Queen's University
Department of Electrical and Computer Engineering
ELEC 271 Digital Systems
Fall 2019

Lab 1:
Canonical Representations of Combinational Functions, Algebraic Simplification, Testbench Simulation, and Multiplexer Functions

Objectives

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

- compare canonical logic functions in sum-of-products and product-of-sums representations,
- view the results of logic simplification performed by computer-aided design software,
- apply the axioms, theorems, and properties of Boolean algebra for logic simplification,
- acquire initial experience with simple testbench simulations for logic verification,
- investigate different logic expressions for, and the behavior of, multiplexer circuits, and
- explore features of the VHDL language for logic specification, synthesis, and simulation.

This description includes a worksheet that each student must have during the scheduled laboratory session and that each student must complete (with reasonable neatness) during the session in order to earn credit.

Preparation BEFORE Your Scheduled Laboratory Session

- *Lab 0 is intended to provide initial guided instruction in using the Quartus II CAD software, hence individual completion of that independent learning exercise is part of the expected preparation for the first formal in-laboratory activity in this description for Lab 1.*
- *Template VHDL code is provided below for use during the Lab 1 activity. Each individual student will prepare two files and have them ready on your network storage or USB drive at the start of your scheduled laboratory session for use during the in-lab procedure.* In this manner, each student will have individual learning benefits, and each student will be prepared with no excuses in the event of someone else being absent or in the event of lost/forgotten USB drive.

- In a file entitled `lab1.vhd`, prepare the following template for later modification.

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

entity lab1 is
     port (?, ?, ? : in  std_logic;
           f       : out std_logic);
end entity;

architecture sop of lab1 is

begin

f <=       (not ?  and  not ?  and  not ?)     -- m0
      or   ...
      or   (    ?  and      ?  and      ?);    -- m7

end architecture;
```

The '?' characters are placeholders for signal (or input variable) names that will be provided to you at the beginning of your scheduled laboratory session as part of a logic function specification. You will replace each '?' with the appropriate signal name.

Note the name `sop` for the architecture to reflect sum-of-products representation of the function.

The template indicates how you will complete appropriate lines for each minterm of the function that you will be given. The '--' comments on the right are suggested as your labelling of any minterms from m_0 to m_7 to aid you in correct preparation of the VHDL code. The '...' indicates that there may be other rows for other minterms in your particular logic function. The function you will be given may or may not include m_0 and/or m_7. *Make the necessary changes for your function*.

Note the use of separate lines to implement the overall OR of the product terms, and the vertical alignment of the AND terms within parentheses across the rows.

These aspects reflect an expectation for you to prepare professional, well-structured descriptions to enable visual inspection by you and by teaching assistants in order to detect errors.

You will also add additional code to the above VHDL description during the laboratory activity.

- In another file entitled `tb_lab1.vhd`, prepare the following template for later modification.

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

entity tb_lab1 is
  -- no external ports/pins for a simulation testbench entity
end entity;

architecture testbench of tb_lab1 is

component lab1
  port (?, ?, ? : in  std_logic;
        f       : out std_logic);
end component;

-- signals connecting to system under test; can be same names as ports
signal ?, ?, ?, f : std_logic;

begin

lab1_under_test : lab1  -- instantiate the system under test
  port map (            --   and map entity ports to testbench signals
             ? => ?,    --   using the format port_name => signal_name
             ? => ?,
             ? => ?,
             f => f
            );

-- this VHDL process drives the simulated progression of time
--   and dictates the input signal values for each interval
--   (there are more sophisticated methods to go through the
--    binary count sequence for the inputs - for later work)
test_driver : process
begin
  ? <= '0';  ? <= '0';  ? <= '0';  wait for 20 ns;
  ? <= '0';  ? <= '0';  ? <= '1';  wait for 20 ns;
     ⋮          ⋮          ⋮             ⋮
     ⋮          ⋮          ⋮             ⋮  -- fill in the rest
     ⋮          ⋮          ⋮             ⋮
  ? <= '1';  ? <= '1';  ? <= '1';  wait;    -- last one has no delay
end process;                                -- which means 'forever'

end architecture;
```

This testbench description uses VHDL syntax similar to a circuit description, but it does *not* represent an actual circuit for implementation. Instead, it describes a desired sequence of software-controlled behavior to dictate input values presented to an abstract circuit representation that is to be tested through simulation. There are more sophisticated features that can be used for testbench files (e.g., reading input signal settings from a file, using for loops and signal vectors to more compactly step through the binary counting sequence). This initial testbench experience will use a rather simple approach that is reasonable for a small example but would not be suitable for a large number of input signals.

You will replace the '?' characters with the appropriate signal names, based on your function.

There will also be one more line to add/modify in this file during the laboratory activity.

Part 1: Implementation of Canonical Sum-of-Products Function in Hardware

- On your USB drive, create a folder *<USB drive letter>*:\ELEC271\LAB1.
- Place your template file `lab1.vhd` in the above folder.
- Start Altera Quartus II. Create a new project called `lab1` in *<USB drive letter>*:\ELEC271\LAB1.
- In the New Project Wizard dialog box, when you reach the Add Files step, use the "…" button to select *only* your `lab1.vhd` file. Be certain to click on the Add button to make it appear in the list.
- In the device selection step, choose *Cyclone III*, the *FBGA* package, *484* for the pin count, *6* for the speed grade, and then click on *EP3C16F484C6* to highlight that choice in blue. Click *Next*.
- In the EDA Tool Settings, select *ModelSim-Altera* for the tool and *VHDL* for the format.
  Do *not* select the option for automatic gate-level simulation, as this feature will not be used.
- Click *Next*. Then, for the Summary, click *Finish* to close the New Project Wizard.
- From the main menubar, select *Assignments → Device…* and then *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 LVTTL.*
- Select *OK* to close the Device/Pin Options dialog box and *OK* again to close the Device box.
- *On your worksheet*, write the *compact Σ form* with the minterms of the canonical sum-of-products representation for the logic function given to you for this laboratory activity.
- *On your worksheet*, write the *full expansion* of the canonical sum-of-products representation, being meticulous so that you correctly reflect each specified minterm. *Double-check your work*.
- *On the back of your worksheet*, write the *full truth table* for the logic function for later reference.
- From the main menubar, select *File → Open...* and select your `lab1.vhd` file for editing.
- Modify the incomplete VHDL specification of the logic function using the full function expansion. Ensure that the AND terms properly reflect the given minterms for a correct logic specification.
- From the main menubar, select *File → Save* to save the modified file. *Keep the file open.*
- Select *Processing → Start Compilation* from the main menubar in order to synthesize the circuit.
- After synthesis, select *Assignments → Pin Planner*. Using the Terasic DE0 User Manual, identify the SW2/1/0 pins. Assign these pins to the first (SW2), second (SW1), and third (SW0) inputs for your logic function. Identify the pin for LEDG0 and assign this pin to the function output. Close the Pin Planner. Select *Processing → Start Compilation* **again** to resynthesize with pin assignments.
- Plug the USB cable into the DE0 board. Turn on the power. Select *Tools → Programmer.* Ensure "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip. (If *Start* is greyed out after proper USB-Blaster selection, close Quartus, restart, reopen project.)
- Use your truth table to step through all eight input valuations for the function, setting SW2, SW1, and SW0 for each minterm. Verify that the output is 1 or 0 correctly for each valuation. If there is an error, inspect your code, make corrections, and resynthesize for hardware testing again.
- In the Quartus II menubar, select *Tools → Netlist Viewers → **RTL** Viewer*. Study the circuit schematic shown on the screen. This representation is the initial transformation by Quartus II of your text-based VHDL description. Despite possible differences from the strict canonical two-level sum-of-products representation, *convince yourself that it is indeed your specified function*.
- After studying the initial netlist view, close the schematic window.
- In the menubar, select *Tools → Netlist Viewers → Technology Map Viewer (Post-**Mapping**).* This view reflects simplification performed by Quartus II. Double-click the blue LOGIC_CELL_COMB box to view the simplified combinational logic. Study the simplified logic to understand it.
- *On your worksheet*, write the Boolean expression for the *simplified* logic function from Quartus II.

See course Webpage for additional step to properly establish Quartus/ModelSim tool linkage.

Part 2: Initial Testbench Simulation using the ModelSim-Altera Software Tool

- The Lab 0 activity used waveform input specification for computer simulation of logic circuitry. The ability to draw waveforms on the screen was provided by front-end Quartus software. To perform the simulation, however, the Quartus software invokes back-end software from another vendor. Then, when the simulation is complete, the Quartus front-end software generates output waveforms for on-screen display. The back-end simulation software is called ModelSim and it is from Mentor Graphics. ModelSim is a widely used simulator in industry, hence it is valuable to acquire some expertise in using it, just as with the Quartus II software for FPGA chips. Altera (now Intel) has a business arrangement with Mentor Graphics for a version of ModelSim that is tailored for use with Quartus II and Altera/Intel FPGA chips, hence the tool is called "ModelSim-Altera."
- In this exercise, you will not use graphical waveform input specification. Instead, you will use an approach that is more typical of real-world engineering in industry: *testbench simulation*. This approach involves preparing a special description – in the same hardware design language as the circuit specification – that provide inputs to an instance of the circuit to model its output behavior. Sophisticated testbenches *automate simulation and detection/reporting of errors relative to the desired output behavior.* Good testbenches also provide *ease of repeatability* as a system under development is modified/enhanced. Ultimately, testbenches enable correct *hardware* operation.
- In the Quartus II menubar, select *File → New...* to edit your `tb_lab1.vhd` file. The Quartus text editor is used here for convenience; this file will *not* be synthesized by Quartus for an FPGA.
- In the testbench VHDL file, replace all '?' characters with the specific names of the three inputs. List the input signals in alphabetic order from left to right to follow the convention in this course.
- Select *File → Save* to save the changes to `tb_lab1.vhd`. ***Keep this file open in the editor***.
- Using Microsoft Windows Explorer, navigate to your <*USB drive letter*>`:\ELEC271\LAB1` folder.
- In the Quartus II menubar, select *Tools → Run Simulation Tool →* ***RTL*** *Simulation*.
- After ModelSim opens, the lower *Transcript* part of the ModelSim window will show a `ModelSim>` prompt. Maximize the ModelSim Window for viewing convenience. Click in the *Transcript* window. Enter the following commands line by line, pressing the Enter key after *each* one.

Note the forward slash characters.

```
vcom  -93  -work work  {<USB drive letter>:/ELEC271/LAB1/tb_lab1.vhd}

vsim  tb_lab1

add  wave  *

run  200ns
```

There is a shortcut for the first command. Use the scrollbar on the right of the Transcript window to go backwards and find an almost identical `vcom` command for `lab1.vhd`. Use the mouse to highlight this command, press Ctrl-C to copy it, scroll forwards to the prompt at the bottom of the Transcript Window, click at the prompt location, then press Ctrl-V to paste the copied text. Click just before the `lab1.vhd` text and type `tb_` in front of it. Press Enter to accept the command.

- After the `run` command, ModelSim will show a waveform display. Click in the waveform window. Adjust the display by using the '–' key to zoom out and see more of the waveforms and by using the '+' key to zoom back in. When zoomed in, use the left and right arrow keys to move towards the beginning or towards the end of simulation time. Zoom to show all eight input valuations. Compare the simulated output for each valuation against with the expected value in the truth table on the back of your worksheet. Note how the automated simulation has the same effect of you manually setting the hardware input switches.

- *Keep ModelSim open*. Do not close any windows.
- When you are repeatedly testing your system in ModelSim, you don't need Quartus resynthesis. In the ModelSim *Transcript* window, just type the commands below (with Enter after each).

```
vcom  -93  -work work  {<USB drive letter>:/ELEC271/LAB1/lab1.vhd}
restart
```

  After `restart`, a dialog box will appear to ask which items to 'keep' from the previous run. Leave all boxes checked and select *OK*. The *Transcript* window should show that `lab1` is reloaded.
- Type `run 200ns` to resimulate and adjust the resulting new waveform display as necessary. Thus, you can easily and reasonably quickly identify/correct issues in your system or testbench.

Part 3: Testbench Simulation and Synthesis of Canonical Product of Sums

- *On your worksheet*, use either the truth table on the back or the compact ∑ sum-of-products form on the front to write the *compact* Π *form* for the maxterms in the product-of-sums representation of the function. Then, carefully write the *full expansion* for the product-of-sums representation and *check your work*. Recall that for the product-of-sums representation, an input variable with a value of 1 is complemented in the maxterm sum (the opposite of what is done for minterm products).
- Return to Quartus II and the editor window for `lab1.vhd`. Presently in the file, there is the `entity` definition and the `architecture` definition for the sum-of-products representation. Now, *below* the `end architecture` line, introduce a *new* architecture definition while leaving the first definition unchanged. Copy/paste the existing definition if you wish, but edit the new one.
- Set the name of the new architecture to `pos` (it must be distinct from the previous `sop` name).
- In a similarly structured and commented manner as for the minterms in the sum-of-products representation, write VHDL code for product-of-sums representation with one line per *maxterm*, using your worksheet as your guide. *Double-check your typing to be certain there are no errors.*
- When finished, your `lab1.vhd` file should have the `entity` definition, the initial `architecture` definition named `sop`, and a new `architecture` definition named `pos` at the end of the file.
- Select *File* → *Save* to save the change. *Keep the file open*.
- Do *not* resynthesize the implementation in Quartus II. You are currently engaged in *simulation*.
- With two different architectures available for the system under test, *it is necessary to specify which architecture to use*, otherwise a default selection will be made (which may not be the desired one).
- In Quartus II, switch to the editor window for `tb_lab1.vhd`. After the `end component` line, leave a blank line, type the following text, and leave another blank line for readability.

```
for all:  lab1  use entity work.lab1(pos);
```

  This statement is informing the software tool that all instances of the `lab1` entity in the testbench file must use the `pos` architecture, which is the desired one for this particular part of the exercise.
- Select *File* → *Save* to save the change. *Keep the file open*.
- Return to ModelSim and click in the *Transcript* window. *Use the up-arrow key* and text editing to execute the two following commands (press Enter after each one).

```
vcom -93 -work work {<USB drive letter>:/ELEC271/LAB1/lab1.vhd}
vcom -93 -work work {<USB drive letter>:/ELEC271/LAB1/tb_lab1.vhd}
```

  Both files must be recompiled within ModelSim because both files have been modified.
- Next, type the `restart` command to repeat the simulation for the modified VHDL files.

After `restart`, a dialog box will appear to ask which items to 'keep' from the previous run. Leave all boxes checked and select *OK*. The *Transcript* window should show that `lab1` is reloaded.

- Before examining the waveform output, first identify the *Sim* window that is directly above the *Transcript* window. In the *Instance* column of the *Sim* window, locate the `tb_lab1` entity name. Click on the '+' symbol at the left of the name to expand the hierarchy encompassed by it. You should see the `lab1_under_test` instance name, and to the right of that name, in the *Design Unit* column, you should see `lab1(pos)` reflecting the `for all` statement you typed above. If necessary, click-and-drag on the vertical line to the right of "Design Unit" to widen the column.
- Having confirmed that the product-of-sums form of the function is being simulated, examine the waveform/transcript displays to be certain that the simulated function outputs match the expected outputs for each of the input valuations. The automated verification will indicate any errors.
- The final step for this part is to return to Quartus II and proceed with re-synthesis of the `lab1` entity for FPGA chip implementation, having modified the VHDL file earlier in this part. As before, go to the main menubar and select *Processing → Start Compilation*.
- Note that the VHDL code for *synthesis* has not indicated which of the two architectures to use.
- Which architecture does Quartus II select for an instance of an entity? The official default choice for the VHDL language (and hence *any* tool from *any* vendor that supports VHDL) is to use the *last architecture encountered*, i.e., the last one found in an entity file with multiple architectures.
- Therefore, the *product-of-sums* representation is used here because it is the last architecture that is listed in the `lab1.vhd` file. (You can use `for all` in synthesis but another file will be needed.)
- After synthesis, select *Tools → Netlist Viewers →* ***RTL*** *Viewer* and confirm product-of-sums form.
- Then, select *Tools → Netlist Viewers → Technology Map Viewer (Post-****Mapping****)*.
- *On your worksheet*, write the simplified logic expression generated by Quartus II.
- *Keep both Quartus II and ModelSim open for later use.*
- You have now experienced simple testbench simulation and you have seen more aspects of the VHDL language that make it a versatile means for circuit implementation and simulation.

Part 4: Algebraic Simplification of Boolean Logic Expressions

- The synthesis steps performed in previous parts by Quartus II involved the software transforming a given logic description to a simpler but equivalent form so as to reduce resource consumption when implemented in hardware. Such simplification is much more important for large systems.
- In this part, you will use the space provided *on the worksheet* to perform *step-by-step algebraic simplification* to definitively confirm the correctness of the final expression from the software tool, and to practice the formal application of the theory of Boolean algebra that is part of this course.
- Examine the full expansion that you have written for the *sum-of-products* representation on your worksheet and identify an appropriate axiom/theorem/property to apply from Boolean algebra and write the resulting expression in the first space under the "simplification" section of the worksheet. *To the right of the expression, write the name or label of the relevant axiom/theorem/property.*
- Continue to manipulate the expressions and document your algebraic steps on your worksheet, line by line, until you obtain the same simplified expression as you extracted from Quartus II.
- Repeat the algebraic simplification procedure for the *product-of-sums* representation on your worksheet until you obtain the same simplified expression as you extracted from Quartus II.
- As you pursue this part, seek to develop an appreciation of the principle of duality as you consider the algebraic steps for each of the two representations, and refine your grasp of the most relevant and effective algebraic manipulations for logic simplification.

Part 5: Alternative VHDL Specifications for 2-to-1 Multiplexer Circuits

- In Quartus II, select *File* → *Close* ***Project*** to finish with the work of the previous parts.
- Select *File* → *New Project Wizard…* to prepare a new project for this part in the same folder as before. Use a different project name: `mux21`. (The folder and project names can be different.)
- Do not click Next yet. Before proceeding further in new project creation, click on the button labelled *Use Existing Project Settings…*
- In the new dialog box, put a checkmark in the box for "Copy settings from specified project …"
- Click on the circle beside "Use settings from last opened project," which should show the name of the project used in the previous parts. Then click on *OK*.
- If you see warnings about changing assignments due to selecting a new device, click on *OK*.
- You will then see a warning about the current directory already containing a Quartus II project. Provided that appropriate care is taken in handling of any VHDL files that are shared between multiple projects in the same directory, it can be convenient to allow the use of a single directory. In this case, no files will be shared between projects, so there is no concern. Thus, in response to the question "Do you want to select a *different* directory?" you can answer *NO* to proceed.
- For the Add Files step, you have no VHDL code yet, so you can click on *Next*.
- For the Device step, the correct Cyclone III chip should already be highlighted in light gray.
- *It is still necessary, however, to explicitly change the settings for unused pins and the voltage.* Thus, you will make the appropriate changes after the New Project Wizard completes.
- Click on *Next* twice to move to the EDA Tool Settings and Summary steps, then click on *Finish*.
- From the main menubar, select *Assignments* → *Device…* and then *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 LVTTL.*
- Select *OK* to close the Device/Pin Options dialog box and *OK* again to close the Device box.
- From the menubar, select *File*→ *New…* In the New dialog box, select *VHDL File* and click OK.
- Type in the text for a *complete* VHDL specification for an entity called `mux21` with inputs *s*, *x*, *y* and outputs *f*, *g*. You can choose a useful name for the architecture (`logic`, `behavior`, etc.).
- For the architecture body, provide *two* signal assignment statements. The first statement should be for *f*(*s,x,y*) and should specify the output using `and`, `or`, `not` keywords for gate-level logic. The second statement should be for *g*(*s,x,y*) and should use VHDL `when…else…` syntax for a higher-level specification of multiplexer functionality (see the counter VHDL code in Lab 0).
- Make the specifications for *f*(*s,x,y*) and *g*(*s,x,y*) consistent with each other (i.e., equivalent). In other words, ensure the following: *s*=1 selects input *x*, *s*=0 selects input *y*.
- 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.
- Using the Terasic DE0 User Manual, make the following pin assignments:
  SW2 for *s*, SW1 for *x*, SW0 for *y*, LEDG1 for *f*, LEDG0 for *g*.
- Select *Processing* → *Start Compilation* **again** to resynthesize with the pin assignments.
- Ensure that the DE0 is on and the USB cable is connected. Select *Tools* → *Programmer.* Ensure "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip.
- Set SW2 (input *s*) to 0. Cycle through all four combinations for SW1 (*x*) and SW0 (*y*).
  Ensure that both outputs LEDG1 (*f*) and LEDG0 (*g*) show the value of SW0 (*y*) for the four cases.
- Set SW2 (input *s*) to 1. Cycle through all four combinations for SW1 (*x*) and SW0 (*y*).
  Ensure that both outputs LEDG1 (*f*) and LEDG0 (*g*) show the value of SW1 (*x*) for the four cases.

- In Quartus II, select *Tools → Netlist Viewers →* ***RTL*** *Viewer* to inspect your logic specification.
- *On the back of your worksheet*, draw the schematic in the space provided.
- Select *Tools → Netlist Viewers → Technology Map Viewer (Post-****Mapping****)* to see the final outcome from processing by Quartus II.
- *On the back of your worksheet*, draw the final schematic in the space provided.
- Can you recognize what the Quartus II software has done to produce the final technology-mapped circuit from the initial representation derived from your original VHDL description? Discuss/debate this issue with your partner(s) to ensure an appreciation of the capability of such design software.

ELEC 271 Lab 1 Worksheet Name: _____

Given canonical *sum-of-products* function: *f*(, ,) =

Full expansion: *f*(, ,) =

Quartus II post-mapping simplified function: *f*(, ,) =

Equivalent canonical *product-of-sums* function: *f*(, ,) =

Full expansion: *f*(, ,) =

Quartus II post-mapping simplified function: *f*(, ,) =

Step-by-step algebraic simplification of canonical *sum of products* to match Quartus II function:

f(, ,) =

=

=

Step-by-step algebraic simplification of canonical *product of sums* to match Quartus II function:

f(, ,) =

=

=

Truth table for given function on previous page:

RTL viewer schematic for mux outputs

Post-mapping viewer schematic for mux outputs