CSE 100/L Logic Design
Fall 2025

Lab Assignment 3

Code Submission deadlines:

- 100% Before the end of your section Tuesday, October 21st.
- 95% after the end of your section until 11:59 PM Tuesday, October 21st.
- 80% after 11:59PM Tuesday, October 21st.
- 60% after 11:59PM Wednesday, October 22nd.
- 40% after 11:59PM Thursday, October 23rd.
- 20% after 11:59PM Sunday, October 26th.
- 0% after 11:59PM Monday, October 27th.

Demo deadline:

- Code may be demonstrated until the end of the next lab assignment. You must submit before you demo, and the code you submit must be the code you demo. You may be required to download the code from Canvas to demo.

Write-Up deadline:

- There is no write-up submission for this lab assignment.

Academic Integrity Policy:

The Academic Integrity Policy is stated in the syllabus on Canvas. Students are responsible for reading the syllabus. Not reading the syllabus is not an excuse for academic misconduct.

- Work submitted in CSE 100 must be yours alone. Sharing/sending code, submitting code written by others, and submitting (any) code from generative AI **are all violations of academic integrity**.
- Working together on a whiteboard, on paper, debugging waveforms, explaining concepts and/or examples, is encouraged, **as long as no solution code is shared**.

Verilog Operator and Procedural Block Constraints:

In this lab you can use any of the following combinational operators:

- All combinational logic must be implemented using only `assign` statements.
- Bit-wise Operators: &, |, ∼, ∧, and ∼ ∧
- Concatenation and Replication: {} and {{}}

Your designs must be **synchronous with the system clock** specified in the lab. This means:

- use only positive edge-triggered flip-flops (FDRE),
- not use asynchronous clears or pre-sets of any sequential elements,
- connect only **the system clock** as input to the clock pins of any sequential components,
- not connect **the system clock** as the input to any other logic.
- you may **not** use any procedural blocks in your design (i.e. `always@`, `always_ff@`, `always_comb`).

You may only use **`assign`** statements and FDREs in your design. Use of disallowed operators or procedural blocks will be considered behavioral and will result in a **score of 0**!

Design Overview

In Lab 3 you will build:

- a 4-bit binary counter which counts up and down and is loadable,
- an edge-detector,
- a ring counter to control the 7-segment displays,
- a selector to choose one out of four 4-bit buses, and
- re-use the 7-segment display module you created in Lab 2, but it must be your own code.

These parts will be assembled as below to build a 16-bit counter and display its content in hexadecimal on the four 7-segment displays, plus two LEDs.

The leftmost LED (`led[15]`) will be lit when the 16-bit counter is at its highest value, hex `0xFFFF`, and the rightmost LED (`led[0]`) will be lit when the counter is at its lowest value 0.
The counter will:

- increment each time pushbutton `btnU` is pressed,
- decrement each time pushbutton `btnD` is pressed, and
- count up continuously, while `btnC` is held down **except** in the range `0xFFFC` to `0xFFFF`.
- Incrementing at value `0xFFFF` will result in `0x0000` and decrementing from `0x0000` will result in `0xFFFF`.

Your counter is also **loadable**: when pushbutton `btnL` is pressed, the 16-bit number determined by the position of the rightmost 16 switches (`sw[15:0]`) is loaded into the counter.

Read all of the instructions below before beginning any design.

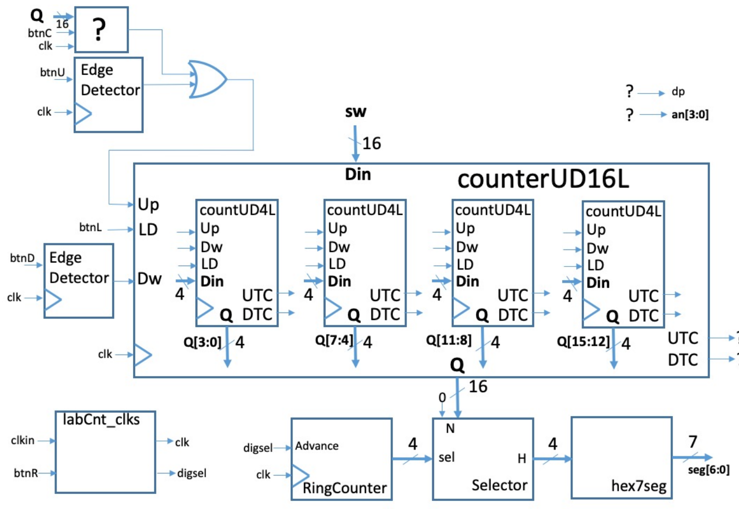

Figure 1: Lab 3 overview

4-bit Counter Design

Start by designing a 4-bit binary loadable counter **countUD4L** using the **FDRE** module that is part of the "built-in" Unisim library.

Your 4-bit counter, **countUD4L**, should have 5 inputs:

- `clk_i`: The system input clock
- `up_i`: Increment input port
- `dw_i`: Decrement input port
- `ld_i`: Load control input port
- `Din_i`: Data input port, that will be loaded into the counter on the positive clock edge if `ld_i` is high

Your 4-bit counter has 3 outputs:

- a 4-bit bus, `Q_o` which is the current value held by the counter,
- the signal `utc_o` (Up Terminal Count) which is 1 only when the counter is at `4'b1111` (15 in decimal), and
- the signal `dtc_o` (Down Terminal Count) which is 1 only when the counter is at `4'b0000`.

Below is the module port list for the **countUD4L** module:

```
module countUD4L(
    input clk_i,
    input up_i,
    input dw_i,
    input ld_i,
    input [3:0] Din_i,
    output [3:0] Q_o,
    output utc_o,
    output dtc_o
);
```

You must **simulate your 4-bit counter before proceeding.**

- Make sure it doesn't count when `up_i` is low and `dw_i` is low.
- Make sure it loads whenever `ld_i` is high regardless of `up_i` or `dw_i`.
- Make sure `utc_o` is high only when the counter bits are all 1's and `dtc_o` is high only when the counter bits are all 0's.

You will need a clock to simulate a sequential circuit such as a counter. You can borrow some of the lines from the Verilog code in the simulation step (below) to provide a clock.

16-bit Counter Design

To build your 16-bit counter, use a few additional gates to connect four instances of your 4-bit counter to make a 16-bit counter as in the overview figure. Your 16-bit counter will have the same ports as the 4-bit counters except that its input and port ports will be 16-bit vectors. Its `utc_o` output should be 1 when your 16-bit counter is at hex `0xFFFF` and its `dtc_o` output should be 1 when your 16-bit counter is at `0x0000`.

Selector

Rather than using four **hex7seg** symbols, one for each digit, and merging their outputs (as we did in Lab 2), it is more economical to use only one of these symbols and feed it 4-bit values one at a time for each of the four digits.

The **Selector** has a 16-bit bus, `N_i[15:0]`, along with a 4-bit control input, `Sel_i[3:0]`. Its output is a 4-bit bus, `H_o[3:0]` which is a 4-bit range of the `N_i[15:0]` bus. Specifically:

```
H_o is N_i[15:12] when Sel_i=(1000)
H_o is N_i[11:8] when Sel_i=(0100)
H_o is N_i[7:4] when Sel_i=(0010)
H_o is N_i[3:0] when Sel_i=(0001)
```

Below is the module port list for the **selector** module:

```
module selector(
    input [3:0] Sel_i,
    input [15:0] N_i,
    input [3:0] H_o,
);
```

When more than one of the control inputs is high, or they are all 0, the output `H_o` can be anything ("Don't Care").

Ring Counter

A ring counter holds a bit vector which has a single 1 bit. It has a control input, `advance_i`, along with a clock input, `clk_i`. If `advance_i` is high on the positive clock edge then all bits shift to the right. The 0th bit will replace the top bit, shifting the bits in a ring.

Chapter 4 of the text has a section on "Registers" that describes the **Ring Counter**. You will need to make sure that one of the FFs has a 1 on reset. In the text, this is accomplished by placing two inverters around one of the FFs, though there are other ways to do this.

Below is the module port list for the **ring_counter** module:

```
module ring_counter(
    input clk_i,
    input advance_i,
    output [3:0] ring_o,
);
```

Edge Detector

Pushbuttons `btnU` and `btnD` will be connected to the input of an **Edge Detector**. An **Edge Detector** will generate a high value for one clock cycle if the past two inputs consist of a 0 followed by a 1. Design a small sequential circuit to implement the **Edge Detector**.

Below is the module port list for the **edge_detector** module:

```
module edge_detector(
    input clk_i,
    input button_o,
    output edge_o,
);
```

Recommended Design Procedure

1. Design your top level such that it has the following inputs:

   - `clk` - this is the 100MHz clock on the BASYS3 Board
   - `btnR` - this button will be connected the built-in global reset of the Artix 7 FPGA
   - `btnU` - when pressed this button causes the counter to increment by just one
   - `btnD` - when pressed this button causes the counter to decrement by just one
   - `btnC` - when pressed this button causes the counter to advance continuously **except in the FFFC to FFFF range**
   - `btnL` - when pressed this button causes the counter to load the value determined by the switches on the clock edge
   - `sw` - this 16 bit vector determines the value loaded into the counter when btnL is pressed

   and the following outputs:

   - `seg` - this 7 bit vector controls the segments in the 7-segment display,
   - `dp` - this controls the dp segments in the 7-segment display,
   - `an` - this 4 bit vector controls the 4 digits of 7-segment display,
   - `led[15]` - this is the leftmost LED below the switches and should display UTC.
   - `led[0]` - this is the rightmost LED below the switches and should display DTC.
   - NOTE: Set `led[14:1]` to 0 so that they will be connected to 0, and not display a random value.

   Below is an example of what the top level ports list can look like:

```
module top(
    input clk,
    input btnU,
    input btnD,
    input btnR,
    input btnC,
    input btnL,
    input [15:0] sw,
    output [15:0] led,
    output [3:0] an,
    output [6:0] seg,
    output dp
)
```

2. In your top level, add and connect the modules for your **Edge Detector**, **Ring Counter**, **16-bit Counter**, **Selector** and **hex7seg**. You will need to provide logic so that depressing btnC does not advance the counter when it is in the range `0xFFFC` to `0xFFFF`.

3. Connect the `clk` inputs of the Edge Detector and Counters to a net named **clkin**. This is the **system clock** for the design. It is the only signal which can be used as a clock in your design.

4. Download **labCnt_clks.v**, found in the Lab 3 folder in the Lab Resources in Canvas, into your project directory.

5. In the Vivado Project Manager, add it to your project. Make sure you select the option to copy it into your project.

6. Add an instance of the module **labCnt_clks** to your top level as follows:

```
labCnt_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel));
```

The signal `clkin` is your system clock. The signal `digsel` should be used to advance the Ring Counter; it should not be used as a clock!!!

Pushbutton `btnR` should be connected only to the `greset` input of **labCnt_clks** module, no where else!

7. Edit the constraint file. You'll need to setup the clock as in Lab 2 (the external clock is labeled `clkin`) and uncomment the lines for the inputs/outputs we are using.
8. Implement your design. Even if you are not ready to download the bitstream to the board, running these two last steps, **Implement and Generate Bitstream**, can catch errors that may be more difficult to find through simulation.
9. Simulate your design.

- Providing values for the input `clkin` yourself would be tedious. Instead copy and paste the following text into your Verilog Testbench to generate a periodic signal for `clkin`:

```
parameter PERIOD = 10;
parameter real DUTY_CYCLE = 0.5;
parameter OFFSET = 2;

initial    // Clock process for clkin
begin
   #OFFSET
   clkin = 1'b1;
   forever
   begin
       #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
   end
end
```

- In the testbench add values for your pushbuttons and vary them over time. Advance time by a multiple of 100ns between changes. The **clkin** will only become active after 1000ns, so advance time by at least that amount before varying any inputs. Make sure pushbutton btnC is high and low for multiple consecutive clock cycles in your testbench. Pushbuttons btnU and btnD should also to held high for multiple clock cycles. You will want to add the 16-bit bus output of your counter to the Waveform Viewer.
- Before attempting to demo your design please also simulate the **testTC.v** file, found in the Lab 3 Canvas folder, that verifies your UTC goes high (at about 1.3ms), stays high and then returns low as required. After simulating this testbench for 1.4ms, check the value of TX_ERROR; it should still be 0. (The TA will be annoyed if she/he spends the 40 seconds to discover that your TC is not working and you have not simulated this testbench.)

1. ***Implement your design, configure the FPGA and demonstrate your design and simulation to the TA.***
2. Remember to archive your project. Files left on the PC are not 100% guaranteed to reappear when you login next time.
3. ***Submit your archive of the exact project used in your demonstration to the "Lab Assignment 3: Demo and Code Submission" assignment in Canvas.***
4. **Important** Please remember to turn off the power to the BASYS3 board when you are done.

ran

Rubric

There is no partial credit for this lab. All students must completely satisfy the lab description, as many components are used in later labs.