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

Lab 2: 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 experience in preparing and performing 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 activity involves the use of the worksheet included in the description. Each student must have a separate paper copy of the worksheet ready at the beginning of the laboratory session. Each student must complete it (with reasonable neatness) during the session, *otherwise no credit will be granted.*

Preparation BEFORE Your Scheduled Laboratory Session

- *You will individually prepare three 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.

  At the start of a laboratory session, each student must have a *printed* worksheet ready to use, otherwise the initial attendance credit will not be granted.

- In a file entitled `lab2.vhd`, prepare the following template for later modification.

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

entity lab2 is
    port (?, ?, ? : in  std_logic;
          f       : out std_logic);
end entity;

architecture sop of lab2 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 a file entitled `signal_values.txt`, prepare the following template for later modification.

```
000
?
001
?
010
?
011
?
100
?
101
?
110
?
111
?
```

Type the content *exactly* as shown above with *no blank lines* between the given lines of text.

The first row reflects the binary input valuation 000 for the input variables, and the '?' in the second represents the corresponding output value for the logic function that you will be given.

The remainder of the file has a similar interpretation for the rows, i.e., input valuation followed by the corresponding output value for that input valuation.

Therefore, the '?' characters in this file are placeholders for 0 or 1 output logic values. Based on the function that you will be given, you will replace each '?' with either 0 or 1, as appropriate.

The simulation that you will perform in Part 2 will involve an automated process that will use the contents of this file to drive a circuit under test and to verify that it produces correct output values.

As will be stated in Part 2, the software tool that will be used for simulation is ModelSim, which is a separate product by a different company, Mentor Graphics. Altera (now Intel) has partnered with Mentor Graphics to develop a version of this tool for distribution and use with Quartus II.

- In a file entitled `tb_lab2.vhd`, prepare the following testbench driver for logic simulation.

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

library std;
use std.textio.all;

entity tb_lab2 is
  -- no external ports/pins for a simulation testbench entity
end entity;

architecture testbench of tb_lab2 is

component lab2
  port (?, ?, ? : in  std_logic;
        f       : out std_logic);
end component;

-- define the input/output signals for this testbench,
-- all of which have the same names as the ports of the entity under test
signal ?, ?, ?, f : std_logic;

begin

lab2_under_test : lab2   -- instantiate the system under test
   port map (            -- and map entity ports to testbench signals
             ? => ?,
             ? => ?,
             ? => ?,
             f => f
            );

-- process to read vectors from a file and drive the system under test
--  using assertions to verify the correct output for each vector

test_driver_with_assertions: process

variable line_in, line_out : line;
variable i                 : integer;
file     input_file        : text is "signal_values.txt";
variable in_vector         : std_logic_vector(2 downto 0);
variable out_vector        : std_logic_vector(0 downto 0);

begin

  -- loop for the number of input/output vector pairs in file
  for i in 0 to 7 loop

    -- retrieve input vector and assign to variable
    readline (input_file, line_in);
    read (line_in, in_vector);
```

… continued on the next page …

- The continuation of `tb_lab2.vhd` is shown below. The two pages of code should be in one file.

```
    -- prepare portion of output line
    write (line_out, string'("input vector: "));
    write (line_out, in_vector);

    -- retrieve output vector and assign to variable
    readline (input_file, line_in);
    read (line_in, out_vector);

    -- prepare remainder of output line and send to output
    write (line_out, string'("  output vector: "));
    write (line_out, out_vector);
    writeline (output, line_out);

    -- assign individual signals from input vector and delay
    ? <= in_vector(2);
    ? <= in_vector(1);
    ? <= in_vector(0);
    wait for 10 ns;

    -- check generated output signal against testbench output value
    assert (f = out_vector(0))
      report "output did not match"
      severity error;

    -- additional delay before handling the next input vector
    -- allows for centering of error symbol on waveform display
    wait for 10 ns;

  end loop;

  -- after loop, force inputs to zero for remainder of simulation time
  ? <= '0';
  ? <= '0';
  ? <= '0';
  wait;

end process;

end architecture;
```

When you prepare the content of this file, double-check your text entry to ensure that you have not left out any lines, particularly across the split of the content between the two pages. 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. The description uses VHDL language features for: accessing information in a text file, testing simulated output against the expected output, and producing human-readable messages that allow a designer to easily interpret the outcome of the simulation.

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

In Part 3, you will add one line of code to this file after you use it initially in Part 2.

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

- On your network storage, create a folder `Z:\My Files\ELEC271\LAB2`.
- Place your template files `lab2.vhd`, `signal_values.txt`, `tb_lab2.vhd` in the above folder.
- Start Altera Quartus II. Create a new project called `lab2` in `Z:\My Files\ELEC271\LAB2`.
- In the New Project Wizard dialog box, when you reach the Add Files step, use the "…" button to select *only* your `lab2.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 `lab2.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.
- 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.
- *Leave the DE0 powered on with the chip configured so that you can later demonstrate your work to earn part of the completion credit for this exercise.*
- 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.

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

- The Lab 1 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 – in order to provide inputs to an instance of the circuit and to verify the outputs. The intent is to *automate the simulation process for reliable error detection and ease of repeatability* whenever changes and enhancements are made to a system under development, so that there is substantially higher likelihood of correct *hardware* operation for a complex system.
- In the Quartus II menubar, select *File → Open...* to edit your `tb_lab2.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. Note the following order for the `in_vector` assignments: bit 2 should be the *first* input variable, bit 1 should be the *second* variable, and bit 0 should be the *third* variable.
- Select *File → Save* to save the changes to `tb_lab2.vhd`. *Keep this file open in the editor*.
- Select *File → Open...* to edit your `signal_values.txt` file (again, this file is not for synthesis).
- Using the truth table from your worksheet, carefully change '?' characters to 0 or 1 and *check your changes*. This file provides input values and expected output values for simulation and verification.
- Select *File → Save* to save the changes to `signal_values.txt`. Close this file.
- Using Microsoft Windows Explorer, navigate to your `Z:\My Files\ELEC271\LAB2` folder.
- Right-mouse-click on the icon for `signal_values.txt` and select *Copy*.
- Find the `simulation` folder icon in the Windows Explorer view and double-click on it.
- You should now see the `modelsim` folder icon. Double-click on it to enter this folder.
- With the Windows Explorer view now showing the `simulation\modelsim` folder contents, right-mouse-click on a clear area of the `modelsim` folder view, then select *Paste*. Ensure that a copy of your `signal_values.txt` file has indeed been pasted in the `modelsim` 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  {Z:/My Files/ELEC271/LAB2/tb_lab2.vhd}

vsim  -msgmode both  tb_lab2

add  wave  *

run  200ns
```

There is a shortcut for the `vcom` command. Use the right-hand scrollbar of the Transcript window to go backwards and find an almost identical `vcom` command for `lab2.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 `lab2.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 so that eight input valuations are visible up to the point that all inputs are forced to zero. 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.
- Return to Quartus II, and select the editor window for the `lab2.vhd` file that you kept open. In the logic expression for the function, *introduce an intentional error* by typing a `not` keyword in front of the first input variable that does not have one already. Select *File* → *Save*. *Keep the file open*.
- Do *not* resynthesize the implementation in Quartus II. You are currently engaged in *simulation*.
- Return to the ModelSim *Transcript* window. Type the commands below (with Enter after each).

```
vcom  -93  -work work  {Z:/My Files/ELEC271/LAB2/lab2.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 `lab2` is reloaded.

- Type `run 200ns` and adjust the resulting new waveform display to view all eight valuations. Note how the intentional error that was introduced in the logic description has caused red arrow(s) to appear above the interval(s) where the generated output does not match the expected output.
- Furthermore, examine the *Transcript* window and use the scrollbar on the right to view the text messages generated from the `tb_lab2.vhd` testbench file. The `write`/`writeline` commands show the progress as the input/output vectors are collected from the `signal_values.txt` file, and the `report` command indicates when a generated output fails to match an expected output.
- Return to Quartus II and the editor window for `lab2.vhd`. Restore the logic description to its original correct form by deleting the erroneous `not` keyword you introduced previously.
- Select *File* → *Save* to save the change. *Keep the file open*.
- *Keep ModelSim open*. Do not close any windows.

Credit Checkpoint

Having completed hardware implementation and simulation of the sum-of-products implementation, show the following items to a teaching assistant for credit.

☐ completed worksheet page up to this point in the exercise for each student

☐ demonstration of canonical sum-of-products function in FPGA chip using SW2, SW1, SW0

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 `lab2.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 `lab2.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*.
- 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_lab2.vhd`. After the `end component` line, leave a blank line, type the following text, and leave another blank line for readability.

```
for all:  lab2  use entity work.lab2(pos);
```

  This statement is informing the software tool that all instances of the `lab2` 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* to scroll through the command history to find and repeat the two following commands (press Enter after each one).

```
vcom -93 -work work {Z:/My Files/ELEC271/LAB2/lab2.vhd}
vcom -93 -work work {Z:/My Files/ELEC271/LAB2/tb_lab2.vhd}
```

- Next, type the `restart` command to repeat the simulation for the modified VHDL files.
- 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_lab2` entity name. Click on the '+' symbol at the left of the name to expand the hierarchy encompassed by it. You should see the `lab2_under_test` instance name, and to the right of that name, in the *Design Unit* column, you should see `lab2(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 `lab2` 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 `lab2.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 for `pos`.
- *Keep both Quartus II and ModelSim open for later use.*
- You have now experienced *industrial-style* 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.

Credit Checkpoint

Show the following items to a teaching assistant for credit.

☐ completed worksheet up to this point in the exercise for each student

☐ `lab2.vhd` file with two different architectures; `tb_lab2.vhd` selecting `pos` version

☐ ModelSim indication of `pos` version in *Sim* window, and error-free waveform display

After obtaining credit, *close ModelSim* but keep Quartus II open.

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 1).
- 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*=0 selects input *x*, *s*=1 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 SW1 (*x*) 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 SW0 (*y*) 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.
- Do you recognize what the Quartus II software has done to produce the final post-mapping circuit from the initial RTL representation derived from your original VHDL description? Discuss/debate this issue with your partner(s) to enhance your appreciation of the capabilities of this design software.

Credit Checkpoint

☐ VHDL file with two multiplexer alternatives; working hardware implementation

☐ completed worksheet for each student

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(, ,) =

=

=

Name: ____

Truth table for given function considered on previous page of this worksheet:

RTL viewer schematic for mux outputs

Post-mapping viewer schematic for mux outputs