Lab 1: Verilog Synthesis and FSMs

Contents

0 Introduction

In this lab you will be introduced to behavioral Verilog, and one of the more important tasks of the tools, logic synthesis. Completing this lab will give you experience describing hardware quickly, and at a high level. In addition you will take a close look at the logic that is produced by the CAD tools, and any inefficiencies produced in the translation from text to FPGA implementation.

In this lab we have given you the framework to build a two digit combination lock. Your job will be to implement three separate modules and complete the lock. The inputs to your lock will consist of a rotary dial and two push buttons. The rotary dial, represented by the signals `FPGA_ROTARY_INCA` and `FPGA_ROTARY_INCB`, is used to select a digit of the combination. The rotary dial push button, represented by the signal `FPGA_ROTARY_PUSH` is used to enter the digit selected. LEDs are used to indicate that the lock has been opened, and also that an incorrect combination has been entered.

1 Prelab

Note: Please make sure to complete the items in this section *before* coming to lab. You most likely will not finish the lab during your section if you do not do the prelab.

For the prelab, complete the following items:

- **Read this entire handout thoroughly.** In particular, the Lab Details section gives you lots of valuable details on the circuit you must build. Please also

read the Verilog FSM Tutorial.

- **Download the lab files.**
- **Write your Verilog ahead of time.** You are responsible for implementing Lab2Lock and Lab2Counter.
- **Answer the prelab questions.**

Questions

1. How many D-type flip-flops do you think the synthesis tools will infer in `Lab2Lock`? Think about this carefully. What will D-type flip-flops be used for in this module?

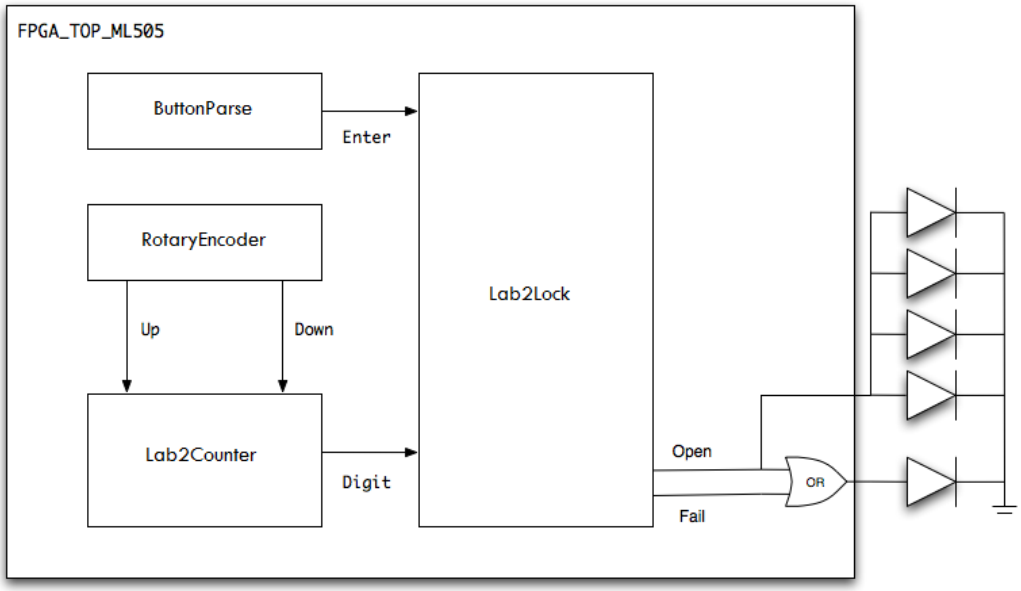
2. Given the verilog you came up with for `Lab2Counter`, what do you think the logic synthesis tools will produce? Suggest a circuit that implements `Lab2Counter`. Draw an RTL diagram of this circuit on the back of the checkoff sheet. You need not go in to great detail - simply show a very high level structure of the circuit.

2 Design Details

This section describes in detail what you will need to design for this lab.

Overview

A diagram of the circuit we want you to implement is below.



Most of the modules have been provided for you; you are only responsible for implementing Lab2Lock and Lab2Counter.

One possible point of confusion is the existence of two separate reset signals, `LockReset` and `SystemReset`. This separation exists to allow the lock to be reset independently of the rest of the system, such as the counter, in the event that the user would like to input another combination. The Lab2Lock module is reset by both the `SystemReset` and the `LockReset`.

ButtonParse

The module has been built for you.

Raw inputs from push buttons are often very noisy, oscillating for a short time before settling on a value. This module filters is input into a clean 1-cycle pulse. It is a fairly complex circuit; you do not need to understand the internal details of how it works.

Refer to the following table for the `ButtonParse` port specification.

Signal	Width	Dir	Description
`Clock`	1	In	The clock
`Reset`	1	In	Resets the state of the `ButtonParse`
`Enable`	1	In	Enables the output
`In`	Variable	In	Raw input from a button
`Out`	Variable	Out	Filtered single pulse output

RotaryEncoder

The module has been built for you.

The `RotaryEncoder` module parses raw input from the rotary encoder (which is the wheel found on the lower right corner of the ML505 board). This module decodes changes in the 2-bit state of the encoder in to pulses of either the `Up` or the `Down` signal. It is important to note that due to the encoder works, **the signal will be pulsed 4 times for each click of the wheel**.

Refer to the following table for the `RotaryEncoder` port specification.

Signal	Width	Dir	Description
`Clock`	1	In	The clock
`Reset`	1	In	Resets the state of the `RotaryEncoder`
`A`	1	In	Raw rotary encoder signal from `FPGA_ROTARY_INCA`
`B`	1	In	Raw rotary encoder signal from `FPGA_ROTARY_INCB`
`Up`	1	Out	Pulses high during clockwise clicks of the wheel.
`Down`	1	Out	Pulses high during counter-clockwise clicks of the wheel.

Lab2Counter

We will be using the `Lab2Counter` in conjunction with the Rotary Encoder as a means of inputting the digits of the combination into our combination lock. We will increment the `Count` output of the lock when the wheel is clicked once counter-clockwise, and decrement the wheel is spun clockwise. Slightly complicating the task of designing this module is the fact that, as noted in the specification of the `RotaryEncoder`, we only want the `Count` output to change once for every *4* clicks of the `Increment` or `Decrement` signal. You will have to figure out a way to deal with this in your design.

Signal	Width	Dir	Description
`Clock`	1	In	The clock
`Reset`	1	In	Resets the count to 0
`Increment`	1	In	Increment pulses from the `RotaryEncoder`
`Decrement`	1	In	Decrement pulses from the `RotaryEncoder`
`Count`	4	Out	The current value of the counter

Lab2Lock

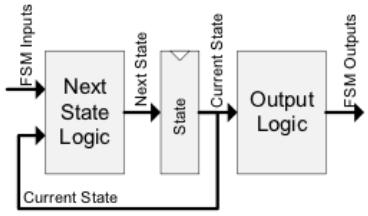
This module is responsible for maintaining the state of the lock and generating outputs which indicate the lock's status (these will be pulled out to LEDs).

This module should detect DIGIT1 and DIGIT2 being entered in the correct order. You should use the verilog `localparam` statement in order to define the values of these constants inside your module. The following snippet shows how this is done:

```
/* Inside the Lab2Lock module */
localparam DIGIT1 = 4'h2;
localparam DIGIT2 = 4'h3;
```

Please use the above values for the combination to your lock to make it easier to test.

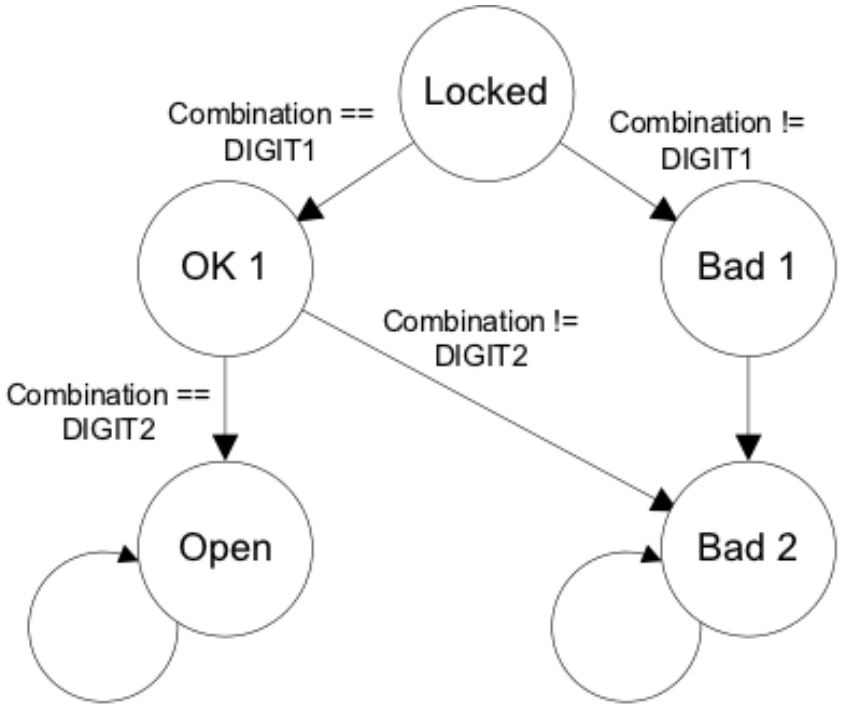
You will build this module as a finite state machine. In particular, you should build a **Moore** machine to accomplish this task. A Moore machine depends only on its current state to generate its output (not, for instance, on both its current state and its current inputs). Therefore, your `Lab2Lock` will have the following high level organization:



Remember the tips we have given you for building FSMs in verilog. If you're not absolutely sure about how to proceed, please review the comprehensive Verilog FSM Tutorial.

Both the port specification and the state transition diagram for this module follow.

Signal	Width	Dir	Description
`Clock`	1	In	The clock
`Reset`	1	In	Resets the lock to a set state
`Enter`	1	In	Tells the lock to read in a new digit
`Digit`	4	In	A digit of the combination
`State`	3	Out	The state of the FSM, for debugging purposes
`Open`	1	Out	Indicates that the lock is open
`Fail`	1	Out	Indicates that the user has entered the incorrect combo



3 **Analysis**

You should now have a complete and working combination lock. In order to build this circuit you did not need to specify the individual implementation of each functional unit. Instead, you provided the synthesis tool with a high level textual representation of the behaviour you wanted, and the synthesizer did it's best to match this description. You now have enough experience to have seen how drastically high level synthesis can affect development times. You will now take a look at how well the synthesizer does it's job.

1. In the prelab you speculated a possible circuit for synthesized Lab2Counter. Find this module in the RTL schematic and compare the circuit created by the synthesis tool to your own.

2. Collect some general information about the design

1. Number of occupied SLICEs
2. Number of SLICE LUTs
3. Number of SLICE registers (used as flip-flops)