Andrea Asprer

Benjamin Avalos

Will Spieler

CS M152A, Lab 4

# Lab 3 Report: Stopwatch Report

## Ⅰ. Introduction

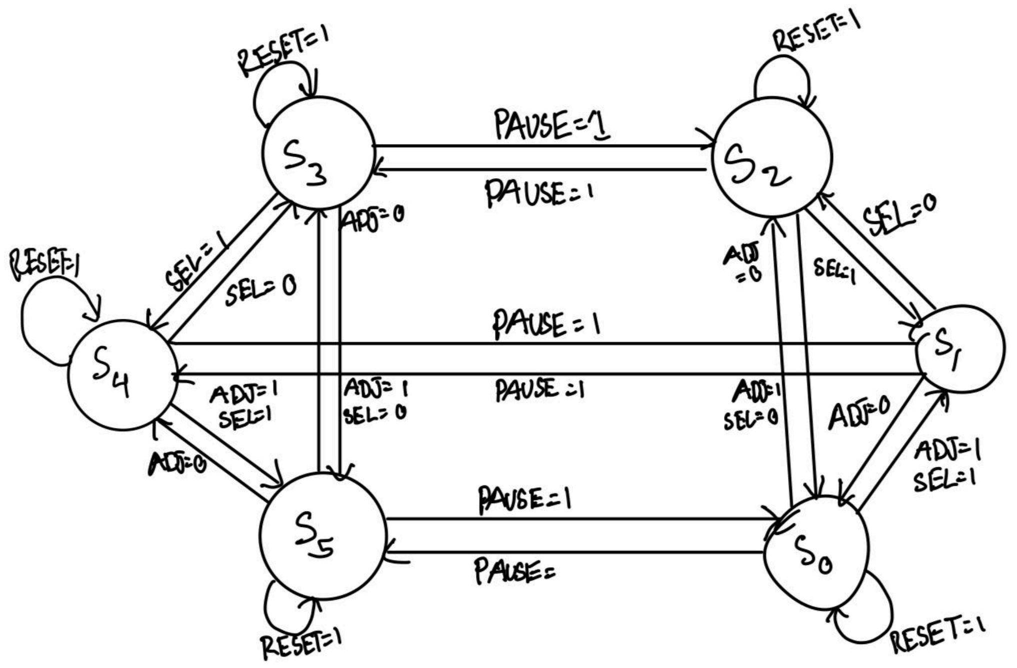
In this lab, we worked on building a stopwatch circuit on the Basys 3 FPGA board. This project gave us hands-on experience with the FPGA design process, including hardware description language development, clock signaling, debouncing, finite state machine design, and seven-segment display multiplexing. We used concepts from previous labs and applied them to handle real-world design challenges, such as timing constraints, synchronization issues, and signal stability.

Our stopwatch implementation was designed as a simple clock that counts minutes and seconds, showing the time on the Basys 3's onboard seven-segment display. Some of the important functionalities include time adjustments with SEL and ADJ switches, pausing and resuming the count with a PAUSE button, and resetting the clock with a RESET button. In order to implement these features, our design includes multiple clock signals, a debouncing module to make sure inputs are stable, a counter module to track the time, and a display model to correctly format the time values.

One important part of this lab was designing an FSM to control the stopwatch under certain conditions. The FSM is responsible for transitioning between normal counting adjustment mode, paused state, and reset conditions, making sure that the operation is as smooth as possible. To verify our results, we included simulations to ensure the stopwatch worked well under various scenarios, such as normal operation, adjustments, pausing, and resetting.

In the following sections, we will discuss the design process in more detail, including the FSM implementation, clocking strategies, and display logic. Additionally, we will explain our testbench results to prove the accuracy of our design.

## Ⅱ. Design Description



For this lab, we implemented the above finite state machine. State $S_0$ is the initial state, and we have the counter count up at a rate of 1Hz. In this state, we can move to a couple different states depending on **SEL** and **ADJ.** If SEL = 1 and ADJ =1, we go to $S_1$, if SEL = 0 and ADJ = 1, state $S_2$, or state $S_5$ if PAUSE = 1. State $S_1$ is seconds adjustment mode, and we have the seconds increment at a rate of 2Hz, with the seconds display blinking at a rate of 1.5Hz, with the minutes frozen. From $S_1$, we can go to $S_0$ from by setting ADJ = 0, state $S_2$ through setting SEL = 0, or state $S_4$ by setting PAUSE = 1. $S_2$ is our minutes adjustment mode with the seconds frozen. The minutes increment at a rate of 2Hz and the minutes display is blinking at a rate of 1.5Hz. In this state, we can go to $S_0$ by setting ADJ = 0, $S_1$ with SEL =0, or $S_3$ by setting PAUSE = 1. The paused versions of $S_2$ $S_1$ and $S_0$ are $S_3$, $S_4$, and $S_5$ respectively. Each of these states are able to return to their unpaused .

The top module for our implementation is **stopwatch.v,** which takes input signals **clk, SEL, ADJ, RESET**, and **PAUSE,** with outputs **cathode** and **anode.** This module integrates various submodules which work to process input signals, manage our counter, as well as convert counter values into their appropriate cathode and anode values.

The first module, **clock_divider,** is responsible for generating clock signals with different frequencies from a single input clock. It takes in a system clock and a reset signal, and splits them into four signals: **clk_2hz, clk_1hz, clk_fast,** and **clk_blink.** To achieve this, we increment the counters on every clock tick. In our **stopwatch_constants** file, we have predetermined thresholds which, when reached, resets the counter to zero and toggles the corresponding clock signal. Each of these four signals is output from this module for other modules to use.

Our next module, **debouncer,** has the job of debouncing and stabilizing an input signal using the master clock. From filtering out the noise of the buttons or switches, we ensure that only stable signals are recognized. It takes in clock signal **clk** and input signal **signal_i,** to output a debounced signal **signal_f.** Using the intermediate registers **store, temp_signal, and cur_signal**, we are able to deal with metastability. The input signal is first stored into the store register, and the rest of the module reads from the register during the posedge of clk. This signal is compared to the current signal of stopwatch. If it's changed, then we begin to start a counter, and continue to increment while the signal remains constant. If the signal fluctuates before the counter reaches **DB_COUNT,** the counter resets. This ensures that an input signal must remain stable for 1ms before being considered valid. Once the counter reaches this threshold, the new stable signal is stored in **cur_signal** and passed to the rest of the module through **signal_f.** This module is used for each of the input signals **ADJ, SEL, RESET,** and **PAUSE,** and the debounced output is assigned to **sel, adj, rst,** and **pause** respectively.

The **sel_adj** module is used to select the appropriate clock signal based on whether the adjustment mode (**adj)** is active or not. It switches between clk_2hz (the faster clock for adjustments) and clk_1hz for regular time counting. If adj =1, the module outputs clk_2hz to hasten time modifications, allowing quick adjustments to minutes or seconds. Otherwise, it outputs clk_1hz for its normal operations. This ensures that the stopwatch can efficiently switch between counting and adjustment modes as needed.

The next module is **counter.** This module takes in inputs **clk_1hz, clk_2hz, sel, adj, rst,** and **pause,** and outputs **minutes_tens, minutes_ones, seconds_tens,** and **seconds_ones.** The module tracks the time in both minutes and seconds, storing them as 4-bit values for each digit. The module checks if the **pause** signal is high, and in this case flips the bit of **is_paused,** which communicates that the stopwatch is paused. If **rst** is high, then all counters will be reset to 0, which serves to reset the stopwatch. The two clock signals it takes in, clk_1hz and clk_2hz, are for normal counting and adjustments respectively. These clock signals select the appropriate one using the **sel_adj** module. If adj is low, it means that the stopwatch is in its normal mode. During this mode, the one's place of the seconds value is incremented by one. If the value has reached its maximum of 9, then it will loop back to 0 and increment the ten's place value by one. If the ten's has reached its maximum of 5, it will then loop back to 0 as well and increment the one's place of the minute value by one. This pattern is continued for the ten's place of the minute value. These digits are then output from the module for translation into 7-segment display format.

The following module, **get_cathode,** is the one that does this translation. It takes the input **display_state** and outputs the 7 bit **cathode** value. There are ten possible values for these digits to correspond to, and it uses a case statement to map each digit (0–9) to a predefined 7-bit cathode pattern, which is stored in **stopwatch_constants**. This module is applied to each of the digits output by the counter module. This module is essential for translating the stopwatch's time values into a readable format.

**create_display** is the final module, and it is responsible for generating the seven-segment display that shows the stopwatch's time values. This module takes in the clock signals **clk_fst** and **clk_blnk**, the input signals **sel and adj**, and the cathode values from **get_cathode**. In the end, it outputs a single cathode and anode value. It takes in cathode values for each digit (minutes and seconds), and cycles through them rapidly using **clk_fast.** This module uses a 2-bit switch register to determine the digit that is currently being displayed, as well as sequentially updating its anode and cathode values. During adjustment mode, the module continues to check **clk_blink,** and if the value is high, the selected digits cathode is replaced with an empty display, which serves to create a blinking effect to the viewer.

## Ⅲ. Simulation Documentation

Our biggest challenge in this lab was troubleshooting why the display did not blink in adjustment mode. Although writing a testbench was not required, we developed one to debug the issue with the blinking feature of the stopwatch. By analyzing the waveforms, we confirmed that all clock signals, except for the blinking clock, were functioning correctly. The blinking clock remained low throughout the simulation, leading us to suspect something wrong with our clock divider module.

Our testbench was designed to simulate and verify the behavior of the stopwatch. It tests different modes, including normal operation, adjustments for minutes and seconds, pausing, unpausing, and resetting.

The following waveform shows all the various clocks oscillating:

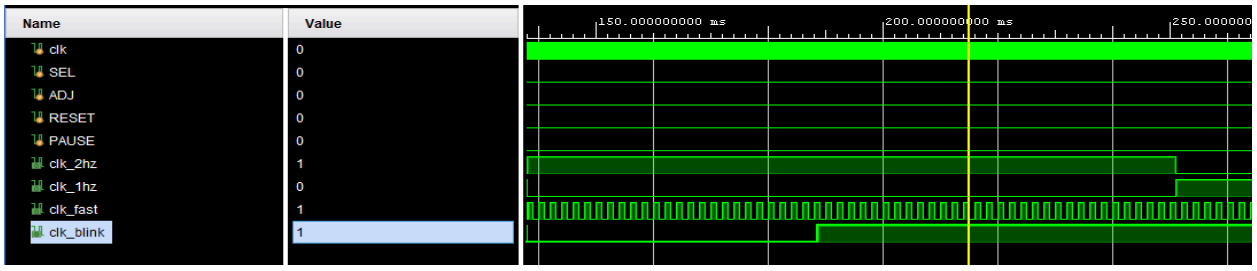


Figure 1: waveform diagrams of all clocks

The following is the stopwatch operating in normal mode:

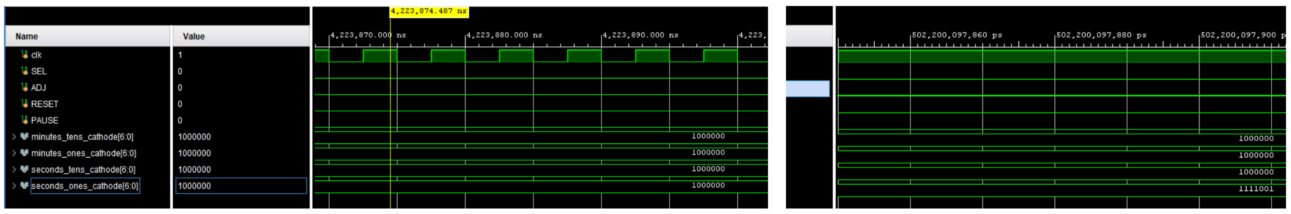
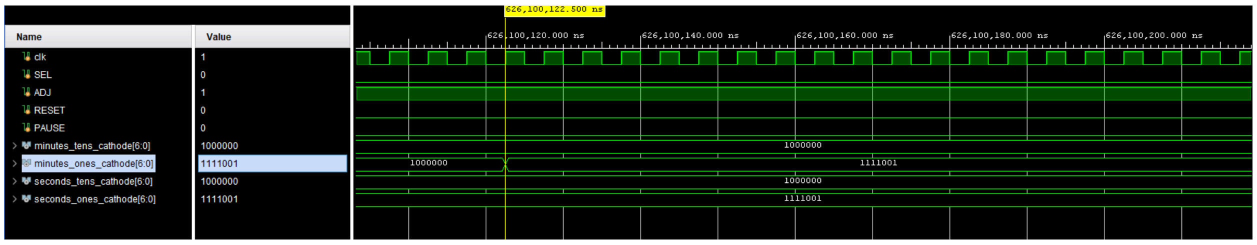


Figure 2: waveform diagrams before and after 1 second in normal mode

The signals for SEL, ADJ, and PAUSE are all low, so this means that we are in normal mode. Let’s focus on the seconds_ones_cathode array which starts with 1000000 which displays 0 on the display. Then after a second, it changes to 1111001 which displays 1 which verifies that our stopwatch is working.

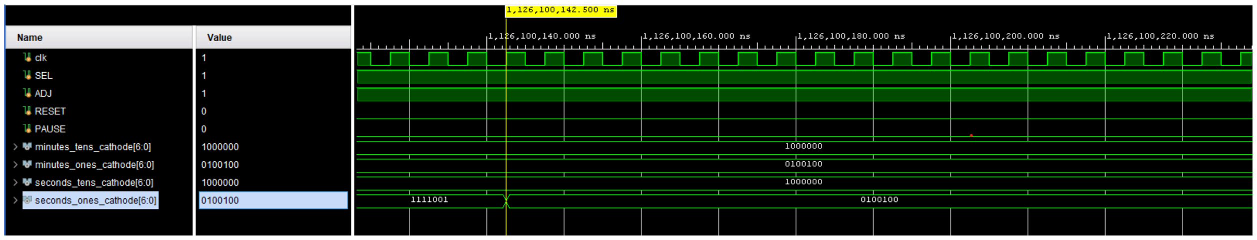
Next, are the waveforms for the stopwatch operating in adjustment minutes mode:



| Name | Value |
|---|---|
| clk | 1 |
| SEL | 0 |
| ADJ | 1 |
| RESET | 0 |
| PAUSE | 0 |
| minutes_tens_cathode[6:0] | 1000000 |
| minutes_ones_cathode[6:0] | 1111001 |
| seconds_tens_cathode[6:0] | 1000000 |
| seconds_ones_cathode[6:0] | 1111001 |

The signal for ADJ is high and SEL is low so we are in minutes adjustment mode. As we can see before half a second passes, our initial value for the one's place of the minute value is 0. Then it switches to 1. We have now verified that the minute's adjustment is working.

We can verify the seconds adjustment mode with a similar process. These are the following waveforms:



| Name | Value |
|---|---|
| clk | 1 |
| SEL | 1 |
| ADJ | 1 |
| RESET | 0 |
| PAUSE | 0 |
| minutes_tens_cathode[6:0] | 1000000 |
| minutes_ones_cathode[6:0] | 0100100 |
| seconds_tens_cathode[6:0] | 1000000 |
| seconds_ones_cathode[6:0] | 0100100 |

Here we see that the signal for ADJ is still on high but SEL is now on high, signaling we are in the correct mode. The initial value for the one's place for the second value is 1 from the normal mode simulation, we see after half a second it increments to 2. All modes have been verified in our simulation.

## IV. **Conclusion**

This lab gave us valuable experience in designing and implementing a stopwatch on the Basys 3 FPGA board using Verilog. Through this project, we applied key digital design concepts, including finite state machines, clock signal generation, debouncing, and seven-segment display multiplexing. By including these features, we were able to create a stopwatch that accurately tracks the time, allows for adjustments, and gives the user the ability to pause and reset the stopwatch.

One of the main challenges we faced during the design process was debugging the blinking display in adjustment mode. To fix this, we developed a testbench and analyzed the simulation waveforms. We realized that the blinking clock signal was not working correctly due to issues with the clock divider module. By debugging and going through this process, we learned how important simulation and waveform analysis is throughout the FPGA design process.

Also, we learned about FSMs and their role in controlling the behavior of the system. The FSM gave us permission to handle different operational states, making sure the transitions between normal counting, adjustment mode, pause, and reset were as smooth as possible. Creating the FSM made us realize how important state management and sequential logic are in digital systems.

In conclusion, this lab strengthened our ability to design, implement, and debug digital circuits using Verilog. This experience with FPGA timing, input signals, and display control will be helpful to us in future classes where we have to use embedded systems and digital applications.