

Queen's University
Department of Electrical and Computer Engineering
ELEC 271 Digital Systems
Fall 2018

Lab 1:
Introduction to Digital Logic Circuits,
FPGA Technology, and Computer-Aided Design Software

Copyright © 2018 by Dr. Naraig Manjikian, P.Eng.
All rights reserved.

*Any direct or derivative use of this material
beyond the course and term stated above
requires explicit written consent from the author,
with the exception of future private study and review
by students registered in the course and term stated above.*

Objectives

This introductory laboratory activity for *ELEC 271 Digital Systems* provides the opportunity for students to:

- obtain an initial awareness of combinational and sequential logic circuitry used in digital systems,
- compare text-based and graphical representation of basic logic functions and behaviors,
- initiate an exploration of the technology of field-programmable gate-array (FPGA) chips,
- understand the relation between the simulation of logic circuits and their operation in hardware,
- comprehend individual binary signals and the representation of numbers as binary bit vectors,
- acquire initial experience with industrial-calibre computer-aided design (CAD) software tools, and
- become familiar with the main physical components of the Altera DE0 board used in laboratory activity.

To obtain the best learning benefit, [read this description thoroughly](#) before arriving at your lab session.

During the lab session, each member of a lab group should have an opportunity to use the software. There are seven parts to this exercise; each member of the lab group can pursue each part in a round-robin fashion. As the term progresses, all students should become confident in using all of the software features explored here.

Part 1: Basic Combinational Logic Circuit Description and Implementation in FPGA Chip

- Click on the Windows icon in the lower left corner, and type “quartus” as the search text. From the list of icons that appears, click on “Altera Quartus II (32-bit)” or “Altera Quartus II (64-bit)” to start the software.
- In the dialog box shown in the middle of the screen, click on *Create a New Project*. Alternatively, close the dialog box (click on “X” in upper right corner), and select *File → New Project Wizard* from the menubar. When the New Project dialog box appears, skip the Introduction portion by clicking on *Next*.
- Now enter `z:\MyFiles\ELEC271\LAB1\COMB_LOGIC` as the working directory for the project
- Enter `comb_logic` as the name of the project. Click on *Next*. Say ‘yes’ when asked to create the new working directory. For the Add Files part, there are no existing files to add now, so click on *Next* again.
- To select the device that is found on the DE0 board, choose *Cyclone III* from the list of device families, choose the *FBGA* package, choose *484* for the pin count, choose *6* for the speed grade, and then finally click on the first of the four devices listed (EP3C16F484C6) to highlight it in blue, and click *Next*.
- Click *Next* for EDA Tool Settings without making any changes. Finally, for the Summary, click *Finish*.
- From the main menubar, select *Assignments → Device...* and then click on *Device and Pin Options...*
- For *Unused Pins*, change the option for ‘Reserve all unused pins’ to *As input tri-stated*.
- For *Voltage*, change the ‘Default I/O standard’ to *3.3-V LVTTTL*.
- Select *OK* to close the Device and Pin Options dialog box. Select *OK* again to close the Device dialog box.
- From the main menubar, select *File → New...* In the New dialog box, select *VHDL File* and click on *OK*.
- Enter the text in the box into the editor. It is a description of a simple logic circuit in the VHDL language.

```
library ieee;
use ieee.std_logic_1164.all;

entity comb_logic is
  port (v, w, x, y : in  std_logic;
        f, g       : out std_logic);
end entity;

architecture synth of comb_logic is
begin
  f <= v and w;
  g <= x or y;
end architecture;
```

(Definitions needed by the VHDL language for the ‘std_logic’ signal type.)

(Definition of a logic ‘entity’ with labelled input/output port signals; these signals will be assigned to input/output pins of the FPGA chip.)

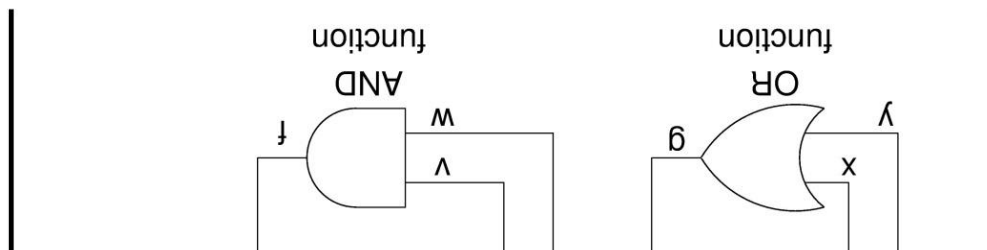
(Definition of the logic behavior to be synthesized for this entity; one output pin will be the combinational logic ‘AND’ of two inputs; another output pin will be the combinational logic ‘OR’ of two inputs.)

- From the main menubar, select *File → Save*. The Save As dialog box will appear. Click on *Save*.

- Select *Processing → Start Compilation* from the main menubar in order to synthesize the circuit.
- After synthesis, select *Assignments → Pin Planner* from the menubar. Enter the pin locations at right. Close the Pin Planner (click on ‘X’ in upper corner).
- Select *Processing → Start Compilation* again from the main menubar to resynthesize with pin locations.

| | Node Name | Direction | Location |
|-----|-----------|-----------|----------|
| out | f (LEDG0) | Output | PIN_J1 |
| out | g (LEDG9) | Output | PIN_B1 |
| in | v (SW0) | Input | PIN_J6 |
| in | w (SW1) | Input | PIN_H5 |
| in | x (SW8) | Input | PIN_E4 |
| in | y (SW9) | Input | PIN_D2 |



- Plug the USB cable into the DE0 board. Turn on the power with the red button on the side of the board.
- From the menubar, select *Tools → Programmer*. Ensure “USB-Blaster” appears beside *Hardware Setup*.
- Press the *Start* button to program the FPGA chip with the synthesized circuit from the description above.
- Align the labelled template below with the bottom edge of the Altera DE0 board and its switches/LEDs.
- Experiment with the switch inputs and the resulting output changes on the LEDs. *Understand the behavior* of the ‘AND’ and ‘OR’ logic functions that are implemented in the circuit that is now configured in the chip.



Part 2: Schematic View of Logic Circuit Specification and Chip Planner View of Implementation

- On the main Quartus menubar, click on *Tools* → *Netlist Viewers* → **RTL Viewer**. Study the schematic that appears on the screen and relate it to the VHDL specification that you entered in the preceding part. It is the result of processing the VHDL specification by the Quartus software. It is presented in a graphical form for humans, but the software internally uses complex data structures to represent arbitrarily large circuits. As users of such software, we rely on its creators ensuring correctness in how it processes descriptions for chip implementations from human input. Our key concern is the correctness of our own circuit descriptions.
- On the main Quartus menubar, click on *Tools* → *Chip Planner*. This new window reflects the internal organization of the Cyclone III chip. The toolbar on the left has various icons including a selection arrow, a \pm magnifying glass for zooming, and a hand icon for moving the view up/down/left/right at current zoom.
- On the right of the window is the Layers Setting box. Click to put checkmarks beside Routing Details and beside Logic Details. These options tell the software to draw more details on the screen when zoomed in.
- At the initial view-the-entire-chip zoom level, you should see a large number of rectangular boxes. Each box represents a logic array block (LAB) which has within it 16 logic elements (LEs). Two of the LABs on the screen should have a darker blue color indicating that at least one LE within that LAB is being used.
- Click on the \pm magnifying glass icon. Press and hold the left mouse button to click-and-drag a small box around one of the two LABs mentioned above. Release the button, and note how the connection details appear after zooming in. Click on the left/right mouse buttons to adjust the zoom. Use the hand icon (or the scrollbars) to adjust the on-screen view. You should see that one of the LEs within the LAB is being used.
- Click on the arrow icon on the left, then *double-click* on the used LE within the LAB. The LE details should appear, consisting of a four-input look-up table (LUT) with two inputs used, and a single storage flip-flop that is not used here. As the course proceeds, you will learn more about these components in the chip.

Part 3: Waveform Simulation of Combinational Logic Circuit

- On the main Quartus menubar, click on *File* → *New*. In the dialog box that appears, select *University Program VWF* under the tab labeled *Verification/Debugging Files*. Click on *OK*.
- A waveform editor window should now appear. A good habit to develop is to save new files right away with an appropriate name. Thus, click on *File* → *Save As ...* in the waveform editor menubar. In the filename dialog box, *navigate* to the `COMB_LOGIC` folder, enter `test_comb_logic.vwf`, and click on *Save*.
- Position the mouse pointer in the area below the *Name* heading. Click on the right mouse button to obtain a popup menu. Select *Insert Node or Bus ...* and then select *Node Finder ...* to open a second dialog box.
- Select *Pins:all* from the *Filter* list (and note the "*" wildcard for pattern matching beside the *Named* entry).
- Click on *List* to produce a complete list of all the inputs and outputs for the design created in Part 1.
- One by one, select a signal under *Nodes Found*, and click on the > button to move it to *Selected Nodes*. Follow the order **v**, **w**, **f**, **x**, **y**, **g**. After completing the selections, click on *OK* to close Node Finder.
- Click on *OK* in the remaining dialog box, and the waveform display should then show six waveforms.
- The waveform display has vertical dashed lines delineating time intervals of 10 nanoseconds. Using these lines as a guide, you will now set the logic levels for the two inputs of each gate to the four possible input combinations (00, 01, 10, and 11) in order to test functional (i.e., zero-delay) behavior with a simulation.
- For inputs **v** and **x** one at a time, use the left mouse button to highlight the interval 160-320 nanoseconds and then click on the  level-setting button in the waveform editor toolbar to change the waveform level.
- For inputs **w** and **y** one at a time, use the left mouse button to highlight the interval 80-160 nanoseconds and then click on the  level-setting button in the waveform editor toolbar to change the waveform level. Repeat the process to select the interval 240-320 nanoseconds and click on the "1" level-setting button.
- Select *File* → *Save* from the waveform window menubar to save the updated waveform file.
- Select *Simulation* → *Run Functional Simulation* from waveform window menubar. A text window will appear temporarily as the simulation software executes, then a new read-only output waveform editor will appear.
- First, examine the waveform for output **f** and verify that it reflects the AND function of inputs **v** and **w**. Then, examine the waveform for output **g** and verify that it reflects the OR function of inputs **x** and **y**. Because the outputs change *instantaneously* with zero delay after input changes, we have a *functional* simulation. In contrast, a *timing* simulation would have delay before output changes to reflect real circuits.

Part 4: Description/Implementation of Binary Counter with Arithmetic and Storage Elements

- Select *File* → *New Project Wizard* from the menubar, and skip the Introduction portion by clicking on *Next*.
- Now enter `z:\MyFiles\ELEC271\LAB1\COUNTER` as the working directory for the project.
- Enter `counter` as the name of the project. Click on *Next*. Say 'yes' when asked to create the new working directory. For the Add Files part, there are no existing files to add now, so click on *Next* again.
- To select the device on the DE0 board, follow the same instructions given in Part 1 of this exercise.
- Click *Next* for EDA Tool Settings without making any changes. Finally, for the Summary, click *Finish*.
- From the main menubar, select *Assignments* → *Device...* and then click on *Device and Pin Options...*
- For *Unused Pins*, change the option for 'Reserve all unused pins' to *As input tri-stated*.
- For *Voltage*, change the 'Default I/O standard' to 3.3-V LVTTTL.
- Select *OK* to close the Device and Pin Options dialog box. Select *OK* again to close the Device dialog box.
- From the main menubar, select *File* → *New...* In the New dialog box, select *VHDL File* and click on *OK*.
- Enter the text in the box into the editor. It is a description of a circuit with arithmetic and storage elements.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity counter is
    port (clk, reset_n, sw0 : in std_logic;
          q : out std_logic_vector(9 downto 0));
end entity;

architecture synth of counter is

    signal full_count : std_logic_vector(31 downto 0);

begin

    the_full_counter: process (clk, reset_n)
    begin
        if (reset_n = '0') then
            full_count <= (others => '0');
        elsif (rising_edge(clk)) then
            full_count <= full_count + 1;
        end if;
    end process;

    q <= full_count(31 downto 22) when (sw0 = '0')
        else full_count(21 downto 12);

end architecture;
```

(Definitions needed by the VHDL language for the 'std_logic' and 'std_logic_vector' types, and also the definition of the '+' operator.)

(Definition of a logic entity with the necessary input/output port signals that will be assigned to pins of the FPGA chip.)

(Definition of a 32-bit binary count vector that will be incremented at a frequency of 50 MHz.)

(Definition of the storage and arithmetic for the vector defined above; on reset, all bits are set to zero; on each rising edge of the clock signal,



the count vector will be incremented.)

(Definition of the connections to ten LEDs as two different subsets of bits from the vector, depending on the setting of an input switch.)

- From the main menubar, select *File* → *Save*. The Save As dialog box will appear. Click on *Save*.
- Select *Processing* → *Start Compilation* from the main menubar in order to synthesize the circuit.
- After synthesis, select *Assignments* → *Pin Planner* to enter the pin locations below, then close Pin Planner.

| in | clk | Input | PIN_G21 |
|-----|--------------|--------|---------|
| out | q[9] (LEDG9) | Output | PIN_B1 |
| out | q[8] (LEDG8) | Output | PIN_B2 |
| out | q[7] (LEDG7) | Output | PIN_C2 |
| out | q[6] (LEDG6) | Output | PIN_C1 |
| out | q[5] (LEDG5) | Output | PIN_F1 |
| out | q[4] (LEDG4) | Output | PIN_F2 |







| | | | |
|-----|-------------------|--------|--------|
| out | q[3] (LEDG3) | Output | PIN_H1 |
| out | q[2] (LEDG2) | Output | PIN_J3 |
| out | q[1] (LEDG1) | Output | PIN_J2 |
| out | q[0] (LEDG0) | Output | PIN_J1 |
| in | reset_n (BUTTON0) | Input | PIN_H2 |
| in | sw0 | Input | PIN_J6 |

- Select *Processing* → *Start Compilation* again from the main menubar to resynthesize with pin locations.
- From the menubar, select *Tools* → *Programmer*. Ensure "USB-Blaster" appears beside *Hardware Setup*.
- Press the *Start* button to program the FPGA chip with the synthesized circuit from the description above.
- Observe the difference in the speed of changes to the LEDs, depending on the SW0 switch setting.
- Observe how the count starts from zero when BUTTON0 is pressed, regardless of the SW0 setting.

Part 5: Schematic View of Counter Specification and Chip Planner View of Implementation

- On the main Quartus menubar, click on *Tools* → *Netlist Viewers* → **RTL Viewer**. Study the schematic that is shown on the screen and relate it to the VHDL specification of the counter circuit you prepared in Part 4.
- The counter circuit consists of both combinational logic and sequential logic. Sequential logic uses storage elements that can hold values for arbitrary periods of time, whereas combinational logic generates outputs purely as a function of current inputs. The sequential circuitry in the counter consists of *flip-flops* forming a multi-bit *register* to hold the current count value. The combinational circuitry consists of an *adder*, and a *multiplexer* for the *q* output. The adder output is connected to the counter register input. On each rising clock edge, the register's flip-flops accept new input values and hold those values until the next rising edge.
- On the main Quartus menubar, click on *Tools* → *Chip Planner*. Enable the options to show Routing Details and Logic Details. Use the toolbar icons to explore the LABs and LEs for the counter implementation. This circuit uses more on-chip resources, including flip-flops in multiple LEs. But this relatively small FPGA chip with more than 15,000 LEs is still almost entirely unused, hence it can accommodate much larger systems.

Part 6: Waveform Simulation of Binary Counter

- In `counter.vhd`, change `(31 downto 22)` to `(9 downto 0)`. Click *Processing* → *Start Compilation*. The reason for this change (and *re-synthesis*) is that simulation is much slower than hardware speed, and seeing meaningful output changes in simulation requires using the 10 least-significant bits of the vector.
- On the Quartus menubar, click *File* → *New*. Then, select *University Program VWF* and click on *OK*.
- In the waveform editor, click on *File* → *Save As ...*, then *navigate* to the `COUNTER` folder. For the filename, enter `test_counter.vwf`, and then click *Save*.
- Position the mouse pointer in the area below the *Name* heading. Click on the right mouse button to obtain a popup menu. Select *Insert Node or Bus ...* and then select *Node Finder ...* to open a second dialog box.
- Select *Pins:all* from the *Filter* list (and note the "*" wildcard for pattern matching beside the *Named* entry).
- Click on *List* to produce a complete list of all the inputs and outputs for the circuit specified in Part 3.
- One by one, select a signal under *Nodes Found*, and click on the > button to move it to *Selected Nodes*. Follow the order **clk**, **reset_n**, **sw0**, **q**. For *q* specifically, select the output signal **group** just once for all 10 outputs together. After completing selections, click on *OK* to close *Node Finder*.
- Click on *OK* in the remaining dialog box, and the waveform display should then show four waveforms.
- Click on the *clk* waveform *label*, which should highlight its entire waveform. Then, click on  in the toolbar.
- In the Clock dialog box, change the default period to 20 ns. Click on *OK* to have a periodic *clk* waveform.
- Click on the *reset_n* waveform *label* to select the full waveform, then click on  to make it 1 for its entirety.
- Use the mouse to highlight the first 20 ns  of *reset_n* then click on  to make the beginning 0. 
- Click on the *sw0* waveform *label* to highlight the entire waveform, then click on  to make it all 0.
- Select *File* → *Save* to save the waveform file. Then, select *Simulation* → *Run Functional Simulation*.
- In the output waveform diagram, use *View* → *Zoom In/Out* to observe the incrementing output values for *q*.

Part 7: Modification of Counter Hardware Implementation for Pushbutton Clock Input

- For this final part of the exercise, the counter VHDL code will be used again, but you will make various changes. The `counter.vhd` file should be the one from Part 6 that has the change of `(9 downto 0)`.
- Modify the `counter.vhd` file to introduce the material highlighted below for a new entity port definition.

```
entity counter is
  port (clk, reset_n, sw0 : in std_logic;
        pushbutton1 : in std_logic;
        q : out std_logic_vector(9 downto 0));
end entity;
```

- Then, introduce two new signals within the architecture, as highlighted below.

```
signal full_count : std_logic_vector(31 downto 0);
signal push_clk_count : std_logic_vector(31 downto 0);
signal tmp, push_clk : std_logic;
```


- Within the existing process definition, make the two changes highlighted below.

```
the_full_counter : process (push_clk, reset_n)
begin
    if (reset_n = '0') then
        full_count <= (others => '0');
    elsif (rising_edge(push_clk)) then
        full_count <= full_count + 1;
    end if;
end process;
```

- Finally, introduce the two *new* processes given below *within* the architecture section (the order is not important). Some labels have been emphasized with dotted lines to ensure that you enter them correctly.

```
the_push_counter : process (clk, reset_n)
begin
    if (reset_n = '0') then
        push_clk_count <= (others => '0');
    elsif (rising_edge(clk)) then
        push_clk_count <= push_clk_count + 1;
    end if;
end process;

the_delay : process (push_clk_count(20), reset_n)
begin
    if (reset_n = '0') then
        tmp <= '1';
        push_clk <= '1';
    elsif (rising_edge(push_clk_count(20))) then
        tmp <= pushbutton1;
        push_clk <= tmp;
    end if;
end process;
```

- Select *File* → *Save*. Then, select *Processing* → *Start Compilation* to check for errors and update the ports.
- After successful compilation, select *Assignments* → *Pin Planner* and enter G3 for pushbutton1.
- Close the Pin Planner and select *Processing* → *Start Compilation* **again** to resynthesize with the new pin.
- Select *Tools* → *Programmer* and press the *Start* button to program the chip with the revised circuit.
- Ensure that SW0 is set to 0 so that the LED outputs reflect bits 9...0 of output vector *q*.
- BUTTON0 on the DE0 board is still the reset signal. But BUTTON1 is the source of the input clock signal. In Part 3, the input clock signal had a frequency of 50 MHz from an oscillator circuit on the DE0 board. Now, the input clock is your finger pressing on BUTTON1, hence the effective frequency is much lower.
- Press and hold BUTTON1. Now, watch the LEDs as you release BUTTON1. The output change occurs when you release the pushbutton because that causes a low-to-high voltage transition that corresponds to one of the rising edges shown in the diagram in Part 3. Continue to experiment with BUTTON0 behavior.

Closing Comments

- The relatively short laboratory experience in this first exercise for *ELEC 271 Digital Systems* is a reflection of more than 60 years of innovation/development in the areas of electronics to fabricate logic circuits and chips, printed-circuit board technology to build systems, software technology to provide sophisticated design tools, and computer organization/hardware to provide the platforms to execute the design software.
 - This exercise has also provided an initial exposure to the theoretical concepts of this course through the graphical representations of logic circuits, and the corresponding text-based descriptions in a form suitable for processing by computer-aided design software. Engineering processes such as system synthesis from behavioral description and functional simulation are other aspects that this exercise has highlighted.
 - Finally, this exercise has provided the opportunity to begin acquiring practical expertise with laboratory hardware and related design software that will be used in this course, and subsequent courses as well.
- *From this experience, you should derive the motivation and focus to learn in more depth about the various aspects that have been introduced by the procedures described in this document. In doing so, you can help increase the likelihood of a successful outcome in this course, and subsequent courses as well.*