EECS150: Lab 3, Verilog Synthesis & FSMs

UC Berkeley College of Engineering
Department of Electrical Engineering and Computer Science

September 13, 2010

1 Time Table

ASSIGNED	Friday, September 10^{th}
DUE	September $21^{st} - 22^{nd}$, at beginning of your assigned lab section

2 Objectives

In this lab, you will be working with behavioral Verilog HDL (hardware description language) to describe a digital system. After writing your Verilog, you will debug your circuit visually using the provided framework. Finally, after your circuit is working correctly, you will analyze the circuit produced by the logic synthesis tool from behavioral Verilog.

Through completing this lab, you will gain experience designing at a high level. Additionally, you will learn about logic synthesis, the conversion of a textual description of a circuit's behavior or structure in an HDL, like Verilog, to a real circuit implementable on an FPGA. Lastly, you will learn how to avoid inferring circuits prone to map poorly to FPGA resources.

3 Addendum to the CAD Flow

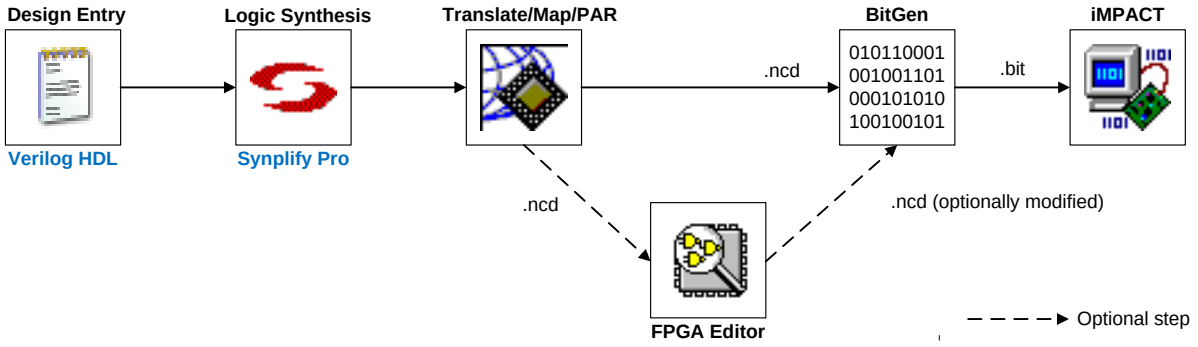
In Lab1, you became acquainted with portions of the Xilinx Virtex5 platform. In Lab2, you worked at a higher level of abstraction. Rather than mapping and placing a design onto an FPGA by hand, you described the structure of your circuit using structural Verilog HDL, which the Place and Route tools implemented. This lab is designed to expose you to another level of abstraction: logic synthesis. Synthesis allows the designer to produce very complex designs without completely specifying the hardware inferred. A designer can specify the behavior of a circuit, rather than its structure, relying on the synthesis tool to derive a circuit implementation. We will be using Verilog HDL as means of describing the structure and behavior of a circuit, and XST as the logic synthesis tool.

Figure 1 shows the tool flow that will be used in this lab. In this exercise, you will be introduced to **Logic Synthesis**. You were briefly introduced to each of the tools in this toolflow in previous labs.

In this lab, you will only be using a subset of Verilog. This is the subset you are enouraged to use in EECS150, as it helps prevent a great number of pitfalls. For more information, refer to the Verilog tutorials on the Documents page of the website.

For this lab we have given you a framework for a combination lock. Your job will be to implement two modules to complete the lock. In order to do this, you will need to use good Verilog style, and to become at least somewhat proficient with the CAD tools.

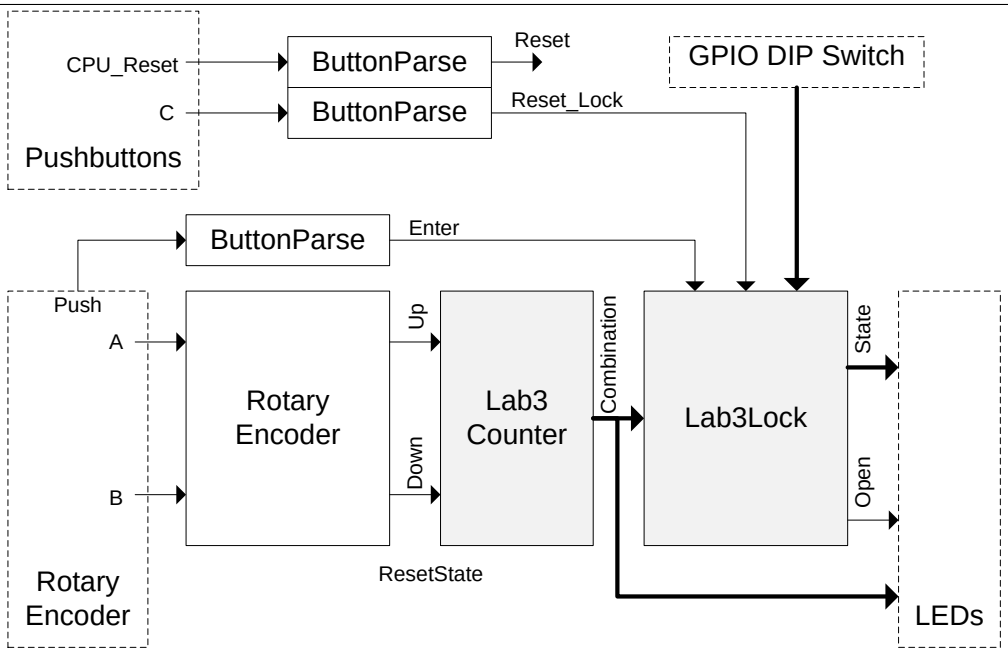
Figure 1 Verilog HDL ⟶ .bit file tool flow.



4 Introduction

For this lab we have given you a framework for a combination lock. Your job will be to implement three modules to complete the lock. In order to do this, you will need to use good Verilog style, and to become at least somewhat proficient with the CAD tools. You will need to exercise caution not to lose sight of the underlying platform: you need to know what your Verilog is synthesized to on the FPGA. You will be describing the combination lock using behavioral Verilog HDL. The inputs to the lock consist of **a rotary encoder** and **two pushbuttons**. The **rotary encoder** (FPGA_ROTARY_INCA,FPGA_ROTARY_INCB) is used to set one digit of the combination. The **rotary encoder button** (FPGA_ROTARY_PUSH) is used to enter a digit of the combination. The two buttons are **ResetLock** (GPIO_COMPSW[0]) which is used to reset lock to a specified state (for debugging), and **Reset** (FPGA_CPU_RESET_B) which acts as the system reset.

Figure 2 Block diagram of the Lab3Top.



The combination lock, itself, is 4 bits wide and controlled by a 2-digit combination. To operate the lock, a user would:

1. **Set the code** using the **rotary encoder** to the **first digit** and **press the rotary encoder button**.

2. **Set the code** using the **rotary encoder** to the **second digit** and **press the rotary encoder button**.
3. The lock will **Open**.
4. The lock can be reset to the inital locked state at any time via **ResetLock** (GPIO_COMPSW[0]).

When someone gets the combination wrong it would go like this:

1. **Set the code** using the **rotary encoder** to a **wrong digit** and **press the rotary encoder button**.
2. **Set the code** using the **rotary encoder** to **any digit** and **press the rotary encoder button**.
3. The lock will report an **Error**.
4. The lock will stay in this state until the user **presses ResetLock** (GPIO_COMPSW[0]).

5 Prelab

Please make sure to complete the prelab before you attend your lab section. **You will not be able to finish this lab in 3hrs otherwise!**

1. **Read this entire handout thoroughly.**
   - Pay particular attention to Section 7 as it describes in detail the circuits you must create.
2. **Examine the Verilog provided for this weeks lab.**
   (a) Make sure you **understand Lab3Top.v** as it instantiate your modules, and **handles I/O**.
3. **Write your Verilog ahead of time.**
   (a) **Lab3Lock.v**
   - Implements the functionality of a combination lock using a finite stage machine (FSM).
   - See Section 7.4 for details.

   (b) **Lab3Counter.v**
   - Increments or decrements a value based on input values.
   - Used to set a combination digit with the rotary encoder.
   - See Section 7.3 for details.
4. You will need the **entire 3hr lab** to **debug** your Verilog and **analyze** the results of synthesis!
   - Debugging may take longer than you expect!

5.1 Questions

After writing your Verilog, and (before) examining the results of logic synthesis, please answer the following questions.

1. How many D-type flip-flops do you think the synthesis tools infer in Lab3Lock? 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 Lab3Counter, what do you think the logic synthesis tools will produce? Suggest a circuit that implements Lab3Counter. Draw an RTL diagram of this circuit on the back of the checkoff sheet. As an example, you may refer to the RTL view of any synthesized circuit (In ISE, navigate to **"Synthesize - XST"** → **"Launch Tools"** → **"View RTL Schematic"**). You need not go into great detail - simply show a very high-level structure of a circuit.

Write down your answers on the checkoff sheet before proceeding.

6 Circuit Debugging

Debugging circuits visually can be a daunting task. Fortunately, we are working with a relatively small finite state design. As our first step, we must resolve trivial typos and remaining bugs in our circuit.

Most of the bugs can be caught and fixed before your design is loaded onto the board:

1. Read your verilog carefully, paying special attention to state transition logic and capitalization on wire names. Make sure your **always** blocks are either **always** @ ($*$) or **always** @ (**posedge** Clock). Using an always block that is not one of these two types may cause odd non-deterministic bugs if you aren't careful. Make sure your **always** @ ($*$) state machine assigns **all variables** in **all cases**. Faliure to do so may also cause inexplicable behavior.

2. If synthesis is not completing due to errors, open the synthesis log (**"Synthesize - XST"** → **"View Synthesis Report"**) Search for "@E", a marker used to identify errors. Correct the errors until only the "@END" search result remains.

3. Use the RTL schematic tool (In ISE, navigate to **"Synthesize - XST"** → **"Launch Tools"** → **"View RTL Schematic"**) to verify all connections. The RTL schematic shows your synthesized circuit as a set of interconnected high-level building blocks. Most Verilog wiring errors are usually obvious here. Note: use the mouse wheel to pan and zoom across your circuit quickly. You have used a very similar tool (the technology schematic) in Lab 2.

Once your synthesized design seems entirely correct, you are ready to move on to hardware verification. Once again, we will be doing this visually.

1. If you suspect a bug, think what a few possible causes are before touching the Verilog.

2. Change only one thing at a time.

3. Try to derive the bubble-arc diagram for the Lab3Lock FSM by observing its black-box behavior. Compare this diagram against the original.

7 Lab Procedure

We expect you to write your Verilog ahead of time. Since Verilog is nothing more than a bunch of standard text in a file with a *.v extension, **you can do this part of the lab entirely from home** in your favorite text editor. Alternatively, you can connect to Kramnik or Iserver for remote access to Xilinx tools. There is a tutorial on using Windows Remote Desktop on the CS150 Website.

If none of the above sound appealing, you can come to the lab and use the tools there. For those of you who like maintaining a single Xilinx Project Navigator project for each lab, you can even **create the project ahead of time and write your Verilog from within Project Navigator**.

Remember to **manage your Verilog, projects and folders well**. Doing a poor job of managing your files can cost you **hours of rewriting code**, if you accidentally delete your files.

7.1 Debouncer

You should keep in mind that Debouncer has been built for you. We are providing this documentation to document the operation of this module.

The Debouncer module is fairly complex. You do not need to understand all details of this module. This is a highly parameterized module which can be used to clean up groups of related buttons used for human input. Raw input from buttons is usually very noisy, and oscillates for a short time before settling. This module filters this input into a clean 1-cycle pulse.

Refer to Table 1 for the Debouncer's port specification.

Table 1 Port Specification for Debouncer

Signal	Width	Dir	Description
Clock	1	I	The Clock signal
Reset	1	I	The lock Reset signal
Enable	1	I	The enable signal for this module.
In	Param	I	Raw input from the buttons
Out	Param	O	Cleaned up pulse signals from the buttons.

Table 2 Port Specification for RotaryEncoder

Signal	Width	Dir	Description
Clock	1	I	The Clock signal
Reset	1	I	The system Reset signal
A	1	I	Raw Rotary Encoder signal read from FPGA_ROTARY_INCA
B	1	I	Raw Rotary Encoder signal read from FPGA_ROTARY_INCB
Up	1	O	Pulses high every click of the wheel clockwise.
Down	1	O	Pulses high every click of the wheel counter-clockwise.

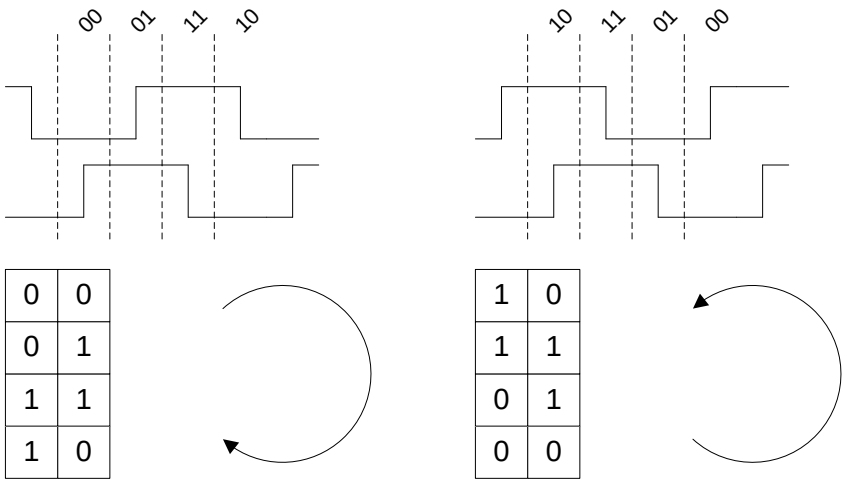
7.2 RotaryEncoder

You should keep in mind that RotaryEncoder has been built for you. We are providing this documentation to document the operation of this module.

The RotaryEncoder module parses raw input from the rotary encoder (lower right corner of the XUPv5 or ML505 boards). This module traces the 2-bit gray-code state of the encoder and detects when the wheel has been rotated clockwise or counter-clockwise. The module then pulses either the Up or the Down output. Note: due to the physical design of the RotaryEncoder and the implementation of this module, the pulses will occur too often to be used directly. You will deal with this problem in Lab3Counter. A continuous rotation of the wheel clocwise, for example, would produce a sequence of pulses on the Up output. Please feel free to examine the internal structure of this module.

Refer to Table 2 for the RotaryEncoder's port specification.

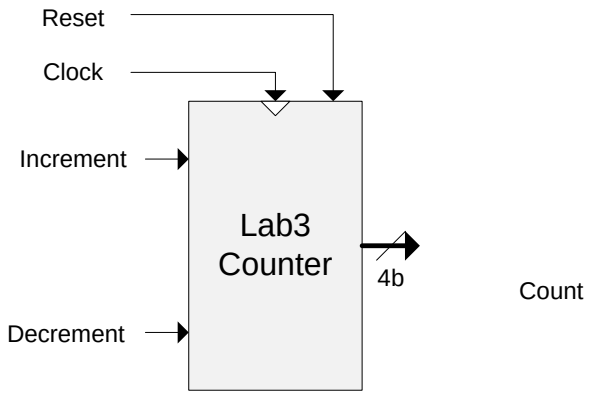
Figure 3 Interpreting the Output of a Rotary Encoder.



7.3 Lab3Counter

We will be using Lab3Counter in conjunction with the Rotary Encoder as 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 spun counter-clockwise, and decrement when the wheel is spun clockwise. It is important to note that the Up and Down pulse outputs of the RotaryEncoder module will be signaled too frequently to be useful. In fact, only 1 out of 4 pulses should increment or decrement the Count output. You will solve this problem by keeping a 6-bit counter internal to the Lab3Counter, exposing only the top 4 bits. The 2 low-order bits will not be exposed. This will ensure that the combination digit will increment or decrement only once per click of the rotary encoder wheel. The current digit will be displayed via 4 LEDs.

Figure 4 The Lab3Counter module.



You will need to implement this module using behavioral constructs (an **always** block, in particular). This will force the synthesis tools to derive an implementation based on your description of the circuit's functionality, rather than structure. You will analyze the synthesized circuit later in this lab.

Refer to Table 3 for the Lab3Counter's port specification.

Table 3 Port Specification for Lab3Counter

Signal	Width	Dir	Description
Clock	1	I	The Clock signal
Reset	1	I	Reset Count to **4'h0**
Increment	1	I	The **Count** will increment every clock cycle **Increment** is high
Decrement	1	I	The **Count** will decrement every clock cycle **Decrement** is high
Count	4	O	The current value of the counter

7.4 Lab3Lock

In this lab you will be building primarily the Lab3Lock module, a relatively simple FSM. This module is responsible for maintaining the **state of the lock** and generating the **outputs to show the lock's status** to the user.

For this lab, you may design the lock to use any combination of two HEX digits. We have provided a sample combination below:

You will build this FSM as a Moore machine 6. This means that the output of this circuit will depend solely on the current state. We will maintain a clear separation between the three major components of this FSM: the next state logic will be described within an **always** @ ($*$) block, the register will be described by an **always** @ (**posedge** Clock) block, and the output logic will use continuous assign statements. Remember that any variable on the left hand of an assignment inside an always block must be declared as a "reg".

Figure 5 The Lab3Lock module.

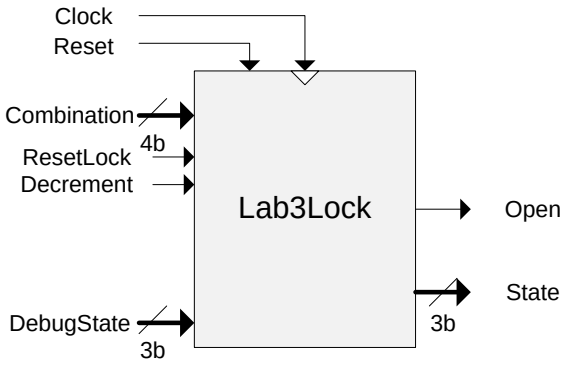


Table 4 Port Specification for Lab3Lock

Combination Digit	
Digit 1	0x2
Digit 2	0x3

Refer to the state transition diagram 7. ("bubble and arc diagram") for the Lab3Lock finite state machine. Give meaningful names to your states using **localparam** macros (similar to #define in C-like languages).

Refer to Table 5 for the Lab3Lock's port specification.

Double-check this module for misspelled wires and erraneous state transition logic.

8 Analysis

You should now have a working Combination lock. In order to build this circuit, you did not have to specify all details of its implementation. Instead, the synthesis tool used a description of your circuit's function, and output a physical netlist that implements this functionality. In other words, the synthesis tool took a description of what you want your modules to do, and invented a circuit that preforms these functions.

Logic synthesis greatly accelerates digital design, as a designer can offload some details such gate-level design and low-level optimization to the CAD tools. How good is the synthesis tool? We cannot rely on this software to implement our circuit until we have answered that question.

Figure 6 A High-Level View of a Moore FSM.

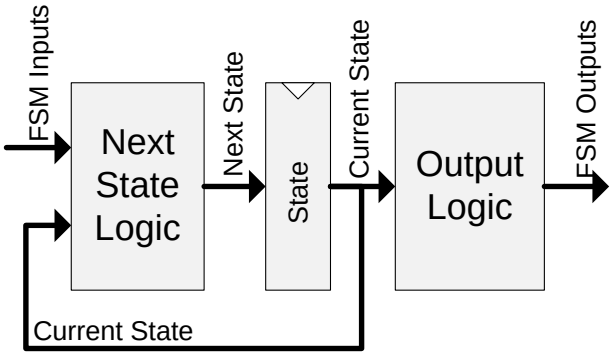


Figure 7 The State Transition Diagram for Lab3Lock.

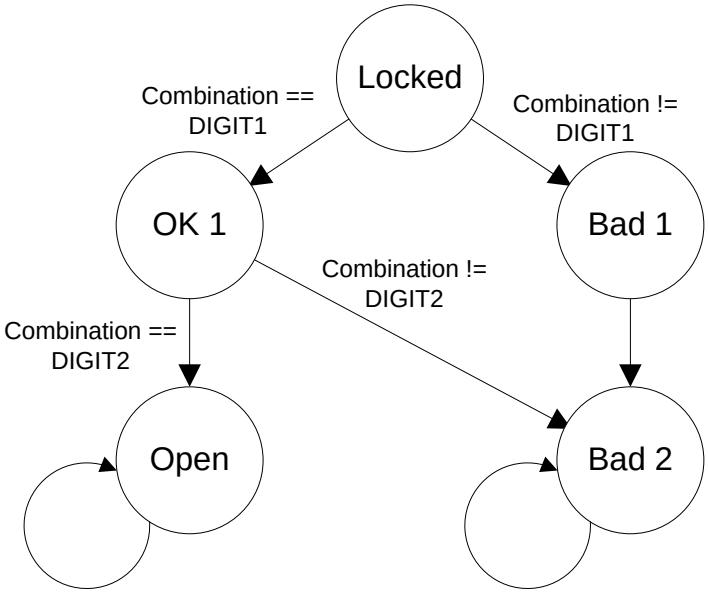


Table 5 Port Specification for Lab3Lock

Signal	Width	Dir	Description
Clock	1	I	The Clock signal
Reset	1	I	The system Reset signal
DebugState	3	I	Used to force the lock into a particular state for debugging (optional).
ResetLock	1	I	Forces the lock into the locked state (or to a the DebugState). Closes the lock.
Enter	1	I	Commits a digit entered via the Rotary Encoder to the lock.
Combination	4	I	A digit of the combination, specified via the Rotary Encoder to the lock.
State	3	O	Outputs the current FSM state for debugging purposes.
Open	1	O	This signal remains high when the lock is open.

You will analyze the circuits synthesized from your Verilog to help answer this question.

1. You inferred a state machine in Lab3Lock using behavioral Verilog. Now, examine the **technology schematic** for the next state logic, and extract the logic equations for the state transition logic. In other words, look at the gate-level implementation, and derive the boolean expression that defines each bit of next state.

2. In the technology schematic diagram for Lab3Lock, find the **flip-flops** representing the current state of the FSM. These flip-flops are the only stateful primitives in the finite state machine. Find these flip-flops in FPGA Editor, and look at what else their slices contain. Look at the contents of a few neighboring slices. Is the logic densely packed in the placed and routed implementation?

3. In the prelab, you have speculated a possible circuit for synthesized Lab3Counter. Find this module in the RTL schematic and compare the circuit created by the synthesis tool to your own.

4. Collect some general information about the synthesized design. Of particular interest are the following:

   (a) # of occupied SLICEs.

   (b) # of SLICE LUTs.

(c) # of SLICE registers (used as flip-flops).

5. Finally, examine the placed and routed circuit. Open the Lab3 circuit in FPGA Editor and answer the following questions:

   (a) Where is the circuit placed on the chip? Why do you think the CAD tools placed the circuit this way?

   (b) Is the circuit closely packed or distributed throughout the FPGA fabric?

9 Extra For Experts

Note: this section is optional. You do not have to do this section.

If you are done with the lab and would like a challenge, we encourage you to try the following: Alter the finite state machine in Lab3Lock to allow a user to change the lock's combination once the lock is opened.

A good solution will use two additional states.

Rev.	Name	Date	Description
H	Michael Eastham	9/13/2010	Fixing minor typos.
G	Austin Doupnik	9/5/2010	Updated for Fall 2010 and XST.
F	Kyle Wecker	2/4/2010	Updated to match new source style for Spring 2010.
E	Ilia Lebedev& John Wawrzynek	2/1/2009	Rewrote lab for use with the ML505 and XUPv5, intended to be given before the Synthesis lab.
D	Chris Fletcher & Greg Gibeling	6/3/2008	Rewrote lab in LaTeX; Updated figures to reflect ISE v. 10.1; Fixed general errors/typos/grammar; Fixed formatting: working to clean up and integrate with RAMP docs.
C	Greg Gibeling	1/18/2004	Added errata from Fall 2004;Migrated to Lab3
B	Greg Gibeling	7/10/2004	Complete Rewrite of Lab3;Based on the old Lab3
A	Multiple	-	Original Lab3 from Fa02-Sp04;Spring 2004: Greg Gibeling;Fall 2003: Greg Gibeling;Spring 2003: Sandro Pintz;Fall 2002: John Wawrzynek& L.T. Pang

10 Lab 3 Checkoff

ASSIGNED	Friday, September 10^{th}
DUE	September $21^{st} - 22^{nd}$, at beginning of your assigned lab section

Man Hours Spent	**Total Points**	**TA's Initial**	**Date**	**Time**
	/100		/ /	
Name	**SID**	**Section**		

1. PreLab ..... _____ (25%)
   (a) How many D-type flip-flops will the synthesis tool infer in Lab3Lock? ..... _____
   (b) Suggest a circuit that implements Lab3Counter. (Draw an RTL diagram). ..... _____
2. Quality of Verilog
   (a) Lab3Lock ..... _____ (15%)
   (b) Lab3Counter ..... _____ (10%)
3. Working Synthesized Lock ..... _____ (25%)
4. Analysis ..... _____ (25%)
   (a) Write down the next-state logic expressions by examining the synthesized Lab3Lock. _____

   _____

   _____

   _____

   (b) Write down the locations of the flip-flops containing the current state of Lab3Lock. _____

   _____

   (c) Look at the RTL for th synthesized **Lab3Counter** circuit and compare it with your prelab design. ..... _____
   (d) # of occupied SLICEs. ..... _____
   (e) # of SLICE LUTs. ..... _____
   (f) # of SLICE registers (used as flip-flops). ..... _____
   (g) Describe the placement of the design (using FPGA editor). Where is the circuit located on the chip? Is the circuit clustered or distributed? ..... _____

   _____