

DUE DATE: Thursday, December 1st @ 11:59PM

DAV Lab 3

Sequential Logic

[Introduction](#)

[Reference Material](#)

[Getting Help](#)

[Checkoff Requirements](#)

[Simulating Sequential Logic](#)

[Design Constraints](#)

[Part 1: Stopwatch](#)

[Part 2: Timer](#)

[Part 2a: The Timer State Machine](#)


[Buzzer Setup](#)


[Part 2b: Creating the Timer](#)

Introduction

In lecture three, we went over several of the key components that are necessary when designing sequential logic. For this lab, you'll be using a bunch of those concepts to create a stopwatch and a timer that you can use to track how much time you have left in your final exams.

Reference Material

 DE10-Lite_User_Manual.pdf

 Lecture 3 - Sequential Logic

Getting Help

You can contact us either on Discord, through email, or in-person during our lab hours.

Tim Jacques: TJ178#5214

tjacques888@g.ucla.edu

Siddhant Gupta: Condolences#1271

siddhantgupta@g.ucla.edu

Our lab hours can be found at <https://ieebruins.com/lab>

Checkoff Requirements

In this lab, you'll be making two FSM diagrams and a quick video to submit to the form linked below. In addition, you have to show us your final working timer during our lab hours or virtually on Discord.

Simulating Sequential Logic

You should test your clock divider in simulation (using a testbench). Of course, you can't use the clock on the FPGA in simulation so you will need to create your own clock signal within the testbench to act as the input.

You can create a clock reg (remember to initialize it to 0 or 1) and then in an "always begin/end" block flip your clock every x nanoseconds using delays (**note this is different from always@** and ONLY used in testbenches to create clock signals).

Your simulation will be infinitely long if you keep toggling your clock in the **always** block. If we want our simulation to end after a fixed duration we can add an initial begin/end block. These blocks run only one time.

Example excerpts from a testbench:

```
initial begin //we run this once at the very beginning of execution
    #10000 $stop; //after 10000 time ticks, we end simulation.
end
always begin //this kind of block only works in simulation and should not be in any other module
    # 10 clock = ~clock; //we toggle the clock every 10 nanoseconds forever
    //this runs in parallel to the initial begin so it will stop after 10000 nanoseconds
end
```

Design Constraints

In DAV, we don't want to just have code that is functional in simulation. Verilog is hardware design so we must also be aware of what the design synthesizes to. We have the following requirements for all labs starting from this point on:

1. **Your designs must not infer any latches.** Your combinational logic should be wrapped in **always_comb** instead of always@(*) and your sequential logic should infer registers (triggered by a clock rather than an arbitrary signal)
2. When designing with state machines, you should follow the suggestions in the slides. Your **combination logic for each state should be in a case statement**. You should use **parameters** to name your states rather than arbitrary values. Your **sequential logic should be very simple and minimal and use non-blocking assignments**. For this lab, the only registers inferred should be for your **state** and a **counter** to track the time. Everything else should be combinational.

Note for this lab, there will not be a skeleton provided. You should set up a new project but remember that **you can reuse modules or code from Lab 2 wherever you deem it helpful!**

Part 1: Stopwatch

For the first part of the lab, you will be developing a stopwatch that runs on your board. Your inputs will be a start/stop button, a reset button, a speed switch, and the built-in 50MHz clock source. Your output will be the seven segment display on your board that will display the running time. Two digits are reserved for the minutes, two digits are reserved for the seconds, and the last two digits should be displaying hundredths of a second.

ie. 1 0 3 9 5 5 would be 10 minutes, 39 seconds, and 55 hundredths of a second.

The behavior of your stopwatch will be as follows:

- If the reset button is pressed at any time, the stopwatch will return to the initial condition where time is not running and display all zeroes.
- If the start/stop button is pressed it should toggle between the running and paused states.
 - In the running state, the display should continuously be counting up with your clock.
 - In the paused state, the display should be stopped on whatever time was displayed when the stopwatch transitioned to the pause state.
- One switch on your board should select which speed your display should be incrementing in. Either natural time (100 Hz) or double speed (200Hz). The double speed setting exists to allow you to check if your display is working faster than waiting a whole minute :)

Tips:

- For this, you'll need to use a clock divider as mentioned in lecture to lower the input clock of 50 MHz down to the 100 Hz and 200Hz we need to count in hundredths of a second and at double speed.
 - We recommend you instantiate two clock dividers and multiplex their outputs. If you are curious, you can check out <https://www.chipverify.com/verilog/verilog-parameters> to learn good practice ways to make reusable modules!
- Remember that the buttons are active low, which means that they output HIGH when not pressed.
- You can reuse most of the seven segment display code from the previous lab with a few tweaks!
 - Specifically, remember the conversion from seconds to minutes is different than from centiseconds to seconds (base 100 vs base 60)

Checkoff:

For both of these, you'll submit them to the Google Form in [Checkoff Requirements](#) along with part 2.

1. Draw a diagram for the state machine for this part. Your diagram should have appropriate labels for the button press conditions (0 vs 1 where 1 is pressed)
2. Record a short clip of your stopwatch working. This clip should include the transition between 59 seconds and 1 minute, start/stop button behavior, the switch changing the speed, and reset button behavior. Please keep this as short as possible to keep storage usage down!

Part 2: Timer

Part 2a: The Timer State Machine

In this part of the lab, you'll be creating a simple timer. Like we saw in lecture 3, many systems around us can be modeled as state machines! Timers are one of these. Here's what the state machine for our timer should look like:



Oh no! I accidentally dropped the diagram into the blender! Here's a text description of the state machine. Can you recreate the diagram for next year's DAV students?

Here's what each state does:

1. SET:

During the set state, the user can use the switches to control how much time the timer will have to countdown. To keep this simple, you can simply treat the switches as a single input for the number of seconds to countdown. The display should show the value of the current input switches in the format mm:ss:00. (The last two digits are for the hundredths of a second, however since we're inputting in seconds those should stay zero in this state)

If the user presses start/stop in this state, the timer starts, and switches to the run state.
If the user presses reset, we go back to the SET state (and nothing really will happen).

2. RUN:

During the run state, the timer counts down from the amount of seconds set in the SET state.

DUE DATE: Thursday, December 1st @ 11:59PM

Changing the switches should not affect anything at this point.

If the user presses start/stop in this state, the timer pauses and switches to the PAUSE state.

If the user presses reset in this state, the timer goes back to the SET state.

3. PAUSE:

During the PAUSE state, the timer is paused, and time does not count down. The display flashes and displays the amount of time remaining.

If the user presses start/stop in this state, the timer resumes and continues counting down in the RUN state.

If the user presses reset in this state, the timer goes back to the SET state.

4. BEEP:

During the BEEP state, the buzzer beeps, and the display flashes 00:00:00 to indicate the timer has finished.

If the user presses start/stop in this state, nothing happens.

If the user presses reset in this state, the timer goes back to the SET state.

Checkoff: Submit your finished state diagram to the Google Form in [Checkoff Requirements](#) along with your Part 1 diagram and clip.

Buzzer Setup

Before you actually make your timer, we need to set up the hardware for your buzzer. This is really simple, since the buzzer just needs an oscillating signal to make noise, which just happens to be what a 50% duty cycle clock signal looks like on an output pin. To connect the buzzer, all you have to do is connect one side to ground, and the other to any of the pins on the FPGA (Like Arduino_I00 for example!). Your kit should include one buzzer, a breadboard, and two jumper cables to make this possible. If you're missing anything, please DM your leads.

Part 2b: Creating the Timer

Now that you know how the timer is supposed to work, go ahead and give it a shot!

Tips:

- To make the buzzer beep, it's easiest to connect it to a clock generator that generates the frequency you want the buzzer to make. Make sure that your clock is 50% duty cycle so it creates a nice square wave! Feel free to find a good frequency at <https://onlinetonegenerator.com/>
- To make the display flash on and off, think about some bit tricks you could use to overwrite what the display module is outputting.
- As explained in lecture, it's easiest to use a `case` statement to create the state machine!

DUE DATE: Thursday, December 1st @ 11:59PM

Checkoff:

Show us your working timer! You can either come to our lab hours or set up a time to Discord video call us to demonstrate all of the required functions.