DUE DATE: 11/10 @ 11:59PM

DAV Lab 2

Introduction to the DE10-Lite

Introduction

It's finally time to get your FPGAs out and do something fun with them! In this lab, you'll be playing around with some of the basic peripherals that are connected to the FPGA on our development board. For now, we'll stick to combinational logic to keep things simple and make sure that we're able to get a grasp of the whole process. This lab will require a bit of reading the datasheet, so we recommend keeping that handy for reference.
By the end of this lab, you should be able to write testbenches to ensure your code is correct before you put it on the hardware, write and upload programs to your FPGA, have experience with module hierarchies, and have a working seven-segment module that you can easily drop into future designs.

Reference Material

DE10-Lite_User_Manual.pdf Uploading Code to the DE10-Lite

Skeleton Code

The skeleton code can be found in this folder.

DUE DATE: 11/10 @ 11:59PM

Getting Help

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

Tim Jacques: TJ178#5214
Siddhant Gupta: Condolences#1271

Our lab hours can be found at https://ieeebruins.com/lab

Checkoff Requirements

For this lab, there are several checkoff photos that you need to take. All of these photos will be submitted at the same time to the form below. The last checkpoint needs to be performed live over discord video call or in lab hours.

DUE DATE: 11/10 @ 11:59PM

Part 1: Switches and LEDs

For this part of the lab, create a design that simply lights up the 10 LEDs corresponding to the 10 switches beneath them. To do this, you'll need to set the pins for the LEDs high or low depending on the value of each of the switch pins. Recall that we can treat GPIO simply as input / output ports to our top module, so your module should just take in 10 single-bit inputs for the switches and output 10 single-bits for the LEDs.

You can find the pin assignments for everything in the user manual.

Hint: How would we directly wire up the switch inputs to the LED outputs?

For more information on pin setup and programming your FPGA, follow the instructions here.
Remember when programming the FPGA that your top level should be `labTwoDesign.sv`.

No checkoff for this part!

Part 2: The Seven-Segment Digit

In this part of the lab, you'll be designing a module that converts a 4-bit number into the correct seven-segment output to display numbers on a digit of the seven-segment display.

The Logic

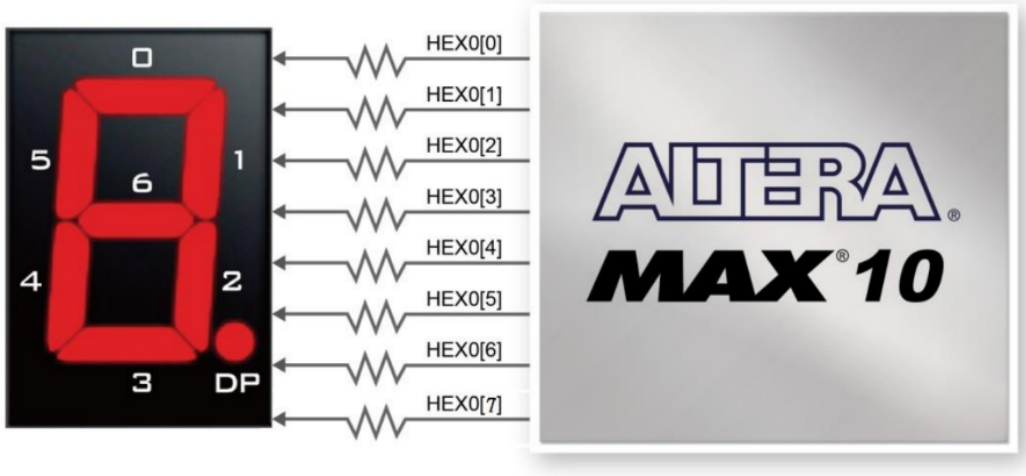


Figure 3-17 (p 29)

As you've probably seen before, a seven-segment display is, well, made up of seven segments plus the dot. Like the LEDs in the last part, these segments are controlled simply by setting the pin on the FPGA that they're connected to either high or low. Your task is to encode a digit 0-9 into the correct pin outputs to display a number.

DUE DATE: 11/10 @ 11:59PM

For now, start by creating a module that encodes only one digit. This will require a 4-bit input value (which can hold up to sixteen different values, but we're only going to use ten), and the 8-bit output value for each of the segments.

Hints:

- You should use entirely combinational logic for this task. Feel free to use any approach you like, including simple assign statements or just a case block.
- The order of the segments (aka which pin corresponds to which) is detailed in the diagram above and in the datasheet.
- The segments are ACTIVE LOW, which means that they're only on when you set a pin to 0.

Side note, if you have fifteen minutes to kill and want to learn about the art of seven segment displays, this is a really cool video :) - Tim

Testing the Digit Module

Although we've been telling you that testbenches are always the best way to check before programming the hardware, sometimes for simple modules like this it's easier just to upload to hardware.

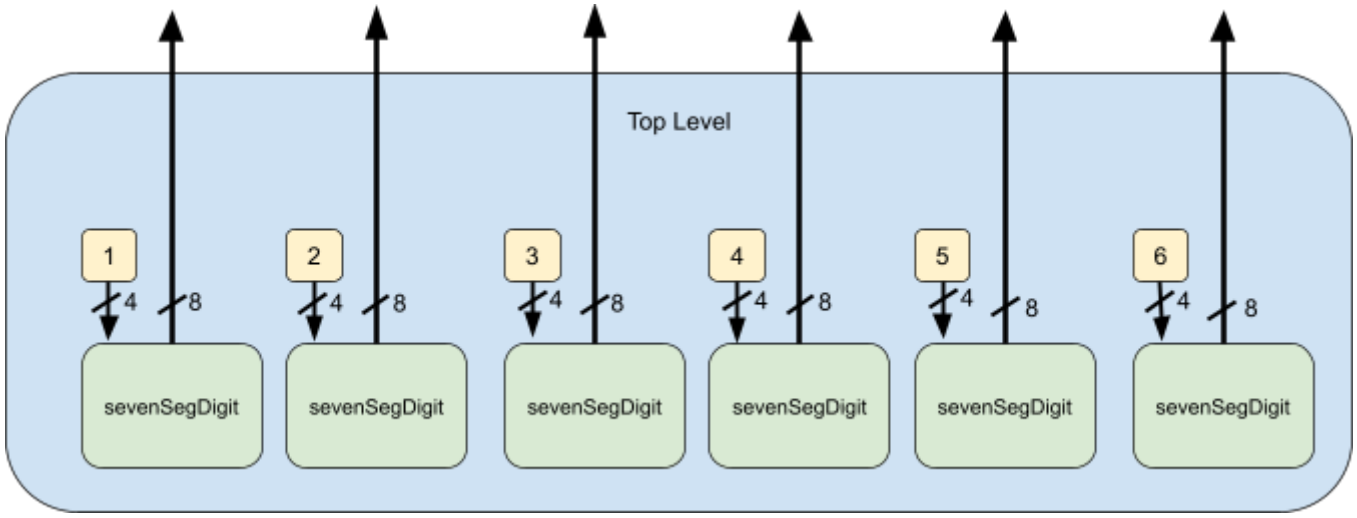
To test your module in hardware, edit the top level to set the registers `input0`, `input1`, … `input5` to hardcoded values. To start, you can set the individual modules to 1, 2, 3, 4, 5, & 6 respectively so you can see if your digits work properly. Once you check those, change your hard coded inputs to include the other four numbers (7, 8, 9, & 0) so you can check all of the possible values.

Hint: You can hardcode registers either in their declaration (this works similarly to an initial value) or in an `always_comb` block.

```
reg [3:0] input0 = 4'd3;
```

OR

```
always_comb begin
    input0 = 4'd3;
end
```



DUE DATE: 11/10 @ 11:59PM

Checkoff: Once you're done, take a photo of your FPGA displaying some numbers (doesn't matter which, just that all of the digits are different)!

Part 3: The Seven-Segment Display (6 digits!)

For this part of the lab, you'll be taking that digit converter we just created and making it more useful. Using the power of submodules, we can easily create a decoder that takes in a single decimal value and outputs it onto our screen!

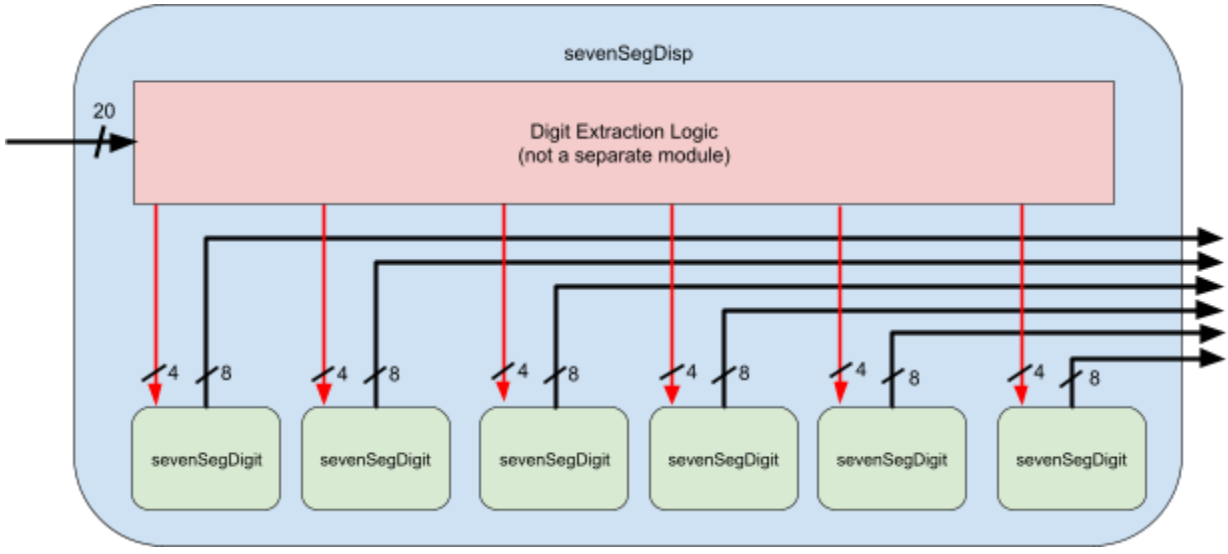
The Logic

Now, if we want to display six-digit unsigned decimal numbers, we'll need a module that controls everything at once. This module will take in a 20-bit number (log2(999999) = 19.9) and output the six 8-bit values to control the whole display.

As an example, if the input is `20'd128` or `20'b00000000000010000000`, the output would make this display show `000128`.

Hints:

- To extract the individual digits, you can use modulus and divide in Verilog!



Top Level not shown

Testing the Full Display Module

Once you think that your logic is correct, we can test our module by using it in a hardware test. In your top-level module, use the 10 switches as inputs to the bottom 10 bits of the display module you just created. You can hardcode the top 10 bits to zero since we're not using them here.

DUE DATE: 11/10 @ 11:59PM

Now, if your logic is correct, the display should show the number input in binary using the switches as a decimal number on the display. ie. if you switch on the bottom 3 switches, the number 7 should be displayed.

Checkoff: Once you're done, take a photo of your FPGA displaying a number represented by the switches on the bottom.

DUE DATE: 11/10 @ 11:59PM

Part 4: The Mini-ALU

In this part of the lab, you'll be designing your own mini arithmetic logic unit that has two operations! While normal ALU's have many more different operations to provide more ways to play with numbers, we'll keep it simple for this lab. In this part, you'll also learn how to use Questa's waves function to better testbench different modules that you create.

The Logic

This ALU will take in two 4-bit inputs, and 1-bit for the operation. It will output one 20-bit result.

When operation is..

- 0: result is the sum of inputs A + B
    - ex: if A is 4 and B is 5, result is 9
- 1: result is input A left shifted by input B
    - ex: if A is 4 in decimal (000100 in binary) and B is 2, result is 16, or 010000 in binary.

More on left shifting:
If A is 3 in decimal, it is 0...0011 in binary. If B is 2, we shift all the bits of A to the left twice and add 0s on the right to fill. So, 3 shifted left by two bits is 0...1100 which is 12 when converted to decimal.

Hints:

- Since our output is 20 bits and our A is only 4, we don't need to worry about running out of bits after left shifting in this module! The max input is A=15 and B=15, which means the max output is 15 * 2^15 or 491520. This nicely fits perfectly into our six digit display.

- In Verilog, left shifting can be written as the following: `out = data << numBitsToShift`
    - More on shift operators here.

The Testbench

You didn't think we'd let you get away doing this entire lab without a testbench, did you?

Once you think that your logic is correct, you can now start writing the testbench for this miniALU! This section will be more tutorial-oriented, so if you already have experience with test benching, feel free to skip the tutorial and write your testbench on your own.

The point of the testbench is to make sure that your logic is correct without having to fully compile and upload onto the hardware. This can make debugging much easier, and allows you to peer into the intricacies of hardware design. If you come to ask for help on any lab, the first questions your leads will ask you will be about your testbench. If you ask a question without making a testbench first, we'll just tell you to go make a testbench, as they usually make a problem obvious!

To learn about how to make proper testbenches, follow the instructions in these slides.

DUE DATE: 11/10 @ 11:59PM

For this module, it is recommended that you take a shot at using the waves method of test benching. Your testbench would generate two inputs programmatically, and display the output of both operations for each input. Then, you can visually check using Questa if the operations are working as expected.

Checkoff: **Take a screenshot of your working testbench showing values from both operations.**

You can now submit your photos from parts 2 and 3 along with this screenshot to the form linked in Checkoff Requirements.

DUE DATE: 11/10 @ 11:59PM

Part 5: The Calculator

Now that we've created all the pieces, you can create the final product – a basic 2 operation calculator.

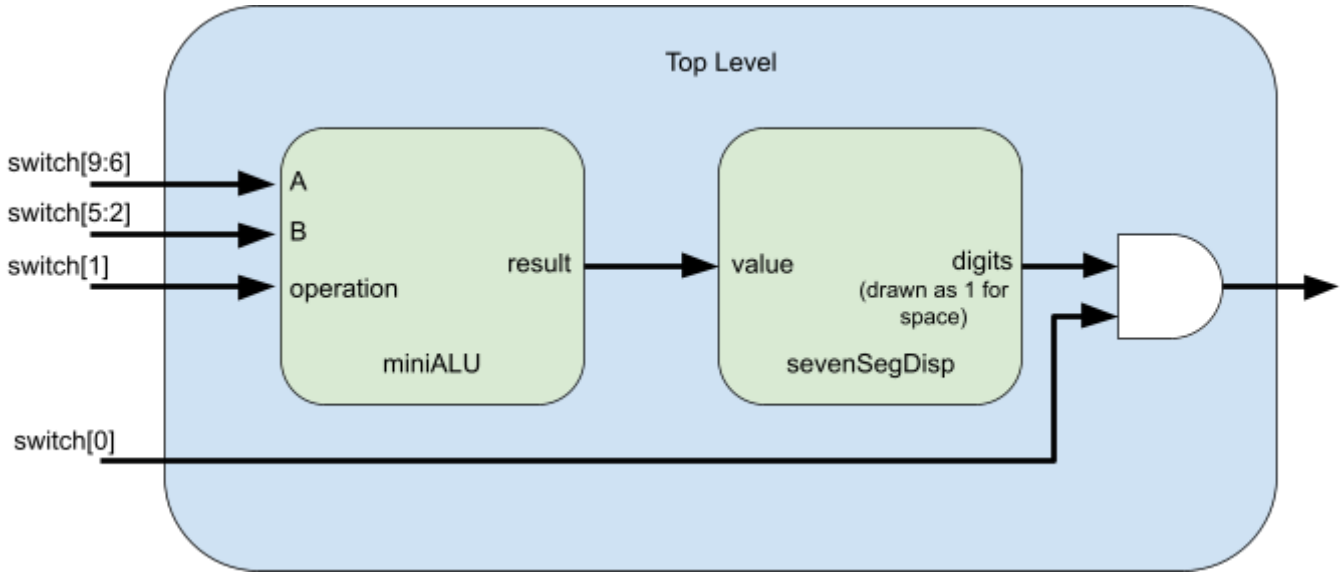
For this, you'll need to use some of the modules from the last couple parts:

- The miniALU, to take in inputs from the switches and perform some math!
- The sevenSegDisp, to take in the output from the miniALU and show it on the display.

This part should be quick, since it's just wiring everything together.

All 10 switches will be used for these operations:

- The top 4 for inputA
- The next 4 for inputB
- The second to last for operation
- The last to turn the display on and off



}

Checkoff: Show us your working calculator live! You can either come to our lab hours or set up a time to Discord video call us to demonstrate all of the working parts of the calculator.