Lab #4: Introduction to Computer Aided Design
(Verilog Entry using Gate Level Primitives)

Revised 2_07_11

Overview
We have begun to utilize the BaSYS board. We have entered circuit data using schematic capture and found that to be ineffective for large circuits. Thus, we will now investigate entering function data using a hardware description language. The language of choice for us is Verilog.

Before beginning this module, you should...

• Be familiar with reading and constructing basic logic circuits;
• Understand logic equations, and how to implement a logic circuit from a logic equation;
• Know how to operate Windows computers and Windows programs.

After completing this module, you should...

• Understand how basic CAD tools are used in basic circuit design;
• Be able to enter functions using gate level primitives with Verilog.
• Be able to simulate any logic circuit using the Xilinx internal simulator;
• Be able to examine the output of a logic simulator to verify whether a given circuit has been designed correctly.

This module requires:

• A Windows PC
• The Xilinx ISE software
• A Digilent BaSYS board
• Keyboard

Contains material © Digilent, Inc
The Design Process

An idea for a new circuit design rarely proceeds directly from concept to flawless implementation. Rather, during the design phase, several potential circuits are considered, constructed, and evaluated. These prototype circuits are intended to give the designer greater insight into a circuit’s behaviors and characteristics before a final design is selected. These prototype circuits are intended to give the designer greater insight into a circuit’s behaviors and characteristics before a final design is selected. In the early days of digital design, prototype circuits were sketched on paper and then constructed from discrete components or simple integrated circuits (like the 7400 devices used in earlier labs). But over the last 30 years, the use of computer aided design (CAD) tools to specify and design digital circuits has made such methods obsolete. With the onset of the computer age, engineers learned they could be far more productive by designing a virtual circuit on a computer instead of actually building it. Design flow using CAD tool is shown on the following figure.
Design capture
CAD tools are useful throughout the engineering design process, and they benefit simple logic designs and complex system designs alike. In the early stages of a design, CAD tools allow designers to capture circuit definitions on a computer using any one of several different entry modes.

1. **HDL Design**: a “Hardware Definition Language” allows highly behavioral descriptions.
   - enable quick implementation
   - must be transformed to a structural representation before a circuit implementation since it contains no information about the structure of a circuit

2. **Schematic**: require highly structural designs.
   - yields a description that can be accurately simulated and directly implemented
   - all components and interconnections can take significant effort to create

A class of computer programs called *synthesizers* can transform the behavioral description to gateway level circuits, thereby freeing design engineers to focus on other design tasks. (In the lab, ‘synthesize’ function in the Xilinx ISE does this job) Although synthesizers use rules and assumptions that allow for a wide range of behavioral definitions, several studies have shown that they are nevertheless able to produce structural descriptions that are better than most engineers can produce. HDL usually offers gate level primitive for low level circuit design.

Test and Verify (or Circuit Simulation)
Constructing circuits from discrete components can be somewhat time consuming, and often are of limited value in providing insight into circuit performance. Yet it is difficult to gain confidence in a circuit’s performance without actually testing and measuring its various characteristics. Simulators allow engineers to experiment with a circuit design, and challenge it with a wide array of inputs and operating assumptions before undertaking the job of actually building it. Further, complex circuits like modern microprocessors use far too many components to assemble into a prototype circuit – they simply could not have been built without the heavy use of simulators. Simulators need two kinds of input – a description of the virtual circuit that includes all of the gates (or other components) and interconnections, and stimulus input describing how the circuit’s inputs are to be driven over time. (In our lab, the test bench waveform in Xilinx ISE program enables this type of test)

Implement and Test
Once the simulation is correct and all design requirements have been confirmed, the design can be implemented and verified in hardware. This process involves the use of various meters, oscilloscopes, and other test and measurement equipment. The intent of the verification process is to ensure that the design still meets specification after it has actually been constructed in its target hardware. Several problems, such as slow operation, electronic noise, or excessive power consumption may be encountered for the first time after the circuit is actually constructed, which requires modification of the circuit design. As we experience in past lab, FPGA programming significantly reduce the actual circuit implementation and testing.

Finally, most modern CAD tools have a top-level graphical interface called a framework from which all other required CAD tools can be launched. Such a framework is well illustrated in the Xilinx design tools used in this lab – all the steps that need to be taken from the beginning to the end of a design process are presented in outline form.

Contains material © Digilent, Inc.
Verilog

Brief background
The Verilog Hardware Description Language (HDL) is a language for describing the behavior and structure of electronic circuits, and is an IEEE standard (IEEE Std. 1364-1995). Verilog is used to simulate the functionality of digital electronic circuits at levels of abstraction ranging from stochastic and pure behavior down to gate and switch level, and is also used to synthesize (i.e. automatically generate) gate level descriptions from more behavioral descriptions. Verilog is commonly used to support the high level design (or language based design) process, in which an electronic design is verified by means of thorough simulation at a high level of abstraction before proceeding to detailed design using automatic synthesis tools. Verilog is also widely used for gate level verification of ICs, including simulation, fault simulation, and timing verification.

Simple example

```
module circuit1(X, Y, Z, F);

input wire X;
input wire Y;
input wire Z;
output wire F;

wire Z_not, X_and_Y;
not(Z_not, Z);
and(X_and_Y, X, Y);
or(F, X_and_Y, Z_not);
endmodule
```

- First line is comment line. ‘//’ or ‘/**’ can be used like c type language.
- ‘;’ means an end of statement and is required at end of every statement.
- The program begins with 'module' statement containing the name of module, here ‘circuit1’
  - Module name cannot start with number like ‘1bitadder’
  - Case sensitive
  - Cannot include space like ‘card checker.’ Use under-bar like ‘card_checker’
- A list of input and output list are followed to represent their types and directions.
  - Type: wire, reg (register)
  - Type wire is default and can be ignored (i.e. input (wire) X;)
  - Direction: input, output, inout (bi-directional signal)
- ‘not’, ‘and’, ‘or’ is gate level primitives that works as function.
  - not(Z_not, Z) : performs inverting operation on Z and store the result on Z_not
  - and(X_and_Y, X, Y) : performs ‘and’ operation on X and Y, and store the result on X_and_Y
- There is another approach (more descriptive) which will be covered later. We use this in this lab.
- Several Verilog reference manuals are linked on the lab webpage. However, the descriptions above are enough for this lab.
DIGITAL ROULETTE

A deck of cards has 52 cards with four sets (Hearts, Spades, Clubs, and Diamonds) and values of Ace through Ten, Jack, Queen, and King. We change the rule for this time. In digital roulette, the player will lose if he/she draws a TWO, THREE or QUEEN. Also, the player will win if he/she draws a SEVEN or JACK.

Implementation Detail

Assuming that the cards can be represented in binary, four bits are used to represent the cards. The number cards can be represented with their binary equivalent and the face cards can be represented as the decimal values (Jack => 11, Queen => 12, King => 13). These values can be encoded to the binary equivalent as well. Thus, using

- **TWO** = A'B'CD' (0010₂ = 2₁₀)
- **THREE** = A'B'CD (0011₂ = 3₁₀)
- **TWELVE** = ABC'D' (1100₂ = 12₁₀)
- **LOSE** = TWO + THREE + TWELVE

You might have question what is the relationship between left hand side and right hand side in the equation. We call this minterm notation canonical format. We want to have a WIN signal when we have 2, 3, and 12. Decimal value 2 can be represented in binary form, 0010, 3 in 0011, and 12 in 1100, respectively. These equations are written in minterm notation canonical format and can be minimized further. Minimizing the equations will reduce the amount of circuitry needed for the functions. Bonus problem will be given for the minimization of the functions.
Hierarchical Design

Can we reuse the module that we already in different module for convenience or to build a more complex system? The answer is YES! Verilog allows you to define and describe a 'module', which can then be included into other, higher-level designs. Using entities, it is possible to both hide complexity and to manage changes more effectively (modify a single instance of a component, effecting multiple uses of the component). Hierarchical design is a design made up of a collection of smaller pieces (similar to using subroutines in software development). By adopting hierarchical design approach it is possible to reuse common elements, and segment a complex design into smaller pieces. In this lab, we will make two Verilog modules which checks lose condition and win condition in separate files and use them in higher module which is card checking module. Next diagram shows the how hierarchical design is applied on our lab.

Card_checker module calls the win_check and lose_check module inside the modules. We use switches and LEDs to check the results. The switches defined in the user constraint file that we use, ‘BaSYS2_ECE3714.ucf’, are grouped together such as SW<7>, SW<6> ... SW<0> or SW [7:0] in Verilog notation. When a group of wires are used to transport data with the same name, it is called a bus (BOLD arrows). The wires are named with the same name, but given different numbers to indicate the wire, i.e.SW(2), SW(1), SW(0). For this lab, we will use the toggle switches (SW[3]-SW[0]) and LEDs (LD[2]-LD[1]). Signals generated by 4 switches(SW[3:0]) are passed into the card checker module, which maps 4 switches into common variables, A, B, C, and D. Each result (F and G) determined by the modules (win_check and lose_check) are passed to BaSYS board after mapped into LD[2] and [1].
Lab #4: Introduction to Verilog using Gate Level Primitives

Circuit Design (Verilog)

Specification (Three modules)

- lose_check: determine the output signal, LOSE, by the signal A, B, C, and D which makes 2, 3, or 12
  - Inputs: A, B, C, and D
  - Output: LOSE
- win_check: determine the output signal, WIN, by the signal A, B, C, and D which makes 7, or 11
  - Inputs: A, B, C, and D
  - Output: WIN
- card_checker: test two modules by using switches and LEDs of BaSYS board. This module turns on LD2 if WIN signal is asserted, or LD1 if LOSE signal is asserted.

lose_check module

1. Create a new project, i.e., Lab4 and select the top level source as HDL (not schematic)

2. Create a new source file, i.e. lose check, and choose Verilog module.
Lab #4: Introduction to Verilog using Gate Level Primitives

3. Define port name like following. Be careful on the direction.

4. Add ‘BaSYS2_ECE3714.ucf’ file at the ‘Add Existing Sources’ step.

5. Write a Verilog functions TWO, THREE, TWELVE, and LOSE referring to the ‘simple example.’
   a. Write functions inside module and endmodule
   b. Hint: Use four ‘not’ functions, three ‘and’ functions, and one ‘or’ function

6. After writing the code, synthesize it by clicking ‘Synthesize – XST’ menu on the process tab.

7. If errors are found, go to the module and fix them.
Lab #4: Introduction to Verilog using Gate Level Primitives

**win_check module**

1. Click ‘Create New Source’ on Process tab.

![Image of Process tab](image1)


![Image of Create New Source wizard](image2)

3. Define input and output port list as the same number 3 of ‘lose_check’ module except ‘LOSE’ signal.

4. Write a Verilog functions SEVEN, ELEVEN, and WIN referring to the ‘simple example.’
   
   a. Hint: four ‘not’ functions, two ‘and’ functions, and one ‘or’ function

5. After writing the code, synthesize it by clicking ‘Synthesize – XST’ menu on the process tab.

6. If errors are found, go to the module and fix them.
Lab #4: Introduction to Verilog using Gate Level Primitives

card_checker module (by Hierarchical design)

1. Click ‘Create New Source’ on Process tab.
2. Create ‘card_checker’ Verilog file.
3. Define input and output port list like following figure
   a. Check on ‘Bus’ and put numbers in MSB(most significant bit) and LSB

4. Right click the ‘card_checker’ on the Source tab, and choose the ‘Set As Top Module’ like following.
   a. (NOTE‼) In this way, you can designate which module is the major module which calls other sub modules.

Contains material © Digilent, Inc.
Lab #4: Introduction to Verilog using Gate Level Primitives

5. Write a Verilog code like following

```verilog
module card_checker(SW, LD);
    input [3:0] SW;
    output [2:1] LD;

    win_check W1(SW[3], SW[2], SW[1], SW[0], LD[2]);
    lose_check L1(SW[3], SW[2], SW[1], SW[0], LD[1]);
endmodule
```

a. win_check is module name where W1 is the name of this instantiation, which mean the module can be used multiple times as the need arises.
b. win_check is consist of 4 inputs and 1 output (A,B,C,D,WIN) so the appropriate switches and LED needs to put in the function call (inside parentheses).
c. Order matters‼ win_check W1(LD[2], SW[3], SW[2], SW[1], SW[0]) will cause an error because of the disorder of input and output position.
d. The original form of the function call
   - win_check W1(.A(SW[3]), .B(SW[2]), .C(SW[1]), .D(SW1[0]), .WIN(LD[2]))
   - lose_check L1(.A(SW[3]), .B(SW[2]), .C(SW[1]), .D(SW1[0]), .LOSE(LD[2]))
   - Can be used either original form or contracted form. Original form is safer, and contracted form is more efficient.

e. Don’t forget ; (semi colon) at the end of function call.

6. After writing the code, synthesize it by clicking ‘Synthesize – XST’ menu on the process tab.
7. If errors are found, go to the module and fix them.
Lab #4: Introduction to Verilog using Gate Level Primitives

Circuit Simulation for Verification

1. Create a new test bench waveform using some name, i.e. `lab4_test`, and associate with ‘check_card’ module.
2. Change the ‘Initial Length of the Test Bench’ to 1500ns.

3. Expand SW signal by clicking + sign on the left of SW[3:0]. Adjust the signal like following.
   a. Hint: if you right click on the gray regions of the top signal, you can choose different representation like unsigned decimal that helps you easily recognize the number. Here we have the number, 2, 3, 12, 7, 11, 4 to test.

4. Simulate the wave by using ‘Simulate Behavioral Model’ under the ‘Xilinx ISE Simulator’ option on the Process tab. Make sure you first click the test bench wave form file, i.e., `lab3_test.tbw`.
5. Look at the results signal, and check the signal. If not correct, check you code and fix it.
Configuring the FPGA and Test

Creating a Programming File

1. After done simulation, the program is ready to generate programming file (.bit). In this lab, we use keyboard to test the function.
   a. Download lab4.zip file from the ‘Lab file’ folder of the lab website, and extract the files.
   b. Add all files in the project by using ‘Add Existing Source’ on Process tab
   c. Set ‘card_checker_top’ as top module similar to the step 4 of ‘card_checker’ module.

2. **(Important)** In order to work with BaSYS board correctly, you need to remember two things in mind.
   a. Make sure you have a right constraint file (BaSYS2_ECE3714.ucf or BaSYS_ECE3714.ucf).
   b. Right Click on ‘Implementation’ on Process tab, and go to property menu. You will see the figure like following. Check ‘Allow Unmatched LOC constraints’ option. This option tells the hardware that there are some pins that we do not use so as to ignore it.

4. You can check Warnings or Errors on Transcript window of the Xilinx ISE program. If you have error, check your Schematic design and two steps above.
5. If there is no major error, you can find the ‘bit’ file on your working folder. The name follows you project name, i.e., card_checker_top.bit. Check it out.
Download a Programming File on FPGA chip

1. Connect the USB cable that comes with the Digilent BaSYS board kit to the computer and to the BaSYS board.
2. Connect the keyboard cable to the PS/2 port on the BaSYS board.
3. Turn on the power switch of the board.
4. Select a product in the ‘Connect’ in list box.
   
   ![Connect and Product Selection](image)

5. Choose the program (card_checker_top.bit) by using ‘Browse’ button either of FPGA and PROM, which are two options provided by BaSYS board
   a. FPGA: Direct configuration from the computer. Fast configuration but volatile that means the program is erased from the memory
   b. PROM: Indirect configuration from the flash memory in the board. Slow configuration but the program is stored on the board so that it will run the stored program when you connect the power.
6. Check the results on the board. The keyboard setting can be found on the ‘scancode_converter’ module.
   a. Check ‘7’, ‘J’ keys turn on LD2(WIN), and ‘2’, ‘3’, ‘Q’ on LD1(LOSE).

![Code Converter](image)

7. If the results are not correct, check you schematic and compile it again.
8. Check off your results with TA. If you cannot finish this within the time, you can take a picture of the board with your student ID and include them in the lab report.

Minimizing Equation to Use Less Circuits (20 points)

Minimize the functions of the WIN and LOSE signal which work the same, but with a smaller number of circuits. Modify the modules, lose_check and win_check, to have the minimized equations. Describe how much you can minimize the circuit while keeping same results. Show the results on the testbench waveform. Make sure you set the ‘card_checker’ as top module to generate testbench waveform.