

## CSE 260M Lab #2

In this laboratory, starting with your Vivado project from Homework 4, you will add circuitry to your difference engine implementation that computes values of the 4th-order polynomial

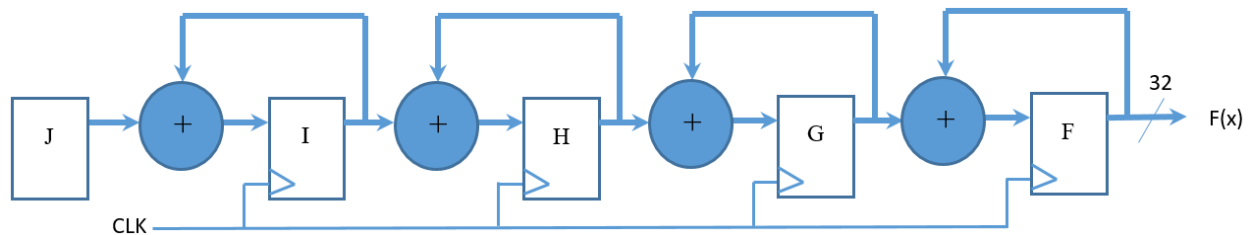
$$f(x) = 4,194,304 - 491,520x + 17,920x^2 - 240x^3 + x^4 \text{ for } x=0, 1, 2, \dots,$$

to display the four roots of the quartic equation

$$4,194,304 - 491,520x + 17,920x^2 - 240x^3 + x^4 = 0$$

on the 7-segment display on the Digilent Artix-7 demo board.

The difeng.vhd file used in Homework 4 implements the difference engine from "The Difference Engine" document on Canvas shown in block diagram form below. Not shown in this block diagram, but included in the VHDL description of the difference engine, is the logical ability to reset the difference engine to the initial compute state associated with  $F(0)$  by asserting an external "reset\_l" input. Make sure you understand how the basic difference engine from Homework 4 is initialized using reset\_l (the "\_l" implies that reset is asserted low).



Because reset\_l can be asserted asynchronously to the clock, the implementation of Homework 4 has a problem that could result in the difference engine, when tested in hardware, starting out incorrectly: Some of the flip-flops in the registers may see reset\_l removed prior to a rising clock edge, while others may not, due to the fact that reset\_l has to be routed around the FPGA die to all of the flip-flops using physical wires. To fix this issue, as well as to eliminate possible metastable behavior in the flip-flops because of a setup or hold time violation, you will need to include the reset synchronizer discussed in class in your VHDL description.

As a first step for Lab 2, add the reset synchronizer VHDL process from the "Reset Synchronizer" document on Canvas to your difeng.vhd source file. **Do not, under any circumstances, integrate the reset synchronizer into the registers process!** You will need to edit your original difference engine process to use "reset\_l\_sync" instead of "reset\_l" so the synthesized difference engine uses the synchronized reset instead of the asynchronous reset. **Many students end up adding the synchronizer but then fail to make this edit, so be careful!**

In addition to adding the reset synchronizer, for Lab 2, you will need to edit the "registers" process to add a new 8-bit register called "X." X(7:0) should be an 8-bit standard logic vector that is reset to zero by reset\_l\_sync and then incremented by one each time F, G, H, and I are

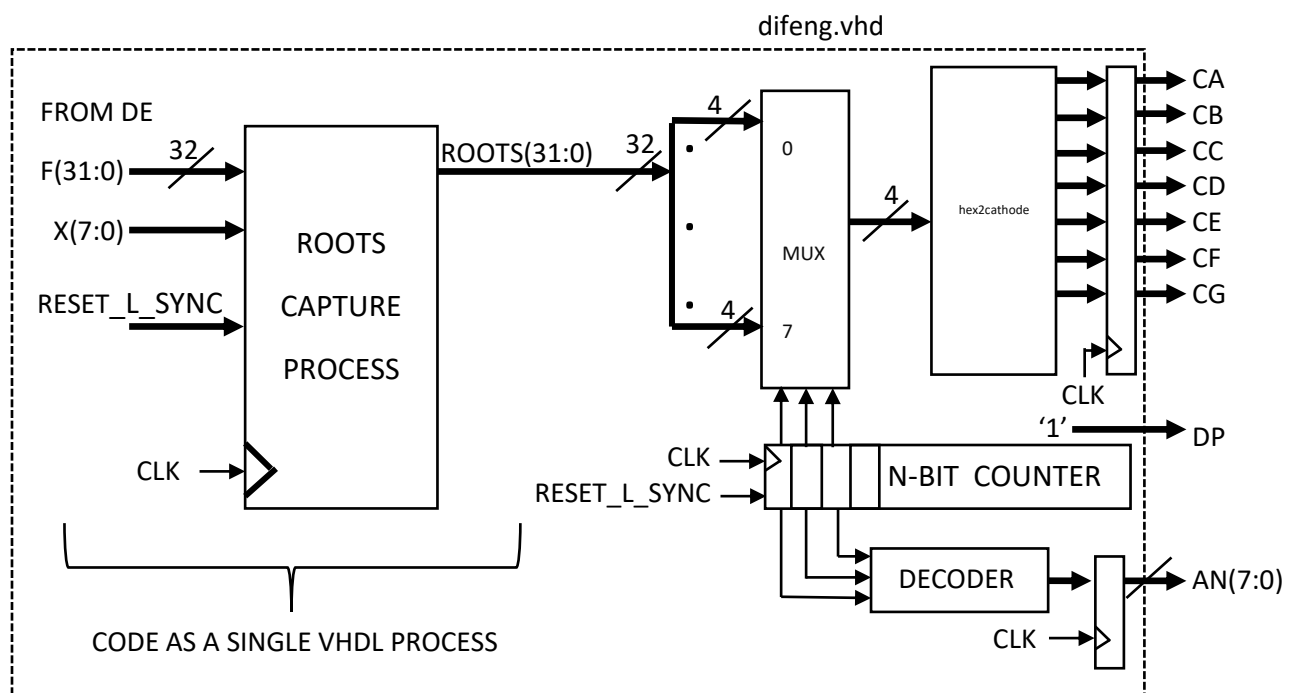
updated inside the difference engine process. This creates the value "x" associated with each computed value of  $f(x)$ . When  $f(x) = 0$ ,  $x$  is a root of the quartic equation. Here, it is assumed that the roots of the quartic equation will be real and less than 256 so they can be represented using an 8-bit (unsigned) value. Your modified difference engine process should always update on a rising clock edge after `reset_l_sync` is de-asserted, keeping the process "elegant" and the logic generated able to run as fast as possible in the target FPGA.

Your Lab 2 design will not have  $F(31:0)$  as an output:  $F$  will be an internal signal like  $G$ ,  $H$ ,  $I$ , and  $J$  in this new design. Edit the `difeng.vhd` entity to remove  $F$  from the entity and add the internal signal  $F$  you will need in your final design. Do not worry that your entity has no outputs in this intermediate state: Your design will simulate just fine even without outputs!

With your reset synchronizer and  $X$  register added to your VHDL description, and  $F$  changed from an output to an internal signal, run Vivado and simulate your modified difference engine. You should see  $F$  update as it did in Homework 4, and, in addition, you should see  $x$  update as well. Make sure  $F(x)$  has the correct value associated with each value of  $x$ !

It cannot be stressed how important completing the base design described above is relative to how easy it will be to implement the remainder of the project. Debugging your entire design will be far more difficult if your base design does not work properly, so make sure your base design is working properly before continuing.

Once your base design is solid, add display circuitry to your implementation to display the roots of the quartic equation. Base your VHDL implementation of the display circuitry on the block diagram shown below. Here,  $X(7:0)$  and  $F(31:0)$  are generated by your "registers" process.



A separate process, shown in the block diagram above, is required to capture the first four (4) values of  $X$  associated with the four roots of the quartic equation. The `reset_l_sync` input, when asserted at a rising clock edge, should force `ROOTS(31:0)` to zero. After `reset_l_sync` is removed, the first occurrence of  $f(x) = 0$  should cause `ROOTS(7:0)` to be assigned the current value of  $X$ , the second occurrence of  $f(x) = 0$  should cause `ROOTS(15:8)` to be assigned the current value of  $X$ , etc., until four values have been saved, after which updates to `ROOTS` stops. With this approach, the polynomial values produced by the difference engine can, and will ultimately, overflow and become incorrect, but it simply will not matter since `ROOTS` will never again update unless the entire computation process is started again from the beginning via asserting of `reset_l_sync`. Make sure you understand why the polynomial calculation will overflow! **Note: You do not need to use "variables" to implement the ROOTS process!**

The four 8-bit values held in the 32-bit `ROOTS` register are displayed as a single 32-bit hexadecimal number using the remaining circuitry in the block diagram. As an example, if the four roots are  $X = 5, 9, 15$ , and  $127$ , the hexadecimal value displayed will be `7F0F0905`.

The display circuitry is based on an  $n$ -bit binary counter **that must be implemented using a dedicated process**. This  $n$ -bit counter is sized (in terms of the number of bits in the counter) so that the upper three (3) bits can be used to multiplex between the eight (8) different 4-bit `ROOTS` nibbles to be displayed at the optimal rate required for display (refer to the dev board manual): For each of the eight (8) different combinations of the upper three bits, a different 4-bit value is fed to the cathode logic block. The code for this block (`hex2cathode.vhd`) will be supplied by the instructor. The three (3) upper bits also set the associated anode output low. Output flip-flops are used to eliminate any “glitches” caused by races/hazards. **You are required to code these output flip-flops in a separate process.**

**Implement the decoder shown in the block diagram above using a `SELECT` statement coded in the concurrent section of your VHDL architecture. Code the mux shown using a second `SELECT` statement coded in the concurrent section of your VHDL architecture.**

For Lab 2, you will need to create one subcomponent: The `hex2cathode` module. You will need to create “internal” signals `an_int`, `ca_int`, etc., that can be used as inputs to the output flip-flops.

You can copy the XDC file from your Lab 1 Vivado project and place it in your Vivado project folder to use as the starting point for the XDC file you need for Lab 2. Rename your XDC file `difeng.xdc` and add it to your Vivado project after making the appropriate edits. Your XDC file should contain definitions for all the pins on your final design (and only for the pins used on the final design). Alternatively, you can use the GUI approach from Lab 1 to create a new XDC file if you so desire.

The following will also be required in your XDC file for Lab 2:

- ➔ Assign the clock to the pin associated with the 100 MHz clock provided on the board
- ➔ Assign the reset\_l signal to the pin associated with the “CPU RESET” pushbutton

Use these two lines at the top of your XDC file as you did in Lab 1:

```
set_property CFGBVS VCCO [current_design]
set_property CONFIG_VOLTAGE 3.3 [current_design]
```

Replace the “virtual clock” constraint you used in Lab 1 with a constraint for the 100 MHz clock on the board:

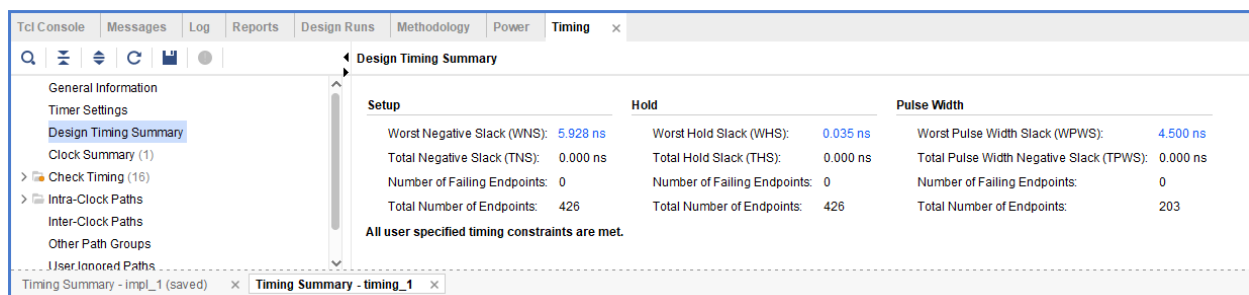
```
create_clock -period 10.000 -name clk -waveform {0.000 5.000} [get_ports clk]
```

Simulate your design to make sure it works properly and that the ROOTs register captures the correct values. Take a screen shot of the simulation showing the instant the difference engine computes the last root value with the yellow cursor placed at exactly the point where the F register is set to zero and X is set to the value of the 4th root of the quartic equation. Show all of the signals in your design in your screen shot. Your screen shot must be named "simulation.png" to receive credit. Screen shots with any other name will receive zero credit.

So that everyone gets the same computation time in their screen shot, use a 100 MHz clock in your simulation, and force reset\_l low for 55 ns before releasing it. Note: **You never force internal signals like reset\_l\_sync! You only need to force reset\_l and clk!**

Implement your design, and generate a .BIT file for testing with a TA.

Document your timing results by capturing a screen shot of the “Design Timing Summary” produced by Vivado (look in Open Implemented Design ➔ Report Timing Summary). An example of what is required is shown. Your timing results should be similar but will not be identical to these results. With positive slack, the tools indicate that your design will run at the required 100 MHz clock frequency. **If you have negative (red) slack, you have a problem that must be fixed prior to submission.**



Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 5.928 ns	Worst Hold Slack (WHS): 0.035 ns	Worst Pulse Width Slack (WPWS): 4.500 ns
Total Negative Slack (TNS): 0.000 ns	Total Hold Slack (THS): 0.000 ns	Total Pulse Width Negative Slack (TPWS): 0.000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 426	Total Number of Endpoints: 426	Total Number of Endpoints: 203

All user specified timing constraints are met.

Submit the following via Canvas for grading prior to the submission deadline:

- 1) Your difeng.vhd VHDL source file\*;
- 2) Your difeng.xdc Vivado constraint file\*;
- 3) Your difeng.bit file\*.
- 4) Your Zipped Vivado project folder;
- 5) Your simulation screen shot showing the exact instant the F register is set to zero and x changes to the 4th root of the polynomial (name your file simulation.png)\*\*;
- 6) A screen shot of your "Design Timing Summary" (name your file timing.png)\*\*;

\* These files will be in your Zipped Vivado project folder and should not be submitted separately.

\*\* Place these files inside your Vivado project folder, i.e., do not submit them to Canvas separately. Place them in the root of the folder, not in a subdirectory.

Students are required to test their lab in person with a TA prior to the submission deadline. Students are required to have done a full submission prior to testing. Do not email BIT files to the TAs: You must bring your BIT file on a flash drive.

## GRADING RUBRIC

Properly created Vivado project folder: 10%

Commented VHDL implementation of the DE, the roots capture process, and display logic: 40%

Accurate XDC constraint file: 10%

Simulation screen shot: 10%

Design timing summary screen shot: 10%

Properly functioning .BIT file tested prior to the submission deadline with a TA: 20%