Eric Tang 705579803
Yolanda Chen 005323226

CS 152a Lab 3 Report

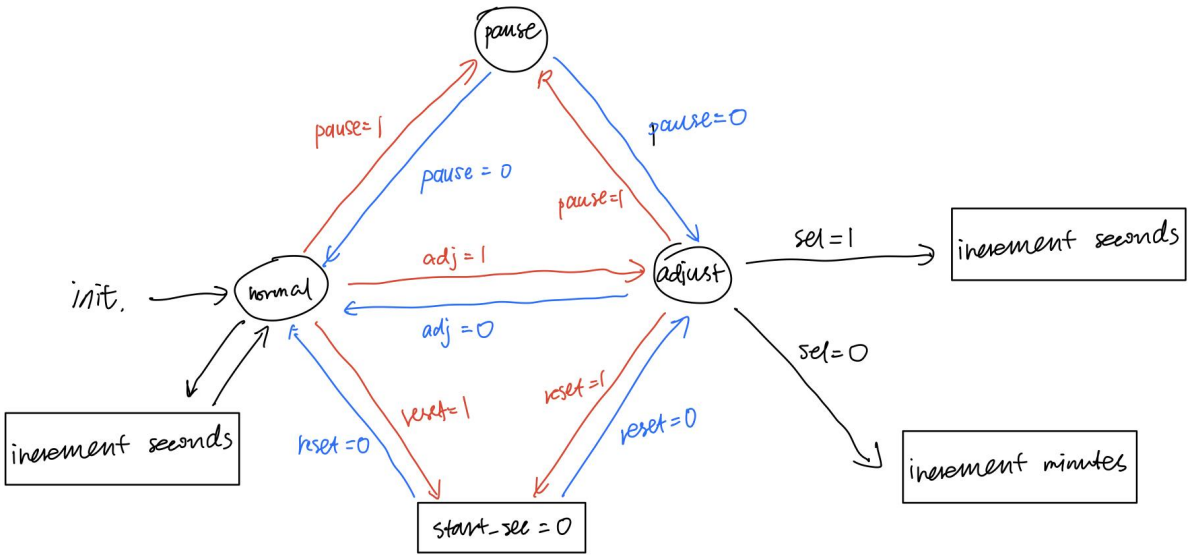
1. **Introduction and requirement (10%). Summarize background information about the lab and the detailed design requirements. It's very important to make sure you are designing the right thing before starting.**

For this lab, we use the Xilinx ISE software to create a stopwatch circuit on the Nexys™3 Spartan-6 FPGA Board. The stopwatch displays in minutes and seconds (e.g. 01:49 means 1 minute and 49 seconds). The user is able to reset, pause, and set the starting minute and second of the stopwatch in *adjust mode*.

When the stopwatch is in *normal mode*, it increments at a frequency of one second. When the stopwatch is in *adjust mode*, the user is able to vary the starting minutes and seconds for the stopwatch.

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.**

Below is a schematic of our state transitions:
Circles are states
Rectangles are actions



Clocks:

Eric Tang 705579803
Yolanda Chen 005323226

Our implementation of stopwatch is made possible by creating different clock signals with different purposes. Using a clock divider module, we created four clocks (**one_hz_clk**, **two_hz_clock**, **segment_clk**, **blink_clk**) from the built-in 100MHz clock in the FPGA.

- **one_hz_clk**: The **one_hz_clock** is a 1Hz clock that is responsible for incrementing the stopwatch every second.
- **two_hz_clk**: The **two_hz_clock** is a 2Hz clock that is responsible for incrementing the seconds or minutes during *adjust mode*.
- **segment_clk**: The **segment_clk** is a 10kHz clock that is responsible for displaying each digit in the seven-segment display very quickly so it seems like the four digits are displayed at the same time. The **segment_clk** is also responsible for dealing with debouncing and metastability.
- **blink_clk**: The **blink_clk** is a 4Hz clock that is responsible for blinking the seven-segment display when in *adjust mode*.

Clock Divider (clk_div.v):
Our clock divider module takes in 2 parameters, 1 input, and creates 1 output.

- Parameters
  - **count_from**: the number we want to start counting from
  - **count_to**: the number we want our clock to count to
- Input
  - **in**: the base clock
- Output
  - **out**: the clock created from the clock divider

Debouncing and Metastability:
To deal with debouncing and metastability, we created a separate 2-bit variable for each of the switches and buttons that is responsible for checking if the user actually presses the button or turns on the switch. For example, for the reset button, we created a variable called **setReset**. **setReset** works similar to a 2-bit predictor. It sets **reset** to True if it detects that the button **btnR** is high twice in a row. By doing so, we eliminate the possibility of changing states because of unstable signals.

Our pseudocode for debouncing and metastability is shown below:

```
input btnR
reg reset;
reg [1:0] setReset;

// for each sampling point
```

```
always @(posedge segment_clk) begin
      shift setReset right by 1
      if (btnR is high)
            Set msb of setReset to 1
      if (setReset == 2'b11)
            Set reset to True
      else
            Set reset to False
end
```

Seven Segment Display:
There is a four-digit display on our FPGA board. From left to right, we named each digit displayed as **seg3**, **seg2**, **seg1**, and **seg0**. **seg3** and **seg2** (the first two digits displayed) represent the number of minutes; **seg1** and **seg0** (the last two digits displayed) represent the number of seconds. The number of minutes will reset to 0 when the stopwatch ticks at 99:59. When we reach 59 seconds on the display, the stopwatch will increment the minutes by 1 and set seconds back to 0. Our logic for setting the correct minutes and seconds is to convert the current time (in seconds) to minutes and seconds every cycle. By doing so, we only need to keep track of one incrementing variable (**start_sec**).

The following is how we determine what number each digit displays every cycle:

```
// start_sec = the total number of seconds at this moment
seg0 = (start_sec%60)%10;
seg1 = (start_sec%60)/10;
seg2 = (start_sec/60)%10;
seg3 = (start_sec/600)%10;
```

To display the time correctly, as mentioned in the **Clocks** section, we set each digit separately in every cycle of **segment_clk** (a very fast clock) and display them in order over multiple cycles. Since **segment_clk** is very fast, the user of the stopwatch cannot tell that each digit is displayed individually. We accomplished this by creating a 2-bit variable **digit_dis** that increments by one in every cycle of **segment_clk**. Since it is a 2-bit value, **digit_dis** will only have values of 00, 01, 10, and 11. Using **digit_dis**, our code will be able to turn on the corresponding four seven-segment display in each cycle.

Reset:

The logic for resetting the stopwatch is simple. Whenever **reset** is True, we set **start_sec** to 0. Reset should be able to work in *adjust mode* and *normal mode*.

Pause:
The logic for pausing the stopwatch is simple. We only increment **start_sec** whenever **pause** is False. Pause should be able to work in *adjust mode* and *normal mode*.

Adjust:
In our implementation, **sw0** (the rightmost switch on the FPGA) determines whether the stopwatch is on *adjust mode* or not. When **sw0** is 0, the stopwatch should be in *normal mode*; when **sw0** is 1, the stopwatch should be in *adjust mode.*

Select:
In our implementation, **sw1** determines whether we are adjusting minutes or seconds on the stopwatch. Note that *select* only works when in *adjust mode*. When **sw1** is 0, we are adjusting the minutes; when **sw1** is 1, we are adjusting the seconds. Adjusting minutes or seconds should not affect the other. In other words, when adjusting minutes, the value of seconds should remain the same and vice versa. To make sure overflowing in minutes and seconds don't affect the other, we use the following code:

```
// if minute is overflowing
if (start_sec/60 == 99) begin
     start_sec = start_sec % 60;
end

// if second is overflowing
if(start_sec % 60 == 59) begin
     start_sec = start_sec - 59;
end
```

User Constraint File (nexys3.ucf):
For this lab, we had to use several components of the FPGA. Here is the list of components we used and their purposes:

clk	The 100MHz base clock for generating the four different clocks used in our implementation.
bntR	The right button on the FPGA used

	for reset.
btnS	The center button on the FPGA used for pause.
sw<0>	The rightmost switch on the FPGA used for *adjust*
sw<1>	The second rightmost switch on the FPGA used for *select*.
an<0>	The seven-segment number display that displays the value of **seg0**.
an<1>	The seven-segment number display that displays the value of **seg1**.
an<2>	The seven-segment number display that displays the value of **seg2**.
an<3>	The seven-segment number display that displays the value of **seg3**.

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

We have encountered several difficulties in this lab. In the beginning, we weren't able to display the digits properly because of a few fundamental confusions. We didn't realize that we can only display one digit at a time, and we needed a high frequency clock to provide the illusion that four digits were displayed simultaneously. Moreover, we mixed up the sign for **an**. It turned out that we needed to set the element in **an** at the corresponding index to 0 for the digit to be displayed. We also had a big shift in terms of our coding structure. We initially planned to use two verilog files to delegate different types of tasks. The first file picks the clock based on the FPGA inputs and calls the function in the other file to execute a certain state (e.g. pause, reset, adjust). However, we combined the files because they complicated things.

Later on we struggled with switching between clocks because we didn't realize that **assign** statements change the wire's value instantaneously as soon as the wires assigned to it change in value. Therefore, we didn't have to put **assign** statements in an **always** block. Furthermore, we tried to switch between **blink_clk** and **segment_clk** in an **always** block by using a clock variable. However, it complicated things and we realized that toggling an additional

variable in a separate **always** block controlled by **blink_clk** is more straightforward.

Toward the end of our lab 3 progress, we forgot to implement the pause and reset function when the FPGA is in adjust mode. After the fix, we also realized that we could only reset the minute and second portion to 1 because we double incremented the **start_sec** value. We also forgot to implement the blinking only for the selected portion when in adjust mode and uncomment **sw** in the ucf file, but these were the problems we fixed easily. Lastly, throughout the lab, we had to change the frequency of several clocks to achieve optimal display.