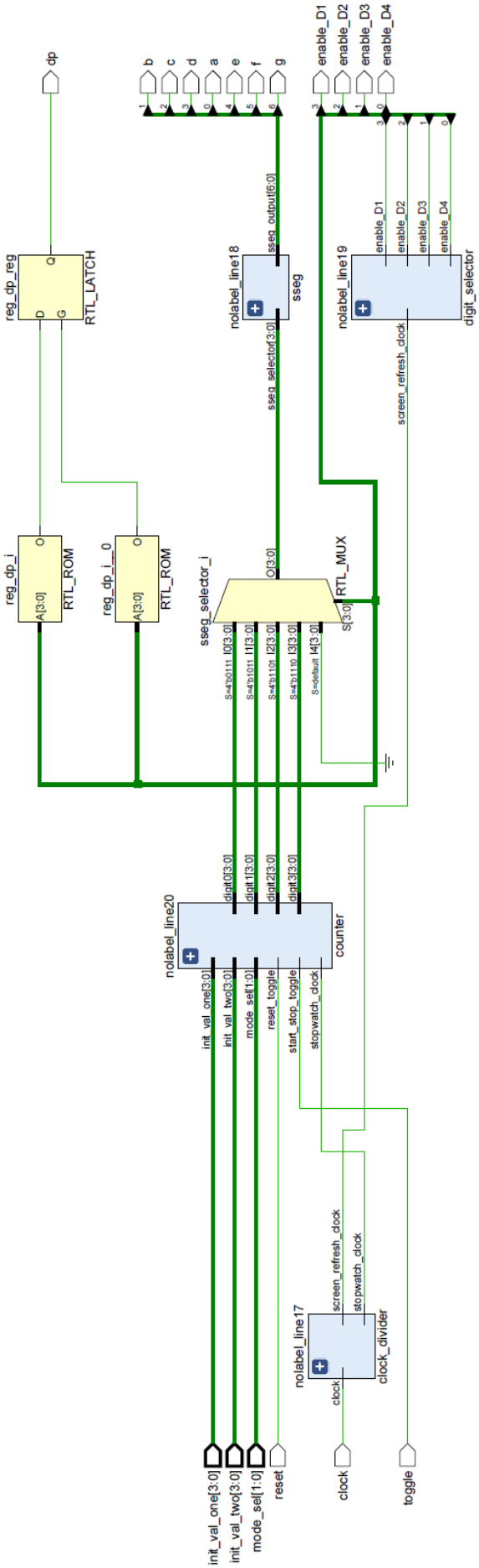
Documentation of Design Process

I began my design process with laying out each of the pieces I'd need to complete the project. These were: a timer that could count milliseconds accurately, a counter that could use this timer to increment or decrement, a way of displaying this counter to the display, and a way of inputting the mode selection, initial values, and the start/stop and reset buttons. These already divided themselves up pretty logically into modules that could function independently of each other for the most part. Obviously, there would be connections between all the modules and building these were the next step. The slow clock would need to travel from the clock divider module to the counter module, and the digits being incremented in the counter module would need to travel to the display module. And then all of these would be conditional on the inputs received from the reset, start/stop, and mode selection buttons and switches.

My first functional attempt at this only had 3 modules, a clock divider, a display module, and a main. These came almost directly from my initial planning. However, this made keeping track of where things were, and how they were connected very difficult. I also had a lot of repetitive code with small adjustments. It worked for the most part, but debugging was a nightmare. The button debouncing wasn't working at all and displaying the digits would only work for counting up. After messing with this iteration for a while I took a step back and decided some code cleanup would be necessary before I could continue debugging. I added an extra module and divided up the work each had a little differently. My clock dividing module now created two separate clocks, one for the stopwatch, one for the display. At first, I hadn't even considered the need for two independent clocks each generated from the board's clock. I thought I would only need the capability of timing the milliseconds. These could now be adjusted very easily and could not function independently of a reset button. Then I copied over the SSEG module we made in Lab 4 and made that its own module. The logic for switching between which digit to display went into its own module. Finally, the counting logic became a completely independent module, which made attacking the reset and counting down issues much easier. I had a separate main that tied all these together and had a small bit of logic for displaying the right digits and the period divider. I could have added this to the SSEG control module, but felt it fit better in the main.

Overall, the initial design wasn't the issue. It was the fact that I rushed into it without enough proper planning, which hurt me when it came to debugging the finer points of the lab. Most of the parts we needed we had already coded in previous Labs but putting them together wasn't as simple has copy-pasting them into a new project. The fact that there were separate modes with independent behaviors meant you had to redesign the overarching logic from the ground up.

Processor Architecture



Verilog Codes and Constraints

Stopwatch Main

```
module stopwatch(
        input clock,
        input toggle,
        input reset,
        input [1:0] mode_sel,
        input [3:0] init_val_one, init_val_two,
        output a, b, c, d, e, f, g, dp,
        output enable_D1, enable_D2, enable_D3, enable_D4
    );

    wire [3:0] digit0, digit1, digit2, digit3;
    reg [3:0] sseg_selector;
    reg reg_dp;
    wire stopwatch_clock;
    wire screen_refresh_clock;

    clock_divider(clock, stopwatch_clock, screen_refresh_clock);
    sseg(sseg_selector, {g, f, e, d, c, b, a});
    digit_selector(screen_refresh_clock, enable_D1, enable_D2, enable_D3, enable_D4);
    counter(
            stopwatch_clock,
            toggle,
            reset,
            mode_sel,
            init_val_one,
            init_val_two,
            digit0,
            digit1,
            digit2,
            digit3
    );

    assign dp = reg_dp;

    always @ (*)
        case ({enable_D1,enable_D2,enable_D3,enable_D4})
            4'b0111:
            begin
                sseg_selector = digit0;
                reg_dp = 1'b1;
            end
            4'b1011:
            begin
                sseg_selector = digit1;
                reg_dp = 1'b1;
            end
            4'b1101:
            begin
                sseg_selector = digit2;
                reg_dp = 1'b0;
            end
            4'b1110:
            begin
                sseg_selector = digit3;
                reg_dp = 1'b1;
            end
            default: sseg_selector = 0;
    endcase
endmodule
```

Counter

```
module counter(
        input stopwatch_clock,
        input start_stop_toggle,
        input reset_toggle,
```

```
    input [1:0] mode_sel,
    input [3:0] init_val_one, init_val_two,
    output reg [3:0] digit0, digit1, digit2, digit3
);

reg button_n_ff;
reg start_stop;
reg reset_n_ff;
reg reset;

always @ (posedge stopwatch_clock) // look for the edge of the button. Use active low logic
begin
    button_n_ff <= start_stop_toggle; //assign button flip flop from button
    if (button_n_ff && !start_stop_toggle) // if button_n_ff = 1 && button_n = 0
            start_stop <= ~start_stop;
    reset_n_ff <= reset_toggle; //assign reset button flip flop from reset button
    if (reset_n_ff && !reset_toggle) // if reset_n_ff = 1 && reset_n = 0
        reset <= 1; //assert reset signal
    else
        reset <= 0; //when the reset button is not negative edge, reset signal is low
end

always @ (posedge stopwatch_clock)
begin
    if (start_stop == 1 && reset == 1)
    begin
        if(mode_sel == 2'b00)
        begin
            digit0 <= 0;
            digit1 <= 0;
            digit2 <= 0;
            digit3 <= 0;
        end
        else if(mode_sel == 2'b01 || mode_sel == 2'b11)
        begin
            digit0 <= 0;
            digit1 <= 0;
            digit2 <= init_val_one;
            digit3 <= init_val_two;
        end
        else if(mode_sel == 2'b10)
        begin
            digit0 <= 9;
            digit1 <= 9;
            digit2 <= 9;
            digit3 <= 9;
        end
    end
    else if(start_stop == 1)
    begin
        digit0 <= digit0;
        digit1 <= digit1;
        digit2 <= digit2;
        digit3 <= digit3;
    end
    else if(start_stop != 1)
    begin
        if (mode_sel <= 2'b01)
        begin
            if(digit0 == 9)
            begin
                digit0 <= 0;
                if (digit1 == 9)
                begin
                    digit1 <= 0;
                    if (digit2 == 9)
                    begin
                        digit2 <= 0;
                        if(digit3 == 9)
                            digit3 <= 0;
                        else
                            digit3 <= digit3 + 1;
                    end
                    else
                        digit2 <= digit2 + 1;
                end
```

```
                else
                    digit1 <= digit1 + 1;
            end
            else
                digit0 <= digit0 + 1;
        end
        else if (mode_sel >= 2'b10)
        begin
            if(digit0 == 0)
            begin
                if (digit1 == 0)
                begin
                    if (digit2 == 0)
                    begin
                        if(digit3 == 0)
                        begin
                            digit0 <= 0;
                            digit1 <= 0;
                            digit2 <= 0;
                            digit3 <= 0;
                        end
                        else
                        begin
                            digit3 <= digit3 - 1;
                            digit2 <= 9;
                        end
                    end
                    else
                    begin
                        digit2 <= digit2 - 1;
                        digit1 <= 9;
                    end
                end
                else
                begin
                    digit1 <= digit1 - 1;
                    digit0 <= 9;
                end
            end
            else
                digit0 <= digit0 - 1;
        end
    end
  end
endmodule
```

SSEG

```
module sseg(
        input [3:0] sseg_selector,
        output reg [6:0] sseg_output
    );

    always @ (*)
    begin
        case(sseg_selector)
            0 : sseg_output = 7'b1000000;
            1 : sseg_output = 7'b1111001;
            2 : sseg_output = 7'b0100100;
            3 : sseg_output = 7'b0110000;
            4 : sseg_output = 7'b0011001;
            5 : sseg_output = 7'b0010010;
            6 : sseg_output = 7'b0000010;
            7 : sseg_output = 7'b1111000;
            8 : sseg_output = 7'b0000000;
            9 : sseg_output = 7'b0010000;
            default : sseg_output = 7'b0111111;
        endcase
    end
endmodule
```

Clock Divider

```
module clock_divider(
    input clock,
    output stopwatch_clock,
    output screen_refresh_clock
    );

    reg [19:0] stopwatch_count; // 19:0 for real time,
    reg [16:0] screen_refresh_count; // 16:0 for real time

    reg tmp_stopwatch_clk;
    reg tmp_screen_refresh_clk;

    assign stopwatch_clock = (stopwatch_count == 1000000) ? 1'b1 : 1'b0;
    assign screen_refresh_clock = (screen_refresh_count == 100000) ? 1'b1 : 1'b0;

    always @(posedge clock) begin
        if (stopwatch_count < 1000000) begin
            stopwatch_count <= stopwatch_count + 1; // count up
        end
        else begin
            stopwatch_count <= 0; // reset the counter
        end
    end

    always @(posedge clock) begin // use for loop to generate the rclk. rclk*refresh = master clock
        if (screen_refresh_count < 100000) begin
            screen_refresh_count <= screen_refresh_count + 1; // count up
        end else begin
            screen_refresh_count <= 0; // reset the refresh counter
        end
    end
endmodule
```

Digit Selector

```
module digit_selector(
        input screen_refresh_clock,
        output enable_D1, enable_D2, enable_D3, enable_D4
    );

    reg [3:0] pattern = 4'b0111;

    assign enable_D1 = pattern[3];
    assign enable_D2 = pattern[2];
    assign enable_D3 = pattern[1];
    assign enable_D4 = pattern[0];

    always @(posedge screen_refresh_clock)
    begin
        pattern <= {pattern[0], pattern[3:1]};
    end
endmodule
```

Test Bench Code

Constraints

```
## Clock signal
set_property PACKAGE_PIN W5 [get_ports clock]
    set_property IOSTANDARD LVCMOS33 [get_ports clock]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clock]
## Switches
set_property PACKAGE_PIN V17 [get_ports {init_val_one[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_one[0]}]
set_property PACKAGE_PIN V16 [get_ports {init_val_one[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_one[1]}]
set_property PACKAGE_PIN W16 [get_ports {init_val_one[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_one[2]}]
set_property PACKAGE_PIN W17 [get_ports {init_val_one[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_one[3]}]
set_property PACKAGE_PIN W15 [get_ports {init_val_two[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_two[0]}]
set_property PACKAGE_PIN V15 [get_ports {init_val_two[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_two[1]}]
set_property PACKAGE_PIN W14 [get_ports {init_val_two[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_two[2]}]
set_property PACKAGE_PIN W13 [get_ports {init_val_two[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {init_val_two[3]}]
set_property PACKAGE_PIN T1 [get_ports {mode_sel[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {mode_sel[0]}]
set_property PACKAGE_PIN R2 [get_ports {mode_sel[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {mode_sel[1]}]

##7 segment display
set_property PACKAGE_PIN W7 [get_ports {a}]
    set_property IOSTANDARD LVCMOS33 [get_ports {a}]
set_property PACKAGE_PIN W6 [get_ports {b}]
    set_property IOSTANDARD LVCMOS33 [get_ports {b}]
set_property PACKAGE_PIN U8 [get_ports {c}]
    set_property IOSTANDARD LVCMOS33 [get_ports {c}]
set_property PACKAGE_PIN V8 [get_ports {d}]
    set_property IOSTANDARD LVCMOS33 [get_ports {d}]
set_property PACKAGE_PIN U5 [get_ports {e}]
    set_property IOSTANDARD LVCMOS33 [get_ports {e}]
set_property PACKAGE_PIN V5 [get_ports {f}]
    set_property IOSTANDARD LVCMOS33 [get_ports {f}]
set_property PACKAGE_PIN U7 [get_ports {g}]
    set_property IOSTANDARD LVCMOS33 [get_ports {g}]

set_property PACKAGE_PIN V7 [get_ports dp]
    set_property IOSTANDARD LVCMOS33 [get_ports dp]

set_property PACKAGE_PIN U2 [get_ports {enable_D1}]
    set_property IOSTANDARD LVCMOS33 [get_ports {enable_D1}]
set_property PACKAGE_PIN U4 [get_ports {enable_D2}]
    set_property IOSTANDARD LVCMOS33 [get_ports {enable_D2}]
set_property PACKAGE_PIN V4 [get_ports {enable_D3}]
    set_property IOSTANDARD LVCMOS33 [get_ports {enable_D3}]
set_property PACKAGE_PIN W4 [get_ports {enable_D4}]
    set_property IOSTANDARD LVCMOS33 [get_ports {enable_D4}]
##Buttons
set_property PACKAGE_PIN U18 [get_ports {toggle}]
    set_property IOSTANDARD LVCMOS33 [get_ports {toggle}]
set_property PACKAGE_PIN T18 [get_ports {reset}]
    set_property IOSTANDARD LVCMOS33 [get_ports {reset}]
```

Simulation Waveforms