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

Lab 2: Logic Optimization with Karnaugh Maps and Experimentation with Sequential Circuitry

Objectives

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

- optimize logic functions with Karnaugh maps for sum-of-products/product-of-sums forms, and
- experiment with sequential circuitry in hardware and simulation.

This activity involves the use of the worksheet included in the description. **Each student** must have a paper copy of the worksheet ready at the beginning of the laboratory session, and each student must complete it (with reasonable neatness) during the session.

VHDL code should be prepared by each student in advance of the laboratory session, for individual learning benefits and also to be ready for any contingency (e.g., lab partner being absent).

Preparation BEFORE Your Scheduled Laboratory Session

Similar to Lab 1, prepare template `lab2.vhd` and `tb_lab2.vhd` files with the following differences.

- For the `lab2` entity, there will be four input ports (unknown names until you are given a function specification), and two output ports `fsp` and `fps`.

- For the `tb_lab2` entity, there will similarly be four signals (unknown names) for connection to the inputs of an instantiated `lab2` entity, and two signals `fsp` and `fps` for connection to the outputs of a `lab2` entity.
- An additional signal `v` is also required, defined as `std_logic_vector(3 downto 0)`.
- The `component` declaration should match the `lab2` entity.
- The `lab2` entity should be instantiated with port mappings that involve the four inputs and two outputs.
- For the `test_driver` process, there will be more test cases (due to having one more input for the given function than in Lab 1), but the approach will be slightly different. Instead of assigning a single-bit value to each signal before a `wait`, a multibit vector will be assigned a multibit value corresponding to the logic values in the four columns of the truth table for a given row. Each of the four individual signals will then be associated with a specific element within the bit vector. *Inside* the `test_driver` process, there will lines of the form

  ```
  v <= "1011";  wait for 20 ns;
  ```

  and *outside* the process, there will four lines of the form

  ```
  ? <= v(2);
  ```

  where each statement will use a different index 3 down to 0. The use of multibit vectors surrounded by double quotes will be more convenient with four input signals and double the number of test cases than in Lab 1.

The above files will be used to apply the logic optimization technique based on Karnaugh maps, as discussed in class. The technique is applicable in sum-of-products representation and also in product-of-sums representation (hence the two outputs described above).

Finally, the counter description in VHDL from Lab 0 will be required. If you have pursued the Lab 0 activity, then you should have the complete project prepared in an appropriate folder, including the pin assignments for the hardware. If not, then follow the instructions in Lab 0 to prepare the counter project with the pin assignments.

The counter example will be used for experimentation on the DE0 hardware and further exploration of more sophisticated simulation techniques using ModelSim and VHDL processes to drive simulation, this time for a sequential circuit that involves clock and reset signals. Although the details of circuitry that can hold information (i.e., maintain a state) have not yet been discussed, the VHDL expression of this functionality was first seen in Lab 0 and the practical experimentation that will be pursued as part of Lab 2 activity provides a basis for the more detailed discussion that will be pursued on this topic.

Part 1: Logic Optimization with Karnaugh Maps and Hardware Implementation

- On your USB drive, create a folder <*USB drive letter*>`:\ELEC271\LAB2`.
- Place files `lab2.vhd` and `tb_lab2.vhd` in the above folder.
- Start Altera Quartus II, and create a new project labelled `lab2` in the folder you created above.
- At the appropriate screen of the New Project Wizard, include only `lab2.vhd` into the project.
- As done in Lab 0 and Lab 1, use the New Project Wizard to complete the setup for the correct Cyclone III chip on the Altera DE0 board. *Also select ModelSim-Altera and VHDL for simulation*.
- After completing the steps in the New Project Wizard, use *Assignments* $\rightarrow$ *Device…* to change settings for *unused pins* and *voltage (refer to previous laboratory descriptions)*.
- *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 exercise.
- *On your worksheet*, generate a Karnaugh map for the minterms with proper row/column labels. Derive the optimized sum-of-products expression and write it in the space provided.
- *On your worksheet*, write the *compact Π form* with the maxterms of the canonical product-of-sums representation for the logic function given to you for this laboratory activity.
- *On your worksheet*, generate a Karnaugh map for the maxterms with proper row/column labels. Derive the optimized product-of-sums expression and write it in the space provided.
- In the Quartus II main menubar, select *File* $\rightarrow$ *Open...* and select your `lab2.vhd` file for editing.
- Complete the entity section for the given input variable names. Then, in the architecture section, type your optimized sum-of-products/product-of-sums expressions for outputs `fsp` and `fps`. *Use parentheses in your logic expressions to ensure your desired order of operations.*
- From the main menubar, select *File* $\rightarrow$ *Save* to save the modified file.
- Select *Processing* $\rightarrow$ *Start Compilation* from the main menubar in order to synthesize the circuit.
- After synthesis, select *Assignments* $\rightarrow$ *Pin Planner*. Using the Terasic DE0 User Manual, identify the SW3/2/1/0 pins. Assign these pins to the first (SW3), second (SW2), third (SW1), and fourth (SW0) inputs for your logic function. Identify the pin for LEDG1 and assign this pin to output `fsp`. Identify the pin for LEDG0 and assign this pin to output `fps`. Close the Pin Planner.
- Select *Processing* $\rightarrow$ *Start Compilation* **again** to resynthesize with the specified pin assignments.
- With the USB cable connected and the DE0 board on, select *Tools* $\rightarrow$ *Programmer.* Ensure that "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip.
- From the worksheet, you know which input valuations are minterms where the function output is 1, and you know the input valuations that are maxterms where the function output is 0. Step through all sixteen input valuations, setting SW3, SW2, SW1, and SW0 appropriately. Verify that the `fsp` and `fps` outputs match *each other* and *the expected result*. If not, check your work and try 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 Quartus II, select *Tools* $\rightarrow$ *Netlist Viewers* $\rightarrow$ ***RTL*** *Viewer* to examine the *initial* schematic.
- Then, select *Tools* $\rightarrow$ *Netlist Viewers* $\rightarrow$ *Technology Map Viewer (Post-**Mapping**)* to examine the result of processing by the software for comparison with your optimized expressions. Just click on the boxes labelled LOGIC_CELL_COMB. Ignore the inputs and outputs; they are pins, and they will have buffers as symbols. Our interest is the logic function implementation, not the pins.
- Finally, select *Tools* $\rightarrow$ *Chip Planner* to view the placement of the logic functions by Quartus II. How many lookup tables are used? Is this what you would expect for the four-input functions?

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

- 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 file, replace the relevant '?' characters with the specific names of the four inputs. For the `v` assignments: bit 3 should be the *first* input variable, bit 2 should be the *second* variable, bit 1 should be the *third* variable, and bit 0 should be the *fourth* variable.
- From the main menubar, select *File → Save* to save the modified testbench file. (The testbench should not be added to the Quartus II project; remove it from the list of files if it is added.)
- Select *File → Save* to save the changes to `tb_lab2.vhd`.
- In the Quartus II menubar, select *Tools → Run Simulation Tool →* ***RTL*** *Simulation*.
- *Refer back to the instructions in Lab 1 for the four commands to type in the Transcript window*.
- For this exercise, use the `vcom` and `vsim` commands suitably adapted for Lab **2**.
- But for the remaining two commands, use the the Use similar commands as done in Lab 1, but with the changes indicated below.

  ```
  add wave -divider "Inputs"  ? ? ? ?  v  -divider "Outputs"  fsp fps

  run  400ns
  ```

  Replace each $\boxed{?}$ with one of the input signals in the correct order.
  Place the `v` signal *after* the four inputs.
  The 'divider' options introduce labelled lines in the display for easier reading of the results.
  The 400ns specification is needed because there are twice as many input valuations as in Lab 1.
- Also refer to the instructions in Lab 1 for how to manipulate the waveform window to view the simulation results. The waveforms for the two optimized output signals should match each other and they should reflect the expected output values for the original function specification.
- Because you should have already verified correct circuit behavior with hardware implementation, simulation errors are likely to be caused by a problem in the testbench file – fix it and try again.
- In this simulation, the four individual inputs are shown in separate waveforms. Also shown is the `v` signal that is a four-bit vector from which the individual inputs are derived. After running the simulation, the waveform for the `v` signal is shown with the default 'radix' of binary. Zoom in if necessary to see the four-bit values for the `v` signal.
- For easier identification of each of the valuations corresponding to the minterm subscripts, the displayed radix for a multi-bit waveform can be changed. *Right*-mouse-click on the `v` name on the left side of the waveform display. In the popup menu that appears, select *Radix → Unsigned* which will show unsigned decimal representation for each four-bit value of the `v` signal. Again, adjust the zoom as necessary to better view the waveform output. Use the decimal labelling of the input cases to reconfirm that the optimized function outputs correspond to the specified minterms.
- *Keep ModelSim open* once the results are correct so that waveforms can be shown for credit.

Credit Checkpoint

☐ completed paper worksheet for each student

☐ demonstration of both optimized functions in FPGA chip using SW3, SW2, SW1, SW0

☐ `lab2.vhd` and `tb_lab2.vhd` files

☐ correct ModelSim waveform display

Part 3: Hardware Testing of Counter Circuit from Lab 0 with Pushbutton Clock

- Close ModelSim from the earlier part (it must be closed to properly reopen later).
- The Lab 0 counter circuit was required as preparation for Lab 2. If the instructions from Lab 0 have been followed properly, then the the project for this circuit is in a separate folder, i.e., <*USB drive letter*>`:\ELEC271\LAB1\COUNTER` (the instructor should have been more careful in checking the description, because it should have been `LAB0`, but you can leave the name as it is).
- In Quartus II, open your project for the Lab 0 counter circuit.
- For the pin assignments, the Lab 0 description indicated that the clk input should be G21. For Lab 2, however, initial hardware testing will instead use a pushbutton clock source (i.e., you, the human, will be the clock source). Button 1 on the DE0 board is pin G3.
- Open the Pin Planner and modify the pin assignment for the clk input to use button 1.
- Close the Pin Planner and **resynthesize** the implementation with the revised pin assignment.
- With the USB cable connected and the DE0 board on, select *Tools → Programmer.* Ensure that "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip.
- Set SW0 to 0 so that the lowest 10 bits of the counter output are shown on the LEDs.
- Press *and hold* button **1**. Then, while watching the LEDs closely, release button 1. Repeat the use of button 1 a few more times, and refer back to the Lab 0 description where explanatory comments were provided on the right-hand side of the VHDL code for the counter.
- Holding button 1 down corresponds to the low (logic 0) portion of the clock waveform. Releasing button 1 results in the transition from low (logic 0) to high (logic 1), i.e., a rising edge. Consider this information carefully and relate it to the observed behavior with button 1.
- You can change the setting of SW0 to observe higher bits of the counter output on the LEDs. You should understand the behavior of these higher-bit outputs as you press/release button 1.
- Restore the setting of SW0 to have the lowest bits shown on the LEDs.
- Now press *and hold* button **0** (to the right of button 1). Button 0 is the asynchronous active-low reset input to the system, which has the effect of forcing all of the bits stored in the register portion of the system to zero, hence the LEDs reflect that change. Active-low inputs perform their action when the signal level is low (logic 0), which is the case when button 0 is held down.
- Generate rising edges on button 1 to update the counter state (i.e., the bit values held in the register portion of the system). Experiment again with using the reset input followed by multiple clock edges to become more familiar with the behavior of this type of *sequential logic circuitry*.
- In the Lab 0 description, one of the parts involved using the RTL Viewer to inspect the internal representation derived by the Quartus software tool from the VHDL description. Use the RTL Viewer again to see the schematic representation and relate its parts to the corresponding parts of the VHDL description. Again, the aim is to become more familiar with sequential logic circuity, in this case the inclusion of register/flip-flop elements for holding information in schematic diagrams.

Part 4: Hardware Testing of Counter Circuit from Lab 0 with 50-MHz Clock

- Open the `counter.vhd` file in the Quartus II text editor (i.e., use *File → Open...*).
- Identify the signal assignment statement with `when…else…` syntax for the output.
- Change the `(9 downto 0)` specification for the subset of bits to `(31 downto 22)`. This change is necessary because the 50-MHz clock input to be used is so fast that viewing the lowest bits of the counter output on the LEDs is not useful because the human eye cannot see them changing. Save the modified file.
- Open the Pin Planner. Change the pin assignment for the clk input to G21, which is connected to an external oscillator element on the DE0 board that generates a 50-MHz clock signal.

- Close the Pin Planner and **resynthesize** the implementation with the updated pin assignments.
- With the USB cable connected and the DE0 board on, select *Tools → Programmer.* Ensure that "USB-Blaster" appears beside *Hardware Setup*. Press the *Start* button to program the FPGA chip.
- Try different settings of the SW0 input and observe the behavior of the LEDs.
  Understand the VHDL description and relate it to the observed behavior for changes in SW0.
- Now press *and hold* button **0** (the reset input). Observe the LEDs while you hold button 0 down.
- Now release button 0 and observe the LED behavior.
- What is happening during the time that button 0 is being held down? Think in terms of the input signals to the system, and what the levels of those input signals mean.
  What does releasing of button 0 mean in terms of input signal levels?
  Become familiar with sequential circuit behavior when a high-frequency clock input is involved.

Part 5: ModelSim Testbench Creation for Sequential Circuit Simulation

- Click on *Assignments → Settings...,* then in the dialog box that appears, select *EDA Tool Settings*. For the Simulation setting, choose ModelSim-Altera and make certain that the format is VHDL. Click on OK to close the dialog box. is the formatOpen the counter VHDL file in Quartus II.
- In the Quartus text editor for `counter.vhd`, identify the signal assignment with `when…else…` syntax for the output. Change the `(31 downto 22)` specification for the subset of bits to `(9 downto 0)`. Because simulation is being used for this part, the lowest bits are again selected for viewing because they change most frequently in response to clock edges. Save the modified file.
- **Resynthesize** the implementation to use the revised VHDL *and* to prepare for using ModelSim.
- Click on *File → New...* and select "VHDL file" from the choices available. An empty text view should be visible in Quartus. A testbench file will be created in this window.
- Before typing anything into the text editor, click on *File → Save as...*
- In the dialog box that appears, **uncheck** the box at the bottom that says "add file to project" because this is a testbench file and will *not* be used for synthesis of a chip configuration.
- For the name of the new VHDL file, enter `tb_counter.vhd`. Click on *Save*.
- Use your existing testbench files as a guide to prepare a testbench file that can be used properly with `counter.vhd`. Define the entity and architecture portions of the testbench appropriately. Define the internal signals for connection to the counter inputs/outputs. Instantiate a counter.
- For the `test_driver` process, there will *not* be a set of input valuations from a truth table, as used for testing combinational logic. Instead, the `test_driver` process will have only one purpose, namely to control the setting of the `sw0` input. The body between `process` and `end process` should set `sw0` to `'0'` and then `wait` (forever, because no other change is needed).
- To complete the testbench for a *sequential* circuit such as this counter, *two new processes* are appropriate. One will control the active-low reset signal, and the other will provide a free-running clock signal forever (so that regardless of how long the simulation is run, there is always a clock).
- For the reset signal, create a process called `the_reset` with only two lines between `process` and `end process`. The first line sets `reset_n` to `'0'` then uses a `wait` for 10 ns. The second line then sets `reset_n` to `'1'` followed by a `wait` forever so that the signal remains at logic-1.
- For the clock signal, create a process called `the_clock` with only two lines between `process` and `end process`. The first line sets `clk` to `'1'` then uses a `wait` for 10 ns. The second line then sets `clk` to `'0'` and then *also* uses a `wait` for 10 ns (*not* forever). In simulation, after the second `wait`, the simulation software will repeat from the beginning of the process. The result is a repeating pattern of logic-1 for 10 ns, and logic-0 for 10 ns, which gives a 50-MHz clock.

- Review your `tb_counter.vhd` file and make certain all of the necessary elements are included.
- Save the `tb_counter.vhd` file. Select *Tools → Run Simulation Tool →* ***RTL*** *Simulation*.
- After ModelSim opens, use your previous experience to guide you in giving the four necessary commands for simulating a system. For the waveform display, use the following command to have the relevant signals listed in the desired order: `add wave  clk  reset_n  sw0  q`
  Perform the simulation for 400 ns so that there at least (400 ns) / (20 ns/cycle) = 20 cycles of the 50-MHz clock.clocka reasonab between 100 ns and 200 ns to cover enough successive clock cycles to see multiple output changes.
- Zoom in and out to view the binary values for the output.
- Then, change the 'radix' for the output to unsigned decimal as described earlier in Part 2. Confirm for yourself that the binary numbers previously shown match the unsigned decimal values.
- Finally, change the 'radix' for the output to *hexadecimal*. Because 400 ns or 20 cycles was used for the duration of the simulation, you should see how hexadecimal representation displays values above 9, and also what happens when values exceed 15.

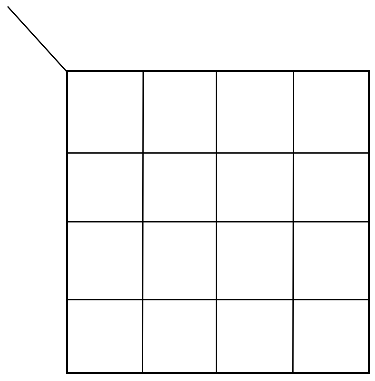
Part 6: Extending the Counter Circuit and Testbench with a Load-Enable Input

- In the Quartus text editor for `counter.vhd`, introduce a new port for the entity called `le` (for load-enable) as an input. In the body of the process within the architecture, add a nested `if` statement that checks if `le` to `'1'` before incrementing the counter value.
- **Resynthesize** the implementation to use the revised VHDL and to check for syntax errors.
- In the Quartus text editor for `tb_counter.vhd`, make the appropriate revisions to reflect the addition of the `le` input (i.e., declaration of the component, definition of the internal signals, and the component instantiation).
- The VHDL processes to model the behavior of the reset and clock inputs will not be modified.
- At the beginning of the `test_driver` process, extend the first line to also set `le` to `'1'`.
  Then, change the `wait` forever to a `wait` for 150 ns.
- Now add a second line to set `le` to `'0'` followed by a `wait` for 100 ns.
- Finally, add a third line to set `le` to `'1'` and `wait` forever (i.e., no further change in `le`).
- Save the `tb_counter.vhd` file.
- Switch to ModelSim (it should have been left open). You have modified the two VHDL files. Recompile them both with the `vcom` command. Then, use `restart` followed by `run` for 400ns.
- Examine the resulting waveforms and understand the behavior and its relation to the VHDL for the counter entity and the input changes in the testbench. *Use this opportunity to truly appreciate the use of the CAD software to implement desired logic functionality and verify correct operation with scripted simulation scenarios, as actual engineers pursue in the real world.* The skills that the laboratory activity of the course seeks to develop will be useful for future courses, for any projects pursued in undergraduate studies or possible graduate studies, and also for future employment.
- The only aspect that testbench simulation has not illustrated so far is automated checking that the simulated output is correct. To this point, it is the human (i.e., you) who is required to examine the waveform output and determine if the behavior is correct. With a few additional features of VHDL, it is possible to have the testbench automatically compare the output at any point with expected output values and report any disagreement as an error for the human to consider for corrections to the logic description. The plan is to illustrate some of this automated checking in a later lab.

SHOW YOUR VHDL CODE AND YOUR MODELSIM WAVEFORM OUTPUT FOR CREDIT.

Given canonical sum-of-products function:

f(, , ,) =



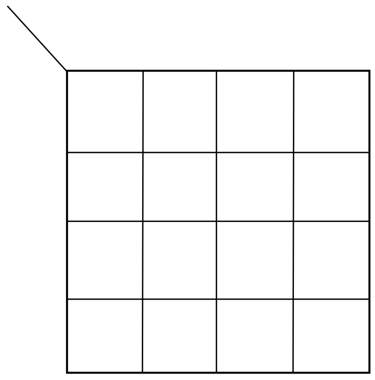
Prime implicant ____________ Essential?

Optimized sum-of-products expression:

____

Equivalent canonical product-of-sums function:

f(, , ,) =



Prime implicant ____________ Essential?

Optimized product-of-sums expression:

____