Name: _____

Collaborators: _____

Thayer School of Engineering • DARTMOUTH

ENGS 31/CoSc 56 24S

LAB 3 — Counters and Some Applications

Due on Canvas by Sunday, April 21st at 11:59PM

Contents

Deliverables

1 Overview

1.1 Place in course

At this point in the course, you have seen lots of combinational logic (adders, muxes, encoder etc.) You have learned about flip flops and registers. You've started to become familiar with VHDL. This lab will tie all of those learnings together.

1.2 Learning Objectives

- Develop a paper design for a system and translate it into VHDL.
- Understand the Vivado design flow
- Validate your design with simulation, RTL analysis and the oscilloscope.
- Understand how frequency dividers are designed and employed.

2 Prelab

There is no formal prelab for this lab, however, if you wish to get a head start, you can skip down to the beginning of section 3.2.3 and do the initial design work there.

3 Procedure

3.1 About this lab

In this lab, you will design, model, validate, and implement a four-bit binary-coded decimal (BCD) counter. BCD comprises a subset of decimal numbers: single digit decimal integers 0-9. The counter will count 0000, 0001, 0010, . . . , 1001, 0000, . . . in binary. As you go through this lab, you will learn more about the Vivado design flow, examine more RTL diagrams (like the ones you saw in Lab 2), analyze a special circuit called a clock divider, and then finally implement and validate your BCD digit counter.

Note!

At several points in this and subsequent labs, you will need to unpack a `.zip` file containing a prebuilt project. Modern(ish) versions of Windows do some super smarmy, misguided things when you double click a `zip` file. In particular, if you double click a `.zip` file, Windows extracts it into a virtual file system. I suspect this is an honest attempt to save "precious" disk space, but I can't rule out pure malice as the motivational force here. At any rate:

Right click the `.zip` file and select `Extract All` . This will create real files.

If you forget to do this, Vivado will appear to work just fine for a while, and then you will get the spinning wheel of doom... You will be sad, I will be sad, Vivado will crash...

3.2 Procedure

3.2.1 Elaboration and Synthesis

To explore the elaboration and synthesis steps of the Vivado design flow, you will work with a variation on the 4-bit (binary, not BCD for now) counter that you explored in lab last week and class this week. It is provided to you in the file `Basic4bitCounter.vhd`. A project is provided for you in the Deliverable 1 folder of Lab 3. Open the project by launching Vivado, clicking `Open Project` and navigating to the file `basic_4bit_counter.xpr`.

1. **Elaboration** is the process of interpreting a VHDL model to create a network of function blocks. The result can be printed as a block diagram. Select `RTL Analysis` → `Open Elaborated Design` . When elaboration is complete, you should see a familiar picture (Fig 1), albeit with slightly different names.

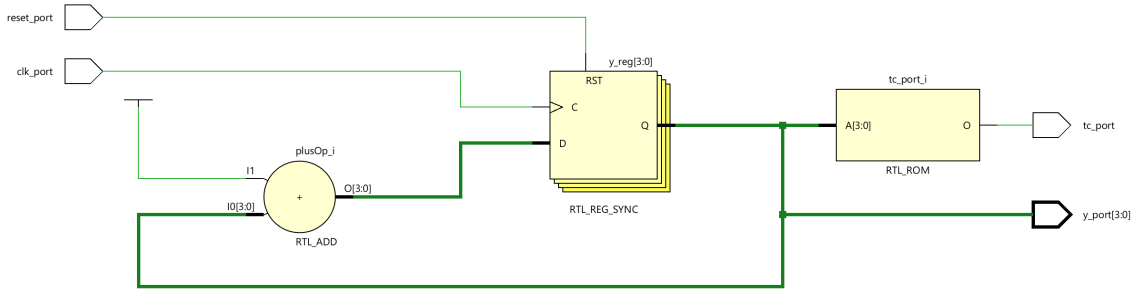


Figure 1: Elaborated Schematic

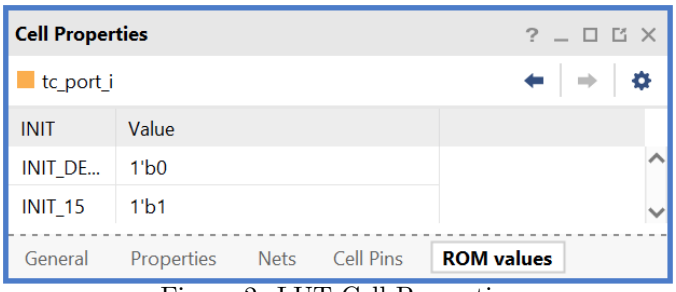


Figure 2: LUT Cell Properties

Compare this diagram with the VHDL model and make sure you understand the correspondence. The block that generates the `tc` output is a "read only memory" (ROM), also called a lookup table (LUT), or interpreted logically, a truth table. If you right-click on the box and select `Cell Properties` , then the ROM Values tab in the Cell Properties window, you should see something similar to Figure 2.

The first line says that by default, the truth table's output is 0. But, when the input is 15, the output is 1. This is the logic that generates the `tc` signal.

2. **Synthesis** is the process of breaking down the functional block diagram inferred from the VHDL into a network of smaller units supplied by the FPGA. The result of this process can also be printed as a block diagram at a finer level of detail.

   In the Vivado Flow Navigator, click `Synthesis → Run Synthesis` . When synthesis is complete, select the `Open Synthesized Design` option in the popup box. Once you've opened the synthesized design, click `Synthesis → Schematic` .

   You should see a schematic similar to the first, but with much more detail. For example, the diagram includes `IBUF` and `OBUF` components. These are created by the tool for each port in your enitity declaration. They are used to connect signals in your design to physical pins on the chip for use by external circuits. We'll come back to these in subsequent labs.

Deliverable 1: LUT analysis *(2 points)*

Inspect the `LUT4` (4 bit look up table) box generating `tc` and compare it with what you observed in the elaborated design (right-click → `Cell Properties` → `Truth Table`). How does this compare to the truth table you found for the D_output for circuit 1 in lab last week?

Deliverable 2: Counter LUT *(3 points)*

The 4-bit register is expanded to four flip flops, and each flip-flop's D input is taken from a LUT. These LUTs implement the truth tables that comprise the next state logic of the counter. Inspect the contents of the LUTs driving `y_reg[0]` and `y_reg[1]` (`y[0]_i_1` and `y[1]_i_1`, respectively). What logic do these LUTs implement? If you're curious, feel free to inspect the remaining LUTs.

3.2.2 Analyzing the clock divider

Navigate to the Deliverable 2 folder and open the provided VHDL file `lab3_clock_generation.vhd` with your favorite text editor. If you are on the lab computers, you can use Notepad++ (right click on the file and `Open with Notepad++`

should be an option). Read through the model. This circuit generates a slower clock from the 100 MHz clock on the Basys 3.

Counters that increment every clock cycle can be used to precisely keep time. Imagine that you extended the maximum count value from the counter in the previous lab from 16 cycles (counting 0 to 15) to 600 cycles (counting 0 to 599). Recall that that counter was clocked at 10 Hz (counting +1 every 0.1 seconds). This new counter would set the terminal count HIGH for one clock cycle every minute. Here, the terminal count would show that a minute had passed.

Our goal is to create a slower clock from a faster clock. Recall that clocks are square waves that are defined by two key parameters:

- The frequency (and it's inverse, the period).
- The duty cycle.

The duty cycle is the ratio of the time that the square wave is high relative to the duration of the whole period. Clocks always have a fixed 50% duty cycle. The signal must be LOW half the time and HIGH half the time.

In the above example that system would have a duty cycle of 1/600, as there are 600 steps in one period, and the output is high for only a single clock cycle. In the last lab, the TC had a duty cycle of 1/16. A counter can generate a periodic pulse, but additional hardware is required to generate a signal with a 50% duty cycle.

In this lab, a 2 Hz clock is desired and will be generated from the 100 MHz external clock on the Basys 3. This clock is generated with counter followed by a toggle flip flop (one that switches state every time it is enabled). It then passes through a special buffer onto the FPGA's clock network. After the lab, if you still have questions about how clock dividers work, please consult Section 14.3 in the eBook for more detail. You will be using clock dividers every week from here on out.

Your goal in this problem is to learn more about how clock dividers work by examining a VHDL model for a simple clock divider (we'll add some more utility into this in subsequent weeks). Part of the goal of this problem is to build confidence reading VHDL. You want to rapidly become comfortable looking at a diagram and knowing what VHDL will describe it, and conversely, looking at VHDL and knowing what circuits it models.

Without opening the elaborated design for this model in Vivado, read through the system clock generation code.

Deliverable 3: Clock Divider Block Diagram *(5 Points)*

Draw a block diagram that highlights how this circuit works.

Deliverable 4: Clock Divider Discussion *(5 Points)*

In your own words, explain how this circuit works.

Deliverable 5: Clock Divider Design *(5 Points)*

Once you have worked out how this circuit works, consider what $CLOCK\_DIVIDER\_TC$ (not $CLK\_DIVIDER\_RATIO$) you would need to generate a $10MHz$ clock from the $100MHz$ clock. How many bits would be needed for the `clk_divider_counter`?

3.2.3 BCD Counter Design

Copy the basic four bit counter from part 1 (`Basic4BitCounter.vhd`) to a file called `bcd_digit.vhd`.

Modify this counter so that it's entity declaration matches that shown below.

```
--------------------------------------------------------------------------------
--Entity Declaration:
--------------------------------------------------------------------------------
entity bcd_digit is
  port(clk_port : in  std_logic;
       reset_port  : in  std_logic;
       enable_port : in  std_logic;
       y_port      : out std_logic_vector(3 downto 0);
         tc_port : out std_logic );
end entity;
```

The new port (`enable_port`) when high will allow the counter to increment and when low will cause it to hold its current value.

Modify the original counter to add the enable functionality. Also modify it so that when it reaches 9 (`''1001''`) it asserts `tc_port` and returns to zero the next time it is enabled.

3.2.4 Implementing the BCD Digit in Hardware and Validating the Design

For the rest of the lab, we will work with the 4-bit BCD counter that you designed in the previous step. Open the project in the Deliverable 3 folder by launching the `.xpr` file with Vivado. If you have questions about this, ask a member of the lab staff. Add your `bcd_digit.vhd` file to the project as a design source. Your `Sources` hierarchy should be similar to that showm in Figure 3.

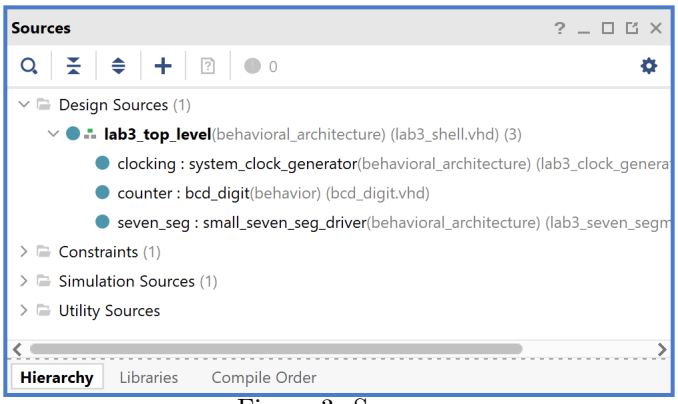


Figure 3: Sources

You will now take your project through the steps needed to create a bit file that can program your FPGA.

Debugging

Vivado lets you know in a variety of ways when you have made an error.

- As you are typing, it underlines erroneous text in red and puts a red mark in the margin to the right of the scroll bar. Hovering over the mark shows a brief description, like `Syntax Error Near ...`.
- When a file is saved, a syntax check is run and errors are reported under the Messages tab at the bottom of the screen. A typical message is `[HDL 9-806] Syntax error near` followed by a hyperlink to the offending line.
- When a file is elaborated or synthesized, any errors, critical warnings, or warnings are reported. Errors and critical warnings must be fixed. Other warnings may or may not be benign. Consult an instructor for guidance on warnings.

1. **Elaboration:** Always examine your elaborated design and check it against the block diagram you began with. This Vivado project connects the clock generation circuit you explored in the previous procedure to your BCD digit. It also wires your BCD digit into a driver for the 7-segment display. A high-level RTL diagram of the system is shown below, which highlights how your BCD digit counter nests into the larger design.

   Open the component that you designed your BCD counter. Confirm that there are no latches present. If there are, check your process blocks in your VHDL for items missing on the sensitivity list or a missing "rising_edge" statement. Check for unexpected dangling or misconnected signals. This step will save you many hours of debugging - never skip it!

   Deliverable 6: BCD Enumeration *(5 points)*

   > Elaborate your project and double click on your counter to view the sub-component separately from the rest of the design. Save an image of the RTL schematic (take a screenshot). Annotate the diagram to identify the register, the reset, the roll-over logic, and the terminal count.

2. **Simulation:** The next step in design validation is simulation, to see if the design is functionally correct. If your testbench and design are properly set up, you will the design file positioned under the testbench and indented in the `Sources` pane between the Flow Navigator and editor windows.

   In the clock generator, comment out the CLOCK_DIVIDER_TC for SYNTHESIS, and uncomment CLOCK_DIVIDER_TC for SIMULATION.

   Using a smaller clock divider value makes the simulation run faster (fewer simulated cycles between TC events). Make sure that you switch back to the correct terminal count for synthesis in the next step, once you have the data you need in this step.

   Following the Vivado simulation tutorial video, run simulation by selecting

   `Flow Navigator → Simulation → Run Simulation → Run Behavioral Simulation`

   When you run a simulation, it runs for 1000ns by default (1us). You will need to run the simulation for a longer time. Do so by clicking on the right facing arrow with a (t) subscript. Use it in conjunction with the text box beside it to run your sim for another 25μs or so.

   Use the waveform as a guide to detecting and correcting any bugs. If you have questions about adding a signal to the simulation window, let a member of the lab staff know!

Deliverable 7: Waveforms *(5 points)*

> Take a screenshot of the waveform. You may need to zoom in to see the numerical values of the count sequence in the waveform. In that case, make a couple of screenshots to get everything. Annotate the printouts to show important features like the roll-over behavior at 9 instead of 15 (F). Make sure that the name of each waveform is clearly visible.

3. **Synthesis, Implementation, Bitstream:** Take your design through the rest of the flow. At each stage, read the messages tabs. You must clear all errors and critical warnings at each stage. If you have a question about whether a warning you receive is benign, ask a member of the lab staff.

   Remember: In the clock generator, comment out the CLOCK_DIVIDER_TC for SIMULATION, and uncomment CLOCK_DIVIDER_TC for SYNTHESIS.

4. **Hardware Validation:** A design is not considered to be working until it is tested in hardware. Program your FPGA with the bitfile that you generated, then connect your circuit to the digital scope. The counter and the TC are routed to the same pins as last week—on Pmod header JB.

   Deliverable 8: Hardware Validation *(10)*

   > Take a screenshot that shows that your circuit is working as intended and show a member of the lab staff to obtain their signature. Annotate the printouts to show important features like the roll-over behavior at 9 instead of 15 (F). Make sure that the name of each waveform is clearly visible

   Staff Signature _____

   Deliverable 9: Correct Code *(5)*

   > Submit your final code and corrected paper design. If nothing changes from the prelab, simply resubmit the section of the homework.

Congratulations! You've just realized your first fully functional digital design!

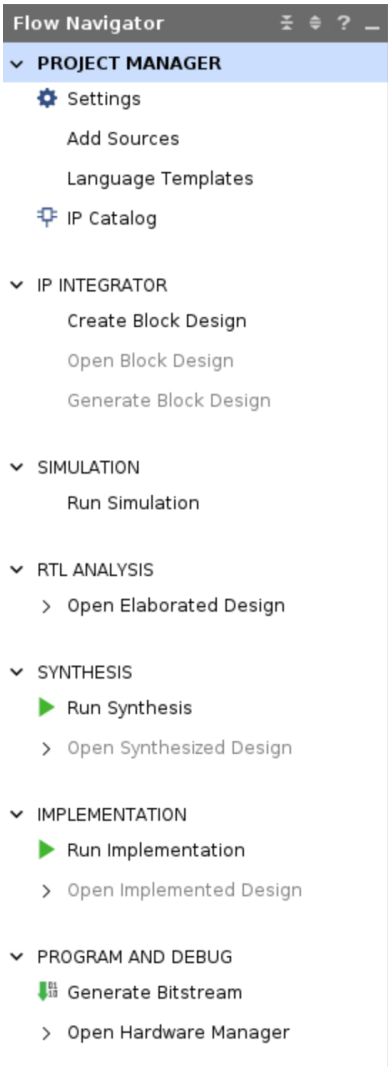
4 Postlab

Submit the deliverables and screenshots (as a single PDF please) to Canvas.

A Design Flow

This appendix provides a brief overview of the Vivado Design flow.

This is the Vivado Flow Navigator, and it will be how you access different design and verification tools. Our design flow will typically follow this general trend but will have a few differences.



1. You should always start with a design on paper, where you document a full solution to the problem you are trying to solve. This will involve some combination of high-level block diagrams, circuit schematics, state diagrams, and truth tables. You should not start coding until you have done this and feel confident in your solution. *You should also not ask for help without a paper design.*
2. *TRANSLATE* your paper design into a behavioral model with VHDL.
3. Create a Vivado project and elaborate the design with the RTL Analysis tool in the design flow. This will generate a set of schematics that you can compare to your paper design. Check for common issues, like latches, dangling wires, or improper wiring.
4. Write a testbench file, then simulate the design. Verify that the timing diagram matches your expectations, and check for edge cases. Errors that occur will appear in the TCL (pronounced "tickle") console on the bottom bar.
5. Take the design through synthesis, and check the message console on the bottom bar. Clear all errors, critical warnings, and problematic warnings (some warnings will be ok, but it's hard to tell at first. Ask a member of lab staff if you are unsure!). Make sure that you turn off "info."
6. Write a constraints file, and run implementation. Check the message console on the bottom bar. Clear all errors, critical warnings, and problematic warnings (some warnings will be ok, but it's hard to tell at first. Ask a member of lab staff if you are unsure!).
7. Run generate bitstream. Check the message console on the bottom bar. Clear all errors, critical warnings, and problematic warnings (some warnings will be ok, but it's hard to tell at first. Ask a member of lab staff if you are unsure!).
8. Open the hardware manager and program your FPGA. Test your design in hardware, using the scope if you need to. If you need to use the scope, compare against simulation results.