



**ELECTRICAL AND ELECTRONICS ENGINEERING  
&  
COMPUTER ENGINEERING**

**EEE 248 | CNG 232**

**Logic Design**

**21 | FALL | 22**

**LABORATORIES**

## 1 TABLE OF CONTENTS

<b>REGULATIONS</b>	<b>3</b>
<b>EXPERIMENT #1</b>	<b>4</b>
<b>INTRODUCTION TO DIGITAL DESIGN ENTRY, SIMULATION, AND IMPLEMENTATION</b>	<b>4</b>
<b>1.1 OBJECTIVE</b>	<b>4</b>
<b>1.2 PRELIMINARY WORK</b>	<b>4</b>
1.2.1 VERILOG HDL	4
1.2.2 GATES	4
1.2.3 COMPLEX GATES	5
1.2.4 WORD PROBLEM: BAKERY PROFIT CALCULATOR	5
<b>1.3 INTRODUCTION TO VERILOG</b>	<b>7</b>
1.3.1 STRUCTURAL (GATE-LEVEL) SPECIFICATION	8
1.3.2 BEHAVIOURAL DESCRIPTION	8
1.3.2.1 Continuous approach	9
1.3.2.2 Procedural approach:	9
<b>1.4 EXPERIMENTAL WORK</b>	<b>10</b>
1.4.1 EXPERIMENTAL SETUP	10
1.4.2 BASIC GATES SCHEMATIC ENTRY, SIMULATION, AND IMPLEMENTATION	10
1.4.2.1 Launch Quartus II Project Navigator	10
1.4.2.2 Create a New Project	10
1.4.2.3 Enter the schematic	11
1.4.2.4 Compiling the design	11
1.4.2.5 Pin Assignment	11
1.4.2.6 Simulation	14
1.4.2.7 Functional Simulation	15
1.4.2.8 Timing Simulation	15
1.4.2.9 The Programming And Configuration Task Is Performed As Follows In Order To Implement The Design Using The Programmable (Fpga) Device	15
1.4.2.10 Validate your design	15
1.4.3 COMPLEX GATES VERILOG HDL DESIGN, SIMULATION AND IMPLEMENTATION	16
1.4.3.1 Create A New Project	16
1.4.3.2 Enter the schematic	16
1.4.4 BAKERY PROFIT CALCULATOR VERILOG HDL DESIGN, SIMULATION, AND IMPLEMENTATION	17
1.4.4.1 Create A New Project	17
1.4.4.2 Enter The Schematic	17
1.4.4.3 Modelsim Verilog Code And Simulation	18
<b>1.5 LIST OF DELIVERABLES</b>	<b>18</b>
1.5.1 PREWORK	18
1.5.2 EXPERIMENTAL	18
<b>1.6 REFERENCES</b>	<b>19</b>

## 2 REGULATIONS

- Students are not permitted to perform an experiment without doing **the preliminary work** before coming to the laboratory. It is **not allowed** to do the preliminary work at the laboratory during the experiment. **Students, who do not turn in the complete preliminary work printout at the beginning of the laboratory session, cannot attend the lab. No “make-up” is given in that case.**
- No food or drink in the lab.
- There may be a quiz before each laboratory session, which will start promptly at the beginning of the lab. **Students who miss the quiz cannot attend the lab. No “make-up” is given in that case.** There won't be any extensions in the quiz time for latecomers. Therefore, students have to be at the lab on time for the quiz.
- Only the following excuses are valid for taking lab make-up:
  1. **Health Make-up:** Having a health report from METU Medical Centre.
  2. **Exam Make-up:** Having an exam coinciding with the time of the laboratory session. The student needs to notify the instructor in advance if this is the case.
- Experiments will be done **individually**. The lab instructor will inform you of any part that you can do in groups.
- **Cheating or plagiarism is not tolerated. Plagiarism is a form of cheating as is using someone else's written word with minor changes and no acknowledgement. If you are caught cheating or plagiarising, you will at the very least receive a zero for the whole experiment. Disciplinary action may be taken.**
- Students who miss the lab **two times** without a valid excuse get zero as the laboratory portion of the course grade.
- *Those who fail to get a satisfactory score from the laboratory portion may fail the class. This score is expected to be 70% but may be adjusted up or down with the initiative of the course instructor.*

## EXPERIMENT #1

### INTRODUCTION TO DIGITAL DESIGN ENTRY, SIMULATION, AND IMPLEMENTATION

#### 2.1 OBJECTIVE

The purpose of the first laboratory exercise is getting familiar with Quartus II Project Navigator, DEO Demo Board, and the associated toolset to facilitate the digital logic design. The student will complete a step-by-step tutorial in the first part of the experiment to implement complex gates (XOR, NAND, NOR) on a Cyclone III FPGA (Field Programmable Gate Array) using schematic entry, simulation, and FPGA programming tools. Verilog will be utilised in the second part of the experiment to implement a 1-bit full adder designed in the preliminary work. The relationship between majority-voter, odd-detector and full-adder logic will also be studied.

#### 2.2 PRELIMINARY WORK

##### 2.2.1 VERILOG HDL

Read through Section 1.3 to acquire general familiarity with Verilog. This is a very brief summary, and good references are available in the library if you need further information.

##### 2.2.2 GATES

1. For the logic gates given below.
  - i. Derive the truth table and Boolean expression for each logic gate.
  - ii. Draw the corresponding logic schematics for all gates.
2. Refer to MODELSIM™ tutorial for Verilog HDL coding and simulation.
  - i. Write a structural Verilog code for each logic gate and simulate using Modelsim. (In your report show your simulation outputs.)

a. NOT	b. AND	c. OR	d. NAND
e. NOR	f. XOR	g. XNOR	
3. Write a structural Verilog code and implement all logic gates together as depicted in figure, simulate and verify the functionality using Modelsim. Compare your results with 1.i and 1.ii. This implementation will then be used in LAB2&3 as logic unit.

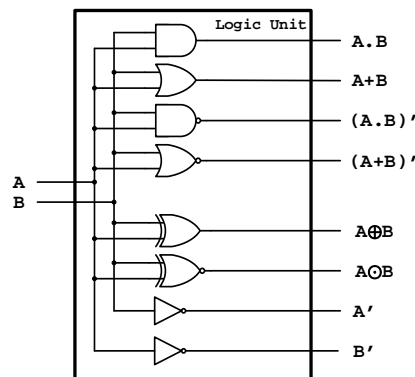


Figure 1: Logic Unit

### 2.2.3 COMPLEX GATES

Table 1 (a-b) provide the truth tables corresponding to a 3-input minority gate and 3-input even-detector.

*Table 1:1.2.3. Complex Gates*

(a) 3-input Minory Gate				(b) 3-input Even Detector			
a	b	c	MN-out	a	b	c	EVEN-out
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	0
1	0	1	0	1	0	1	1
1	1	0	0	1	1	0	1
1	1	1	0	1	1	1	0

$$Q = (A + B) (AB)'$$

- Briefly explain in words the function of each logic block represented by the truth tables (a) and (b) i.e., describe the condition for which the output becomes '1' for each function.
- Using AND, OR, and inverter operators, derive a Boolean expression for each of the two outputs as a function of the inputs:
  - MN-out (a, b, c) = ?
  - EVEN-out (a, b, c) = ?
- Sketch a logic schematic implementation for each of your answers in 1.2.3,2.
- Given the logic XNOR function in 1.2.2, redesign the 'EVEN-out' output using XNOR gates only i.e., first express EVEN-out as an expression with one or more XNOR operations, and then sketch a schematic.

### 2.2.4 WORD PROBLEM: BAKERY PROFIT CALCULATOR

In this design, you will implement a simple digital circuit to compute total profits made by a bakery based on the kinds of deserts that they are baking.

Bakery owner has to follow the set of guidelines below for setting up the bakery.

- Baker can bake donuts, brownies, eclairs, and croissants on his bakery but cannot produce all at the same time.
- The bakery does not have enough space to bake more than 2 different types of deserts. Therefore, his bakery never has more than 2 types of deserts at any particular time.
- Each type of the desert earns a certain level of profit for the bakery. Donuts earn him a profit of 4 units, brownies earn him a profit of 2 units, eclairs earn a profit of 3 units and croissants being small livestock earn him a profit of 1 unit.
- The total profit of the bakery is the sum of profits made by each type of desert that is baked with the exceptions provided below.
- Brownies and croissants have very different ingredients, so Baker needs to spend extra for the sourcing of ingredients for each separately. Therefore, if he plans to bake brownies and croissants together then his profits go down by 1 unit.
- Similarly, if he bakes donuts and croissants his profits go down by 1 unit because of extra sourcing costs. Additionally, donuts have a chance to require more labor than croissants on the bakery. This further reduces his profits by 1 unit.
- However, croissants and eclairs do very well if baked together. They have many similarities especially with respect to the kind of ingredients they have. Also due to the huge demand for eclairs and croissants, the profit they make goes up by 1 unit if croissants and eclairs are baked together.

Your objective is to design a digital circuit which takes in the types of deserts on the bakery and outputs a binary number representing the corresponding profit. The four input bits are C (Croissants), E (Eclairs), D (Donuts), and B (Brownies), and the output bits are, P2, P1, P0, such that P2 is the most significant bit. Begin by creating a truth table containing all of the input combinations. Because there are four input bits, your table should contain  $2^4 = 16$  rows. Then fill in the output columns of your truth table based on the guidelines provided above. For outputs that are not possible due to restrictions on the bakery, insert 'X' to represent "don't cares." Once your truth table is complete, use Karnaugh Maps as discussed in class to simplify the Boolean algebra expressions for each of the output bits. To conclude the design, draw a schematic of the final digital circuit. Be sure to label all inputs and outputs.

### 2.3 INTRODUCTION TO VERILOG

Verilog, standardized as **IEEE 1364**, is a hardware description language (HDL) used to model electronic systems. It is most commonly used in the design and verification of digital circuits at the logic (register-transfer) level of abstraction. Verilog is very similar to the software programming languages like C, but it is used to describe logic gates and their interconnections i.e., it describes hardware instead of software. Similar to C, it is composed of reserved keywords and its own syntax. Verilog is case sensitive. A Verilog design may be in a single block, or in several blocks. Each block in Verilog is created using the keyword **module** and ends with the keyword **endmodule**. A name always follows module keyword which is called an identifier. Identifiers are names given to all elements of a piece of Verilog code so that they can be referenced. After the identifier, a port list should be present to declare the name of the input and output ports of the block. The next two lines of the code are dedicated to distinguishing the input and output ports, all of which first appeared in the module port list. The following is an example of defining a new block:

```
module Simple_Circuit (A,B,C,D,E);
    output D,E;
    input  A,B,C;

    lines to describe the functionality of the module;

endmodule
```

In this code, the first line indicates a definition of a new block or **module**, named “Simple\_Circuit”. The last line marks the end of the block. The port list in the first line contains the names of all interface signals. The next lines, **output** and **input**, describe the type of interface signals. An interface signal is typically either an input or an output but can be bidirectional in some cases, in which case it would be declared as **inout**. To describe the functionality of the module there are two standard methods called “structural specification” and “behavioural specification”. All interconnects between logic gates are described using keyword **wire** in the structural specification. One may think of this as node-by-node description of the logic circuit. Each internal node corresponds to a **wire**. Therefore, a structural code would describe how different logic gates of the block are connected. A behavioural code would describe how the block behaves instead. The behavioural specification can be defined in two different ways: Continuous assignment and procedural assignment. The following sections demonstrate simple coding in Verilog for the logic circuit shown in Figure 2, using structural and behavioural methods.

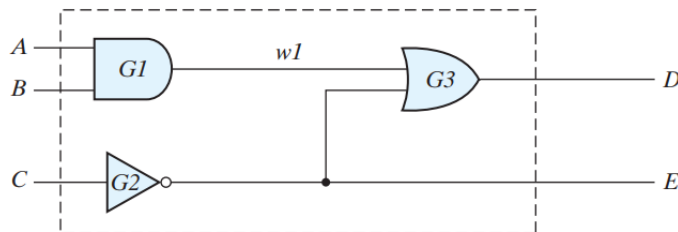


Figure 2: A simple logic block with three inputs A, B, C and two outputs D, E.

### 2.3.1 STRUCTURAL (GATE-LEVEL) SPECIFICATION

The structural specification is very similar to describing the block diagram using text, that is instantiating all blocks and defining the interconnects using **wires** for nodes. The following is the structural specification of the block shown in Figure 2.

```
module Simple_Circuit (A,B,C,D,E);  
output D,E;  
input A,B,C;  
wire w1;  
and G1(w1,A,B);  
not G2(E,C);  
or G3(D,w1,E);  
endmodule
```

The circuit in this example has one internal node, w1, which is declared with the keyword **wire**. The structure of the circuit is specified by a list of (predefined) primitive gates, each identified by a descriptive keyword (**and**, **not**, **or**) that corresponds to modules that are defined in an available library. The elements of the list are referred to as instantiations of a gate, each of which is referred to as a gate instance. Each gate instantiation consists of an optional name (such as G1, G2, etc.). Note that the gate instance names are optional, and in general make it easy to reference the gates. This means we could, for example, define the and gate as follows without an instance name:

```
and (w1, A, B)
```

The outputs of a primitive gate are always listed first, followed by the inputs. However, the order of inputs and outputs is not important. The module description ends with the keyword **endmodule**. Each statement must be terminated with a semicolon, but there is no semicolon after **endmodule**. In the structural specification, the order of the gate instantiations in the model has no significance and does not specify a sequence for computation.

### 2.3.2 BEHAVIOURAL DESCRIPTION

The behavioural approach to modelling hardware components is different from the structural method in that it does not necessarily reflect how the design is implemented. It is basically the black-box approach to modelling. It accurately models what happens on the inputs and outputs of the black box, but what is inside the box (how it works) is irrelevant. The behavioural description is usually used in two ways in Verilog. First, it can be used to model complex components that would be tedious to model using other methods. Second, the behavioural capabilities of Verilog can be more powerful and are more convenient for some designs. In the early part of the design process one typically uses high-level behavioural models to quickly evaluate an algorithm, and simulate it by feeding real data to it. Behavioural descriptions are supported in two different types, 1- Continuous, in which Boolean expressions are used to describe input-output relations of the block, and 2- Procedural in which similar to software programming languages loop and condition statements such as if-else, do-while, case, etc. can be used. With the procedural approach, the keyword **always** is used, followed by a list of signals in the parenthesis. This list is called sensitivity list and defines the list of signals that Verilog should monitor to determine if an event has occurred before recomputing the contents of the procedure. The statements (or elements) after **always** are used to compute the outputs of the block from its inputs whenever a signal in the sensitivity list changes. The following codes illustrate behavioural specification of the block in Figure 2 using continuous and procedural approaches.



### 2.3.2.1 CONTINUOUS APPROACH

```
module Simple_Circuit (A,B,C,D,E);  
output D,E;  
input A,B,C;  
assign E=!C;  
assign D=(!C) || (A && B);  
endmodule
```

### 2.3.2.2 PROCEDURAL APPROACH:

```
module Simple_Circuit (A,B,C,D,E);  
output D,E;  
input A,B,C;  
reg D,E;  
always @(A,B,C);  
if (C==0) begin E=1'b1; D=1'b1; end  
else if (A==1 & B==1) begin E=1'b0; D=1'b1; end  
else begin E=1'b0; D=1'b0; end  
endmodule
```

Note that to assign value to the output signals a special syntax is used as **1'bx**. In this syntax the first number represents the number of bits, b indicates that the value is a binary value, and the last number is the value. Therefore, **1'b0** refers to a one-bit binary with value 0. In addition, the target output of a procedural assignment statement must be of the **reg** data type. Contrary to the wire data type, whereby the target output of an assignment may be continuously updated, a **reg** data type retains its value until a new value is assigned (whenever a signal in the sensitivity list changes). In order to have more than one statement within an **if**, **else if**, or **else** condition, multiple statements are bracketed together using the **begin...end** keywords. **always** is one of the most confusing structures in Verilog. One may think of this as '*always look for events or changes in the signals in the sensitivity list, and do not execute any of the elements that follow unless there is an event or change*'. When '=' sign is used to assign signals in the elements that follow **always**, then assignments happen in sequential order, as in high-level programming. Hence '=' is called a *blocking* assignment. An alternative that will be discussed in a future laboratory is using '<=', a *non-blocking* assignment. Non-blocking assignments in elements happen in parallel (all at the same time).

## 2.4 EXPERIMENTAL WORK

### 2.4.1 EXPERIMENTAL SETUP

Verify to make sure your workbench has all of the following items:

- A Personal Computer (PC) with Altera Quartus II ISE 13.0 Service pack 1 Project Navigator
- DEO Demo Board with Cyclone III EP3C16F484C6 FPGA installed on a card with 10 input toggle switches, three pushbuttons, and four 7-Segment LED displays, among other components.
- A cable connected between the demo board and the PC using USB interface in order to transfer design information from the PC to the Demo Board.

### 2.4.2 BASIC GATES SCHEMATIC ENTRY, SIMULATION, AND IMPLEMENTATION


#### 2.4.2.1 LAUNCH QUARTUS II PROJECT NAVIGATOR

#### 2.4.2.2 CREATE A NEW PROJECT

Create a New project which will target the FPGA device on the DE0 Demo Board. To create a new project:

1. Select **File > New Project Wizard...** select New Quartus II Project. The New Project Wizard appears.
2. Look at the Introduction and click **Next**.
3. Type **lab1gates** in the Project Name field. Enter or browse to a location (directory path) for the new project. You can select the directory name as **<YourInitials>\_lab1**. Click **Next**.
4. You are given the chance to add existing design files to the project (if any). As you do not have any files to add click **Next**.
5. Family and Device settings window appears. Select Cyclone III as the family first. You will see a list of available devices in the list table. Select **EP3C16F484C6** and click **Next**.
6. The user can specify any third-party tools that should be used. A commonly used term for CAD software for electronic circuits is EDA tools, where the acronym stands for Electronic Design Automation. This term is used in Quartus II messages that refer to third-party tools, which are the tools developed and marketed by companies other than Altera. Since we will rely solely on Quartus II tools for this lab, we will not choose any other tools. Press **Next**.
7. A summary of the chosen settings appears. Press **Finish**, which returns to the main Quartus II window, but with lab1gates specified as the new project, in the display title bar.
8. The Quartus II Graphic Editor can be used to specify a circuit in the form of a block diagram. Select **File > New** and choose **Block Diagram/Schematic File** and click **OK**. This opens the Graphic Editor window. The first step is to specify a name for the file that will be created. Select **File > Save As**. In the box labelled File name type **lab1gates**, to match the name given to the project, which was specified when the project was created. Put a checkmark in the box **Add file to current project**. Click **Save**, which puts the file into the directory **<YourInitials>\_lab1** and leads back to the Graphic Editor window.

#### 2.4.2.3 ENTER THE SCHEMATIC

1. Double-click on the blank space in the Graphic Editor window or click on the **icon** in the toolbar that looks like an **AND** gate. A pop-up window appears. Expand the hierarchy in the Libraries box.
2. First, expand **libraries**, and then expand the library **primitives**, followed by expanding the library **logic** which contains the logic gates.
3. Place the **and2**, **or2**, and **not** (inverter) gates as many times as needed for the designs from the preliminary work **Section 1.2.2**.
4. Use the same procedure for choosing **input and output port** symbols from the library **primitives/pin**.
5. **Assign names to the input and output port symbols** as follows: Make sure nothing is selected by clicking on an empty spot in the Graphic Editor window. Point to the word **pin\_name** on top of the input symbol and double-click the mouse. A dialogue box will appear. Type the pin name, let's say **x1**, and click OK. Similarly, assign the name **x2** to the other input and **f** to the output pin. It is possible to change the name of an element by selecting it first, and then double-clicking on the name and typing a new name.
6. Finally **Connecting Nodes with Wires**: Click on the  icon in the toolbar to activate the Orthogonal Node Tool. Position the mouse pointer over the right edge of one of your input pins. Click and hold the mouse button and drag the mouse to the right until the drawn line reaches the pinstub at the input of one of the gates.

#### 2.4.2.4 COMPILING THE DESIGN

1. Run the Compiler by selecting **Processing > Start Compilation**, or by clicking on the toolbar icon that looks like a purple triangle
2. The successful (or unsuccessful) compilation is indicated in a pop-up box. Acknowledge it by clicking **OK**.

#### 2.4.2.5 PIN ASSIGNMENT

1. Pin assignments are made by using the **Pin Planner**. Select **Assignments > Pin Planner** to start.
2. Select **x1** as the first pin to be assigned. To do this, double-click on the box in the column labelled **Location** to the right of **x1** entry. A drop-down menu appears. Scroll down and select **PIN\_J6**. Instead of scrolling down the menu to find the desired pin, you can just type the name of the pin (**J6**) in the Location box.
3. Use the same procedure to assign input **x2** to **pin H5** and output **f** to **pin J1**
4. To save and close the assignments made, choose **File > close**.
5. Recompile the circuit, so that it will be compiled with the correct pin assignments. Run the Compiler by selecting **Processing > Start Compilation**.

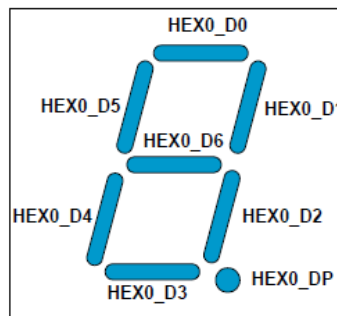
*Table 2:DE0 Test Box Input Switch Cyclone III FPGA pin mapping*

Signal Name	FPGA Pin No.	Description
SW[0]	PIN_J6	Slide Switch[0]
SW[1]	PIN_H5	Slide Switch[1]
SW[2]	PIN_H6	Slide Switch[2]
SW[3]	PIN_G4	Slide Switch[3]
SW[4]	PIN_G5	Slide Switch[4]
SW[5]	PIN_J7	Slide Switch[5]
SW[6]	PIN_H7	Slide Switch[6]
SW[7]	PIN_E3	Slide Switch[7]
SW[8]	PIN_E4	Slide Switch[8]
SW[9]	PIN_D2	Slide Switch[9]

*Table 3:DE0 Test Box Input Button Cyclone III FPGA pin mapping*

Signal Name	FPGA Pin No.	Description
BUTTON [0]	PIN_H2	Pushbutton[0]
BUTTON [1]	PIN_G3	Pushbutton[1]
BUTTON [2]	PIN_F1	Pushbutton[2]

*Table 4:DE0 Test Box HEX Output Cyclone III FPGA pin mapping for HEX0*




Signal Name	FPGA Pin No.	Description
HEX0_D[0]	PIN_E11	Seven Segment Digit 0[0]
HEX0_D[1]	PIN_F11	Seven Segment Digit 0[1]
HEX0_D[2]	PIN_H12	Seven Segment Digit 0[2]
HEX0_D[3]	PIN_H13	Seven Segment Digit 0[3]
HEX0_D[4]	PIN_G12	Seven Segment Digit 0[4]
HEX0_D[5]	PIN_F12	Seven Segment Digit 0[5]
HEX0_D[6]	PIN_F13	Seven Segment Digit 0[6]
HEX0_DP	PIN_D13	Seven Segment Decimal Point 0


*Table 5:DE0 Test Box Output LED Cyclone III FPGA pin mapping*

Signal Name	FPGA Pin No.	Description
LEDG[0]	PIN_J1	LED Green[0]
LEDG[1]	PIN_J2	LED Green[1]
LEDG[2]	PIN_J3	LED Green[2]
LEDG[3]	PIN_H1	LED Green[3]
LEDG[4]	PIN_F2	LED Green[4]
LEDG[5]	PIN_E1	LED Green[5]
LEDG[6]	PIN_C1	LED Green[6]
LEDG[7]	PIN_C2	LED Green[7]
LEDG[8]	PIN_B2	LED Green[8]
LEDG[9]	PIN_B1	LED Green[9]


#### 2.4.2.6 SIMULATION

1. Open the Waveform Editor window by selecting **File > New > University Program VWF**.
2. The Associated Source page shows that you are associating the test bench waveform with the source file xor gate. Click **Next**.
3. The Waveform Editor window opens. Save the file under the name **lab1gates.vwf**; note that this changes the name in the displayed window.
4. Set the desired simulation to run from 0 to 500 ns by selecting **Edit > End Time** and entering 500 ns in the dialogue box that pops up. Selecting **View > Fit** in Window displays the entire simulation range of 0 to 500 ns in the window. You may wish to resize the window to its maximum size.
5. Next, include the input and output nodes of the circuit to be simulated: Click **Edit > Insert > Insert Node or Bus**. It is possible to type the name of a signal (pin) into the Name box, but it is easier to click on the button labelled **Node Finder**. The Node Finder utility has a filter used to indicate what type of nodes are to be found. Since we are interested in input and output pins, set the filter to Pins: all. Click the List button to find the input and output nodes
6. Click on the **x1** signal in the Nodes Found box, and then click the **>** sign to add it to the Selected Nodes box on the right side of the window. Do the same for **x2** and **f**. Click **OK** to close the **Node Finder** window, and then click **OK** in the **Insert Node or Bus** window. This leaves a fully displayed Waveform Editor window. (If you did not select the nodes in the same order as explained above, it is possible to rearrange them. To move a waveform up or down in the Waveform Editor window, click on the node name (in the Name column) and release the mouse button. The waveform is now highlighted to show the selection. Click again on the waveform and drag it up or down in the Waveform Editor).
7. We will now specify the logic values to be used for the input signals **x1** and **x2** during the simulation. The logic values at the output **f** will be generated automatically by the simulator. To make it easy to draw the desired waveforms, the Waveform Editor displays (by default) vertical guidelines and provides a drawing feature that snaps on these lines (which can otherwise be invoked by choosing **Edit > Snap** to Grid or  icon).
8. We will use four 100-ns time intervals to apply the four test vectors. We can generate the desired input waveforms as follows:
  - Click on the waveform name for the **x1** node. Once a waveform is selected, the editing commands in the Waveform Editor can be used to draw the desired waveforms. (Commands are available for setting a selected signal to 0, 1, unknown (X), high impedance (Z), don't care (DC), inverting its existing value (INV), or defining a clock waveform). Each command can be activated by using the **Edit > Value** command, or via the toolbar for the Waveform Editor. The Edit menu can also be opened by **right-clicking on a waveform name**.
9. Set **x1** to **0** in the time interval 0 to 100 ns, which is probably already set by default. Next, set **x1** to **1** in the time interval 100 to 200 ns. Do this by pressing the mouse at the start of the interval and dragging it to its end, which highlights the selected interval, and choosing the logic value 1 in the toolbar. Next, set **x1** to **0** in the time interval 200 to 300 ns. Next, set **x1** to **1** in the interval 300 to 400 ns.
10. Now, Make **x2 = 1** from 200 to 400 ns. Observe that the output **f** is displayed as having an unknown value at this time, which is indicated by a hashed pattern; its value will be determined during the simulation. Save the file.

#### 2.4.2.7 FUNCTIONAL SIMULATION

1. To perform the functional simulation, select **Simulation > Run Functional Simulation**. The Quartus II simulator takes the inputs and generates the outputs defined in the lab1gates.vwf file.
2. It is also possible to run a functional simulation by using the icon . At the end of the simulation, Quartus II software indicates its successful completion and displays a Simulation Report. If your report window does not show the entire simulation time range, click on the report window to select it and choose **View > Fit in Window**.

#### 2.4.2.8 TIMING SIMULATION

1. Having ascertained that the designed circuit is functionally correct, we should now perform the timing simulation to see how the physical signal delays in the chosen FPGA device will affect the output.
2. To perform the timing simulation select **Simulation > Run Timing Simulation**. It is also possible to run a timing simulation by using the icon . The simulator should produce the resulting waveforms.

#### 2.4.2.9 THE PROGRAMMING AND THE CONFIGURATION OF (FPGA) DEVICE

1. Flip the **RUN/PROG** switch into the **RUN** position.
2. Select **Tools > Programmer**. Here it is necessary to specify the programming hardware and the mode that should be used.
3. If not already chosen by default, select **JTAG** in the **Mode** box. Also, if the **USB-Blaster** is not chosen by default, press the **Hardware Setup...** button and select the **USB-Blaster** in the window that pops up.
4. Observe that the configuration file **lab1gates.sof** is listed in the window.
5. If the file is not already listed, then click **Add File** and select it.
6. Note that the device selected needs to be **EP3C16F484**, which refers to the particular FPGA model on the DE0 board.
7. Click on the **Program/Configure** checkbox
8. Now, press **Start** in the window. This loads the file on the processor.

#### 2.4.2.10 VALIDATE YOUR DESIGN

1. The final step is the hardware validation to make sure the functionality of the FPGA chip is as you expected based on your design.
2. Use the switches and LEDs you have programmed in step 1.4.2.5 to test all possible combinations of inputs.
3. Once you are convinced your design works, go back, and **add 2-input NOR and NAND designs** to your schematic. You can use the same inputs or define completely different inputs (switches). However, ensure your outputs (LEDs) are distinct for each of XOR, NAND, and NOR.

***Demonstrate the hardware functionality on the DE0 board to your lab instructor.***




### 2.4.3 COMPLEX GATES VERILOG HDL DESIGN, SIMULATION AND IMPLEMENTATION

#### 2.4.3.1 CREATE A NEW PROJECT

1. Select **File > New Project Wizard...** The New Project Wizard appears.
2. Type **lab1complex** in the Project Name field. Enter or browse to a location (directory path) for the new project.
3. You are given the chance to add existing design files to the project (if any). As you do not have any files to add click **Next**.
4. Family and Device settings window appears. Select Cyclone III as the family first. You will see a list of available devices in the list table. Select **EP3C16F484C6** and click **Next**.
5. Click **Next** to skip choosing third-party tools.
6. Press **Finish**, which returns to the main Quartus II window, but with lab1complex specified as the new project, in the display title bar.
7. Quartus II text editor will be used this time to define the circuit using Verilog code. Select **File > New**, choose **Verilog HDL File**, and click **OK**. This opens the Text Editor window. The first step is to specify a name for the file that will be created. Select **File > Save As**. Type **lab1complex** in the box labelled file name to match the name given to the project. Put a checkmark in the box **Add file to the current project**. Click **Save**, which puts the file into the project directory and leads back to the Text Editor window.

#### 2.4.3.2 ENTER THE SCHEMATIC

1. Double-click on the blank space in the Graphic Editor window or click on the **icon** in the toolbar that looks like an **AND** gate. A pop-up window appears. Expand the hierarchy in the Libraries box.
2. First, expand **libraries**, and then expand the library **primitives**, followed by expanding the library **logic** which contains the logic gates.
3. Place the **and2**, **or2**, and **not** (inverter) gates as many times as needed for the complex gate designs from the preliminary work **Section 1.2.3**.
4. Use the same procedure for choosing **input and output port** symbols from the library **primitives/pin**.
5. **Assign names to the input and output port symbols** as follows: Make sure nothing is selected by clicking on an empty spot in the Graphic Editor window. Point to the word **pin\_name** on top of the input symbol and double-click the mouse. A dialogue box will appear. Type the pin name, let's say **x1**, and click OK. Similarly, assign the name **x2** to the other input and **f** to the output pin. It is possible to change the name of an element by selecting it first, and then double-clicking on the name and typing a new name.
6. Finally **Connecting Nodes with Wires**: Click on the  icon in the toolbar to activate the Orthogonal Node Tool. Position the mouse pointer over the right edge of one of your input pins. Click and hold the mouse button and drag the mouse to the right until the drawn line reaches the pinstub at the input of one of the gates.
7. Follow the compilation, simulation, implementation, and validation steps as described in sections 1.4.2.4 - 1.4.2.10.
8. **Demonstrate the complex gates simulation and hardware validation results to your lab instructor.**




## 2.4.4 BAKERY PROFIT CALCULATOR VERILOG HDL DESIGN, SIMULATION, AND IMPLEMENTATION

### 2.4.4.1 CREATE A NEW PROJECT

1. Select **File > New Project Wizard...** The New Project Wizard appears.
2. Type **lab1bakery** in the Project Name field. Enter or browse to a location (directory path) for the new project.
3. You are given the chance to add existing design files to the project (if any). As you do not have any files to add click **Next**.
4. Family and Device settings window appears. Select Cyclone III as the family first. You will see a list of available devices in the list table. Select **EP3C16F484C6** and click **Next**.
5. Click **Next** to skip choosing third-party tools.
6. Press **Finish**, which returns to the main Quartus II window, but with lab1bakery specified as the new project, in the display title bar.
7. Quartus II text editor will be used this time to define the circuit using Verilog code. Select **File > New**, choose **Verilog HDL File**, and click **OK**. This opens the Text Editor window. The first step is to specify a name for the file that will be created. Select **File > Save As**. Type **lab1bakery** in the box labelled file name to match the name given to the project. Put a checkmark in the box **Add file to the current project**. Click **Save**, which puts the file into the project directory and leads back to the Text Editor window.

### 2.4.4.2 ENTER THE SCHEMATIC

1. Double-click on the blank space in the Graphic Editor window or click on the **icon** in the toolbar that looks like an **AND** gate. A pop-up window appears. Expand the hierarchy in the Libraries box.
2. First, expand **libraries**, and then expand the library **primitives**, followed by expanding the library **logic** which contains the logic gates.
3. Place the **and2**, **or2**, and **not** (inverter) gates as many times as needed for the bakery profit calculator design from the preliminary work **Section 1.2.4**.
4. Use the same procedure for choosing **input and output port** symbols from the library **primitives/pin**.
5. **Assign names to the input and output port symbols** as follows: Make sure nothing is selected by clicking on an empty spot in the Graphic Editor window. Point to the word **pin\_name** on top of the input symbol and double-click the mouse. A dialogue box will appear. Type the pin name, let's say **x1**, and click OK. Similarly, assign the name **x2** to the other input and **f** to the output pin. It is possible to change the name of an element by selecting it first, and then double-clicking on the name and typing a new name.
6. Finally **Connecting Nodes with Wires**: Click on the  icon in the toolbar to activate the Orthogonal Node Tool. Position the mouse pointer over the right edge of one of your input pins. Click and hold the mouse button and drag the mouse to the right until the drawn line reaches the pinstub at the input of one of the gates.
7. Follow the compilation, simulation, implementation, and validation steps as described in sections 1.4.2.4 - 1.4.2.10.
8. **Demonstrate the bakery profit calculator simulation and hardware validation results to your lab instructor.**

#### 2.4.4.3 MODELSIM VERILOG CODE AND SIMULATION

1. Refer to MODELSIM™ tutorial for Verilog HDL coding and simulation.
2. Write a Verilog HDL code using any one of the hardware description methods you studied in prelab, based on your preference: Structural, behavioural continuous or behavioural procedural.
3. Compile and Simulate your design.
4. **Demonstrate your simulations to your lab instructor.**

#### 2.5 LIST OF DELIVERABLES

##### 2.5.1 PREWORK

1. 1.2.2 Gates
  - Truth table and Boolean expression for all the gates
  - Logic Schematics for all the gates
  - Verilog code for all the gates
  - Testbench and ModelSim™ simulation screenshots for all the gates
  - Verilog code for the logic circuit in Figure 1.
  - Testbench and ModelSim™ simulation screenshots for the Logic Circuit.
  - Pin planner Screenshot
2. 1.2.3 Complex gates
  - Explain the function of each logic block (Table 1:1.2.3. Complex Gates)
  - Boolean expression for the functions MN-out and EVEN-out using AND and OR gates
  - Schematic implementation for the functions MN-out and EVEN-out using AND and OR gates
  - Use XNOR gates to create the schematic for EVEN-out
  - Verilog code for MN-out and EVEN-out
  - Testbench and ModelSim simulation screenshots of MN-out and EVEN-out
  - Pin planner Screenshot
3. 1.2.4 Bakery Profit Calculator
  - Truth table
  - KMaps and the Boolean expression of each output
  - Schematic of the circuit
  - Verilog Code of the circuit
  - Testbench and ModelSim simulation screenshots of the Circuit
  - Pin planner Screenshot

##### 2.5.2 EXPERIMENTAL

1. 1.2.2 Demonstrate and explain the schematic and simulation output of the gates.
2. 1.2.3 Demonstrate and explain the schematic and simulation output of MN-out and EVEN-out.
3. 1.2.4 Demonstrate and explain the schematic and ModelSim simulation output of the Bakery Profit Calculator

## 2.6 REFERENCES

- DE0 Board User Manual, Terasic Technologies Inc.
- Digital Design with An Introduction to the Verilog HDL, 5<sup>th</sup> Edition, M. Morris Mano & Michael D. Ciletti, Pearson
- Quartus II Introduction Using Schematic Design, ALTERA.