Contents

1 Introduction
   1.1 Pre-laboratory preparations
   1.2 Safety in the laboratory
   1.3 Laboratory ground rules
   1.4 Lab assessment
   1.5 Laboratory notebook
   1.6 General laboratory report format

2 Experiment 1
   2.1 Binary Numbers
   2.2 Binary Coded Decimal (BCD) Numbers

3 Experiment 2
   3.1 AND Logic Gates
   3.2 OR Logic Gates

4 Experiment 3
   4.1 NOT Logic Gates
   4.2 NOR Logic Gates
   4.3 Ex-OR Logic Gates

5 Experiment 4
   5.1 Sum of Product Expressions
   5.2 Sum of Product (Sum of Product (SOP)) Expressions
   5.3 Product of Sum Expressions
   5.4 Product of Sum (Product of Sum (POS)) Expressions

6 Experiment 5
   6.1 Basic Combinational Logic

7 Experiment 6
   7.1 Full Adder
   7.2 Full Subtractor

8 Experiment 7
   8.1 Verilog HDL and Xilinx ISE
   8.2 Logic Gates Implementation

9 Experiment 8
   9.1 Full Adder Implementation on Field Programmable Gate Array (FPGA)
   9.2 2×4 Decoder Implementation on FPGA

10 Experiment 9
   10.1 4×1 Multiplexer Implementation on FPGA
   10.2 D-Flip Flop Implementation on FPGA

11 Experiment 10
   11.1 Counter Implementation on FPGA

12 Experiment 11
   12.1 Sawtooth Generator Using Digital To Analog Converter (DAC) and FPGA
13 Experiment 12

13.1 Sine Wave Generator Using DAC and FPGA

14 Acronyms
1 Introduction

This manual has been prepared to serve as a laboratory manual for EE-172/CS 130 Digital Logic Design course for electrical engineering students. The manual consists of a set of experiments designed to allow students to build, and verify digital circuits and systems. This set of experiments cover relevant topics prescribed in the syllabus and are designed to reinforce the theoretical concepts taught in the classroom with practical experience in the lab. By the end of the course, students are expected to have a good understanding of digital logic design and implementations.

1.1 Pre-laboratory preparations

Every week before lab, each student should read over the laboratory experiment and work out the various calculations, etc. that are outlined in the so called Pre-Lab. The student should refer to the book: Thomas L. Floyd “Digital Fundamentals” tenth edition for the fundamental theory.

1.2 Safety in the laboratory

To minimize the risk of electric shock and other hazards, the experiments were designed for low-voltage. Despite this careful design one should never assume that electric circuits are safe, because a even only a few milli-amps of current through the body can be lethal. For your safety, you must follow safety rules:

- Turn off power before working on circuits.
- Make sure that the transformers and equipments are plugged into utility lines, and do not expose wires, if you see exposed wires, report them. Check with the instructor if you are not certain about the procedure.

1.3 Laboratory ground rules

Students are expected to comply with the rules and regulations stated in the following list:

- Return parts and jumper wires to correct bins when you are finished with them.
- Do not put suspected defective parts back in the bins. Give them to the Lab Under Graduate Research Assistant (URA) for testing or disposal.
- Report all equipment problems to Lab Instructor or Lab URA.
- Most experiments have several parts; students must alternate in doing these parts as they are expected to work in group.
- Each student must have a laboratory notebook. The notebook should be a permanent document that is maintained and witnessed properly, and that contains accurate records of all lab sessions.
- Laboratory and equipment maintenance is the responsibility of not only the Lab URA but also the students. A concerted effort to keep the equipment in excellent condition and the working environment well-organized will result in a productive and safe laboratory.

1.4 Lab assessment

Lab total = 100%

- Experiment 1 to 12 = 6% each. Total experiment weight = 12 × 6% = 72%.
- Lab test = 18%.
• Lab participation = 10%

Fine grain assessment of each lab:

• Pre lab = 1%
• Lab measurements (result tables) = 2%
• Lab report = 3%

1.5 Laboratory notebook

The laboratory notebook is a record of all work pertaining to the experiment. This record should be constructed such that another person, with similar technical background, can duplicate the experiment and obtain the same results by simply following the descriptions in your laboratory notebook. Record all activities and the results directly into the notebook during the experiment. Do not use scratch paper for recording data. Do not trust your memory to fill in the details at a later time.

Guidelines for the laboratory notebook

• Use the laboratory notebook for the Pre-Lab exercises.
• State the experiment objective.
• Draw the circuit diagram and indicate the values of resistances etc. which were used.
• Take a note of all the measuring instruments that were used during the experiment.
• Mention the equations used.
• Create tables to record the readings, including the units.
• Present your calculations in a need and systematic fashion. Do this is an organized manner.
• Attach appropriate graphs.
• Be concise.
• Mistakes should not be erased. Just bracket them and make a short note explaining the problem.
• Make entries as the lab progresses; don’t assume that you can fill it in later. The instructor will ask you to see these entries it during the lab.
• Date every page.
• Underline all important results.
• Attach simulation results and hand calculations to your note book.
• Draw figures using pencil before you come to the lab, so that you can make corrections to it in case you need to do so by erasing and redrawing. This will ensure tidy and neat work.
• Prepare the READING TABLE using pencil and ruler and not just by sketching lines. Sketching gives rise to crooked lines and gives the lab notebook a haphazard look.
• Take a few short notes (2-3 lines), which explain some of the problems you encountered while doing the experiment. This will help you write better reports.
1.6 General laboratory report format

Following the completion of each laboratory exercise in Digital Logic Design course, a report must be written and submitted for grading. The purpose of the report is to document the activities which led to both design and demonstration in the laboratory. The reports should be complete in the sense that all information required to reproduce the particular experiment is contained within. Writing useful reports is an essential part of becoming an engineer. In both academic and industrial environments, reports are the primary means of communication between engineers.

Typed Reports are required. Any drawings done by hand must be done with neatness, using a straight edge and drawing guides wherever possible. Free hand drawings will not be accepted. Prelab results should be reported in the provided sheets at the end of the manual. It is your responsibility to obtain the instructor’s signature and to include a signed sheet with your final experiment report. Each student must submit an individual report based on individual effort.

There is no one best format for all technical reports but there are a few simple rules concerning technical presentations which should be followed. Adapted to this laboratory they may be summarized in the following list:

- Title page;
- Introduction;
- Experimental procedure;
- Experimental data;
- Discussion;
- Conclusions.

A detailed descriptions of these items is provided in the following sections.

Title page

The title page should contain the following information:

- Your name;
- Student ID;
- Course number (including section);
- Experiment number and title;
- Date submitted;
- Instructor Name(s).

Introduction

It should contain a brief statement which indicates the objectives, or goals of the experiment. It should also help to guide the reader through the report by stating, for example, that experiments were done with three different circuits or consisted of two parts etc. or that additional calculations or data sheets can be found in the appendix, or at the end of the report.

Procedure

This section describes the experimental setup and how the measurements were made. Include here circuit schematics with the component values. Mention instruments used and describe any special measurement procedure that was used.
Results and questions

This section of the report should answer any questions presented in the lab handout. Any tables and/or circuit diagrams, that represent experiment results, should be referred to and discussed/explained with detail. All questions should be answered clearly in paragraph form. Any unanswered questions, from the lab handout, will result in loss of points on the report.

The best form of presentation of some of the data is graphical. In engineering presentations a figure is often worth more than a thousand words. There are some simple rules concerning graphs and figures which should always be followed. If there is more than one figure in the report, the figures should be numbered. Each figure must have a caption following the number. For example, “Figure 1.1: Logic gates”. In addition, this markup will greatly help you to learn how to use headers and figures in Microsoft Word.

Discussion

The discussion section is a critical part of the report, because it testifies that the report author, i.e. the student, understood the experiments and its purpose. This part of the report should compare the expected outcome of the experiment, such as derived from theory or computer simulation, with measured values. A prerequisite for such a comparison may be data analysis or manipulation.

When comparing experimental data with numbers obtained from theory or simulation, make it very clear which is which. A discrepancy between simulation results and measurement data does not necessarily mean that your experiment was a failure. The results will be accepted, provided that you can account for the discrepancy. Your ability to read scales may be one limitation. The value of some circuit components may not be well known and a nominal value given by the manufacturer does not always correspond to reality. However, very often the reason for differences between the expected and measured values lies in either experimental or theoretical procedure or in not taking into account all factors that enter into analysis.

Conclusion

A brief conclusion summarizing the work done, theory applied, and the results of the completed work should be included here. Data and analyses are not appropriate for the conclusion.
2 Experiment 1

2.1 Binary Numbers

Pre-Lab
1. What are binary numbers?
2. Why do we need binary numbers?
3. What are the applications of binary numbers?
4. What are BCD numbers?
5. Why do we need BCD numbers?
6. What are the applications of BCD numbers?

Objective
To demonstrate different encoding schemes. A counting number is represented with the binary format and with the BCD format.

Equipment
2. Precision Digital Multi Meter (DMM).
3. Integrated Circuit (IC) type 7493 4-bit ripple counter.
4. Light Emitting Diode (LED) Bar.
5. Function Generators.

Schematics
1. The Binary counter IC is shown in Figure 1.
2. Figure 2 shows the schematic diagram for this experiment.

Binary Count

Procedure
1. Turn off the power supply.
2. Connect the IC type 7493 as shown in Figure 2. Pin 14 is connected to Switch (SW).
3. Turn the power on and observe the four indicator LEDs. The 4-bit number in the out is incremented by one for every pulse generated by pushing the push button SW.
4. Disconnect the input of the counter at pin 14 from SW and connect it to the function generator.
5. Set the frequency selector to 1 Hz. This will provide an automatic binary up count.
6. Record the observations on Table 1.
Table 1: Data Table

<table>
<thead>
<tr>
<th>Count</th>
<th>( Q_A )</th>
<th>( Q_B )</th>
<th>( Q_C )</th>
<th>( Q_D )</th>
<th>Count</th>
<th>( Q_A )</th>
<th>( Q_B )</th>
<th>( Q_C )</th>
<th>( Q_D )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Questions
1. Explain the function of 7493 IC?
2. What is the applications of 7493 ICs?
3. What is a ring counter?
4. How can we convert binary numbers into BCD numbers?

2.2 BCD Numbers

Theory Overview

BCD counter is ten state sequence and does not have a regular pattern, unlike a straight binary counter. A single stage BCD counter such as the 7447 counts from decimal 0 to decimal 9 and it is capable of counting up to a maximum of nine pulses. Note also that a digital counter may count up or down, or count up and down (bidirectional), depending on an input control signal.

The BCD Count

BCD code consisting of four binary digits. The 7447 designation refers to the binary weight of the four digits or bits used. For example, $2^3 = 8$, $2^2 = 4$, $2^1 = 2$ and $2^0 = 1$. The main advantage of BCD code is that it allows for the easy conversion between decimal and binary forms of numbers.

Equipment
1. Adjustable DC Power Supply.
2. Precision DMM
3. IC type 7447 BCD counter.
4. LED Bar.
5. Push buttons.

Schematics
1. The BCD counter IC is shown in Figure 3.
2. Figure 4 shows the schematic diagram for this experiment.
Procedure

1. Turn off the power supply.

2. Connect the IC type 7447 as shown in Figure 4.

3. Turn the power on and observe the LED bar indicator. The sequence of LEDs are changed by every input generated by pressing the buttons (A, B, C, D) following the sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 in binary value.

4. Record the observations on Table 2.

Questions

1. Give the binary and BCD code representation of the following numbers,
   a) 5
   b) 9
c) 10  
d) 12  
e) 14

2. Solve the following expressions
   a) \(101010)_{10} + 4)_{10} = ?)_{2},\)
   b) \(8)_{10} - 3)_{10} = ?)_{2}.\)
   c) \(01001101)_{2} + 1001001011)_{2} = ?)_{2}\)
3 Experiment 2

Pre-Lab

1. What is a logic gate?
2. What are basic gates?
3. What are universal gates?
4. What is the primary motivation for using Boolean algebra to simplify logic expressions?

3.1 AND Logic Gates

Objective

This lab is an introduction to digital electronics. This includes nomenclature of digital ICs, specifications, study of the data sheet, concept of $V_{cc}$ and ground, verification of the truth tables of logic gates using Transistor Transistor Logic (TTL) Integrated Circuits (ICs). Boolean functions can be practically implemented by using electronic gates. The following points are important to understand.

- Electronic gates require a power supply.
- Gate inputs are driven by voltages having two nominal values, e.g. 0 V and 5V representing logic 0 and logic 1 respectively.
- The output of a gate provides two nominal values of voltage only, e.g. 0 V and 5 V representing logic 0 and logic 1 respectively. In general, there is only one output to a logic gate except in some special cases.
- There is always a time delay (in nano-seconds) between an input being applied and the output responding.

Theory Overview

The AND logic gate is a basic digital logic gate that implements logical conjunction, it behaves according to the truth table given in Table 3. A HIGH output (logic 1) results only if both the inputs to the AND gate are HIGH. If neither or only one input to the AND gate is HIGH, a LOW (logic 0) output results. In other words, the AND function effectively finds the minimum between two binary digits. Therefore, the output is always logic 0 except when all the inputs are HIGH.

The logic symbol or Boolean expression for an AND gate is the multiply sign ($\cdot$). It describes the AND operation on two inputs. In the expression below, $A$ and $B$ are the inputs and $Y$ is the output.

$$Y = A \cdot B$$

Equipment

1. Adjustable DC Power supply.
2. DMM
3. IC type 7408 AND gate.
4. LED and switches.
1. The truth table for an AND gate with two inputs is given in Table 3.

2. Schematic symbol of AND logic gate IC is shown in Figure 5.

3. Figure 6 shows the schematic diagram for this experiment.

**Schematics**

1. The truth table for an AND gate with two inputs is given in Table 3.

2. Schematic symbol of AND logic gate IC is shown in Figure 5.

3. Figure 6 shows the schematic diagram for this experiment.

**Procedure**

1. Place the breadboard gently on the observation table.
2. Fix the IC which is under observation on the half separator line of breadboard, so there are no shorts.

3. Connect a wire to the main voltage source ($V_{cc} = 5 \text{ V}$) whose other end is connected to last pin of the IC (14th pin from the notch).

4. Connect the ground pin of the IC (7th pin from the notch) to the ground terminal provided by DC power supply.

5. Give the input at any one gate of the IC i.e. 1st, 2nd, 3rd or 4th gate by using connecting wires. (In accordance to the IC provided).

6. Connect output pins to an LED.

7. Switch on the power supply.

8. If LED glows then output is true, if it does not glow, the output is false, which is numerically denoted as logic 1 and logic 0 respectively. They should test its truth table provided in Table 3.

9. Complete Table 4 with two inputs.

10. Complete Table 5 with three inputs.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Questions
1. What is the purpose of a truth table and algebraic functions?
2. If we were to build a truth table for a 16-input AND gate, how many different combinations of inputs would we have? Explain.
3. How is a logic probe used to troubleshoot digital ICs?

3.2 OR Logic Gates

Objective

Theory Overview
The OR logic gate which is also known as Inclusive-OR gate, is a type of digital logic gate that has an output which is normally at (logic 0) and only goes HIGH to a (logic 1) when one or more of its inputs are at (logic 1). The output, Y of the OR gate only returns LOW (logic 0) again when ALL of its inputs are at a (logic 0). In other words for a logic OR gate, any HIGH (logic 1) input will give a HIGH (logic 1) output.

The logic symbol or Boolean expression for an OR gate is the add sign (+). It describes the OR operation on two inputs. In the expression below, A and B are the inputs and Y is the output.

\[ Y = A + B \] (2)

Equipment
1. Adjustable DC Power supply.
2. DMM.
3. IC type 7432 OR gate.
4. LEDs and switches.

Schematics
1. The truth table for an OR gate with two inputs is given in Table 6.
2. The schematic symbol of an OR logic gate IC is shown in Figure 7.
3. Figure 8 shows the schematic diagrams for this experiment.
Table 6: Truth table for the OR Gate

<table>
<thead>
<tr>
<th>Input (A)</th>
<th>Input (B)</th>
<th>Output (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Procedure

1. Place the breadboard gently on the observation table.
2. Fix the IC which is under observation on the half separator line of breadboard, so that there are no shorts.
3. Connect the wire to the main voltage source ($V_{cc} = 5$ V) whose other end is connected to last pin of the IC (14th pin from the notch).
4. Connect a ground of the IC (7th pin from the notch) to the ground terminal provided by the DC power supply.
5. Provide input to any one of the gates include the ICs i.e. 1st, 2nd, 3rd, 4th gate by using connecting wires. (In accordance to the IC provided).
6. Connect output pins to a LED.
7. Switch on the power supply.
8. If the LED glows then the output is true, if it does not glow then output is false, which is numerically denoted as logic 1 and logic 0 respectively. They should test its truth table shown in Table 6.
9. Complete Table 7 with two inputs.
10. Complete Table 8 with three inputs.
Table 7: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions

1. Sensors are used to monitor both pressure and temperature of a chemical solution stored in a large tank. The circuitry for each sensor produces a HIGH voltage when a specific threshold value is exceeded. An alarm is requiring a LOW voltage input must be activated when either the pressure or the temperature is excessive.
   
a) Produce the system truth table.
b) Design a circuit for this application.

2. What is the purpose of an inverter in a digital circuit?
4 Experiment 3

Pre-Lab

1. What is the purpose of using NOT gates?
2. What is the difference between NOR and Ex-OR gates?
3. What are the applications of Ex-OR gates?
4. Provide a truth table for an Ex-NOR gate with three inputs?

4.1 NOT Logic Gates

Objective

This lab is an introduction to digital electronics. This includes nomenclature of digital ICs, specifications, study of the data sheet, concept of \( V_{cc} \) and ground, verification of the truth tables of logic gates using TTL Integrated Circuits (ICs). Boolean functions can be implemented with electronic gates. The following points are important.

- Electronic gates require a power supply.
- Gate inputs are driven by voltages having two nominal values, e.g. for TTL 0 V and 5V representing logic 0 and logic 1 respectively.
- The output of a gate provides two nominal values of voltage only, e.g. 0 V and 5 V representing logic 0 and logic 1 respectively. In general, there is only one output to a logic gate except in some special cases.
- There is always a time delay (in nano-seconds) between an input being applied and the output responding.

Theory Overview

A NOT gate (also called Inverter) is an other type of logic gate. It takes one input signal and produces an inverted output signal. In logic, there are usually two states, logic 0 and logic 1. The NOT gate sends logic 1 as output, if it receives logic 0 as input. Alternatively it received logic 1 as input, and sends logic 0 as output. Generally, below 0.5 V is logic 0, and 4 V to 5 V is logic 1. Several inverters are packaged in an integrated circuit. The mathematical expression is

\[
Y = \overline{A} \tag{3}
\]

Equipment

1. Adjustable DC Power Supply.
2. DMM.
3. IC type 7404 NOT gate.
4. LEDs and switches.

Schematics

1. Table 9 shows the truth table for a NOT gate with an input.
2. Figure 9 shows the schematic symbol of a NOT logic gate.
3. Figure 10 shows the schematic diagrams for this experiment.
Procedure

1. Place the breadboard gently on the observation table.

2. Fix the [IC] which is under observation on the half separator line of breadboard, so there are no shorts.

3. Connect a wire to the main voltage source ($V_{cc} = 5\, V$) whose other end is connected to last pin of the [IC] (14th pin from the notch).

4. Connect the ground pin of the [IC] (7th pin from the notch) to the ground terminal provided by DC power supply.

5. Give the input at any one gate of the [IC] i.e. 1st, 2nd, 3rd or 4th gate by using connecting wires. (In accordance to the [IC] provided).

6. Connect output pins to an LED.

7. Switch on the power supply.

8. If the [LED] glows then output is true, if it does not glow, the output is false, which is mathematically denoted as logic 1 and logic 0 respectively. Verify the truth table shown in Table 9.

9. Complete Table 10.
Table 9: Truth table for the NOT Gate

<table>
<thead>
<tr>
<th>Input (A)</th>
<th>Output (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 10: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>Measured Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions

1. Identify the type of gate from the following equation,

\[ Z = \overline{A} \cdot B + A \cdot \overline{B} \]

2. Provide an implementation of a NOT gate using NAND gates.

4.2 NOR Logic Gates

Objective

This includes nomenclature of digital ICs, specifications, study of the data sheet, concept of \( V_{cc} \) and ground, verification of the truth tables of logic gates using TTL Integrated Circuits (ICs). Boolean functions can be practically implemented by using electronic gates.

Theory Overview

The logic NOR gate or Inclusive-NOR gate is a combination of the digital logic OR gate with that of an inverter or NOT gate connected together in series. The NOR (Not OR) gate has an output that is normally at logic 0 and only goes logic 0 to logic 1, when any of its inputs are at logic 0. The Logic NOR Gate is the reverse or logic 1 form of the OR gate we have seen in Experiment 2.

The Boolean expression for a logic NOR gate is that for logical multiplication. It performs on the complements of the inputs. The Boolean expression for a logic NOR gate is denoted by a plus sign, \(( + )\) with an over line on the expression to signify the NOT or logical negation of the NOR gate. Combining the two operations results in

\[ Y = \overline{A + B} \]  \hspace{1cm} (4)

Equipment

1. Adjustable DC Power Supply.
2. DMM
3. IC type 7402 NOR gate.
4. LEDs and switches.
Table 11: Truth table for the NOR Gate

<table>
<thead>
<tr>
<th>Input (A)</th>
<th>Input (B)</th>
<th>Output (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 11: 7402 - NOR logic gate IC

Figure 12: Schematic

Schematics
1. Table 11 shows the truth table for a NOR gate with two inputs is given in
2. Figure 11 shows the schematic symbol of NOR logic gate IC
3. Figure 12 shows the schematic diagrams for this experiment.

Procedure
1. Place the breadboard gently on the observation table.
Table 12: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

2. Fix the IC which is under observation between the half separator line of breadboard, so there is no shortage of voltage.

3. Connect the wire to the main voltage source (Vcc) whose other end is connected to last pin of the IC (14th place from the notch).

4. Connect the ground of IC (7th place from the notch) to the ground terminal provided by DC supply.

5. Give the input at any one of the gate of the IC: i.e. 1th, 2th, 3th, 4th gate by using connecting wires. (In accordance to IC provided).

6. Connect output pins to the LEDs.

7. Switch on the power supply.

8. If the LED glows then output is true, if it not glows, output is false, which is numerically denoted as 1 and 0 respectively. Verify the truth table shown in Table 11.

9. Complete Table 12 with two inputs.

10. Complete Table 13 with three inputs.

Questions

1. An electronic system will only operate if three switches P, S and T are correctly set. An output signal (X = 1) will occur if R and S are both in the ON position or if R is in the OFF position and S and T are both in the ON position. Design a logic circuit to represent the above situation and also draw the truth table.

2. Provide an implementation of a NOR gate using NAND gates.

4.3 Ex-OR Logic Gates

Theory Overview

An Ex-OR gate (Exclusive OR gate) is a digital logic gate with two or more inputs and one output that performs exclusive disjunction. The output of an Ex-OR gate is true only when exactly one of its inputs is true. If both of an Ex-OR gate’s inputs are false, or if both of its inputs are true, then the output of the Ex-OR gate is false.

If an Ex-OR gate has more than two inputs, then its behavior depends on its implementation. In the vast majority of cases, an Ex-OR gate will output true if an odd number of its inputs is true. However, it’s important to note that this behavior differs from the strict definition of exclusive or, which insists that exactly one input must be true for the output to be true. Mathematically, an Ex-OR gate is modeled as

\[ Y = A \oplus B \]  
(5)
Table 13: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 14: Truth table for the Ex-OR Gate

<table>
<thead>
<tr>
<th>Input (A)</th>
<th>Input (B)</th>
<th>Output (Y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Equipment
1. Adjustable DC Power Supply.
2. DMM.
3. IC type 7486 Ex-OR gate.
4. LEDs and switches.

Schematics
1. Table 14 shows the truth table for an Ex-OR gate with two inputs.
2. Figure 13 shows the schematic symbol of Ex-OR logic gate IC.
3. Figure 14 shows the schematic diagram for this experiment.

Procedure
1. Place the breadboard gently on the observation table.
2. Fix the IC which is under observation between the half separator line of breadboard, so there is no shortage of voltage.

3. Connect the wire to the main voltage source ($V_{cc}$) whose other end is connected to last pin of the IC (14th place from the notch).

4. Connect the ground of IC (7th place from the notch) to the ground terminal provided by DC supply.

5. Give the input at any one of the gate of the ICs i.e. 1th, 2th, 3th, 4th gate by using connecting wires. (In accordance to IC provided).

6. Connect output pins to the LEDs.

7. Switch on the power supply.

8. If LED glows then output is true, if it not glows, output is false, which is numerically denoted as 1 and 0 respectively. Verify the truth table shown in Table 14.

9. Complete Table 15 with two inputs.

10. Complete Table 16 with three inputs.
Table 15: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 16: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions

1. Provide a complete expression with schematic of Ex-OR gate.
2. Provide a complete expression with schematic of Ex-NOR gate.
3. Provide an implementation of a Ex-OR gate using NAND gates.
5 Experiment 4

Pre-Lab

- Simplify the following expressions:
  1. \( X = C + B \cdot C \)
  2. \( Y = (A + C)(A \cdot D + A\overline{D}) + A \cdot C + C \)
  3. \( F = A + (\overline{B} \cdot C \cdot D) \)

- Construct schematic of the following expressions:
  1. \( P = C \cdot (A \cdot \overline{D} + A \cdot D) + A \cdot B \)
  2. \( Q = \overline{A} \cdot B \cdot C + \overline{A} \cdot B \cdot \overline{C} + \overline{B} \cdot D + B \cdot C \cdot D \)

- What are Application Specific Integrated Circuits (ASICs)?

- Simplify the following expressions
  1. \( X = C + B \cdot C \)
  2. \( Y = (A + C)(A \cdot D + A\overline{D}) + A \cdot C + C \)
  3. \( F = A + (\overline{B} \cdot C \cdot D) \)

- Construct schematic of the following expressions
  1. \( P = C \cdot (A \cdot \overline{D} + A \cdot D) + A \cdot B \)
  2. \( Q = \overline{A} \cdot B \cdot C + \overline{A} \cdot B \cdot \overline{C} + \overline{B} \cdot D + B \cdot C \cdot D \)

- What are Application Specific Integrated Circuits (ASICs)?

5.1 Sum of Product Expressions

5.2 Sum of Product (SOP) Expressions

Objective

1. Study the operation of basic logic gates.
2. Obtain Boolean expressions from a logic circuit.
3. Build a logic circuit from Boolean expressions.
4. Simplify Boolean expressions using Boolean Algebra theorems and postulates.
5. Obtain truth tables and compute circuit cost for logic circuits.

Theory Overview

The three basic logic gates are AND, OR, and NOT. These logic gates are the building blocks of all digital circuits. Other logic gates such as NAND, NOR, XOR, XNOR are derived from the three basic logic gates.

An equation in SOP form can be expressed as:

\[
X = \overline{A} \cdot B + \overline{A} \cdot B \cdot \overline{C} + A \cdot C
\]
### Table 17: Postulates of Boolean Algebra

<table>
<thead>
<tr>
<th>Postulate</th>
</tr>
</thead>
<tbody>
<tr>
<td>X + 0 = X</td>
</tr>
<tr>
<td>X + X = 1</td>
</tr>
<tr>
<td>X + 1 = 1</td>
</tr>
<tr>
<td>X + Y = Y + X</td>
</tr>
<tr>
<td>X + (Y + Z) = (X + Y) + Z</td>
</tr>
<tr>
<td>X · (Y + Z) = X · Y + X · Z</td>
</tr>
<tr>
<td>X + Y = X · Y</td>
</tr>
<tr>
<td>X + (Y · Y) = X</td>
</tr>
</tbody>
</table>

**Commutative**

**Associative**

**Distributive**

**DeMorgan**

**Absorption**

![Figure 15: Schematic](image)

**Equipment**

1. Adjustable **DC** power supply.
2. Digital Multimeter **DMM**
3. **IC** type 7408 AND gate, 7432 OR gate, 7404 NOT gate.
4. **LEDs** and switches.

**Schematic**

1. Figure 15 shows the schematic diagram for this experiment.
Table 18: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Expected X</th>
<th>Measured X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Procedure**

1. Implement the logic diagram provided in Figure 15 on the breadboard.
   - Connect inputs A, B and C to slide switches.
   - Connect output X to an **LED**.

2. Flip the switches On / Off and check the output for all 8 possible combinations of inputs A, B and C.

3. Fix the **ICs** that are required on the half separator line of breadboard, so there is no shortage of voltage.

4. Connect the wire to the main voltage source \(V_{cc} = 5 \text{ V}\) whose other end is connected to last pin of the **IC** (14\(^{th}\) place from the notch).

5. Connect the ground of **IC** (7\(^{th}\) place from the notch) to the ground terminal provided by **DC** power supply.

6. Switch on the power supply.

7. If the **LED** light up then output is true, if it stays dark then output is false, which is numerically denoted as logic 1 and logic 0 respectively.

8. Complete the observation Table 18.

**Questions**

1. Simplify the following functions using Boolean rules.
   a) \( L = A \cdot (A + B) + \overline{A} \cdot \overline{A} \cdot \overline{B} \)
   b) \( M = (A + A \cdot B \cdot C + A) \cdot B + \overline{A} \cdot B \cdot C \)

2. Construct schematic of following expressions.
   a) \( F = A \cdot \overline{B} \cdot \overline{C} + A \cdot \overline{B} \cdot C + \overline{A} \cdot B \cdot C + \overline{A} + \overline{B} \cdot C \)
   b) \( G = (E \cdot P + \overline{E} \cdot N)(P \cdot \overline{E} + E \cdot \overline{N}) + P \cdot \overline{N} \)

3. What are propagation gate delays in digital logic **ICs**?
5.3 Product of Sum Expressions

5.4 Product of Sum (POS) Expressions

Theory Overview

The three basic logic gates are AND, OR, and NOT. These logic gates are the building blocks of all digital circuits. Other logic gates such as NAND, NOR, XOR, XNOR are derived from the three basic logic gates. An equation in POS form can me express as

\[ Y = (\overline{A} + B) \cdot (A + B + C) \]  \hspace{1cm} (6)

Equipment

1. Adjustable DC power supply.
2. Digital Multimeter DMM.
3. IC type 7408 AND gate, 7432 OR gate, 7404 NOT gate.
4. LEDs and switches.

Schematics

1. Figure 16 shows the schematic diagrams for this experiment.

Procedure

1. Implement the logic diagram provided in Figure 16 on the breadboard.
   - Connect inputs A, B and C to slide switches.
   - Connect output X to an LED.
2. Flip the switches On / Off and check the output for all 8 possible combinations of inputs A, B and C.
Table 19: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Fix the [IC], which is under observation between the half separator line of breadboard, so there is no shortage of voltage.

4. Connect the wire to the main voltage source \((V_{cc} = 5 \text{ V})\) whose other end is connected to last pin of the [IC] (14th place from the notch).

5. Connect the ground of [IC] (7th place from the notch) to the ground terminal provided by DC supply.

6. Switch on the power supply.

7. If LED light up then output is true and if it stays dark then output is false, which is numerically denoted as logic 1 and logic 0 respectively.

8. Complete the observation Table [19].

Questions

1. Simplify the following functions using Boolean simplification techniques.
   a) \(X = A \cdot (A + B)\)
   b) \(Y = A \cdot B + A \cdot \overline{B} + \overline{A} \cdot B\)
   c) \(Z = (A + \overline{C}) \cdot (A + D) \cdot (B + \overline{C}) \cdot (B + D)\)

2. Construct schematic of following expressions.
   a) \(P = (W \cdot X + W \cdot \overline{Y}) \cdot (X + W) + W \cdot X \cdot (X + \overline{Y})\)
   b) \(Q = A \cdot \overline{B} \cdot C \cdot (A \cdot C + A \cdot \overline{B} \cdot C + A \cdot B)\)

3. What is the fan-out for a digital logic [IC]?
6 Experiment 5

Pre-lab

1. What is the difference between combinational circuits and sequential circuits?
2. Describe some applications of combinational circuits.
3. Complete the expected value column of Table 20. Hint: Refer to Figure 17
4. Complete the expected value column of Table 21. Hint: Refer to Figure 18

6.1 Basic Combinational Logic

Objective

1. Design a combinational logic circuit for a given Boolean output expression.
2. Generate the truth tables for the given combinational circuits.
3. Simplify a combinational logic circuit to its minimum form.
4. Troubleshoot the logic circuits.
5. Apply combinational logic to a system application.

Theory Overview

In digital circuit theory, combinational logic (sometimes also referred to as time-independent logic) is a type of digital logic which is implemented by Boolean circuits, where the output is a pure function of the present input only. This is in contrast to sequential logic, in which the output depends not only on the present input but also on the history of the input. In other words, sequential logic has memory while combinational logic does not.

Combinational logic is used in computer circuits to perform Boolean algebra on input signals and on stored data. Practical computer circuits contain a mixture of combinational and sequential logic. For example, the part of an Arithmetic Logic Unit (ALU), that does mathematical calculations is constructed using combinational logic. Other circuits used in computers, such as half adders, full adders, half subtracters, full subtracters, multiplexers, demultiplexers, encoders and decoders are also made by using combinational logic. Lets consider we have following functions

\[ X = A \cdot B + C \cdot D \cdot E \]

\[ Y = \overline{A} \cdot B \cdot C + \overline{A} \cdot B \cdot \overline{C} \]

Equipment

1. Adjustable DC Power Supply.
2. Digital Multimeter DMM.
3. IC type 7408 AND gate, 7432 OR gate, 7404 NOT gate.
4. LEDs and switches.

Schematics

1. Figure 17 and Figure 18 show the schematic diagrams for this experiment.
Procedure

1. Implement the logic diagram shown in Figure 17 on the breadboard.
   - Connect the inputs A, B, C, D, E to slide switches.
   - Connect the output X to an LED.
2. Flip the switches On / Off and check the output for all 32 possible combinations of inputs A, B, C, D and E.
3. Fix the ICs that are required on the half separator line of breadboard, so there are no shorts.
4. Connect the wire to the main voltage source ($V_{cc} = 5$ V) whose other end is connected to last pin of the IC (14th place from the notch).
5. Connect the ground of IC (7th place from the notch) to the ground terminal provided by DC power supply.
6. Switch on the power supply.
7. If the LED lights up then output is true, if it stays dark then output is false, which is numerically denoted as logic 1 and logic 0 respectively.
8. Complete the observation Table 20.
9. Similarly implement the logic diagram provided by Figure 21 on the breadboard.
   - Connect the inputs A, B and C to slide switches.
   - Connect the output Y to an LED.
10. Flip the switches On / Off and check the output for all 8 possible input combinations A, B and C.
11. If the LED lights up then output is true, if it stays dark then the output is false, which is numerically denoted as logic 1 and logic 0 respectively.
12. Complete the observation Table 21.
Table 20: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Expected X</th>
<th>Measured X</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Expected X</th>
<th>Measured X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 18: Schematic
Table 21: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Expected Y</th>
<th>Measured Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions

Provide the schematic diagram of following expressions:

1. \( F (A,B,C) = (A+B) \cdot (A+\overline{B}+C) \cdot \overline{B} \)
2. \( F (A,B,C,D,E) = \overline{A} \cdot \overline{B} + A \cdot B \cdot C + \overline{A} \cdot \overline{C} \cdot \overline{D} \cdot \overline{E} + A \cdot B \cdot E \)
3. \( F (X_1,X_2) = (X_1+X_2) \cdot (\overline{X_1} + X_1 \cdot X_2) + (\overline{X_2}+X_1 \cdot \overline{X_2}) \)
4. \( F (A,B,C) = \sum (1,2,4,7) \)
5. \( F (A,B,C) = \Pi (0,1,2,7) \)
7 Experiment 6

Pre-Lab

1. What are the differences between a full adder and a half adder?

2. Is it possible to make a full adder using two half adders? If yes then provide a full adder schematic using two half adder circuits?

3. What are the applications of full adders?

4. What are the differences between a full subtracter and a half subtracter?

5. What is the purpose of ‘borrow‘ in a full subtracter?

6. Is it possible to make a full subtracter using two half subtracter? If yes then provide a full subtracter schematic using two half subtracter circuits?

7. What are the applications of full subtracter?

7.1 Full Adder

Objective

Arithmetic circuits such as the digital adders and abstractors are the example of simple combinational logic. This lab demonstrates how to build a digital adder and abstractor using logic gates.

Theory Overview

The most basic arithmetic operation is the addition of two binary digits. There are four possible elementary operations, namely,

\[
\begin{align*}
0 + 0 &= 0 \\
0 + 1 &= 1 \\
1 + 0 &= 1 \\
1 + 1 &= 10_2 \text{ with 1 Carry}
\end{align*}
\]

The first three operations produce a sum of whose length is one digit, but when the last operation is performed the sum is two digits. The higher significant bit of this result is called a carry and lower significant bit is called the sum.

A combinational circuit which performs the arithmetic sum of three input bits is called full adder. The three input bits include two significant bits and a previous carry bit. A full adder circuit can be implemented with two half adders and one OR gate. Equation for a full adder can be expressed as:

\[
\text{Sum} = A \oplus B \oplus C_{\text{in}}
\]

\[
\text{Carry} = (A \oplus B) \cdot C_{\text{in}} + A \cdot B
\]

Equipment

1. Adjustable DC power supply.
2. Digital Multimeter DMM.
3. IC type 7408 AND gate, 7432 OR gate, 7486 Ex-OR gate.
4. Two LED and three slide switches.
Schematic

1. Figure 19 shows the schematic diagrams for this experiment.

Procedure

1. Implement the logic diagram provided by Figure 19 on the breadboard.
   - Connect inputs A, B and C\textsubscript{in} to slide switches.
   - Connect output Sum and Carry to LEDs.
2. Flip the switches On / Off and check the output for all 8 possible combinations of inputs A, B and C\textsubscript{in}.
3. Fix the ICs that are required on the half separator line of breadboard, so there is no shorts.
4. Connect the wire to the main voltage source (V\textsubscript{cc} = 5 V) whose other end is connected to last pin of the IC(14\textsuperscript{th} place from the notch).
5. Connect the ground of IC(7\textsuperscript{th} place from the notch) to the ground terminal provided by DC power supply.
6. Switch on the power supply.
7. If the LED light up then output is true, if it stays dark then output is false, which is numerically denoted as logic 1 and logic 0 respectively.
8. Complete the observation Table 22.

Questions

1. Using only full adders, construct an eight-bit adder. Each full adder has two single-bit inputs with a carry\textsubscript{in} and one bit sum and one bit carry\textsubscript{out} outputs.
2. How many full adders are require to add following numbers and what will be the binary outputs?
   a) 101101\textsubscript{2} + 110100\textsubscript{2}
Table 22: Data Table

<table>
<thead>
<tr>
<th>C&lt;sub&gt;in&lt;/sub&gt;</th>
<th>A</th>
<th>B</th>
<th>Sum</th>
<th>Carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b) a<sub>hex</sub> + 5<sub>oct</sub>
c) 64<sub>10</sub> + 48<sub>hex</sub>
d) cd<sub>hex</sub> + 5f<sub>hex</sub>

7.2 Full Subtractor

Theory Overview

The arithmetic operation, subtraction of two binary digits has four possible elementary operations namely,

- 0 - 0 = 0
- 0 - 1 = 1 with 1 borrow
- 1 - 0 = 1
- 1 - 1 = 0

In all operations, each subtrahend bit is subtracted from the minuend bit. In case of the second operation
the minuend bit is smaller than the subtrahend bit, hence 1 is borrowed.

A combinational circuit which performs the subtraction of three input bits is called full subtracter. The
three input bits include two significant bits and a previous borrow bit. A full subtracter circuit can be
implemented with two half subtracters and one OR gate. Equation for a full adder can be expressed as:

\[
\text{Difference} = A \oplus B \oplus \text{Borrow}_{in}
\]

\[
\text{Borrow}_{out} = (A \oplus B) \cdot \text{Borrow}_{in} + \overline{A} \cdot B
\]

Equipment

1. Adjustable DC power supply.
2. Digital Multimeter DMM
3. IC type 7408 AND gate, 7432 OR gate, 7404 NOT gate, 7486 Ex-Or gate.
4. Two LEDs and three slide switches.
Schematics

1. Figure 20 shows the schematic diagrams for this experiment.

Procedure

1. Implement the logic diagram provided in Figure 20 on the breadboard.
   - Connect inputs A, B and Borrow\textsubscript{in} to slide switches.
   - Connect output Difference and Borrow\textsubscript{out} to the LEDs.
2. Flip the switches On / Off and check the output for all 8 possible combinations of inputs A, B and Borrow\textsubscript{in}.
3. Fix the IC, which is under observation between the half separator line of breadboard, so there is no shorts.
4. Connect the wire to the main voltage source ($V_{cc} = 5$ V) whose other end is connected to last pin of the IC (14\textsuperscript{th} place from the notch).
5. Connect the ground of IC (7\textsuperscript{th} place from the notch) to the ground terminal provided by DC supply.
6. Switch on the power supply.
7. If LEDs light up then output is true and if it stays dark then output is false, which is numerically denoted as logic 1 and logic 0 respectively.
8. Complete the observation Table 23.

Questions

1. A full adder is a sequential circuit or a combinational circuit? Explain?
2. Sketch and describe the circuit of 4-bit carry ripple subtracter?
Table 23: Data Table

<table>
<thead>
<tr>
<th>Borrow&lt;sub&gt;in&lt;/sub&gt;</th>
<th>A</th>
<th>B</th>
<th>Borrow&lt;sub&gt;out&lt;/sub&gt;</th>
<th>Difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
8 Experiment 7

Pre-Lab

• What is [FPGA]?
• What is Verilog Hardware Description Language ([HDL])?
• What is the difference between an Application Specific Integrated Circuits ([ASIC]) and [FPGA]?
• Can [FPGA] work as a high speed digital controller? Why?
• Can [FPGA] be use to implements and re-configure thousands of logic gates? How?

8.1 Verilog HDL and Xilinx ISE

Objectives

1. Introduction to Xilinx ISE and [FPGA]
3. Creating a User Constrain File ([UCF]).
4. Synthesize, floor plan and generate a program file.
5. Program Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Theory Overview

A [FPGA] is an integrated circuit designed to be configured by a customer or a designer after manufacturing, hence it is “field-programmable”. The [FPGA] configuration is generally specified using a [HDL] similar to that used for an [ASIC]. Contemporary [FPGA]s have large resources of logic gates and Random Access Memory ([RAM]) blocks to implement complex digital computations. [FPGA] designs employ very fast inputs/outputs and bidirectional data buses.

Floor planning enables resources allocation within [FPGA] to meet these time constraints. [FPGA]s can be used to implement any logic function that an [ASIC] could perform. The ability to update the functionality after shipping, partial re-configuration of a portion of the design and the low non-recurring engineering costs relative to an [ASIC] design offer advantages for many applications. Let’s consider we have following equations to be implemented in [FPGA]

\[ C = A \cdot B \]

and

\[ E = \overline{A} + B \]

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Schematic

Figure 21 shows the schematic diagram for this experiment.
Procedure

1. To create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

2. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.

3. This brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “and_gate” and save it in a local directory. You can place comments for your project in the description text box.

4. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited for the PGA we have is
   - Family = Spartan 6
   - Device = XC6SLX9
   - Pakage = CSG324
   - Speed = -3

5. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

6. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard. Any modifications to these settings can be made by clicking the Back button.

7. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design.

8. You can add a new or existing source file to the project. To do this, right-click on the target device and select one of the three options for adding source files.

9. Once you have created the file, the HDL Editor Window displays the and_gate.v source code.

10. Schematic in Figure 21 shows that logic gates has two inputs (A,B) and two different outputs (C,E). Port D is a wire which is internally connected from one logic gate to an other. Its gate level verilog code can be provided as,

```verilog
module basic_gates
(input A, input B,
```
Table 24: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Expected C</th>
<th>Expected E</th>
<th>Measured C</th>
<th>Measured E</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

output C, output E);

wire D;
and g1(C, A, B);
or g2(D, A, B);
not g3(E, D);

endmodule

11. Type the code on editor work space and save it.

12. Now, synthesize the code and view Register Transfer Level (RTL) schematic.

13. Now to add an UCF file to your design, drag down the “user constrains” option and double click the Floorplanner Area. The Floorplanner Area will open after some moment.

14. Here, select and set the locations of input and output terminals used in Verilog programming on the FPGA board.
   - A → B3
   - B → A3
   - C → P4
   - E → L6

After setting the site, save it.

15. Now right click on “implement design” and Run. After its successful completion .ucf file will be generated.

16. Next, double click on “Generate Programming File”. After successful completion of Generate Programming File, .bit file will be generated.

17. Now, connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

18. Toggle the DIP switches on FPGA board and LEDs will turn on and off according to the programmed logic. Record the observations in Table 24.
8.2 Logic Gates Implementation

Theory Overview

FPGAs contain programmable logic components called “logic blocks” and a hierarchy of reconfigurable interconnects that allow the blocks to be “wired together”. It is like many (changeable) logic gates that can be inter-wired in many different configurations. Logic blocks can be configured to perform complex combinational functions or merely simple logic gates like AND, OR, NOT and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complex blocks of memory. Now consider we have the following equation to be implement in FPGA

\[ Z = A \cdot B + C \oplus D \]

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Schematic

Figure 22 shows the schematic diagram for this experiment.

Procedure

1. To create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

2. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.

3. This brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “logic gates” and save it in a local directory. You can place comments for your project in the description text box.

4. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited for the FPGA we have is
   - Family = Spartan 6
   - Device = XC6SLX9
   - Pakage = CSG324
   - Speed = -3

5. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

6. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard. Any modifications to these settings can be made by clicking the Back button.

7. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design.
8. You can add a new or existing source file to the project. To do this, right-click on the target device and select one of the three options for adding source files.

9. Once you have created the file, the HDL Editor Window displays the logic_gates.v source code.

10. Schematic in Figure 22 shows that logic gates has four inputs (A,B,C,D) and an output (Z). Ports E, F and G are wires which are internally connected from one logic gate to an other. Its Gate level Verilog code can be provided as,

```verilog
module logic_gates
  (input A,
   input B,
   input C,
   input D,
   output Z);
  wire E,F,G;
  and g1(E,A,B);
  xor g2(F,C,D);
  not g3(G,F);
  or g4(Z,E,G);
endmodule
```

11. Type the code on editor work space and save it.

12. Now, synthesize the code and view RTL schematic.

13. Now to add an UCF file to your design, drag down the “user constrains” option and double click the Floorplane Area. The Floor Planer Area will open after some moment.

- A → B3
- B → A3
- C → B4
- D → A4
- Z → P4

After setting the site, save it.

15. Now right click on “implement design” and Run. After its successful completion .ucf file will be generated.

16. Next, double click on “Generate Programming File”. After successful completion of Generate Program File, .bit file will be generated.
Table 25: Data Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>Expected Z</th>
<th>Measured Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

17. Now, connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

18. Toggle the DIP switches on FPGA and the LED will turn on and off according to the program logic. Record the observations in Table 25.

Questions

1. What are the applications of an FPGA?

2. Write a verilog code for the following expressions:
   
   a) $X = (A \cdot B) + \overline{C}$
   
   b) $Y = (\overline{A} \cdot B) + (A \cdot \overline{B})$
   
   c) $Z = (A + C) \cdot (\overline{A} + B + \overline{C})$
9 Experiment 8

Pre-Lab

- What is the purpose of floor plane area for the FPGA?
- What is the Register Transfer Level (RTL) schematic? Why it is used?
- What are decoders and why do we use decoders?
- What are the differences between decoder and encoder circuits? Explain with examples?
- Provide gate level verilog code for a full adder and a $2 \times 4$ decoder? Hint: Refer to the schematics in Figures 23 and 24.

9.1 Full Adder Implementation on FPGA

Objectives

1. Introduction to both Xilinx ISE and FPGA.
2. Verilog Hardware Description Language (HDL) data flow modeling.
3. Creating a User Constrain Files (UCF).
4. Synthesize, floor plan and generate a program file.
5. Program the Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Theory Overview

The most basic arithmetic operation is the addition of two binary digits. There are four possible elementary operations, namely,

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 10_2$ with 1 Carry

The first three operations produce a sum of whose length is one digit, but when the last operation is performed the sum is two digits. The higher significant bit of this result is called a carry and lower significant bit is called the sum.

A combinational circuit which performs the arithmetic sum of three input bits is called full adder. The three input bits include two significant bits and a previous carry bit. A full adder circuit can be implemented with two half adders and one OR gate.

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Schematic

Figure 23 shows the schematic diagram for this experiment.
Figure 23: Schematic

Procedure

1. To create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

2. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.

3. Executing the previous step brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “full_adder” and save it in a local directory. You can place comments for your project in the description text box.

4. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited on the microBoard is:
   - Family = Spartan 6
   - Device = XC6SLX9
   - Package = CSG324
   - Speed = -3

5. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

6. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard. Any modifications to these settings can be made by clicking the Back button.

7. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design.

8. You can add new or existing source files to the project. To do this, right-click on the target device and select one of the three options for adding source files.

9. Once you have created a verilog file, the HDL Editor Window displays the full_adder.v source code.
Table 26: Input/Output Pin Assignment

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>FPGA Resource</th>
<th>PMOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B3</td>
<td>K12</td>
</tr>
<tr>
<td>B</td>
<td>A3</td>
<td>K13</td>
</tr>
<tr>
<td>C_{in}</td>
<td>B4</td>
<td>F17</td>
</tr>
<tr>
<td>Sum</td>
<td>P4</td>
<td>H12</td>
</tr>
<tr>
<td>Carry</td>
<td>L6</td>
<td>G13</td>
</tr>
</tbody>
</table>

10. The schematic in Figure 21 shows that logic gates has three inputs (A, B and C_{in}) and two outputs (Sum and Carry). Ports D, E and F are wires which are internally connected from one logic gate to another. Its data flow verilog code can be written as,

```verilog
module full_adder
  (input A, 
   input B, 
   input C_{in}, 
   output Sum, 
   output Carry);

  assign {Carry, Sum} = A+B+C_{in};
endmodule
```

11. Type the code on the editor work space and save it.

12. Synthesize the code and view the RTL schematic.

13. Now to add UCF file to your design, drag down the “user constrains” option and double click the Floorplane Area. The Floor Planner Area will open after some time.

14. In the Floorplaner, select and set the locations of input and output terminals used in the verilog code on the FPGA board. Table 26 shows the input and output pin assignments for this code. After setting the sites, save it.

15. Right click on “implement design” and Run. After its successful completion .ucf file will be generated.

16. Double click on “Generate Programming File”. After successful completion of Generate Program File, .bit file will be generated.

17. Connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

18. Toggle the DIP switches on the FPGA board and the LEDs will turn on and off according to the programmed logic. Record the observations in Table 27.
9.2 2×4 Decoder Implementation on FPGA

Theory Overview

In digital electronics, a decoder can take the form of a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. For example, binary coded decimal decoders. Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding. There can be 2 to 4 decoder, 3 to 8 decoder or 4 to 16 decoder. We can form a 3 to 8 decoder from two 2 to 4 decoders.

Similarly, we can also form a 4 to 16 decoder by combining two 3 to 8 decoders. In this type of circuit design, the enable inputs of both 3 to 8 decoders originate from a 4th input, which acts as a selector between the two 3 to 8 decoders. This allows the 4th input to enable either the top or bottom decoder, which produces outputs of D1 through D7 for the first decoder, and D8 through D15 for the second decoder.

Procedure

1. To create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

2. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.

3. Executing the previous step brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “decoder” and save it in a local directory. You can place comments for your project in the description text box.

4. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited on the microBoard:
   - Family = Spartan 6
   - Device = XC6SLX9
   - Package = CSG324
5. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

6. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard.

7. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design.

8. You can add new or existing source files to the project. To do this, right-click on the target device and select one of the three options for adding source files.

9. Once you have created a verilog file, the HDL Editor Window displays the decoder.v source code.

10. The schematic in Figure 24 shows that logic gates has two inputs ($A_0, A_1$) and four outputs ($D_0, D_1, D_2, D_3$). Ports B and C are wires which are internally connected from one logic gate to an other. Its data flow verilog code can be written as,

    ```verilog
    module decoder
    (input A0,
     input A1,
     output D0,
     output D1,
     output D2,
     output D3);
    assign
    D0 = ~A0 & ~A1,
    D1 = A0 & ~A1,
    D2 = ~A0 & A1,
    D3 = A0 & A1;
    endmodule
    ```

11. Type the code in the editor work space and save it. Synthesize the code and view the RTL schematic.

12. Now to add UCF file to your design, drag down the “user constrains” option and double click the Floorplane Area. The Floor Planer Area will open after some time.
Table 28: Input/Output Pin Assignment

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>FPGA Resource</th>
<th>PMOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>B3</td>
<td>K12</td>
</tr>
<tr>
<td>A1</td>
<td>A3</td>
<td>K13</td>
</tr>
<tr>
<td>D0</td>
<td>P4</td>
<td>H12</td>
</tr>
<tr>
<td>D1</td>
<td>L6</td>
<td>G13</td>
</tr>
<tr>
<td>D2</td>
<td>F5</td>
<td>E16</td>
</tr>
<tr>
<td>D3</td>
<td>C2</td>
<td>E18</td>
</tr>
</tbody>
</table>

Table 29: Data Table

<table>
<thead>
<tr>
<th>A1</th>
<th>A0</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

13. In the Floorplaner, select and set the locations of input and output terminals used in the Verilog programming on the FPGA board. Table 28 shows the input and output pin assignments for this code. After setting the site, save it.

14. Right click on “implement design” and Run. After its successful completion .ucf file will be generated.

15. Double click on “Generate Programming File”. After successful completion of Generate Program File, .bit file will be generated.

16. Connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

17. Toggle the DIP switches on the MicroBoard and the LEDs will turn on and off according to the program logic. Record the observations in Table 29.

**Questions**

1. What are the applications of decoders and encoders?

2. Provide a verilog code of 4×2 encoder?

3. Write a verilog code for the following expressions:
   a) F (A,B,C,D) = Σ (3,4,8,10,12,13,14)
   b) G (A,B,C,D) = Π (4,6,7,9,10,11,12,13,14,15)
10 Experiment 9

Pre-Lab

• What is the purpose of Look Up Table (LUT) in the FPGA fabric?
• What is the purpose of Xilinx ISE generated .bit file? Why it is used?
• What are flip flops and why do we use flip flops?
• What are the differences between multiplexer and de-multiplexer? Explain with examples?
• Provide gate level verilog code for a $4 \times 1$ multiplexer and a D-Flip Flop decoder? Hint: Refer to the schematics in Figures 25 and 26.

10.1 $4 \times 1$ Multiplexer Implementation on FPGA

Objectives

1. Verilog Hardware Description Language (HDL) data flow and behavioral modeling.
2. Isim simulation of verilog code.
3. Creating a User Constrain Files (UCF).
4. Synthesize, floor plan and generate a program file.
5. Program the Xilinx Spartan-6, LX-9 MicroBoard FPGA.
6. Breadboard prototyping with FPGA.

Theory Overview

A multiplexer (or mux) is a device that selects one of several analog or digital input signals and forwards the selected input into a single line. A multiplexer of $2^n$ inputs has $n$ select lines, which are used to select which input line to send to the output. Multiplexers are mainly used to increase the amount of data that can be sent over the network within a certain amount of time and bandwidth. A multiplexer is also called a data selector.

A multiplexer makes it possible for several signals to share one device or resource, for example one analog to digital converter or one communication line, instead of having one device per input signal. Conversely, a de-multiplexer (or demux) is a device taking a single input signal and selecting one of many data-output-lines, which is connected to the single input. A multiplexer is often used with a complementary de-multiplexer on the receiving end. An electronic multiplexer can be considered as a multiple-input, single-output switch and a de-multiplexer as a single-input, multiple-output switch. The boolean expression for a $4 \times 1$ multiplexer is defined as

$$Z = (I_0 \cdot \overline{S_0} \cdot \overline{S_1}) + (I_1 \cdot S_0 \cdot \overline{S_1}) + (I_2 \cdot \overline{S_0} \cdot S_1) + (I_3 \cdot S_0 \cdot S_1)$$

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.
3. Breadboard, MicroBoard to breadboard interface connector.
4. Slide switches and LED.
Figure 25: Schematic

Schematic

Figure 25 shows the schematic diagram for this experiment.

Procedure

1. To create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

2. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.

3. Executing the previous step brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “multiplexer” and save it in a local directory. You can place comments for your project in the description text box.

4. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited on the microBoard is:
   - Family = Spartan 6
   - Device = XC6SLX9
   - Package = CSG324
   - Speed = -3

5. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

6. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard. Any modifications to these settings can be made by clicking the Back button.

7. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design. Here you can add new or existing source files to the project. To do this, right-click on the target device and select one of the three options for adding source files.
8. Once you have created a verilog file, the HDL Editor Window displays the multiplexer.v source code.

9. The schematic in Figure 21 shows that logic gates has six inputs ($S_0$, $S_1$, $I_0$, $I_1$, $I_2$ and $I_3$) and an output ($Z$). Its data flow verilog code can be written as,

```verilog
module multiplexer
  (input I0,
   input I1,
   input I2,
   input I3,
   input S0,
   input S1,
   output Z);

assign Z = S1? (S0? I3 : I2) : (S0? I1 : I0);
endmodule
```

10. Type the code on the editor work space and save it.

11. Synthesize the code and view the RTL schematic.

12. Now add a new source file of Verilog Test Fixture to your project. In this source file, you are able to define circuit inputs over time so the simulator knows how to drive the outputs. To add the source file, right-click on the device in the Sources window and choose the New Source option. In the New Source wizard, choose Verilog Test Fixture for the source type and provide a meaningful name.

13. After clicking Next, the following dialog box asks you to select the source file you want to associate with the given test fixture file. This dictates which source file you actually run the simulation on. Complete the new source file creation by clicking Next and Finish.

14. Here you can provide your desired input parameters,

```verilog
// Inputs
reg I0;
reg I1;
reg I2;
reg I3;
reg S0;
reg S1;
wire Z; // Output

multiplexer uut // Instantiate the Unit Under Test (UUT)
  (.I0(I0),
   .I1(I1),
   .I2(I2),
   .I3(I3),
   .S0(S0),
   .S1(S1),
   .Z(Z));

initial begin // Initialize Inputs
I0 = 1;
I1 = 0;
```

55
I2 = 1;
I3 = 0;
S0 = 0;
S1 = 0;

#100; // Wait 100 ns
S0 = 0;
S1 = 0;
#100; // Wait 100 ns
S0 = 1;
S1 = 0;
#100; // Wait 100 ns
S0 = 0;
S1 = 1;
#100; // Wait 100 ns
S0 = 1;
S1 = 1;
end
endmodule

15. After typing this code, simulate behavioral model. Now time line diagram will appear. By clicking Run all, you can observe the functionality of the multiplexer.

16. Now to add UCF file to your design, drag down the “user constrains” option and double click the Floorplane Area. The Floor Planer Area will open after some time.

17. In the Floorplaner, select and set the locations of input and output terminals used in the verilog code on the FPGA board.
   
   - I0 → F14
   - I1 → G14
   - I2 → D17
   - I3 → D18
   - S0 → F15
   - S1 → F16
   - Z → C17

   After setting the sites, save it.

18. Right click on “implement design” and Run. After its successful completion .ucf file will be generated.

19. Double click on “Generate Programming File”. After successful completion of Generate Program File, .bit file will be generated.

20. Connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA

21. Make a fair circuit on breadboard with the FPGA

22. Toggle the slide switches on the breadboard and the LED will turn on and off according to the programmed logic. Record the observations in Table 30.
Table 30: Data Table

<table>
<thead>
<tr>
<th>I_3</th>
<th>I_2</th>
<th>I_1</th>
<th>I_0</th>
<th>S_1</th>
<th>S_0</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

Questions

1. What are the applications of multiplexers and de-multiplexers?
2. Provide a circuit diagram and verilog code of 8 × 1 multiplexer?
3. Provide a circuit diagram and verilog code of 1 × 4 de-multiplexer?

10.2 D-Flip Flop Implementation on FPGA

Theory Overview

A flip flop circuit has two stable states and it can be used to store state information. An other name for a flip flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and it will have one or two outputs. It is the basic storage element in sequential logic. Flip flops are fundamental building blocks of digital electronics systems and they are used in computers, communications, and many other types of systems.

Flip flops are used as data storage elements. Such data storage can be used for storage of state and such a circuit is described as sequential logic. When used in a finite-state machine, the output and next state depend not only on its current input but also on its current state and previous inputs. It can also be used for counting of pulses and for synchronizing variably-timed input signals to some reference timing signal.

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.
3. Breadboard, MicroBoard to breadboard interface connector.
4. Slide switches and LEDs.

Schematic

Figure 26 shows the schematic diagram for this experiment.

Procedure

1. To create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.
2. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.
3. Executing the previous step brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “flip_flop” and save it in a local directory. You can place comments for your project in the description text box.

4. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited on the microBoard:
   - Family = Spartan 6
   - Device = XC6SLX9
   - Package = CSG324
   - Speed = -3

5. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

6. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard.

7. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design. Here you can add new or existing source files to the project. To do this, right-click on the target device and select one of the three options for adding source files.

8. Once you have created a verilog file, the HDL Editor Window displays the flip_flop.v source code.

9. The schematic in Figure 26 shows that logic gates has two inputs (D, clock) and two outputs (Q, Q). Its behavioral verilog code can be written as,

```verilog
module D_ff
(input D,
 input reset ,
 input CLK,
 output reg Q,
 output reg Q_bar );

always @ (posedge CLK or negedge reset )
begin
if ( reset == 0 )
```

Figure 26: Schematic
begin
Q <= 0;
Q_bar <= 1;
end
else if (D == 1)
begin
Q <= 1;
Q_bar <= 0;
end
else
begin
Q <= 0;
Q_bar <= 1;
end
endmodule

10. Type the code in the editor work space and save it. Synthesize the code and view the RTL schematic.

11. Now add a new source file of Verilog Test Fixture to your project. In this source file, you are able to define circuit inputs over time so the simulator knows how to drive the outputs.

12. To add the source file, right-click on the device in the Sources window and choose the New Source option. In the New Source wizard, choose Verilog Test Fixture for the source type and provide a meaningful name.

13. After clicking Next, the following dialog box asks you to select the source file you want to associate with the given test fixture file. This dictates which source file you actually run the simulation on. Complete the new source file creation by clicking Next and Finish.

14. Here you can provide your desired input parameters,

```verilog
module test_D_ff;

// Inputs
reg D;
reg reset;
reg CLK;

// Outputs
wire Q;
wire Q_bar;

// Instantiate the Unit Under Test (UUT)
D_ff uut (.D(D),
.reset(reset),
.CLK(CLK),
.Q(Q),
.Q_bar(Q_bar));

initial begin
// Initialize Inputs
D = 0;
reset = 0;
```

59
Table 31: Input/Output Pin Assignment

<table>
<thead>
<tr>
<th>Input/Output</th>
<th>FPGA Resource</th>
<th>PMOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>C10</td>
<td>C10</td>
</tr>
<tr>
<td>reset</td>
<td>B3</td>
<td>K13</td>
</tr>
<tr>
<td>D</td>
<td>A3</td>
<td>K12</td>
</tr>
<tr>
<td>Q</td>
<td>P4</td>
<td>H12</td>
</tr>
<tr>
<td>Q_bar</td>
<td>L6</td>
<td>G13</td>
</tr>
</tbody>
</table>

CLK = 0;

#100; // Wait 100 ns
reset = 1;
#100; // Wait 100 ns
D = 1;
#100; // Wait 100 ns
D = 0;
#100; // Wait 100 ns
D = 1;
end
always @(*)
#5 CLK <= ~ CLK;
endmodule

15. After typing this code, simulate behavioral model. Now time line diagram will appear. By clicking Run all, you can observe the functionality of your D-Flip flop.

16. Now to add UCF file to your design, drag down the “user constrains” option and double click the Floorplaner Area. The Floor Planer Area will open after some time.

17. In the Floorplaner, select and set the locations of input and output terminals used in the Verilog programming on the FPGA board. Table 31 shows the input and output pin assignments for this code. After setting the site, save it.

18. Right click on “implement design” and Run. After its successful completion .ucf file will be generated.

19. Double click on “Generate Programming File”. After successful completion of Generate Program File, .bit file will be generated.

20. Connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

21. Make a fair circuit on breadboard with the FPGA.

22. Toggle the slide switches on the breadboard and the LEDs will turn on and off according to the program logic. Record the observations in Table 32.
Table 32: Data Table

<table>
<thead>
<tr>
<th>reset</th>
<th>D</th>
<th>Q</th>
<th>Q_bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Questions

1. What are the applications of JK flip flops and SR flip flops?
2. Provide a verilog code of clock synchronized SR flip flop?
11 Experiment 10

Pre-Lab

- What are binary counters? Explain its functionality?
- What is the difference between event driven and clock driven counters?
- Can a counter be use as a clock frequency divider? Explain with example?
- Counter is a combinational circuit or sequential circuit? Explain?

11.1 Counter Implementation on FPGA

Objectives

1. Verilog Hardware Description Language (HDL) behavioral modeling.
2. Isim simulation of verilog code.
3. Creating a User Constrain Files (UCF).
4. Synthesize, floor plan and generate a program file.
5. Program the Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Theory Overview

Counters are widely considered as essential building blocks for a variety of circuit operations such as programmable frequency dividers, shifters, code generators, memory select management and various arithmetic operations. Since many applications are comprised of these fundamental operations, much research focuses on efficient counter architecture design. Counter architecture design methodologies explore tradeoffs between operating frequency, power consumption, area requirements and target application specialization.

Early design methodologies improved counter operating frequency by partitioning large counters into multiple smaller counting modules, such that modules of higher significance (containing higher significant bits) were enabled when all bits in all modules of lower significance (containing lower significant bits) saturate. Initializations and propagation delays such as register load time, AND logic chain decoding and the half incrementer component delays in half adders dictated operating frequency. Subsequent methodologies improved counter operating frequency using half adders in the parallel counting modules.

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.
3. Four channel digital oscilloscope.
4. Breadboard, MicroBoard to breadboard interface connector.

Block Diagram

Figure 27 shows the block diagram for this experiment.
Procedure

1. To create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

2. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.

3. Executing the previous step brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “up_counter” and save it in a local directory. You can place comments for your project in the description text box.

4. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited on the microBoard is:
   - Family = Spartan 6
   - Device = XC6SLX9
   - Pakage = CSG324
   - Speed = -3

5. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

6. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard. Any modifications to these settings can be made by clicking the Back button.

7. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design. Here you can add new or existing source files to the project. To do this, right-click on the target device and select one of the three options for adding source files.

8. Once you have created a verilog file, the HDL Editor Window displays the up_counter.v source code.

9. The schematic in Figure 27 shows that up_counter has two inputs (clk, reset) and an eight bit output (out). Its behavioral verilog code can be written as,

   ```verilog
   module up_counter
   (out,
    clk,
    reset);
   ```
output [7:0] out;
input clk, reset;
reg [8:0] prescaler;
reg [7:0] out;
always @(posedge clk)
if (reset)
begin
    prescaler <= 9'b0;
    out <= 8'b0;
end
else if (prescaler == 9'b110001111)
begin
    prescaler <= 9'b0;
    out <= out + 1'b1;
end
else
begin
    prescaler <= prescaler + 1'b1;
end
endmodule

10. Type the code on the editor workspace and save it.
11. Synthesize the code and view the RTL schematic.
12. Now add a new source file of Verilog Test Fixture to your project. In this source file, you are able to define circuit inputs over time so the simulator knows how to drive the outputs. To add the source file, right-click on the device in the Sources window and choose the New Source option. In the New Source wizard, choose Verilog Test Fixture for the source type and provide a meaningful name.
13. After clicking Next, the following dialog box asks you to select the source file you want to associate with the given test fixture file. This dictates which source file you actually run the simulation on. Complete the new source file creation by clicking Next and Finish.
14. Here you can provide your desired input parameters,

module up_counter_test_fixture;
    // Inputs
    reg clk;
    reg reset;

    wire [7:0] out; // Output
    // Instantiate the Unit Under Test (UUT)
    up_counter uut (.out(out), .clk(clk), .reset(reset));

    parameter PERIOD = 250;
    initial
    begin
        reset = 1; // Initialize Inputs
        #10000; // Wait 10000 ns for global reset to finish
        reset = 0;
    end

64
always
begin
clk = 1’b0;
# (PERIOD/2) clk = 1’b1;
# (PERIOD/2);
end
endmodule

15. After typing this code, simulate behavioral model. Now time line diagram will appear. By clicking Run all, you can observe the functionality of the multiplexer.

16. Now to add UCF file to your design, drag down the “user constrains” option and double click the Floorplane Area. The Floor Planer Area will open after some time.

17. In the Floorplaner, select and set the PMOD locations of input and output terminals used in the verilog code on the FPGA board.
   - out[0] → E18
   - out[7] → F18
   - clk → V10
   - reset → B3

   After setting the sites, save it.

18. Right click on “implement design” and Run. After its successful completion .ucf file will be generated.

19. Double click on “Generate Programming File”. After successful completion of Generate Program File, .bit file will be generated.

20. Connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

21. Mount the FPGA on the breadboard and observe the outputs on the function generator.

22. Turn on reset switch on the MicroBoard and output will appear on the PMOD connector.

23. Connect the four channel oscilloscope with the PMOD and observe the waveforms.

24. Take a clear picture of the waveforms on the oscilloscope and attach it with the lab report.

Questions
1. Write a verilog code for a down_counter.
2. Can counters be used in timer circuitry? Explain with a block diagram?
12 Experiment 11

Pre-Lab

- What is Random Access Memory (RAM)? Explain with example?
- What is Read Only Memory (ROM)? Explain with example?
- What is Flash Memory? Explain with example?

12.1 Sawtooth Generator Using DAC and FPGA

Objectives

1. Verilog Hardware Description Language (HDL) behavioral modeling.
2. Isim simulation of verilog code.
3. Creating a User Constrain Files (UCF).
4. Synthesize, floor plan and generate a program file.
5. Program the Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Theory Overview

Resistor ladder networks provide a simple, inexpensive way to perform digital to analog conversion DAC. The most popular networks is the R/2R ladder. It converts digital voltage information to analog and the R/2R ladder has become the most popular due to the networks inherent accuracy superiority and ease of manufacture. Figure 28 is a diagram of the basic R/2R ladder network with n bits. The “ladder” portrayal comes from the ladder-like topology of the network. Note that the network consists of only two resistor values; R and 2R (twice the value of R) no matter how many bits make up the ladder. The particular value of R is not critical to the function of the R/2R ladder. The binary weighted ladder requires double multiples of R as the number of bits increase. More accurate ratios can be obtained in a resistor network with consistent, similar values as in the R/2R network. The R/2R network provides the most accurate method of digital to analog conversion. [1]

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.
3. Four channel digital oscilloscope.
4. DC power supply.
5. DMM
7. DAC circuit board.

Schematic

Figure 28 shows the schematic diagram for this experiment.
Procedure

1. Launch Xilinx ISE project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

2. Select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited on the microBoard is:
   - Family = Spartan 6
   - Device = XC6SLX9
   - Package = CSG324
   - Speed = -3

3. Open project “up_counter” the **HDL** Editor Window displays the up_counter.v source code.

4. Synthesize the code and view the **RTL** schematic.

5. Now drag down the “user constrains” option and double click the Floorplane Area. The Floor Planer Area will open after some time.

6. In the Floorplaner, select and set the PMOD locations of input and output terminals used in the verilog code on the **FPGA** board.
   - out[0] → E18
   - out[7] → F18
   - clk → V10
   - reset → B3
After setting the sites, save it.

7. Right click on “implement design” and Run. After its successful completion .ucf file will be generated.

8. Double click on “Generate Programming File”. After successful completion of Generate Programming File, .bit file will be generated.

9. Connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

10. Turn on the reset switch on the MicroBoard and output will appear on its PMOD connector.

11. Provide the power supply to the DAC board according to following port connections.
   a) First (Most right port) = -15 V<sub>dc</sub>
   b) Second = +15 V<sub>dc</sub>
   c) Third = Ground
   d) Fourth (Most left port) = Output

12. Mount the FPGA on the DAC Printed Circuit Board (PCB) and observe the outputs on the oscilloscope.

13. Connect the oscilloscope at following three stages of the DAC output and observe the waveform.
   a) Without DC offset
   b) With DC offset
   c) DAC output (Observe the frequency and amplitude)

14. Take a clear picture of the waveforms on the oscilloscope and attach it to the lab report.

**Tasks**

1. Double the frequency of the sawtooth generator.
2. Make a down counter and observe the sawtooth wave.
3. Take a clear picture of the waveforms on the oscilloscope and attach it to the lab report.

**Question**

Conclude what you have learned in the DLD Lab? (One paragraph = around 10 lines)
13  Experiment 12

Pre-Lab

- What is a Digital To Analog Converter (DAC)? Explain its functionality?
- What is the difference between Analog To Digital Converter (ADC) and DAC?
- Can a DAC be used to generate a continuous sine wave? Explain?

13.1  Sine Wave Generator Using DAC and FPGA

Objectives

1. Verilog Hardware Description Language (HDL) behavioral modeling.
2. Isim simulation of verilog code.
3. Creating a User Constrain Files (UCF).
4. Synthesize, floor plan and generate a program file.
5. Program the Xilinx Spartan-6, LX-9 MicroBoard FPGA.

Theory Overview

Resistor ladder networks provide a simple, inexpensive way to perform digital to analog conversion. The most popular networks is the R/2R ladder. It converts digital voltage information to analog and the R/2R ladder has become the most popular due to the networks inherent accuracy superiority and ease of manufacture. Figure 29 is a diagram of the basic R/2R ladder network with n bits. The “ladder” portrayal comes from the ladder-like topology of the network. Note that the network consists of only two resistor values; R and 2R (twice the value of R) no matter how many bits make up the ladder. The particular value of R is not critical to the function of the R/2R ladder. The binary weighted ladder requires double multiples of R as the number of bits increase. More accurate ratios can be obtained in a resistor network with consistent, similar values as in the R/2R network. The R/2R network provides the most accurate method of digital to analog conversion.

Equipment

2. Xilinx Spartan-6, LX-9 MicroBoard FPGA.
3. Four channel digital oscilloscope.
4. DC power supply.
5. DMM.
7. DAC circuit board.

Block Diagram

Figure 30 shows the block diagram for this experiment.
Procedure

1. Open QuantisedSineWave.m MATLAB file and run. A plot of three discrete sinusoidal waveforms will appear. Observe the waveforms.

2. Now launch Xilinx ISE and create a new project, open project navigator either from the desktop shortcut icon or by selecting Start → All Programs → Xilinx Design Tools → ISE Design Suite 14.7 → ISE Design Tools → 64-bit Project Navigator.

3. In Project Navigator, select the New Project option from the Getting Started menu or by selecting Select File → New Project.

4. Executing the previous step brings up a dialog box where you can enter the desired project name and project location. You should choose a meaningful name for easy reference. In this lab, we call this project “SineGenerator” and save it in a local directory. You can place comments for your project in the description text box.

5. The next step is to select the proper Family, Device, and Package for your project. This depends on the chip you are targeting for this project. The appropriate settings for a project suited on the microBoard is:
   - Family = Spartan 6
   - Device = XC6SLX9
   - Package = CSG324
   - Speed = -3

6. Once the appropriate settings have been entered, click Next. The next two dialog boxes give you the option of adding new or existing source files to your project, click Next without adding any source files.

7. Before the new project is created, the New Project Wizard gives you a project summary consisting of the selected specifications you have chosen for the project. Make sure all settings are correct before clicking Finish to end the New Project Wizard. Any modifications to these settings can be made by clicking the Back button.
8. Once the new project is created, two sources are listed under sources in the Design panel: the Project file name and the Device targeted for design. Here you can add new files to the project. To do this, right-click on the target device and select new source.

9. Once you have created a verilog file, the HDL Editor Window displays the SineGenerator.v source code.

10. The block diagram in Figure 30 shows that the top level of SineGenerator has two inputs (clk, reset) and an eight bit output (out).

11. The behavioral verilog code for the up_counter can be written as,

```
module up_counter
(input clk, reset);
output [7:0] out;
input clk, reset;
reg [8:0] prescaler;
reg [7:0] out;
always @(posedge clk)
if (reset)
  begin
    prescaler <= 9'b0;
    out <= 8'b0;
  end
else if (prescaler == 9'b110001111)
begin
    prescaler <= 9'b0;
    out <= out + 1'b1;
end
else
begin
    prescaler <= prescaler + 1'b1;
end
endmodule
```

12. Type the code on the editor work space and save it.

13. Now again add a new source file to your project for SinRom and write following code,

```
module SineRom
(input clk, maddr);

parameter ROM_WIDTH = 8;
parameter ROM_ADDR_BITS = 7;

input clk;
input [ROM_ADDR_BITS-1:0] maddr;
output [ROM_WIDTH-1:0] sample;

//-- (* ROM_STYLE="{AUTO | DISTRIBUTED | BLOCK}" *)
(* ROM_STYLE="distributed" *)
```
reg [ROM_WIDTH−1:0] sine_rom [(2∗ROM_ADDR_BITS)−1:0];
reg [ROM_WIDTH−1:0] sample;
reg [ROM_ADDR_BITS−1:0] addr;

initial
$readmemb("sine_8_1000.txt", sine_rom, 0, 99);

always @(posedge clk)
sample <= sine_rom[maddr];
endmodule

14. Finally, add a new source file to your project for a top level SineGenerator and write following code,

module SineGenerator
(output [7:0] out,
input clk, reset);
output [7:0] addr;

wire [7:0] addr;

// Instantiate the counter module
up_counter uc (.out(addr),
.clk(clk),
.reset(reset));

// Instantiate the counter module
SineRom sr (.sample(out),
.maddr(addr[6:0]),
.clk(clk));
endmodule

15. Synthesize the code and view the RTL schematic.

16. Now add a new source file of Verilog Test Fixture to your project. In this source file, you are able to define circuit inputs over time so the simulator knows how to drive the outputs. To add the source file, right-click on the device in the Sources window and choose the New Source option. In the New Source wizard, choose Verilog Test Fixture for the source type and provide a meaningful name.

17. After clicking Next, the following dialog box asks you to select the source file you want to associate with the given test fixture file. This dictates which source file you actually run the simulation on. Complete the new source file creation by clicking Next and Finish.

18. Here you can provide your desired input parameters,

module SineGenerator_test_fixture;

// Inputs
reg clk;
reg reset;
// Outputs
wire [7:0] out;

// Instantiate the Unit Under Test (UUT)
SineGenerator uut (.out(out),
   .clk(clk),
   .reset(reset));

parameter PERIOD = 250;

initial begin
// Initialize Inputs
reset = 1;

// Wait 1000 ns for global reset to finish
#1000;
reset = 0;
end

always begin
clk = 1'b0;
#(PERIOD/2) clk = 1'b1;
#(PERIOD/2);
end
endmodule

19. After typing this code, simulate behavioral model. Now time line diagram will appear. By clicking Run all, you can observe the functionality of the multiplexer.

20. Now to add UCF file to your design, drag down the “user constrains” option and double click the Floorplaner Area. The Floor Planer Area will open after some time.

21. In the Floorplaner, select and set the PMOD locations of input and output terminals used in the verilog code on the FPGA board.
   - out[0] → E18
   - out[7] → F18
   - clk → V10
   - reset → B3

   After setting the sites, save it.

22. Right click on “implement design” and Run. After its successful completion .ucf file will be generated.

23. Double click on “Generate Programming File”. After successful completion of Generate Program File, .bit file will be generated.
24. Connect the FPGA and then “Configure Target Device” → create new project → select the generated .bit file and program the FPGA.

25. Turn on the reset switch on the MicroBoard and output will appear on the PMOD connector.

26. Mount the FPGA on the DAC PCB and observe the outputs on the function generator.

27. Connect the oscilloscope with the DAC output and observe the waveform.

28. Take a clear picture of the waveform on the oscilloscope and attach it with the lab report.

Questions

1. What is the reason of Romwidth is 7 bits, while it is getting 8 bits outputs. Why its MSB is neglected?

2. Can we use DAC to generate sawtooth waveform? How?

3. What is the advantage of higher resolution DAC?

4. What are the applications of DAC?

5. What are the applications of ADC?
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog To Digital Converter</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application Specific Integrated Circuits</td>
</tr>
<tr>
<td>ALU</td>
<td>Arithmetic Logic Unit</td>
</tr>
<tr>
<td>BCD</td>
<td>Binary Coded Decimal</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital To Analog Converter</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DMM</td>
<td>Digital Multi Meter</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>LUT</td>
<td>Look Up Table</td>
</tr>
<tr>
<td>LED</td>
<td>Light Emitting Diode</td>
</tr>
<tr>
<td>POS</td>
<td>Product of Sum</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RTL</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>ROM</td>
<td>Read Only Memory</td>
</tr>
<tr>
<td>SOP</td>
<td>Sum of Product</td>
</tr>
<tr>
<td>SW</td>
<td>Switch</td>
</tr>
<tr>
<td>TTL</td>
<td>Transistor Transistor Logic</td>
</tr>
<tr>
<td>URA</td>
<td>Under Graduate Research Assistant</td>
</tr>
<tr>
<td>UCF</td>
<td>User Constrain File</td>
</tr>
</tbody>
</table>