**ECE 2504 – Introduction to Computer Engineering (Spring 2015)**

**Design Project 1: Design and Implementation of Combinational Circuits**

***Read the entire specification before you begin working on this project!***

1. **Preliminaries**

# *Honor Code Requirements*

You must complete this project and the associated report *individually*. Do not discuss any aspect of your solution or approach with *anyone* except for your instructor or an ECE 2504 GTA. Consider all information that you derive from your design process to be proprietary. This includes (but is not limited to) *your choice of LED display segments*, *the* *manner in which you implement your driver circuits*, and *the number and types of chips that you use*. Copying, using, or discussing any element of any other person’s design is a violation of the Virginia Tech Honor Code, and will be prosecuted as such.

You may discuss general features of Quartus, your trainer kit, and your parts kit with other students. Direct all other questions to your GTA or to your instructor.

# *Objectives*

After completing this project, you should be able to design, simulate, and implement a multiple-output combinational logic circuit from a specification. You will also gain experience in writing a project report to describe a design process and its results.

# *Project Description*

In Section 1, we observed that binary codes need not correspond to numeric values. As long as a one-to-one correspondence exists between the elements of a binary code and the elements of some other data set, we can encode the elements of the data set using the binary code.

In this project, we wish to design a digital circuit that receives four-bit values as inputs and supplies seven-bit values as outputs. The four-bit input values represent single-digit numbers encoded in *excess-3 code*. The seven-bit output values represent signals whose purpose is to drive a seven-segment LED display. The values will cause the LED display to show *alphabetic* characters. After designing and simulating the driver circuits for the seven-segment display, you will select *a subset of the circuits* to implement on your kit.

# *Preparation*

You will need access to a computer that can run version 13.1 of the Quartus II Web Edition software package. You will also need the A&D Trainer Kit and the ECE 2504 Parts Kit.

Read this project specification completely before beginning any design or wiring. Review the appropriate sections of the A&D Board User Manual and Test Procedure for your trainer kit, along with the datasheets for the IC chips in your Parts Kit and the Basic Breadboard Wiring document. Consult the appropriate sections of Chapters 2 and 3 of the course textbook. If you have any questions on the use of the lab kit or your parts, see a CEL GTA or your instructor.

1. **Using Quartus and Qsim in Project 1**

Before proceeding to the specific technical requirements of Project 1, let’s walk through the Quartus archive for Project 1. The purpose of this section of the specification is to illustrate how to use the design tools, since you will be using them to create the schematics for your logic circuit.

Create a working folder for Project 1 (c:\ece2504\project1), and copy basicmoduleP1.qar to that folder.

Open Quartus. In Quartus, choose File > Open Project. Choose basicmoduleP1.qar and open the archive.

In the Entity window, double-click basicmoduleP1. This will open basicmoduleP1.bdf.

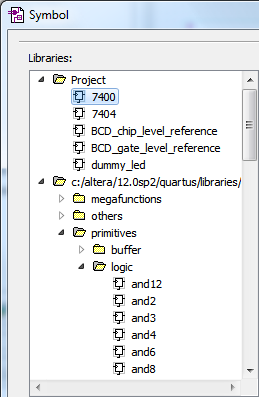
BasicmoduleP1.bdf shows the top-level design. There is a BCD\_gate\_level\_reference component and a BCD\_chip\_level\_reference component. Each one takes a four-bit standard binary-coded decimal number as the input and supply outputs that can drive a seven-segment LED display. Each component connects to a dummy LED, which accepts the driver circuit inputs and displays the corresponding output.

Let’s look at some basic techniques for modifying a BDF file. Figure 1 shows a few of the buttons that you can use to edit the BDF file.



Figure 1.1: A portion of the Circuit Design Toolbar

* To select a component, choose the  button and click on the component you want to select.
* To zoom in and out press . With this cursor selected, left-click to zoom in and right-click to zoom out.
* To add components to a schematic, press  to open the dialogue box shown below:



Chip-level symbols

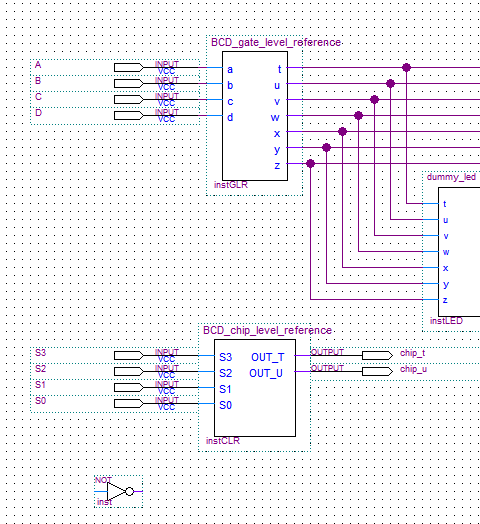
Primitive gates

Figure 1.2: Component Selection

* To wire between pins, use , or click on a pin and drag a wire to the point where you want it to end. Be careful, as dragging wires or parts improperly can result in undesired wire connections.
* To place input and output pins, press the arrow on the  button. Use the pull-down menu to choose the appropriate data direction for the pin.

You should now be able to add your own logic to basicmoduleP1.bdf. Try the following example:

* Click , and select a not gate from Primitive > Logic, and place it into the basicmoduleP1.bdf schematic.



This is the “not” gate.

Figure 1.3: Placement of an inverter

* Use the  button to place input and output pins.



Figure 1.4: Placement of I/O pins. You may need to rotate the pins.

To change a pin name, right-click on the pin and choose Properties. Wire the pins using the  button. Instead of using the  button to place input and output pins one-by-one, you can also right-click on the not gate and choose Generate pins for symbol ports. If you do this, you may still want to edit the pin names.

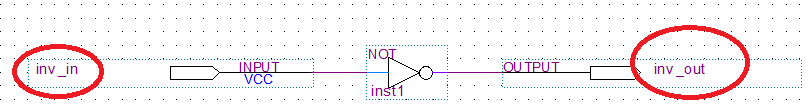


Figure 1.5: Pins with renamed outputs

* Press Ctrl + S to save the BDF file.
* To re-synthesize the project, right-click on Analysis and Synthesis in the Tasks Window and choose Start.

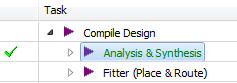
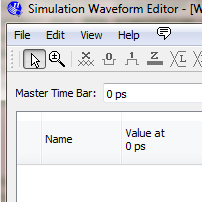


Figure 1.6: Task listing in the Tasks Window

You should now be able to edit the basicmoduleP1.bdf schematics by clicking on the components to open the gate-level and chip-level circuits, adding standalone gates or 7400-series chips, wiring the gates or chips, and assigning input and output pins. You should alter the reference circuits to create your gate-level and chip-level circuits.

Let’s try to simulate a circuit. You have already used Qsim to perform basic simulation operations in Project 0, but let’s look at a brief example of how to observe the inv\_in and inv\_out pins that we just added.

* To begin simulation, choose File > New, then choose University Program VWF under Verification/Debugging Files. This will invoke the Simulation Waveform Editor.
* Double-click the blank area as below to open the Insert Node or Bus dialog box. Click Node Finder…



Double-click blank area

Figure 1.7: Adding Nodes to the Simulation Waveform Editor

* Use the  button to select specific components. Choose Design Entry (All names) as your filter. Select basicmoduleP1 in the Select Hierarchy pop-up window and click OK. Hit List to list all the nodes found. Select inv\_in and inv\_out so that they appear in the Selected Nodes area.

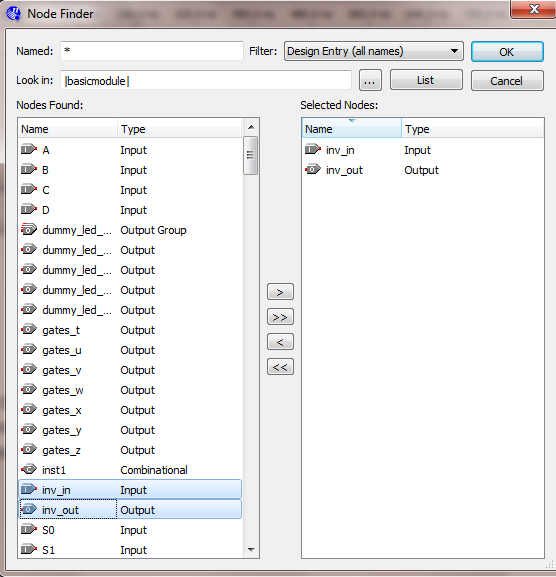


Figure 1.8: Node selection

The result of this step should find inv\_in and inv\_out as the signals you wish to monitor. Choose OK until you have returned to the Simulation Waveform Editor window.

# Figure 9 shows the buttons that you can use to edit waveforms. The buttons are underlined in red.

# 

Figure 1.9: Waveform editing options

# Select part of the waveform using the button. The part you select will be highlighted in blue.

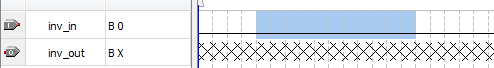


Figure 1.10: Selecting part of a signal for editing

* Clicking on a signal name will highlight the entire length of the signal.



Figure 1.11: Selecting the entire signal for editing

* After choosing some part or the entire length of a signal, we can edit the value of the waveform. Double-clicking the highlighted waveform allows you to edit the value directly. Or you can right click on the highlighted waveform and select Value to see multiple options.

You can use the  buttons to set low and high values (respectively) for a signal. The  button increments the value of a set of signals. For example, if you group two 1-bit signal into a 2-bit signal, you can use  to include all 4 combinations of the signal.

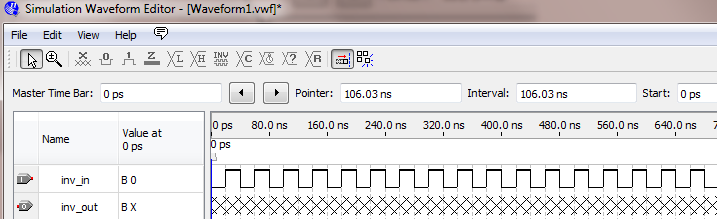


Figure 1.12: Sample input waveform for the inverter simulation

* In the Simulation Wave Editor window, choose File > Save As, give your file a name and click Save.
* Choose Simulation > Functional Simulation. Let the simulation complete. At the end, a window will open containing your simulation result. Figure 13 shows a portion of the simulation of the inverter test.

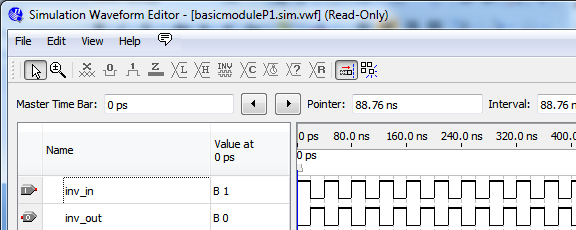


Figure 1.13: Sample output waveform for the inverter simulation

Once you have successfully simulated the inverter circuit, you may delete the inverter and its pins from the circuit schematic

1. **Doing Your Project**

Now that you can use the design tools to both edit and simulate the reference design, let’s return to the general procedure you should follow to design your circuit.

# *Project Requirements*

Let a four-bit binary input code ABCD represent a decimal digit encoded in *excess-3 code*. Only ten of the sixteen four-bit codes correspond to decimal digits. The remaining six combinations represent don’t care conditions. Use the following table to represent the correspondence between decimal digits and codes in this format.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Decimal | - | - | - | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | - | - | - |
| Excess-3 code | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Minterm value | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Figure 2.1: Correspondence between input codes, decimal digits, and minterm values

Note that the correspondence between the four-bit values and the minterm values is the same as it would be if the four-bit values represented standard binary numbers. This is because the minterm representation for a binary combination is the same irrespective of what the bits of the binary combination represent.

Let (t, u, v, w, x, y, z) represent the set of circuit outputs that you will use to drive the inputs of a seven-segment LED display. The outputs are such that when the user applies a valid *excess-3* code as the input, a particular alphabetic character appears on the seven-segment display. The characters should appear as shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0011 → A | 0100 → b | 0101 → C | 0110 → d | 0111 → E |
|  |  |  |  |  |
| 1000 → F | 1001 → g | 1010 → H | 1011 → I | 1100 → J |

Figure 2.1: Complete LED appearances for the ten valid input combinations

(On certain devices, the segments that we have named as “t” through “z” are symbolized as “a” through “g.” Follow the implied alphabetical-order correspondence.)

# To complete Project 1, you must:

* Model a circuit that accepts excess-3 binary-coded decimal inputs and outputs the values that will drive a seven-segment LED display to show the correct alphabetic character.
* Create and simulate a *gate-level circuit* and a *chip-level circuit*.
* Wire your A&D Trainer kit based on your chip-level design, and validate the results of your circuit implementation.

# *Modeling the Circuit*

Design all seven driver circuits to correctly light the segments of the LED display for each valid four-bit input. The seven-segment display in your chipset is such that an input of *logic-0 lights a segment*, while an input of *logic-1 causes a segment to remain unlit*. For example, to display the letter I, segments u and v must be lit, and must therefore receive inputs of 0. Segments t, w, x, y, and z must remain unlit, and must therefore receive inputs of 1. The input that each segment receives is the same as the output of the corresponding circuit driver.

Derive a truth table that shows all input combinations and the corresponding output for each driver circuit. Remember to indicate don’t-care conditions when appropriate. Use the information in the truth table to create your Karnaugh maps, and use your Karnaugh maps to derive logic equations for each output.

# *Creating a Gate-level Circuit*

After you derive the logic equations for your driver circuits, create your design in the BCD\_gate\_level\_reference component of the basicmoduleP1 schematic. Remember to double-click a component to view and modify its internal structure. All seven driver circuits should appear in the same schematic. Use only 2-input NAND gates and inverters in your driver circuits. Show a discrete inverter whenever you need to use one. If you need the complemented version of an input variable, you must use an inverter to derive it.

Provide a screenshot of your final gate-level schematic in your report. Follow the model provided in the gate-level schematic in Quartus. The original circuit provided in the model **does not** correctly implement the requirements of your project. Do not use the circuits in this schematic as elements of your design. You may modify this schematic to create your own. When you make changes to a schematic, you can generate a new symbol file by choosing File > Create/Update > Create Symbol Files for Current File. Follow the instructions described previously for placing new or updated symbols in the schematic window. You may need to rewire the connections and revise the pins in the basicmoduleP1 schematic

*Simulating the Gate-level Circuit*

To simulate your gate-level circuit, connect its outputs to the inputs of a dummy LED and monitor the output of the dummy LED. This should verify the operation of the circuit that you will eventually build. The dummy LED will output a four-bit value that corresponds to the alphabetic character for that four-bit value. Your circuit behavior should be consistent with the table in the Project Requirements section.

For example, if you apply the simulated input ABCD = 0011 (which corresponds to the decimal digit 0 in excess-3 BCD) and obtain the value TUVWXYZ = 0001000 as the 7-segment LED display input (which corresponds to the letter “A”), the dummy LED will encode 0001000 and provide an output of 0011. ***That is, the output of the dummy LED will match the input of your circuit if the circuit produces the correct output.***

The dummy LED will output 1111 for all cases that do not correspond to valid alphabetic characters. If your dummy LED outputs 1111, *or if it outputs a four-bit value that does not match a valid input, you have an error in your circuit.* Your simulation should reveal that the output of the dummy LED matches the input value you provide to your circuit for all valid input combinations. In cases where the dummy LED output does not match the input, use the individual outputs of your circuit - which you should also include in your simulation – to pinpoint the bugs in your circuit. After you verify that the simulation is correct, include a copy of the simulation waveforms in your report.

# *Creating a Chip-level Circuit*

After you verify that all of your driver circuits work, select as many of them as you think you can implement simultaneously using your trainer kit. You must implement at least three complete driver circuits, and you will receive extra credit for implementing more than three. *You are not trying to make three of the ten letters appear completely.* You are trying to implement three of the driver circuits such that the segment in question correctly lights or remains unlit for all ten valid input combinations.

You may not use more than three 7400 2-input NAND gate chips and two 7404 hex inverter chips in your circuit. Your circuit must fit onto one breadboard. Do not use both breadboards. Implement your chip-level design in the basicmoduleP1.bdf schematics. Remember to double-click a component to view and modify its internal structure.

Provide a screenshot of your final chip-level schematic in your report. Follow the model provided in the chip-level schematic in Quartus. The original circuit provided in the model does not correctly implement the requirements of your project. Do not use the circuit in this schematic as an element of your design. You may modify this schematic to create your own. When you make changes to a schematic, you can generate a new symbol file by choosing File > Create/Update > Create Symbol Files for Current File. Follow the instructions described previously for placing new or updated symbols in the schematic window.

*Simulating the Chip-Level Circuit*

After you build your circuit in Quartus, simulate the circuit in the same fashion that you simulated your gate-level circuit. Give appropriate names to the inputs and outputs. Connect the outputs of your chip-level component to a dummy LED.

For the segments that you do not implement at the chip level, connect the outputs of your gate-level component to the dummy LED that you are using to test your chip-level implementation. *The dummy LED needs a complete set of inputs to work properly, thus it is imperative that you obtain a gate-level implementation that works properly before you proceed to the chip-level implementation.* You will need to disconnect certain wires from the dummy LED for the existing chip-level circuit. If you need to, delete the wires connecting the chip-level circuit to the dummy LED and rewire the connections based upon your implementation.

You will know that your chip-level implementation works if you can replace the outputs of your gate-level implementation with their chip-level counterparts and still produce working outputs from a dummy LED. After you verify that the simulation is correct, include a copy of the waveforms in your report.

# *Building the Circuit*

After you simulate your chip-level schematic and verify that it works, build the circuit on your trainer kit. Use the table below to place your inputs. Use the logic indicators as indicated to display your inputs. Read the silkscreen on your trainer kit to locate these input switches and logic indicators. The outputs of the circuit will appear on the LED display.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs | | | | Displayed Inputs | | | |
| S1 | S2 | S3 | S4 | L1 | L2 | L3 | L4 |
| A | B | C | D | A | B | C | D |

*Make sure that you place a resistor in series with the driver circuit output to limit the current flow through the LED segment.* Leaving out the resistor will damage your seven-segment display. Wire your circuit neatly, and color-code your wires to distinguish the various inputs and outputs from one another. Follow the wiring guidelines and wiring grading criteria discussed in the Lab Manual. Wire one driver circuit and verify that circuit’s correct operation before wiring the next circuit. This is simpler than wiring the entire circuit and then trying to find a wiring mistake once the circuit is complete.

If your circuit simulated correctly in Quartus but does not work when you build it, the problem is in your wiring, your chips, or your trainer kit. Remember that power and ground connections are not required for the chips to work in Quartus, but they must be connected correctly for the chips to work on your kit.

1. **Completing Your Project**

# *Circuit Validation*

Once you are satisfied that your circuit works, take your trainer kit, a copy of your chip-level schematic, and a copy of the project validation sheet to the CEL and have a CEL GTA validate the circuit’s operation. *The GTA will not validate your circuit if you do not have your schematic with you.* Validation lines become longer as the project due date approaches, so validate your circuit as early as you can.

During validation, the GTA’s only responsibility is to record your circuit’s output on the validation sheet. *It is your responsibility to make sure that the validation proceeds correctly.* If you have questions during your validation, ask the GTA who is performing the validation.

To help the GTA during the validation of your circuit, label the input switches that you use on your trainer kit, and provide a legend for the seven-segment display. Masking tape and note cards make good labels. Do not assume that the GTA validating your project will know which of the inputs is “A” or which of the segments is “v.” Also, indicate on your validation sheet which driver segments you have implemented.

# *Project Report*

After you have validated your logic circuit, prepare and submit a written lab report that presents a detailed discussion of the project. It should include the design approach you followed, the equations you derived, the driver circuits you chose to implement and how you selected them, the kinds of design decisions you made and the alternatives you considered, your simulation results, your evaluation of the implementation, your observations, and your conclusions.

Make sure that you submit your validation sheet according to the instructions that your instructor provides. If your class uses Scholar as a means of report-submission, you may or may not need to submit a hard-copy of your completed validation sheet. Follow your instructor’s guidance.

Prepare your report using a word processor. *Do not include handwritten items*. Proofread your report to make sure that it is free of spelling and grammar errors. Use the cover sheet included with this specification as the first page of your report. Do not use any other cover page. Please submit your report with completed cover sheet as a PDF file on Scholar.

# *Grading*

The design project will be graded on a 100 point basis, as shown on the project report cover sheet.