ASIC FPGA Prototyping - Counters, "1101" Detector, and Serial-to-Parallel Shift Register

Jared Botte

February 16, 2022

1 Overview

In this lab, we will explore state machines, counters, and shift registers.

2 Setup

To begin, open a terminal window, create a new directory, and move into it. Also, setup the directory structure and copy over the base makefile.

```
mkdir counter-shift-register
cd counter-shift-register
mkdir -p source include
cp $TOOLSDIR/makefile .
```

This will be the standard setup for most labs.

3 New make target

I've added in a new make target that will just compile a design. This is a good way to check that your design compiles before moving on and creating wrapper files. To use this target, run:

```
make %_compile
```

Where `%` is the module name you'd like to compile. This will also compile any modules referenced by the one provided.

4 "1101" Detector

In the first part of the lab, we will be using a state machine to make a "1101" detector. We will provide the device with a stream of 0's and 1's and it should light up any time "1101" is detected in the stream.

4.1 Create a general "1101" detector

Create a new module `detector_1101` with the following parameters:

Name	Direction	Width	Description
clk	input	1 bit	clock signal
n_rst	input	1 bit	reset signal
data	input	1 bit	data stream
state	output	3 bits	the current state
out	output	1 bit	true (1) if "1101" detected in stream

4.2 Create a mapped version

Next, create a module `detector_1101_mapped` that maps the general detector to the FPGA I/O. Use pushbuttons for the `clk` and `n_rst` signals, a switch for the `data` signal, and green LEDs for the `out` and `state` signals.

Confirm that your "1101" detector works as expected.

5 Counters

The rest of this lab will focus on counters. We will begin by having the counter be controlled by a push button, then move on to controlling it from the clock on the DE2-115 development board. To begin, copy over your implementation of the `flex_counter` from ECE 337. This should already be a fully functional n-bit counter, so we'll only need to create a wrapper file.

5.1 Create a 7-segment display module

Since we'll be using the 7-segment displays in this and many future labs, let's create an easy-to-use module called `hex_display` to deal with mapping a 4-bit value to a hex display.

Module `hex_display`:

Name	Direction	Width	Description
value	input	4-bits	The value to display on the screen
disp	output	7-bits	The value sent to the 7-segment display

5.2 Show a 4-bit counter on a 7-segment display

For this part of the lab, choose push-buttons (keys) to act as the clock, reset, and clear signals and display the output of the counter on one of the 7-segment displays. Set the rollover value to `4'hf`, and `count_enable` to `1'b1`. Name your wrapper file `counter_4bit.sv`.

Hint: If your counter seems to be stuck at 0, remember how the push-buttons are connected to the FPGA.

Question: What is the difference in behavior between the `clear` button and the `n_rst` button? What happens when you press them?

5.3 Try it out!

Now that we've got our 4-bit counter working, lets try counting higher!

5.3.1 Challenge 1: Show an 8-bit counter on 2 7-segment displays

Create a new wrapper module, `counter_8bit` that creates and 8-bit flex counter, and shows the result on 2 7-segment displays.

5.4 Connect the counter to the system clock

Hitting a push button to verify all our values work takes a lot of clicks (well, 255 to be specific). Let's connect the counter to the system clock so that it counts automatically. To do this, let's copy the 8-bit counter into a new module called `auto_counter`. Let's also change the clock push button to control `count_enable` instead. Once done, run it on the development board.

5.5 Slowing things down

You probably noticed the displays both show eights. This is because the counter counts fast. Very fast. 50 MHz fast. Try hitting the `count_en` button and seeing different values. Let's try slowing things down a bit. The `flex_counter` module can also be used as a clock divider.

Question: What do we have to divide our clock input by to get a 20 Hz clock?

Question: How can we divide our clock using the flex counter? How many bits wide does the counter need to be to get the clock to 20 Hz? What should the rollover value be set to?

Once you've answered the above questions, try implementing it. Also make the counter a 16-bit counter that uses 4 7-segment displays

6 Serial-to-Parallel Shift Register

For the final part of this lab, we will create a Serial-to-Parallel shift register. Once we've created this shift register, we'll use it both to make a different "1101" detector and as a means of communication with a game controller.

6.1 1-bit Serial-to-Parallel Shift Register

Just like our n-bit adder implementation, we will implement an n-bit shift register by starting with a 1-bit shift register. This is a super simple device, and is just a flip flop. Create a new module `stp_1bit` to implement it. Have it reset to a `1'b1`.

6.2 n-bit Serial-to-Parallel Shift Register

Next create a module called `stp_nbit` which uses a parameter `BIT_WIDTH` and a generate block to create an n-bit shift register using the 1-bit shift register previously made. You should do this the same way you created the n-bit adder in ECE 337.

6.3 1101 Detector without State Machines

Now that we have our n-bit StP Shift register out of the way, let's try using a 4-bit shift register to detect a "1101" input. Create a new module `stp_4bit` and map it the same way you mapped the previous state detector. Instead of the state being displayed on the LEDs, display the parallel output of the shift register.

Think about how you can detect a "1101" using the parallel output of the shift register, and write a statement in your module to detect this and display it on a green LED.