CSM152A Lab 3 Report

Russell Jew (ID: 604477316)
Jeffrey Jiang (ID: 904255069)

Introduction

The main purpose of this lab was to extend our knowledge of Verilog from the combinational logic module that we implemented in Lab 2 to deal with synchronous sequential logic. In addition to implementing sequential logic, in this lab, we were introduced to the UCF file and the 7-segment display, which allowed for us to understand the possibilities of using the user interface options on the FPGA.

As synchronous sequential logic, we need to both deal with clocks and registers that save information. For the first purpose of the lab, we need to derive 4 different types of clock signals (along with the internal clock of 100 MHz) for different purposes. From the initial 100 MHz clock, we derived a 1Hz clock, a 2Hz clock, a 5Hz clock, and a 200 Hz clock. However, unlike the clock, which normally has a duty cycle of 50%, these clocks only keep their rising edge for a single clock cycle of the 100 MHz clock. Therefore, we only use the positive edge of these clocks, as usual, to synchronize the logic. The purpose of each of these clocks will be explained later.

Then we needed to implement a simple timer, which, under default operation, just updates the timer as to how many minutes and seconds have passed since the timer has last been reset. Therefore, this requires the use of the 1Hz clock to increment a counter. This must be displayed correctly on the 7-segment display, which we use the 200 Hz clock to display.

After implementing the most basic timer, we add an adjustment mode, so that the timer can be adjusted to any desired value. This requires the use of two switches which can be used to set whether we want to change the minutes or the seconds of the timer. Using the 2 Hz clock, we increment the timer twice every second, while also making sure that the non-adjusted time is completely paused during this time frame. In the meantime, the timer will also blink five times a second as well, to indicate that it is in adjust mode.

In addition, to make sure that we continue to understand how to use buttons and switches, we must implement a debouncer for the mechanical inputs to make sure that the button presses and switching is accurately done. These buttons will be able to pause and reset the clock as necessary.

Design

We easily split our design into five different portions: clock splitting, counter logic, operation and adjustment logic, debouncer and reset logic, display logic. As a result, we split our code into five different modules: stopwatch.v as the top module, containing all other modules as well as the display logic, counter5.v and counter9.v for the counter logic, clock.v for the clock splitting logic, adjsel.v for the adjustment logic, and debouncer.v for the debouncer logic.

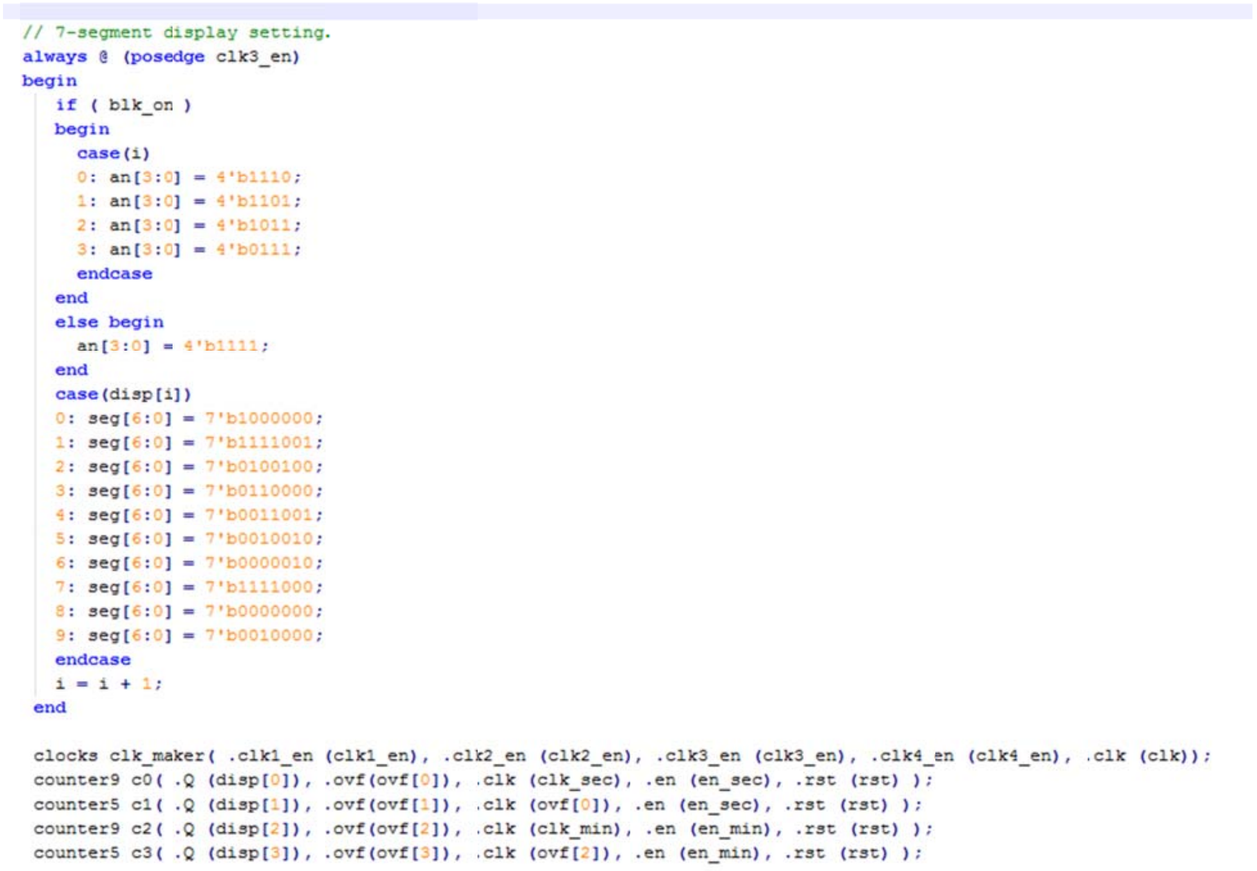
In clock.v, we first implement the logic to split the clock into 4 separate clocks. The four separate clocks will be the backbone to everything else that we use in the following steps. The idea behind splitting the clocks was already shown to us in lab 1 when the lab report asked for those questions. Essentially, we keep a counter of the number of clock cycles that have passed. When a certain number has passed, we set a clock enable as high and reset the counter, as to have a clock pulse with a positive edge every that number of clock cycles. The number we choose depends on the frequency of the positive edge we desire, which just divide 100 MHz by that frequency. The outputs of the clock module, therefore, would be the four clock signals of frequencies 1Hz, 2Hz, 200Hz, and 5Hz.

In debouncer.v, we convert the raw mechanical inputs of the FPGA board into a register in the Verilog code, where we ensure that the button press has become stable before we continue with the remaining logic. Here, we take the input signals btnR, btnS, and sw to determine the reset signal, the pause signal, and the adjustment mode signals adj and sel. The idea behind the debouncer contains two different objectives, which was also asked as a question in lab 1 (although we did not know it was for debouncing purposes). First, we use a slower clock than the 100 MHz internal clock. This allows for us to read less of the noisy data in the transition. Using the 200 Hz clock, we then store the previous four values read from the input. If the transition is made, that is the oldest of the values is 0, but the following values are all 1, we say that the button or switch is successfully turned on. Even with the slower clock rate, using the 200 Hz clock for 4 saves requires only 15 to 20 milliseconds of time, which is more than enough time for a button press or a switch time. Each variable has a slightly different operation for determining value after the switch is determined to be pressed, but that is rather simple.

In counter5.v and counter9.v, we implement two modules, clock5.v and clock9.v which both just give an output for what number they are currently on, and an overflow when the bit overflows past their respective maximum. These modules take in clock, enable, and reset signals. The idea behind these counters are simple: every clock signal the counter just adds one to an internal variable and outputs that. It creates an overflow if the counter has overflowed. The beauty behind using these two outputs is that in the normal stopwatch case, we can input the 1 Hz clock into the lowest digit of the stopwatch and use the overflow of the previous digit for the remaining digits. Then the only complexity behind implementing the counters would be the logic behind setting the enable and the variability of the clocks.

In adjsel.v, we have the adjustment logic. In general, most of this logic is just using the adj, sel, and pause signals coming out of debouncer.v to accurately set the clock and enable of the counters. As mentioned before, under normal conditions (adj = 0), we set the clock of the lowest significant digit to the 1 Hz clock and set the enable of the counters to the inverse of the pause signals. If the adjustment mode is turned on and we selected seconds, the second clock is changed to the 2 Hz clock while the minute counters are disabled. Otherwise if we select minutes, the minute clock is changed to the 2 Hz clock while the second counters are disabled.

Finally all of these are interfaced together in stopwatch.v. Here, we just have a bunch of module calls and the main part is the implementation of the display. Using the 200 Hz clock signal, we cycle through each of the four counters to get the display number and display it. This just uses case signals to display each number for 5 millisecond before rotating to the next number. This provides a 50 Hz flashing LED signal which we cannot detect using the human eye. In addition, the blinking is done in this part, which just alternates a register every time the 5 Hz clock signal turns high if the adjust mode is on or just have it set to 1 if the adjust mode is off.



```
// 7-segment display setting.
always @ (posedge clk3_en)
begin
  if ( blk_on )
  begin
    case(i)
    0: an[3:0] = 4'b1110;
    1: an[3:0] = 4'b1101;
    2: an[3:0] = 4'b1011;
    3: an[3:0] = 4'b0111;
    endcase
  end
  else begin
    an[3:0] = 4'b1111;
  end
  case(disp[i])
  0: seg[6:0] = 7'b1000000;
  1: seg[6:0] = 7'b1111001;
  2: seg[6:0] = 7'b0100100;
  3: seg[6:0] = 7'b0110000;
  4: seg[6:0] = 7'b0011001;
  5: seg[6:0] = 7'b0010010;
  6: seg[6:0] = 7'b0000010;
  7: seg[6:0] = 7'b1111000;
  8: seg[6:0] = 7'b0000000;
  9: seg[6:0] = 7'b0010000;
  endcase
  i = i + 1;
 end

 clocks clk_maker( .clk1_en (clk1_en), .clk2_en (clk2_en), .clk3_en (clk3_en), .clk4_en (clk4_en), .clk (clk));
 counter9 c0( .Q (disp[0]), .ovf(ovf[0]), .clk (clk_sec), .en (en_sec), .rst (rst) );
 counter5 c1( .Q (disp[1]), .ovf(ovf[1]), .clk (ovf[0]), .en (en_sec), .rst (rst) );
 counter9 c2( .Q (disp[2]), .ovf(ovf[2]), .clk (clk_min), .en (en_min), .rst (rst) );
 counter5 c3( .Q (disp[3]), .ovf(ovf[3]), .clk (ovf[2]), .en (en_min), .rst (rst) );
```

Part of stopwatch.v

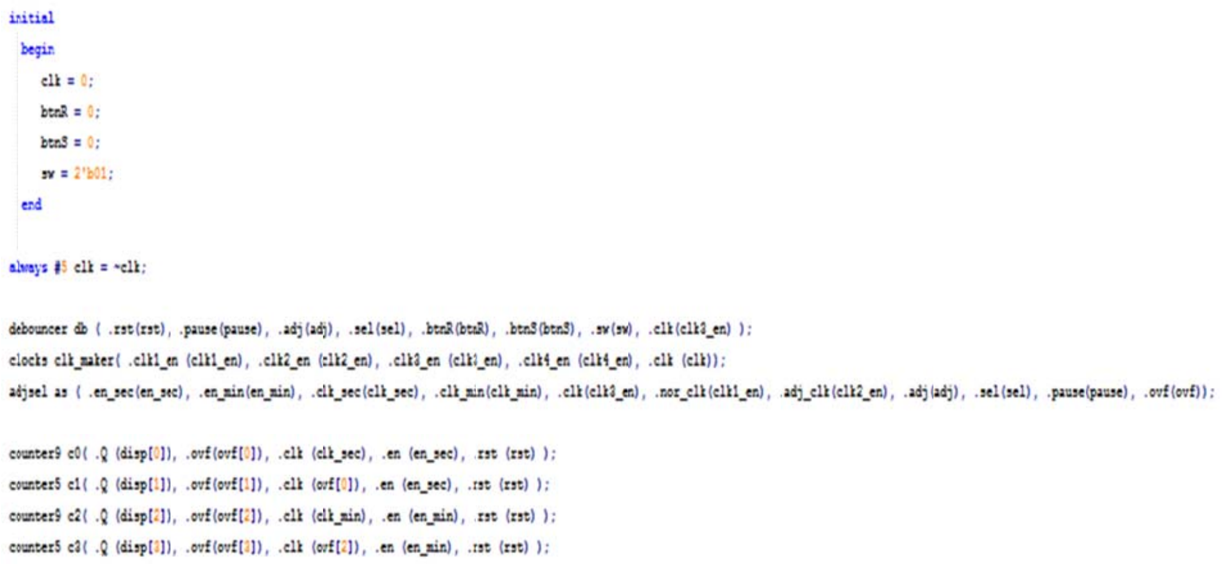
Stopwatch.v is responsible for combining all the modules together and getting them to interact correctly. In the code shown above, stopwatch calls on the counter modules for managing the seconds and minutes of the stopwatch and also calls the clock module to create the different clock frequencies. stopwatch.v is also responsible for displaying the 7-segment display, with the code for that shown above.

.

Simulation

During our design process, we ran into various problems. However, due to the fact that our code is highly modularized, it is very easy to debug our code. The testbench can be accurately changed around to test each module and separately and this can ensure that every signal is done correctly. We were able to create testbenches for each of the modules and ran a simulation on each of them separately to check if they work correctly on their own. At first, many of our problems were simplistic mistakes due to our unfamiliarity of Verilog code and while individual modules worked, when combined they didn't. We found out that this was mainly just due to the fact that we did not initialize our values correctly for various different values and this caused many of our other modules to have unknown or high impedance values which was our biggest error.

Another problem that came up was due to the reset signal. For some reason, when we implemented, the reset button would cause all of our clock signals to stop and not accurately reset the counter modules. Through using the testbench, we came to realize the issues in that the reset was not coming in at the correct timings for the counters and clocks to properly reset the situation and would cause errors in this fashion. Instead we made the reset somewhat “asynchronous” assuming that the press was determined correctly through the debouncer logic. Thus, we have posedge clock or posedge rst to reset the values asynchronously.



```
initial
  begin
    clk = 0;
    btnR = 0;
    btnS = 0;
    sw = 2'b01;
  end

always #5 clk = ~clk;

debouncer db ( .rst(rst), .pause(pause), .adj(adj), .sel(sel), .btnR(btnR), .btnS(btnS), .sw(sw), .clk(clk3_en) );
clocks clk_maker( .clk1_en (clk1_en), .clk2_en (clk2_en), .clk3_en (clk3_en), .clk4_en (clk4_en), .clk (clk));
adjsel as ( .en_sec(en_sec), .en_min(en_min), .clk_sec(clk_sec), .clk_min(clk_min), .clk(clk3_en), .nor_clk(clk1_en), .adj_clk(clk2_en), .adj(adj), .sel(sel), .pause(pause), .ovf(ovf));

counter9 c0( .Q (disp[0]), .ovf(ovf[0]), .clk (clk_sec), .en (en_sec), .rst (rst) );
counter5 c1( .Q (disp[1]), .ovf(ovf[1]), .clk (ovf[0]), .en (en_sec), .rst (rst) );
counter9 c2( .Q (disp[2]), .ovf(ovf[2]), .clk (clk_min), .en (en_min), .rst (rst) );
counter5 c3( .Q (disp[3]), .ovf(ovf[3]), .clk (ovf[2]), .en (en_min), .rst (rst) );
```

Test bench for simulating all the modules

This test bench was used to test all the modules together except stopwatch.v. This test bench ran in simulation so that the values of all the inputs and outputs for each module could be monitored and observed.

Conclusion

This lab expanded on the design techniques and concepts that were used in labs 1 and 2. We were required to design and implement a stopwatch, a sequential system, on the Nexys 3 FPGA board. Besides implementing the standard stopwatch that counts minutes and seconds, the stopwatch also had additional functionality, such as a reset button, a pause button, and an adjust mode. To implement the stopwatch with all the additional functionality, the design had to be split into multiple modules. Two counter modules were needed for counting the seconds and minutes of the timer. A clock module was implemented to create the 4 different clock frequencies that were needed. A module for select and adjust was needed for the adjust functionality. And a debouncer module was needed to deal with the issue of noise when using the reset and pause buttons and the slide switches. The stopwatch module was responsible for combining the different modules together and getting them to interact and work together. It was also responsible for displaying everything on the 7-segment display.

One of the problems that we encountered when designing the stopwatch was getting all the modules to interact with each other correctly. Although we tested all the modules separately and they seemed to work fine on their own, when we combined all the modules together some of them ended up not working. This issue was fixed by properly declaring and initializing all the registers, wires, inputs and outputs for the system and for each module.

An improvement that could be made for this lab would be to make the lab manual more helpful. The lab manual could possibly give more information on how to implement each module and how to display things on the 7-segment display. It could give some suggestions on how to split up the design into multiple modules and also give some pseudo-code for each of the modules to help with getting started. Overall, this lab was a very good extension of the concepts we learned in both lab 1 and lab 2 and also a good introduction to the complete FPGA design flow, going from a large and complicated project specification to the complete design and implementation of the project on the FPGA.