INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

Lab Manual:

DIGITAL SYSTEM DESIGN LABORATORY (AECC06)

Prepared by

Ms S. Sushma (IARE10513)

Electronics and Communication Engineering
Institute of Aeronautical Engineering

December 7, 2021

Contents

Course Information

1.1 Introduction

This course is intended to enhance the learning experience of the student in topics encountered in AECB10. In this lab, students are expected to gain experience in using the basics for design of high-speed ALUs, control and timing circuitry, and asynchronous systems; hands-on system prototyping with HDLs for FPGA devices; current hardware topics related to computer design using modern design methodologies and CAD tools; and principles of system design for testability. How the student performs in the lab depends on his/her preparation, participation, and teamwork. Each team member must participate in all aspects of the lab to insure a thorough understanding of the equipment and concepts. The student, lab teaching assistant, and faculty coordinator all have certain responsibilities toward successful completion of the lab's goals and objectives.

1.1.1 Student Responsibilities

The student is expected to be prepared for each lab. Lab preparation includes reading the lab experiment and related textbook material. If you have questions or problems with the preparation, contact your Laboratory teaching Faculty (LTF), but in a timely manner. Do not wait until an hour or two before the lab and then expect the LTF to be immediately available.

A large portion of the student's grade is determined in the comprehensive final exam, resulting in a requirement of understanding the concepts and procedure of each lab experiment for the successful completion of the lab class. The student should remain alert and use common sense while performing a lab experiment. They are also responsible for keeping a professional and accurate record of the lab experiments in the lab manual wherever tables are provided. Students should report any errors in the lab manual to the teaching Faculty.

The student responsibilities are:

1. Students are required to attend all labs.
2. Students should work individually while performing lab experiments.
3. Students have to bring the lab worksheet whenever they come for lab work.
4. Should take only the lab worksheet, calculator (if needed) and a pen or pencil to the work area.
5. Should learn the prelab questions. Read through the lab experiment to familiarize themselves with the basics of embedded C and 8051 Trainer board.
6. Should utilize 3 hour's time properly to perform the experiment and to verify the results. Do the experiments, draw the graphs if required and take the signature from the faculty.
7. If the experiment is not completed in the stipulated time, the pending work has to be carried out in the leisure hours or extended hours.

8. For practical subjects there shall be a continuous evaluation during the semester for 30 sessional marks and 70 end examination marks.

9. Out of 30 internal marks, 20 marks shall be awarded for day-to-day work and 10 marks to be awarded by conducting an internal laboratory test.

1.1.2 Laboratory Teaching Assistant Responsibilities

The LTF shall be completely familiar with each lab prior to class. The LTF shall provide the students with a syllabus and safety review during the first class. The LTF is responsible for ensuring that all the necessary equipment and/or preparations for the lab are available and in working condition. Lab experiments should be checked in advance to make sure everything is in working order. The LTF should fully answer any questions posed by the students and supervise the students performing the lab experiments. The LTF is expected to award the marks in the lab worksheets in a fair and timely manner. The marks awarded lab worksheets should uploaded in student portal at the same day.

1.1.3 Faculty Coordinator Responsibilities

The faculty coordinator should ensure that the laboratory is properly equipped, i.e., that the teaching assistants receive any equipment necessary to perform the experiments. The coordinator is responsible for supervising the teaching assistants and resolving any questions or problems that are identified by the teaching assistants or the students. The coordinator may supervise the format of the final exam for the lab. They are also responsible for making any necessary corrections to this manual and ensuring that it is continually updated and available.

1.1.4 Lab Policy and Grading

The student should understand the following policy:

ATTENDANCE: Attendance is mandatory for all lab hours as per the academic regulations.

LAB RECORD's: The student must:

1. Thoroughly read the prelab of corresponding experiment before entering to the lab.

2. Come with full preparation of corresponding lab experiment, after reaching the lab you should write the worksheet without seeing the lab manual.

3. Write the all the tabular columns and related calculations after completion of experiment.

4. Take the lab work sheet correction before leaving the lab.

GRADING POLICY:

1. The every week lab worksheet is evaluated for 20 Marks.

2. The 20 Marks are sub dived to 5 parts. i.e. preparation, algorithm/circuit design, program / procedure, execution and Viva. Each part is assigned for 4 marks.

3. Based the student performance the lab, the faculty grade the marks out of 20 marks in each lab.

4. Finally, the faculty grade the marks for day to day evaluation by making the average of conducted laboratory sessions.

5. 10 marks are evaluated from the lab internal exam.

PRE-REQUISTES AND CO-REQUISTIES: The lab course is to be taken during the same semester as AEC111, but receives a separate grade.

1.1.5 Course Goals and Objectives

The Embedded Systems Laboratory is designed to provide the student with the knowledge to use basic input/output devices and techniques with proficiency. These techniques are designed to complement the concepts introduced in AEC016. In addition, the student should learn how to record experimental results effectively and present these results in a lab worksheet.

More explicitly, the class objectives are:

- To gain proficiency in the usage of Keil IDE Tool.
- To enhance understanding the theory of embedded systems concepts including:
  - Operating systems basics,
  - Task scheduling
  - Embedded software development tools,
  - Multiprocessing and multitasking,
  - Task communication/synchronization issues,
  - Advance processors.
- To develop communication skills through:
  - Maintenance of succinct but complete laboratory notebooks as permanent, written descriptions of procedures, results, and analyses.
  - Verbal interchanges with the laboratory instructor and other students.
  - Preparation of succinct but complete laboratory reports.
- To compare theoretical predictions with experimental results and to determine the source of any apparent errors.

1.2 Use of Laboratory Instruments

One of the major goals of this lab is to familiarize the student with the interfacing of various input/output (I/O) devices with the 8051 microcontroller. Some understanding of the lab instruments is necessary to avoid personal or equipment damage. By understanding the device's purpose and following a few simple rules, costly mistakes can be avoided. Most of the instrumentation used in this laboratory is implemented through the interfacing modules, Personal computer, and Cathode ray oscilloscope.

In general, all devices have physical limits. These limits are specified by the device manufacturer and are referred to as the device rating. The ratings are usually expressed in terms of voltage limits, current limits, or power limits. It is up to the engineer to make sure that in device operation, these ratings (limit values) are not exceeded. The following rules provide a guideline for instrument protection.

1.2.1 Instrument Protection Rules

1. Setup Keil μVision 4.0 or above software in personal computer.
2. Be sure the Keil μVision 4.0 or above is installed properly.
3. To initiate the programming you must create a project using the Keil μVision IDE.
4. Selecting the type of device you are working with.
5. Adding C files to your project.
6. Compiling and building the C project using Keil μVision IDE.
7. Generating the hex file using Keil μVision IDE.
8. Burning the hex code into 8051 microcontroller using flash magic.

1.3 Data Recording and Reports

1.3.1 The Laboratory Worksheets

Students must record their experimental values in the provided tables in this laboratory manual and reproduce them in the lab worksheets. Worksheets are integral to recording the methodology and results of an experiment. In engineering practice, the laboratory worksheet serves as an invaluable reference to the technique used in the lab and is essential when trying to duplicate a result or write a report. Therefore, it is important to learn to keep accurate data. Make plots of data and sketches when these are appropriate in the recording and analysis of observations. Note that the data collected will be an accurate and permanent record of the data obtained during the experiment and the analysis of the results. You will need this record when you are ready to prepare a lab worksheet.

1.3.2 The Lab Report

Reports are the primary means of communicating your experience and conclusions to other professionals. In this course you will use the lab report to inform your LTA about what you did and what you have learned from the experience. Engineering results are meaningless unless they can be communicated to others. You will be directed by your LTA to prepare a lab report on a few selected lab experiments during the semester.

Your laboratory report should be clear and concise. The lab report shall be typed on a word processor. As a guide, use the format on the next page. Use tables, diagrams, sketches, and plots, as necessary to show what you did, what was observed, and what conclusions you can draw from this. Even though you will work with one or more lab partners, your report will be the result of your individual effort in order to provide you with practice in technical communication.

Formatting and Style

- The lab report shall be hand written in a lab worksheet.
- The first line of each paragraph should have a left indent.
- All the tables should have titles and should be numbered. Tables should be labeled numerically as Table 1, Table 2, etc. Table captions appear above the table.
- Graphs should be presented as figures. All the figures should have titles and should be numbered. Figure captions appear below the figure. Graphs should have labeled axes and clearly show the scales and units of the axes.
- All the figures and tables must be centered on the page.
- Do not place screenshots of your lab worksheet.

Order of Lab Report Components

COVER PAGE: Cover page must include lab name and number, your name, and the date the lab was performed.
OBJECTIVE: Clearly state the experiment objective in your own words.
Prelab: Indicate the required knowledge for performing the concerned experiment.
EQUIPMENT USED: Indicate which equipment was used in performing the experiment.
BACKGROUND: Give the brief information about corresponding lab experiment.
PROCEDURE: Describe the experimental set up and procedure to perform the experiment.
RESULT: After completion of experiment, provide a summary of what was learned from this

part of the laboratory experiment.
PROBING FURTHER EXPERIMENTS: Questions pertaining to this lab must be answered at the end of laboratory report.

LAB-1 ORIENTATION

2.1 Introduction

In the first lab period, the students should become familiar with the location of equipment and components in the lab, the course requirements, and the teaching instructor. Students should also make sure that they have all of the co-requisites and pre-requisites for the course at this time.

2.2 Objective

To familiarize the students with the lab facilities, equipment, standard operating procedures, lab safety, and the course requirements.

2.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

2.4 Equipment needed

Xilinx software.
Personal computer.

2.5 Procedure

1. During the first laboratory period, the instructor will provide the students with a general idea of what is expected from them in this course. Each student will receive a copy of the syllabus, stating the instructor's contact information. In addition, the instructor will review the safety concepts of the course.

2. During this period, the instructor will briefly review the equipment which will be used throughout the semester. The location of instruments, equipment, and components (e.g. resistors, capacitors, connecting wiring) will be indicated. The guidelines for instrument use will be reviewed.

LAB-2 REALIZATION OF A BOOLEAN FUNCTION

3.1 Introduction

A multi variable Boolean function can be implemented through VHDL in two ways. First one is using primitive gates and the second one is using assign statements.

3.2 Objective

By the end of this lab, the student should learn how to Design and simulate the HDL code to realize three and four variable Boolean functions.

3.3 Prelab

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

3.4 Equipment needed

Xilinx software.
Personal computer.

3.5 Back ground

A multi variable Boolean function can be implemented through VHDL in two ways. First one is using primitive gates and the second one is using assign statements.
Gate primitives are predefined in VHDL, which are ready to use. They are instantiated like modules. There are two classes of gate primitives: Multiple input gate primitives and Single input gate primitives. Multiple input gate primitives include and, nand, or, nor, xor, and xnor. These can have multiple inputs and a single output. Single input gate primitives include not, buf, notif1, bufif1, notif0, and bufif0. These have a single input and one or more outputs.

Assign statements are used to define signal values as Boolean expressions. In the example: out = AS' + BS , out is defined by the function $AS' + BS$, but must be written in VHDL using the AND operator ("&"), OR operator ("|"), the XOR operator ("$\oplus$") and the NOT operator ("~").

3.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table

3.7 VHDL Program

```
// logic gates

library ieee;
use ieee.std_logic_1164.all;
entity vhdl3_1a is port(a,b:in std_logic; x:out std_logic);
end vhdl3_1a;
architecture behavior of vhdl3_1a is
begin
x <= a xor b;
end behavior;


library ieee;
use ieee.std_logic_1164.all;
entity vhdl3_1a is port(a,b:in std_logic; x:out std_logic);
end vhdl3_1a;
architecture behavior of vhdl3_1a is
begin
x<=a and b;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
entity vhdl3_1a is port(a,b:in std_logic; x:out std_logic);
end vhdl3_1a;
architecture behavior of vhdl3_1a is
begin
x<=a or b;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
entity vhdl3_1a is port(a,b:in std_logic;x:out std_logic);
end vhdl3_1a;
architecture behavior of vhdl3_1a is
begin
x<=a nand b;
end behavior;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity vhdl3_1a is port(a, b:in std_logic;x:out std_logic);
end vhdl3_1a;
architecture behavior of vhdl3_1a is
begin
x <= a nor b;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
entity vhdl3_1a is  port(a,b:in std_logic;x:out std_logic);
end vhdl3_1a;
architecture behavior of vhdl3_1a is
begin
x <= a xnor b;
end behavior;

library ieee;
use ieee.std_logic_1164.all;
entity vhdl3_1a is port(a :in std_logic;x:out std_logic);
end vhdl3_1a;
architecture behavior of vhdl3_1a is
begin
x<=not a;
end behavior;
```

3.8 Probing Further Experiments

Q1. What is the difference between main module and test bench module?.

Q2. What are the different tools available for simulation.

Q3. What is meant by universal gate? List them.

Lab 3 – DESIGN OF DECODER AND ENCODER

4.1 Introduction

Encoder circuit basically converts the applied information signal into a coded digital bit stream. Decoder performs reverse operation and recovers the original information signal from the coded bits.

4.2 Objective

To design and simulate the HDL code for the following combinational circuits

1. 3 to 8 Decoder
2. 8 to 3 Encoder (With priority and without priority)

4.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

4.4 Equipment needed

Xilinx software.
Personal computer.

4.5 Background

Program logic for Decoder
A decoder is a multiple-input, multiple-output logic circuit which converts coded inputs into coded outputs, where the input and output codes are different. The input code generally has fewer bits than the output code. Each input code word produces a different output code word, i.e., there is one-to-one mapping from input code words into output code words. This one-to-one mapping can be expressed in a truth table. The most common decoder circuit is an n-to-2n decoder or binary decoder. Such a decoder has an n-bit binary input code and a 1-out-of-2n output code. A binary decoder is used when you need to activate exactly one of 2noutputs based on an n-bit input value. Figure 4.1 shows the general structure of the 3 to 8 decoder circuit and its truth table.

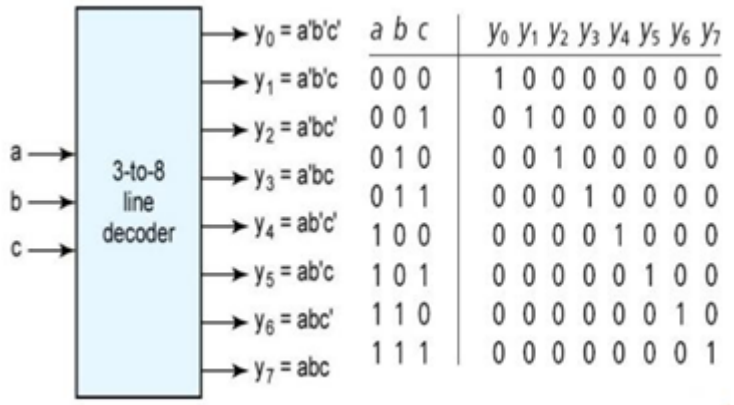


Figure 4.1: General Structure of 3 to 8 Decoder and its truth table

Program logic for Encoder

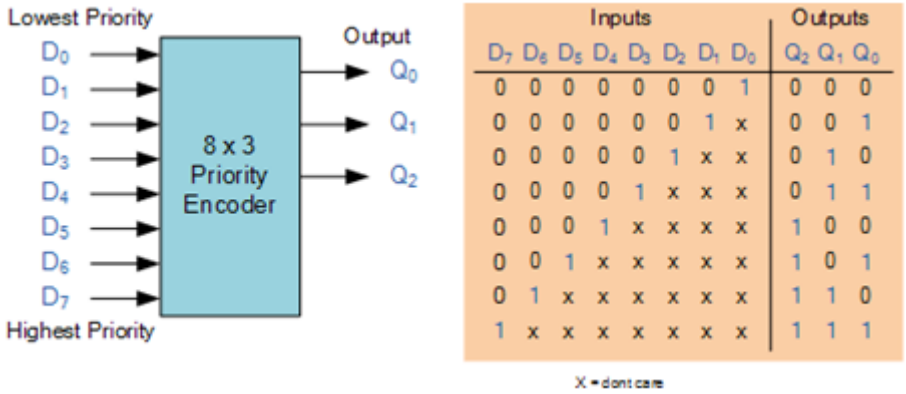
An encoder has M input and N output lines. Out of M input lines only one is activated at a time and produces equivalent code on output N lines. If a device output code has fewer bits than the input code has, the device is usually called an encoder. Example Octal-to-Binary take 8 inputs and provides 3 outputs. For an 8- to-3 binary encoder with inputs D0-D7 the logic expressions of the outputs XYZ are obtained by using the Table.

X = D4 + D5 + D6 + D7 Y= D2 + D3 + D6 + D7 Z = D1 + D3 + D5 + D7

Table 4.1: Truth Table for 8-3 Encoder with D7-D0 inputs

INPUTS								**OUTPUTS**		
D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**X**	**Y**	**Z**
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

One of the main disadvantages of standard digital encoders is that they can generate the wrong output code when there is more than one input present at logic level “1”. The Priority Encoder solves the problems mentioned above by allocating a priority level to each input. The priority encoders output corresponds to the currently active input which has the highest priority. So when an input with a higher priority is present, all other inputs with a lower priority will be ignored. The priority encoder comes in many different forms with an example of an 8- input priority encoder along with its truth table shown in Figure 4.2



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = don't care

Figure 4.2: General Structure of 3 to 8 Decoder and its truth table

Decoder or encoder can be designed using HDL through its truth table in two ways: one is using gate level modeling and another is by behavioral model.

4.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table.

4.7 VHDL Program

```
// 8 to 3 Encoder without priority
entity encoder is
Port ( A:in std_logic_vector(7 downto 0);
B:out std_logic_vector(2 downto 0));
end encoder;

architecture Behavioral of encoder is

begin
process(A)
begin
if A="10000000" then B<="000";
elsif A="01000000" then B<="001";
elsif A="00100000" then B<="010";
elsif A="00010000" then B<="011";
elsif A="00001000" then B<="100";
elsif A="00000100" then B<="101";
```

```
elsif A="00000010" then B<="110";
elsif A="00000001" then B<="111";
else B<="ZZZ";
end if;
end process;
end Behavioral;


// 3 to 8 decoder

entity decoder is
Port ( A:in std_logic_vector(2 downto 0);
B:out std_logic_vector(7 downto 0));
end decoder;

architecture Behavioral of decoder is
begin
process(A)
begin
if A="000" then B<="00000001";
elsif A="001" then B<="00000010";
elsif A="010" then B<="00000100";
elsif A="011" then B<="00001000";
elsif A="100" then B<="00010000";
elsif A="101" then B<="00100000";
elsif A="110" then B<="01000000";
elsif A="111" then B<="10000000";
else B<="11111111";
end if;
end process;
end Behavioral;
```

4.8 Further Probing Experiments

Q1. Create a VHDL module named h6to64 that represents a 6-to-64 binary decoder. Use the treelike structure in which the 6-to-64 decoder is built using nine instances of the 3 to 8 decoder.

Q2. What is the difference between the following two lines of VHDL code?
#5 a = b;
a = #5 b;

Q3. Construct a 4x16 decoder using two 3x8 decoder and additional logic. Show the schematic diagram neatly.

Lab 4 – DESIGN OF MULTIPLEXER AND DEMULTIPLEXER

5.1 Introduction

A Multiplexer is a circuit that accept many inputs but gives only one output. A Demultiplexer functions exactly in the reverse way of a multiplexer i.e., a demultiplexer accepts only one input and gives many outputs.

5.2 Objective

To write HDL codes for an 8X1 multiplexer and 1X8 demultiplexer and verify its functionality

5.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

5.4 Equipment needed

Xilinx software.
Personal computer.

5.5 Background

In the large-scale-digital systems, a single line is required to carry on two or more digital signals – and, of course! At a time, one signal can be placed on the one line. But, what is required is a device that will allow us to select; and, the signal we wish to place on a common line, such a circuit is referred to as multiplexer.

The function of a multiplexer is to select the input of any 'n' input lines and feed that to one output line. The function of a de-multiplexer is to inverse the function of the multiplexer and the shortcut forms of the multiplexer. The de-multiplexers are mux and demux. Some multiplexers perform both multiplexing and de-multiplexing operations. The main function of the multiplexer is that it combines input signals, allows data compression, and shares a single transmission channel.

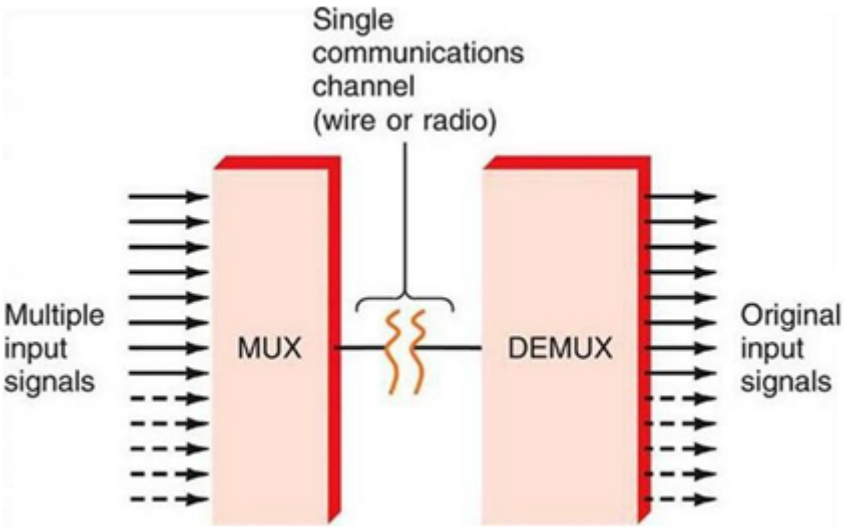


Figure 5.1: Multiplexer and De-multiplexer

The output value of a 8x1 multiplexer can be represented using the equation (4.1)

Y S2 S1 S0 Io S2 S1S0 I1 S2S1 S0 I2 S2S1S0 I3 S2 S1 S0 I4 S2 S1S0 I5 S2S1 S0 I6 S2S1S0 I7 . . . (4.1)
For the combination of selection input, the data line is connected to the output line. The 8x1 multiplexer requires 8 AND gates, one OR gate and 3 selection lines. As an input, the combination of selection inputs are giving to the AND gate with the corresponding input data lines.

In a similar fashion, all the AND gates are given connection. In this 8x1 multiplexer, for any selection line input, one AND gate gives a value of 1 and the remaining all AND gates give 0. And, finally, by using OR gate, all the AND gates are added; and, this will be equal to the selected value.

The demultiplexer is also called as data distributors as it requires one input, 3 selected lines and 8 outputs. De-multiplexer takes one single input data line, and then switches it to any one of the output line. 1-to-8 demultiplexer circuit diagram is shown below; it uses 8 AND gates for achieving the operation. The input bit is considered as data D and it is transmitted to the output lines.

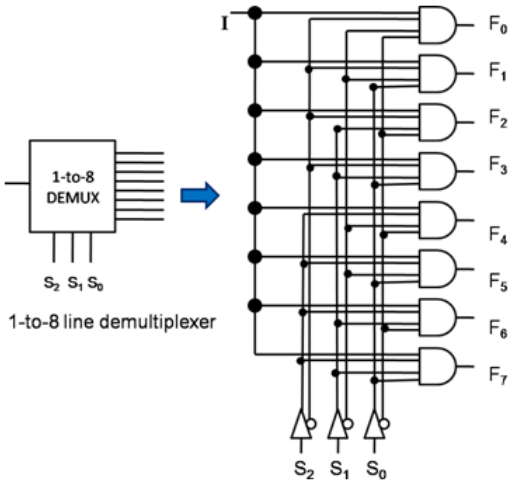


Figure 5.2: Demultiplexer circuit diagram

5.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table.

5.7 VHDL Program

```
// 8:1 multiplexer library ieee;
use ieee.std_logic_1164.all;
entity mux8_1 is
port(din:in std_logic_vector(7 downto 0);sel:in std_logic_vector(2 downt
end mux8_1;
architecture beh123 of mux8_1 is
begin
process(din,sel)
begin
case sel is
when"000"=>dout<=din(0);
when"001"=>dout<=din(1);
when"010"=>dout<=din(2);
when"011"=>dout<=din(3);
when"100"=>dout<=din(4);
when"101"=>dout<=din(5);
when"110"=>dout<=din(6);
when"111"=>dout<=din(7);
when others=>
dout<='z';
end case;
end process;
end beh123;

//1:8 Demultiplexer

architectural behavioral of dmux1 is
begin
y(0)<=f when s="000"else '0';
y(1)<=f when s="001"else '0';
y(2)<=f when s="010"else '0';
y(3)<=f when s="011"else '0';
y(4)<=f when s="100"else '0';
y(5)<=f when s="101"else '0';
```

```
y(6)<=f when s="110" else '0';
y(7)<=f when s="111" else '0';
end behavioral;

demultiplexer:
```

5.8 Further Probing Experiments

Q1. Implement the function f(A,B,C)= Σm(0,1,3,5,7) by using multiplexer.

Q2. Write code for 1x4 Multiplexer using different coding methods.

Q3. Design an OR gate from 2:1 MUX.

Lab 5 – DESIGN OF CODE CONVERTERS

6.1 Introduction

Gray codes are widely used to prevent spurious output from electromechanical switches and to facilitate error correction in digital communications such as digital terrestrial television and some cable TV systems.

6.2 Objective

To Design and simulate the HDL code for the following combinational circuits

a. 4 - Bit binary to gray code converter
b. 4 - Bit gray to binary code converter
c. Comparator

6.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

6.4 Equipment needed

Xilinx software.
Personal computer.

6.5 Background

Binary to gray code converter logic

This conversion method strongly follows the EX-OR gate operation between binary bits. The steps to perform binary to grey code conversion are given bellow.

a. To convert binary to grey code, bring down the most significant digit of the given binary number, because, the first digit or most significant digit of the grey code number is same as the binary number.

b. To obtain the successive grey coded bits to produce the equivalent grey coded number for the given binary, add the first bit or the most significant digit of binary to the second one and write down the result next to the first bit of grey code, add the second binary bit to third one and write down the result next to the second bit of grey code, follow this operation until the last binary bit and write down the results based on EX-OR logic to produce the equivalent grey

coded binary.

Gray to binary code converter logic

This conversion method also follows the EX-OR gate operation between gray & binary bits. The steps to perform gray code to binary conversion are given below.

a. To convert grey code to binary, bring down the most significant digit of the given gray code number, because, the first digit or the most significant digit of the gray code number is same as the binary number.

b. To obtain the successive second binary bit, perform the EX-OR operation between the first bit or most significant digit of binary to the second bit of the given gray code.

c. To obtain the successive third binary bit, perform the EX-OR operation between the second bit or most significant digit of binary to the third MSD (most significant digit) of grey code and so on for the next successive binary bits conversion to find the equivalent.

6.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table.

6.7 VHDL Program

```
// binary to gray code converter
entity btog is
Port ( b : in STD_LOGIC_VECTOR (3 downto 0);
g : out STD_LOGIC_VECTOR (3 downto 0));
end btog;
architecture Dataflow of btog is
begin
g(3)<=b(3);
g(2)<=b(3) xor b(2);
g(1)<=b(2) xor b(1);
g(0)<=b(1) xor b(0);
end Dataflow
//gray to binary converter
entity gtob is
Port ( g : in STD_LOGIC_VECTOR (3 downto 0);
b : inout STD_LOGIC_VECTOR (3 downto 0));
```

```
end gtob;
architecture Behavioral of gtob is
begin
b(3)<=g(3);
b(2)<=b(3)xor g(2);
b(1)<=b(2)xor g(1);
b(0)<=b(1)xor g(0);
end Behavioral;
```

6.8 Further Probing Experiments

Q1. Design BCD to Excess-3 code converter.

Q2. Design a BCD to seven segment code converter.

Q3. Design octal to binary code converter.

LAB-6 FULL ADDER AND FULL SUBTRACTOR DESIGN MODELING

7.1 Introduction

A combinational logic circuit that performs the addition of two single bits is called Half Adder. A combinational logic circuit that performs the addition of three single bits is called Full Adder. A full subtractor is a combinational circuit that performs subtraction of two bits, one is minuend and other is subtrahend, taking into account borrow of the previous adjacent lower minuend bit.

7.2 Objective

To write a HDL code to describe the functions of a full Adder and subtractor.

7.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

7.4 Equipment needed

Xilinx software.
Personal computer.

7.5 Background

A full adder consists of 3 inputs and 2 outputs. Fig 5.1 shows truth table of full adder. Use "assign" keyword to represent design in dataflow style. The output signal expressions can be obtained from the truth table using K-maps.

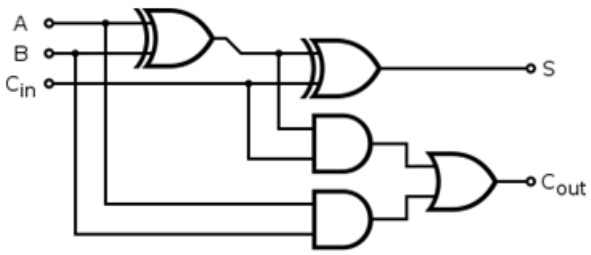


Figure 7.1: Logic diagram for 1-bit full adder

Table 7.1: Truth table for 1-bit full adder

Inputs			**Outputs**	
A	B	Cin	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

This is not practical to perform subtraction only between two single bit binary numbers. Instead binary numbers are always multibits. The subtraction of two binary numbers is performed bit by bit from right (LSB) to left (MSB). During subtraction of same significant bit of minuend and subtrahend, there may be one borrow bit along with difference bit. This borrow bit (either 0 or 1) is to be added to the next higher significant bit of minuend and then next corresponding bit of subtrahend to be subtracted from this. It will continue up to MSB. The combinational logic circuit performs this operation is called full subtractor. Hence, full subs tractor is similar to half subs tractor but inputs in full subs tractor are three instead of two.

Two inputs are for the minuend and subtrahend bits and third input is for borrowed which comes from previous bits subtraction. The outputs of full adder are similar to that of half adder, these are difference (D) and borrow (b).

The combination of minuend bit (A), subtrahend bit (B) and input borrow (bi) and their respective differences (D) and output borrows (b) are represented in a truth table 7.2. The output signal expressions can be obtained from the truth table using K-maps.

Table 7.2: Truth table for 1-bit subtractor adder

Inputs			**Outputs**	
A	B	C (Borrow in)	Difference	Borrow out
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

7.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.

3. Create another module referred as test bench to verify the functionality.

4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table

7.7 VHDL Program

```
// full adder VHDL Program using Dataflow Modelling:

entity FULLADDER is
Port(A,B,C_in: in std_logic;
S,C_out: out std_logic);
end FULLADDER;

architecture Behavioral of FULLADDER is

begin
S<=(A xor B) xor C_in;
C_out<=(A and B) or (C_in and (A xor B));
end Behavioral;

// full adder VHDL Program using Behavioral Modelling:
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FULLADDER_BEHAVIORAL_SOURCE is
Port (A : in  STD_LOGIC_VECTOR (2 downto 0);
      y : out  STD_LOGIC_VECTOR (1 downto 0));
end FULLADDER_BEHAVIORAL_SOURCE;
architecture Behavioral of FULLADDER_BEHAVIORAL_SOURCE is
Begin
process (A)
begin
if (A = "001" or A = "010" or A = "100" or A = "111") then
y(1) <= '1';
 else
y(1) < = '0';
end if;
if (A = "011" or A = "101" or A = "110" or A = "111") then
 y(0) <= '1';
else y(0) <= '0';
end if;
end process;
end Behavioral

// VHDL code for full adder using the structural method using two half a
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity HA is
Port ( A,B : in  STD_LOGIC;
S,C : out  STD_LOGIC);
end HA;
architecture dataflow of HA is
begin
S <= A XOR B;
C <= A AND B;
end dataflow;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ORGATE is
Port ( X,Y : in STD_LOGIC;
Z : out STD_LOGIC);
end ORGATE; architecture dataflow of ORGATE is
begin
Z <= X OR Y;
end dataflow;
library IEEE; use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity FAdder is
Port ( FA, FB, FC : in STD_LOGIC;
FS, FCA : out STD_LOGIC);
end FAdder;
architecture structural of FAdder is
component HA is
Port ( A,B : in STD_LOGIC;
S,C : out STD_LOGIC);
end component;
component ORGATE is
Port ( X,Y: in STD_LOGIC;
Z: out STD_LOGIC);
end component;
SIGNAL S0,S1,S2:STD_LOGIC;
begin
U1:HA PORT MAP(A=>FA,B=>FB,S=>S0,C=>S1);
U2:HA PORT MAP(A=>S0,B=>FC,S=>FS,C=>S2);
U3:ORGATE PORT MAP(X=>S2,Y=>S1,Z=>FCA);
end structural;

//full subtractor using dataflow
entity FULLSUBTRACTOR is
Port(A,B,C:in std_logic;
D,B_out:out std_logic);
end FULLSUBTRACTOR;

architecture Behavioral of FULLSUBTRACTOR is
```

```
begin
Process(A,B,C)
begin
D<=A xor B xor C;
B_out<=((not A) and B) or (C and (A xnor B));
end Process;
end Behavioral;
//full subtractor using Behavioral Modelling
architecture Behavioral of FULLSUBTRACTOR_BEHAVIORAL_SOURCE is
Begin
process (A)
begin
if (A = "001" or A = "010" or A = "111") then
Y <= "11";
elsif (A = "011") then
Y <= "01";
elsif (A = "100") then
Y <= "10";
else Y <= "00";
end if;
end process;
end Behavioral;
```

7.8 Probing Further Experiments

Q1. Write the sum and carry expression for 1-bit full adder.

Q2. Write the difference and barrow out expressions for 1-bit subtractor

Q3. Design a 4-bit ripple carry adder using full adders.

LAB-7 DESIGN OF 8-BIT ARITHMETIC LOGIC UNIT

8.1 Introduction

An arithmetic logic unit (ALU) is at the heart of a modern microprocessor, and the adder cell is the elementary unit of an ALU.

8.2 Objective

An arithmetic logic unit (ALU) is at the heart of a modern microprocessor, and the adder cell is the elementary unit of an ALU.

8.3 Prelab

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

8.4 Equipment needed

Xilinx software.
Personal computer.

8.5 Back ground

An arithmetic logic unit (ALU) is a combinational digital electronic circuit that performs arithmetic and bitwise operations on integer binary numbers. This is in contrast to a floating-point unit (FPU), which operates on floating point numbers. An ALU is a fundamental building block of many types of computing circuits, including the central processing unit (CPU) of computers, FPUs, and graphics processing units (GPUs). A single CPU, FPU or GPU may contain multiple ALUs.

The inputs to an ALU are the data to be operated on, called operands, and a code (opcode) indicating the operation to be performed and, optionally, status information from a previous operation; the ALU's output is the result of the performed operation. In many designs, the ALU also exchanges additional information with a status register, which relates to the result of the current or previous operations.
A number of basic arithmetic and bitwise logic functions are commonly supported by ALUs. Basic, general purpose ALUs typically includes these operations in their properties:

- Arithmetic operations

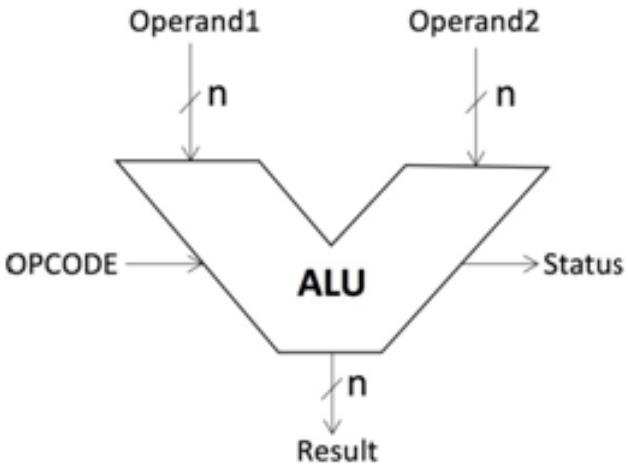


Figure 8.1: Arithmetic logic unit block diagram

- Bitwise logical operations

- Bit shift operations

In this lab, students have to design an 8-bit ALU to implement the following operations:

Control	Instruction	Operation
000	Add	Ouput<= A+B+Cin (Cout is carry)
001	Sub	Output <= A-B-C (Cou is barrow)
010	Or	Output <= A or B
011	And	Output <= A and B
100	Shl	Output <= A[7:0] & '0'
101	Shr	Output <= '0' & A[7:1]
110	Rol	Output <= A[2:0] & A[7]
111	Ror	Output <= A[0] & A[7:1]

Figure 8.2: ALU Instructions

Table 1 also illustrates the encoding of the control input The 4 - bit ALU has the following inputs:

- A: 8-bit input

- B: 8-bit input

- Cin: 1-bit input

- Output: 8-bit output

- Cout: 1-bit output

- Control: 3-bit control input

The following points should be taken care of:

- Use a case statement (or a similar 'combinational' statement) that checks the input combination of "Code" and acts on A, B, and Cin as described in Table1.

• The above circuit is completely combinational. The output should change as soon as the code combination or any of the input changes.

• You can use arithmetic and logical operators to realize your design.

8.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table

8.7 VHDL Program

```
//8 bit ALU

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity alu is
Port ( p,q : in STD_LOGIC_VECTOR (7 downto 0);
r : in STD_LOGIC_VECTOR (2 downto 0);
result : out STD_LOGIC_VECTOR (7 downto 0));
end alu;

architecture Behavioral of alu is

begin
process(p,q,r)
begin
case r is
when "000" => result<=p+q;
when "001" => result<=p-q;
```

```
when "010" => result<=p and q;
when "011" => result<=p or q;
when "100" => result<=p xor q;
when "101" => result<=p nor q;
when "110" => result<=p nand q;
when "111" => result<=not p;
when others => result <="00000000";
end case;
end process;
end Behavioral;
```

8.8 Probing Further Experiments

Q1. Write a HDL code to implement bitwise logical operations using ALU.

Q2. Write a HDL code to implement bit shift operations using ALU.

Q3. Design the 4-bit ALU.

LAB-8 HDL MODEL FOR FLIP FLOPS

9.1 Introduction

A flip-flop circuit can be constructed from two NAND gates or two NOR gates. These flip-flops are shown in Figure. Each flip-flop has two outputs, Q and Q', and two inputs, set and reset. This type of flip-flop is referred to as an SR flip-flop.

9.2 Objective

To write HDL codes for SR, JK, D, T flip flops and verify its functionality

9.3 Prelab

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

9.4 Equipment needed

Xilinx software.
Personal computer.

9.5 Back ground

AEach flip-flop stores a single bit of data, which is emitted through the Q output on the output section side. Normally, the value can be controlled via the inputs to the input side. In particular, the value changes when the clock input, marked by a triangle on each flip-flop, rises from 0 to 1 (or otherwise as configured); on this rising edge, the value changes according to the tables below. Table 9.1 Truth tables of D, T, SR, JK flip flops. Another way of describing the different behavior of the flip-flops is in English text.

D Flip-Flop: When the clock triggers, the value remembered by the flip-flop becomes the value of the D input (Data) at that instant.
T Flip-Flop: When the clock triggers, the value remembered by the flip-flop either toggles or remains the same depending on whether the T input (Toggle) is 1 or 0.
J-K Flip-Flop: When the clock triggers, the value remembered by the flip-flop toggles if the J and K inputs are both 1, remains the same if they are both 0; if they are different, then the value becomes 1 if the J (Jump) input is 1 and 0 if the K (Kill) input is 1. **S-R Flip-Flop:** When the clock triggers, the value remembered by the flip-flop remains unchanged if R and S are both 0, becomes 0 if the R input (Reset) is 1, and becomes 1 if the S input (Set) is 1. The behavior in unspecified if both inputs are 1.

D FF	
D	Q_n
1	0
0	0

T FF	
T	Q_n
0	Q
1	$\overline{Q_n}$

SR FF		
S	R	Q_n
0	0	Q_n
0	1	0
1	0	1
1	1	?

JK FF		
J	K	Q_n
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q_n}$

Figure 9.1: Truth Tables of Flip Flops

9.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table

9.7 VHDL Program

```
//SR flipflop

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity sr_ff  is
port(s,r,clk,clr : in std_logic;
q:inout std_logic:=  0  ;
qbar : out std_logic);
end sr_ff;

architecture beh of sr_ff is
begin
process(clk)
begin
if(clk'event and clk='1') then
if(clr='1') then
q<='0'; qbar<='1';
elsif(clr='0' and s='0' and r='0')then
q<=q;qbar<=not q;
elsif(s='0' and r='1')then
```

```
q<='0';qbar<='1';
elsif(s='1' and r='0')then
q<='1';qbar<='0';
else q<='Z';qbar<='Z';
end if;
end if;
end process;
end behavioral;

//JK flipflop

library IEEE;
use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity jkff is
Port ( jk : in STD_LOGIC_VECTOR(1 downto 0); Clk,clr : in STD_LOGIC;
q : inout STD_LOGIC:='0'); end jkff;

architecture Behavioral of jkff is begin
process(clk,jk) begin
if(clk'event and clk='1') then if(clr='1')then q<='0';
else
case jk is
when "00"=>q<=q; when "01"=>q<='0'; when "10"=>q<='1';
when "11"=>q<= not q; when others=>null;
end case; end if;
end if;
end process;
end Behavioral;

//D flipflop

library IEEE;
use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dff is
Port ( d : in STD_LOGIC; clk : in STD_LOGIC; clr: in STD_LOGIC;
q : input STD_LOGIC:=   0    qbar:out std_logic);
end dff;

architecture Behavioral of dff is begin
process(clk,clr) begin
if(clk'event and clk='1') then if clr='1' then q<='0';
else q<=d; qbar<= not d; end if;
end if;
endprocess;
end Behavioral;

//T flipflop
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL; use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tff is
Port ( t : in STD_LOGIC; clk,clr : in  STD_LOGIC; q : inout STD_LOGIC:='
end tff;
architecture Behavioral of tff is
begin
process(clk,t) begin
if(clk'event and clk='1') then if(clr=  1  )then q<=  0  ;qbar<=  1
elsif(t='0')then
q<=q;qbar<=not q; else  q<=not q;qbar<=q;
if(clk'event and clk='1')
then if(clr=  1  )
then q<=  0  ;
qbar<=  1  ;
elsif(t='0')
then q<=q;
qbar<=not q; else       q<=not q;qbar<=q;
end if;
end if;
end process; end Behavioral;
```

9.8 Probing Further Experiments

Q1. Convert a given J-K flip-flop in to a D flip-flop using additional logic if necessary?

Q2. Convert a given J-K flip-flop in to a T flip-flop using additional logic if necessary?

Q3. Convert a given D flip-flop in to a T flip-flop using additional logic if necessary?

LAB-9 DESIGN OF COUNTERS

10.1 Introduction

A 4-bit Synchronous up counter start to count from 0 (0000 in binary) and increment or count upwards to 15 (1111 in binary) and then start new counting cycle by getting reset. Its operating frequency is much higher than the same range Asynchronous counter.

10.2 Objective

To write HDL codes for the following counters.

i) Binary counter

ii) BCD counter (Synchronous reset and asynchronous reset)

10.3 Prelab

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

10.4 Equipment needed

Xilinx software.
Personal computer.

10.5 Back ground

Counter is a sequential circuit. A digital circuit which is used for counting pulses is known as counter. Counter is the widest application of flip-flops. It is a group of flip- flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.

- Synchronous counters.

Asynchronous counters are called as ripple counters, the first flip-flop is clocked by the external clock pulse and then each successive flip-flop is clocked by the output of the preceding flip-flop. The term asynchronous refers to events that do not have a fixed time relationship with each other. An asynchronous counter is one in which the flip-flops within the counter do not change states at exactly the same time because they do not have a common clock pulse In synchronous counters, the clock inputs of all the flip-flops are connected together and are triggered by the input pulses. Thus, all the flip-flops change state simultaneously (in parallel).

A counter is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number clock pulses arrived. A specified sequence of states appears as the counter output. The name counter is generally used for clocked sequential circuit whose state diagram contains a single cycle. The modulus of a counter is the number of states in the cycle. A counter with m states is called a modulo-m counter or divide-by-m counter. A counter with a non-power-of-2 modulus has extra states that are not used in normal operation. There are two types of counters, synchronous and asynchronous. In synchronous counter, the common clock is connected to all the flip-flops and thus they are clocked simultaneously.

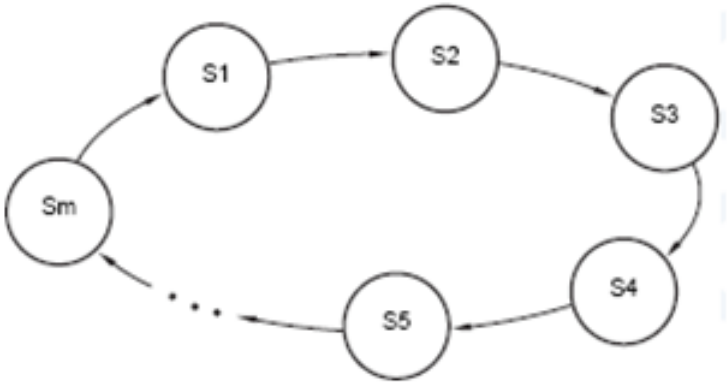


Figure 10.1: General structure of a counter's state diagram – a single cycle

Asynchronous Decade Counters

The modulus is the number of unique states through which the counter will sequence. The maximum possible number of states of a counter is 2nwhere n is the number of flip-flops. Counters can be designed to have a number of states in their sequence that is less than the maximum of 2n. This type of sequence is called a truncated sequence. One common modulus for counters with truncated sequences is 10 (Modules10). A decade counter with a count sequence of zero (0000) through 9 (1001) is a BCD decade counter because its 10-state sequence produces the BCD code. To obtain a truncated sequence, it is necessary to force the counter to recycle before going through all of its possible states. A decade counter requires 4flip-flops. One way to make the counter recycle after the count of 9 (1001) is to decode count 10 (1010) with a NAND gate-and connect the output of the NAND gate to the clear (CLR) inputs of the flip-flops, as shown in Figure 10.2

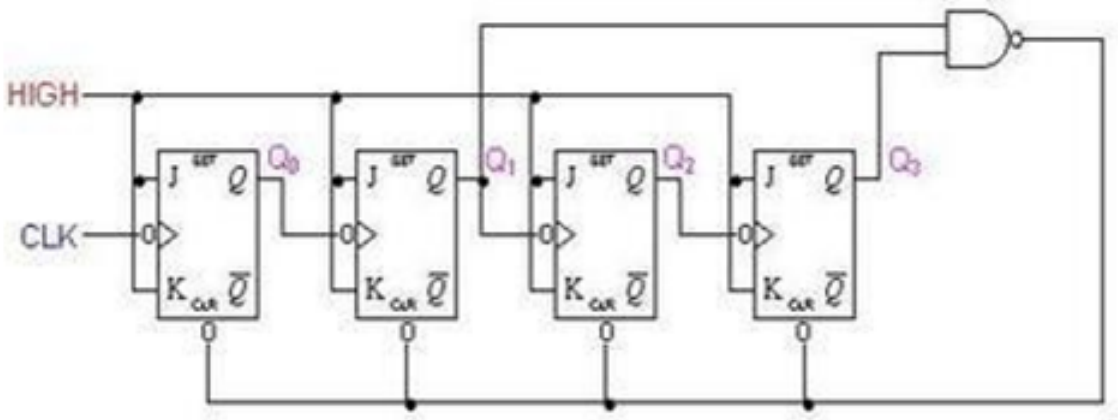


Figure 10.2: Asynchronous Decade Counter

Synchronous Decade Counters

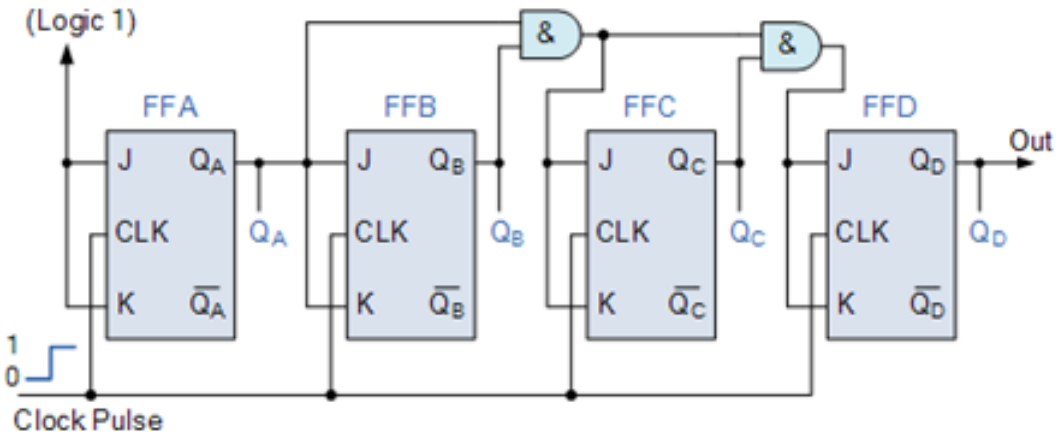


Figure 10.3: synchronous Decade Counter

It can be seen from Figure 10.3, that the external clock pulses (pulses to be counted) are fed directly to each of the J-K flip-flops in the counter chain and that both the J and K inputs are all tied together in toggle mode, but only in the first flip-flop, flip- flop FFA (LSB) are they connected HIGH, logic "1" allowing the flip-flop to toggle on every clock pulse. Then the synchronous counter follows a predetermined sequence of states in response to the common clock signal, advancing one state for each pulse.
The J and K inputs of flip-flop FFB are connected directly to the output QA of flip- flop FFA, but the J and K inputs of flip-flops FFC and FFD are driven from separate AND gates which are also supplied with signals from the input and output of the previous stage. These additional AND gates generate the required logic for the JK inputs of the next stage.
If we enable each JK flip-flop to toggle based on whether or not all preceding flip- flop outputs (Q) are "HIGH" we can obtain the same counting sequence as with the asynchronous circuit but without the ripple effect, since each flip-flop in this circuit will be clocked at exactly the same time.

10.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table

10.7 VHDL Program

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.numeric_STD.ALL;
entity COUNTER is
Port (clk,res:IN STD_LOGIC;
count:INOUT STD_LOGIC_VECTOR(3 downto 0) );
end COUNTER;

architecture Behavioral of COUNTER is
begin
Process(clk,res)
begin
if (res='1') then
count<="0000";
elsif (clk'event and clk='1') then
count<=count+1;
end if;
end process;
end Behavioral;

//asynchrolnous counter using jk flipflop
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity jkc is
port ( clock : in std_logic;
reset : in std_logic;
count : out std_logic_vector(3 downto 0)
);
end jkc;

architecture rtl of jkc is
component jkff
port(
clock : in std_logic;
reset : in std_logic;
j     : in std_logic;
k     : in std_logic;
q     : out std_logic
);
end component;

signal temp : std_logic_vector(3 downto 0) := "0000";

begin
```

```
d0 : jkff
port map (
reset  => reset,
clock  => clock,
j      => '1',
k      => '1',
q      => temp(3)
);

d1 : jkff
port map (
reset  => reset,
clock  => temp(3),
j      => '1',
k      => '1',
q      => temp(2)
);

d2 : jkff
port map (
reset  => reset,
clock  => temp(2),
j      => '1',
k      => '1',
q      => temp(1)
);

d3 : jkff
port map (
reset  => reset,
clock  => temp(1),
j      => '1',
k      => '1',
q      => temp(0)
);

count(3) <= temp(0);
count(2) <= temp(1);
count(1) <= temp(2);
count(0) <= temp(3);
end rtl;
```

10.8 Probing Further Experiments

Q1. Design and implement a synchronous 3 – bit up/down counter using J-K flip- flops.

Q2. Design mod-5 synchronous counter using T FF.

Q3. Implement a Johnson counter.

LAB-10 HDL CODE FOR UNIVERSAL SHIFT REGISTER

11.1 Introduction

a universal shift register is a combined design of serial in serial out (SISO), serial in parallel out (SIPO), parallel in serial out (PISO) , and parallel in parallel out (PIPO).

11.2 Objective

Ro design and simulate the HDL code for universal shift register

11.3 Prelab

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

11.4 Equipment needed

Xilinx software.
Personal computer.

11.5 Back ground

Universal Shift Register is a register which can be configured to load and/or retrieve the data in any mode (either serial or parallel) by shifting it either towards right or towards left. In other words, a combined design of unidirectional (either right- or left- shift of data bits as in case of SISO, SIPO, PISO, PIPO) and bidirectional shift register along with parallel load provision is referred to as universal shift register.

The working of this shift register is explained by the Table 11.1. The corresponding truth table and the wave forms are given by Table 11.2

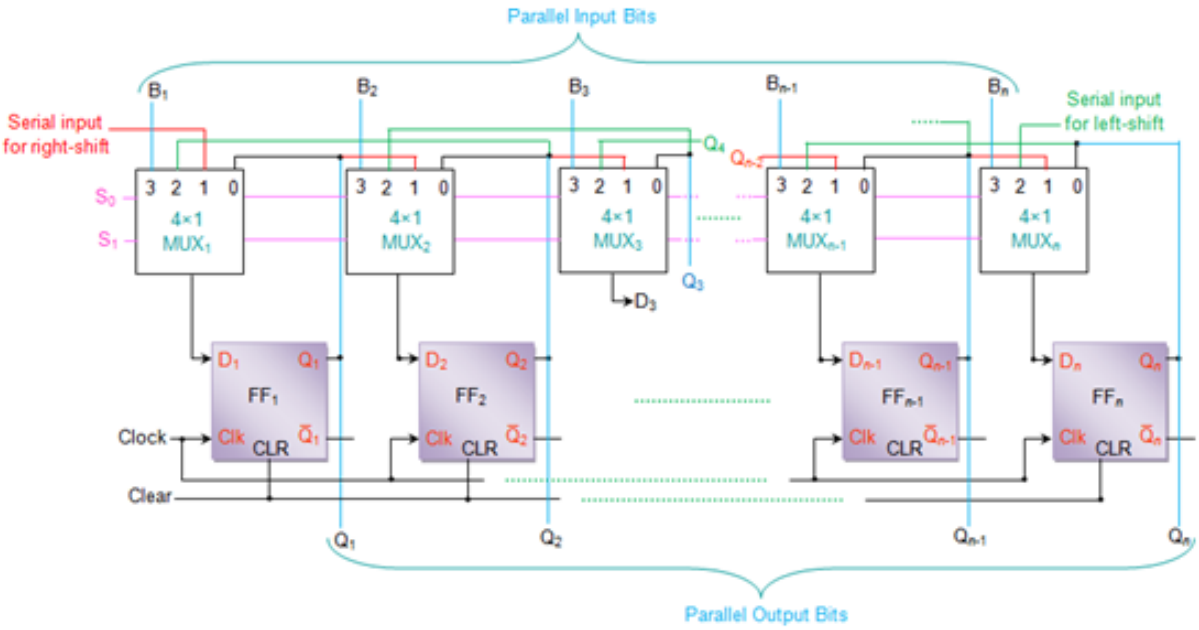


Figure 11.1: N-Bit Universal Shift register

Table 11.1: **Functional table for n-bit universal shift register**

Select Lines		**Functionality**
S_0	S_0	
0	0	No change for any number of clock cycles as the outputs of the flip-flops are back-fed to themselves
0	1	Data bits within the register shift right for each clock tick with the serial input bits being provided at D1 via MUX1
1	0	Data bits within the register shift left for each clock tick with the serial input bits being provided at Dn via MUXn
1	1	Bits of the data word to be stored are fed in parallel format through pin number 3 of each MUX at the rising edge of the clock

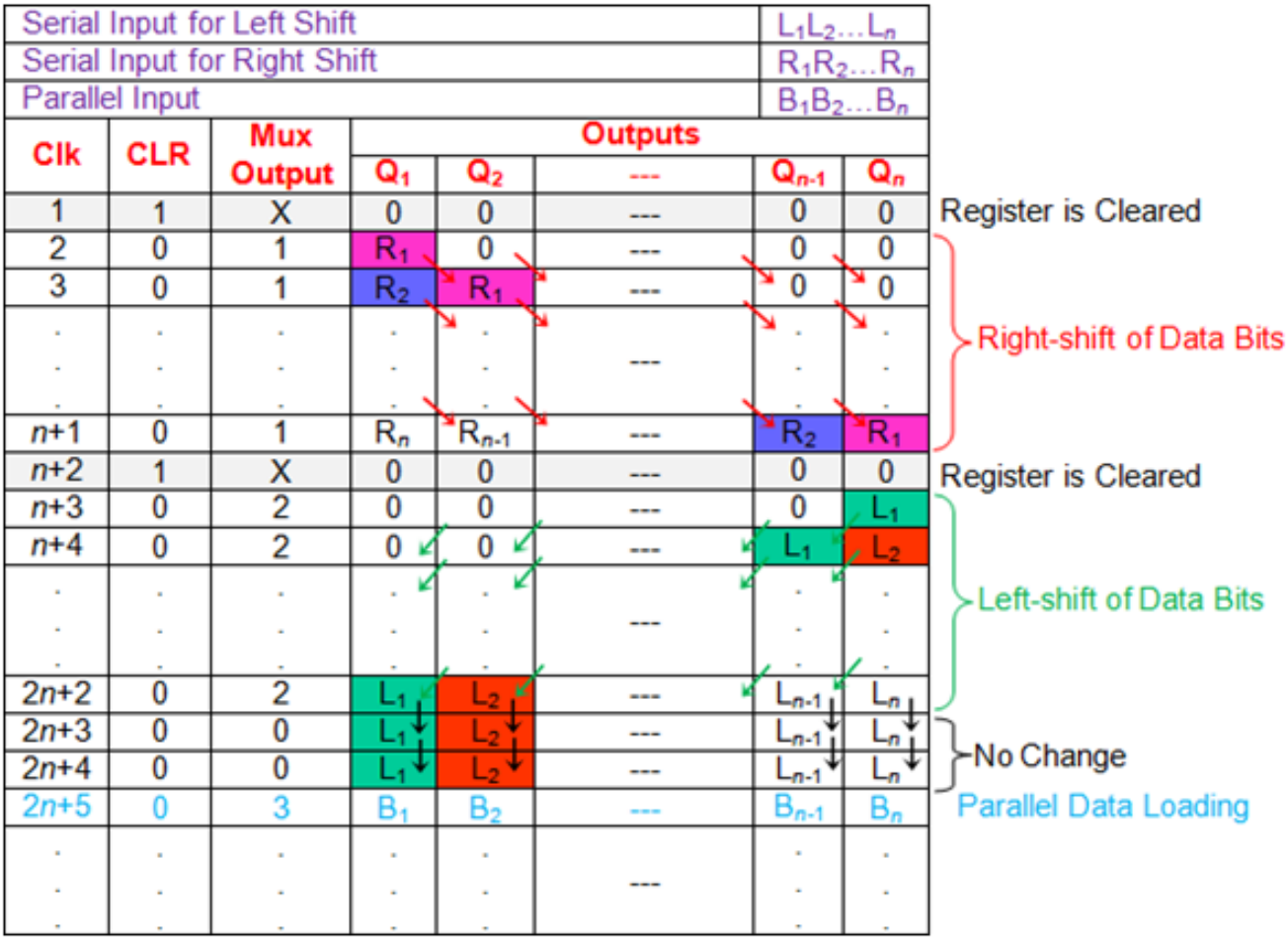


Figure 11.2: Truth table for n-bit universal shift register

11.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table

11.7 VHDL Program

```
entity USR is
Port ( clk : in  STD_LOGIC;
rst : in  STD_LOGIC;
sir : in  STD_LOGIC;
sil : in  STD_LOGIC;
d : in  STD_LOGIC_VECTOR (3 downto 0);
q : out  STD_LOGIC_VECTOR (3 downto 0);
s : in  STD_LOGIC_VECTOR (1 downto 0));
end USR;

architecture Behavioral of USR is
signal temp: std_logic_vector(3 downto 0);

begin
process(rst,clk,s,d,sir,sil)

begin

if rst='1' then
temp<= "0000";
q<= "0000";

elsif (clk='1' and clk'event) then

case s is
-- PARALLEL LOAD
when "11" =>
temp <= d;
q <= temp;

-- SHIFT LEFT        [0] [0] [0] [0]
--                   [0] [0] [0] [sil]
when "01" =>
temp <= d;
temp(3 downto 1) <= temp(2 downto 0);
```

```
temp(0) <= sil;
q <= temp;

-- SHIFT RIGHT        [0] [0] [0] [0]
--                  [sir] [0] [0] [0]
when "10" =>
temp <= d;
temp(2 downto 0) <= temp(3 downto 1);
temp(3) <= sir;
q <= temp;
-- HOLD
when "00" =>
temp <= temp;
q <= temp;

when others => null;

end case;
end if;
end process;

end Behavioral;
```

11.8 Probing Further Experiments

Q1. Design a circular shift right register using JK flip flop.

Q2. Design a circular left right register using JK flip flop.

Q3. Write a HDL code to perform serial in parallel out (SIPO) operation in universal shift register

Lab 11 – HDL CODE FOR CARRY LOOK AHEAD ADDER

12.1 Introduction

A carry-lookahead adder (CLA) or fast adder is a type of electronics adder used in digital logic. The carry-lookahead adder calculates one or more carry bits before the sum, which reduces the wait time to calculate the result of the larger-value bits of the adder.

12.2 Objective

To design and simulate the HDL code for carry look ahead adder

12.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

12.4 Equipment needed

Xilinx software.
Personal computer.

12.5 Background

Ripple-carry addition suffers from an impractical propagation delay cause by the sequential generation of arithmetic carries. In other words, ci+1 is dependent on ci , which is further dependent on ci+1 , etc. The effect of this carry chain is a propagation delay that has a linear dependency on n, the bit width of the adder. Therefore, methods that compute the arithmetic carries in parallel have potential performance benefits over ripple-carry addition.
As the name implies, carry-look ahead is one such technique for high-speed addition that computes arithmetic carries in a parallel fashion. To understand how exactly a carry-look ahead adder works, consider the addition of two numbers, X and Y, such that xi is the ith ith binary digit of X, and yi is the ith binary digit of Y. The (i +1)th arithmetic carry is ci+1 and is computed as follows:

ci+1 = xi yi + xici + yici (1)
= xi yi + (xi + yi)ci (2)

The effect of simply factoring out ci from the last two terms in expression (1) is shown in expression (2). Now observe that conditions exists: ci+1 is logic '1' if either of the two
1. xi yi is logic '1'
2. xi + yi is logic '1'1 and there is a previous carry (i.e. ci =1)

Therefore, xi yi is referred to as generate function because when '1', a carry is generated, while xi + yi is referred to as the propagate function because when '1', it will propagate a carry. In mathematical terms, we see that gi + xi yi (3)
pi + xi + yi (4)
ci+1 + gi + pici (5)

Clearly, expressions (3) and (4) do not depend on the carry in the previous bit position and thus, can be generated in parallel. It turns out, we can write expression (5) for the first four carries in such a way that they, too, do not depend on one another, but rather only depend on the input carry, c0, and the gi and pi . Examine the expressions below to convince yourself of this.
c1 = g0 + p0c0 (6)
c2 = g1 + p1c1 + g1 + p1g0 + p1 p0c0 (7)
c3 = g2 + p2c2 + g2 + p2 g1 + p2 p1g0 + p2 p1 p0c0 (8)
c4 = g3 + p3c3 + g3 + p3 g2 + p3 p2 g1 + p3 p2 p1g0 + p3 p2 p1 p0c0 (9)

Although the expression for ci becomes increasingly complex, the theoretical gate-delay for each of the above expressions, given the gi 's, pi 's, and c0 , is g =2. However, the increased complexity is reflected in the number of inputs to each gate (i.e. the gate fan-in) and the number of gates required. Figure 1 illustrates this point with the gate-level schematic for each of the sub-modules within a 4-bit carry-look ahead adder. One thing to note is that:
pi = xi yi
si = pi ci

In other words, expression (10) is being used in lieu expression (4). It turns out that expression (5) works correctly in either case, and the former allows the Sum to be computed with expression (11). Before moving on, let us try to understand how data flows through the 4-bit carry-look ahead adder. To do so, we enumerate through the steps below:
Data arrives at the Generate/Propagate Unit, and the gi 's and pi 's, are computed in one gate-delay (i.e.$\Delta g = 1$).

The gi 's and pi 's are forwarded to the Carry-Lookahead Unit, which generates all of the carries in two gate-delays, $\Delta g = 2$.

The carries are then fed into the Summation Unit, which computes the sum bits, the si 's, in one gate-delay $\Delta g = 1$.

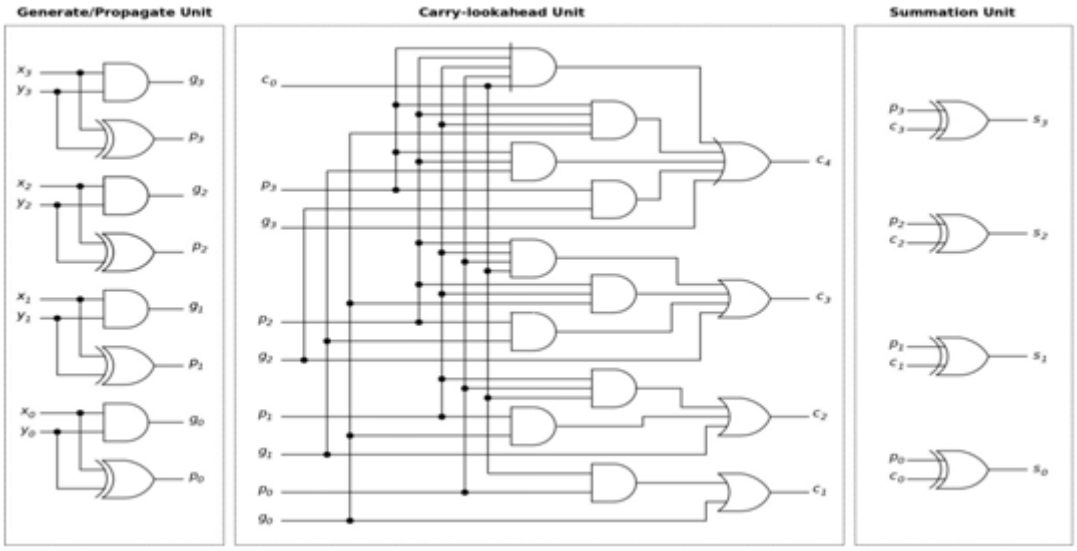


Figure 12.1: Carry-lookahead Adder

For simplicity, we are assuming that all gates have the same delay time. This assumption may or may not be true depending on the target technology that is being used to implement your logic. However, for the sake of comparison with other addition techniques, this model works well. Summarizing the above steps, we can see that the propagation delay for a 4-bit adder is no longer determined by a carry chain and is only four gate-delays, (g= 4). The pre-lab assignment will include an exercise which asks you to look at the gate count of a 4-bit Carry- Lookahead Adder.

12.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table.

12.7 VHDL Program

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Carry_Look_Ahead is
Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
B : in STD_LOGIC_VECTOR (3 downto 0);
Cin : in STD_LOGIC;
S : out STD_LOGIC_VECTOR (3 downto 0);
```

```
Cout : out STD_LOGIC);
end Carry_Look_Ahead;

architecture Behavioral of Carry_Look_Ahead is

component Partial_Full_Adder
Port ( A : in STD_LOGIC;
B : in STD_LOGIC;
Cin : in STD_LOGIC;
S : out STD_LOGIC;
P : out STD_LOGIC;
G : out STD_LOGIC);
end component;

signal c1,c2,c3: STD_LOGIC;
signal P,G: STD_LOGIC_VECTOR(3 downto 0);
begin

PFA1: Partial_Full_Adder port map( A(0), B(0), Cin, S(0), P(0), G(0));
PFA2: Partial_Full_Adder port map( A(1), B(1), c1, S(1), P(1), G(1));
PFA3: Partial_Full_Adder port map( A(2), B(2), c2, S(2), P(2), G(2));
PFA4: Partial_Full_Adder port map( A(3), B(3), c3, S(3), P(3), G(3));

c1 <= G(0) OR (P(0) AND Cin);
c2 <= G(1) OR (P(1) AND G(0)) OR (P(1) AND P(0) AND Cin);
c3 <= G(2) OR (P(2) AND G(1)) OR (P(2) AND P(1) AND G(0)) OR (P(2) AND P
Cout <= G(3) OR (P(3) AND G(2)) OR (P(3) AND P(2) AND G(1)) OR (P(3) AND

end Behavioral;
```

12.8 Further Probing Experiments

Q1. Design a ripple carry adder and mention its disadvantage.

Q2. Design 4-bit ripple carry adder using HDL.

Q3. Design 4-bit carry look ahead adder using HDL.

LAB-12 HDL CODE TO DETECT A SEQUENCE

13.1 Introduction

A sequence detector is a sequential state machine that takes an input string of bits and generates an output 1 whenever the target sequence has been detected.

13.2 Objective

To perform the design flow to generate state machines in VHDL code to detect the given sequence of bits.

13.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

13.4 Equipment needed

Xilinx software.
Personal computer.

13.5 Background

As an illustrative example a sequence detector for bit sequence '1011' is described. Every clock-cycle a value will be sampled, if the sequence '1011' is detected a '1' will be produced at the output for 1 clock-cycle. There are two methods to design state machines, first is Mealy and second is Moore style. We will give you an example for both styles.
Following is the behavior description of the sequencer for a Mealy style implementation and the state diagram is shown in figure 13.1.

When in initial state (S0) the machine gets the input of '1' it jumps to the next statewith the output equal to '0'. If the input is '0' it stays in the same state. When in 2nd state (S1) the machine gets an input of '0' it jumps to the 3rd state withthe output equal to '0'. If it gets an input of '1' it stays in the same state. When in the 3rd state (S2) the machine gets an input of '1' it jumps to the 4th statewith the output equal to '0'. If the input received is '0' it goes back to the initial state.
When in the 4th state (S3) the machine gets an input of '1' it jumps back to the 2ndstate, with the output equal to '1'. If the input received is '0' it goes back to the 3rd state.

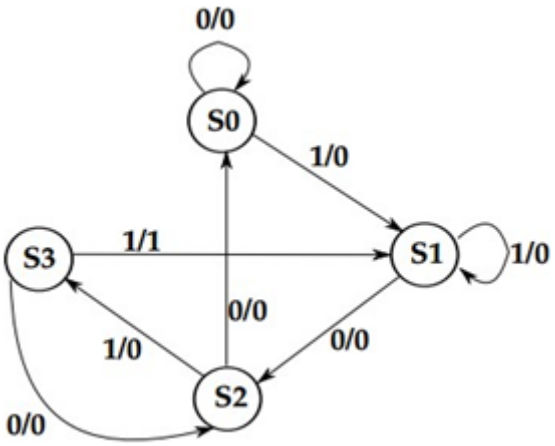


Figure 13.1: Mealy State Machine for Detecting a Sequence of '1011'

Following is the behavior description of the sequencer for a Moore style implementation and the state diagram is shown in figure 13.2:

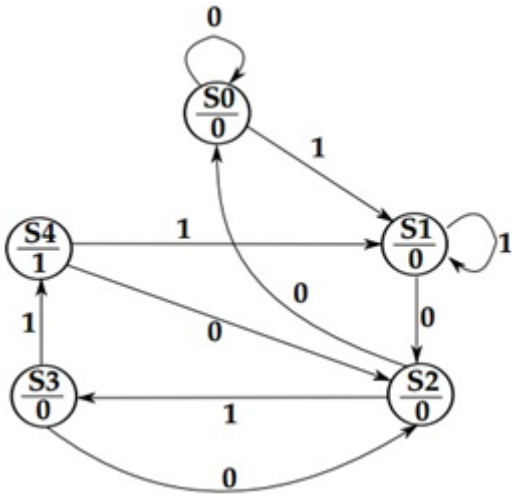


Figure 13.2: Moore State Machine for Detecting a Sequence of '1011'

In initial state (S0) the output of the detector is '0'. When machine gets the input of'1' it jumps to the next state. If the input is '0' it stays in the same state. In 2nd state (S1) the output of the detector is '0'. When machine gets an input of '0'it jumps to the 3rd state. If it gets an input of '1' it stays in the same state. In the 3rd state (S2) the output of the detector is '0'. When machine gets an input of'1' it jumps to the 4th state. If the input received is '0' it goes back to the initial state.

In the 4th state (S3) the output of the detector is '0'. When machine gets an input of'1' it jumps to the 5th state. If the input received is '0' it goes back to the 3rd state.

In the 5th state the output of the detector is '1'. When machine gets an input of '0'it jumps to the 3rd state, otherwise it jumps to the 2nd state.

After designing the state machines the models have to be transformed into VHDL code describing the architecture. Therefore, it is helpful to get an understanding about thebuilding blocks. Figure 13.3 shows the entity for the sequence detector to be developed.The two blocks inside,

i.e., the combinational and the register block is build out ofthe two processes used within the architecture in VHDL. The combinational block decidesthe next state of the FSM according to the current state and the input as well as drives theoutput according to the state (and input for Mealy implementation). The register blocksaves the current state of the FSM. This structure can be used to write the VHDL code.

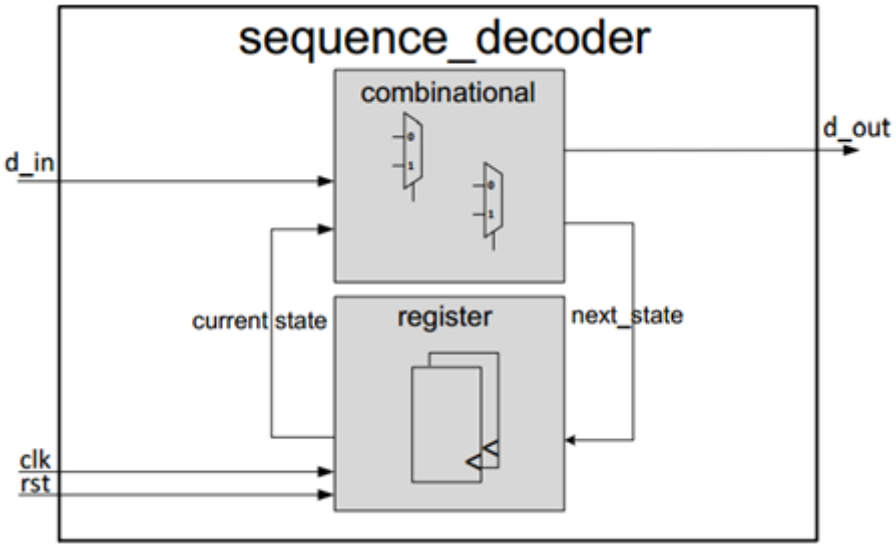


Figure 13.3: Block diagram clarifying the basic building blocks of an FSM

13.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table

13.7 VHDL Program

```
// sequence detector

library ieee;
use ieee.std_logic_1164.all;
entity mealy is
port ( clk : in std_logic;
din : in std_logic;
rst : in std_logic;
dout : out std_logic);
```

```
end mealy;

architecture behavioral of mealy is
type state is (st0, st1, st2, st3);
signal present_state, next_state : state;
begin

syncronous_process : process (clk)
begin
if rising_edge(clk) then
if (rst = '1') then
present_state <= st0;
else
present_state <= next_state;
end if;
end if;
end process;

next_state_and_output_decoder : process(present_state, din)
begin
dout <= '0'; case (present_state) is when st0 =>
if (din = '1') then
next_state <= st1;
dout <= '0';
else
next_state <= st0;
dout <= '0'; end if; when st1 =>
if (din = '1') then
next_state <= st1;
dout <= '0';
else
next_state <= st2;
dout <= '0'; end if; when st2 =>
if (din = '1') then
next_state <= st1;
dout <= '1';
else
next_state <= st0;
dout <= '0'; end if; when others =>
next_state <= st0;
dout <= '0';
end case;
end process;

end behavioral;
```

13.8 Probing Further Experiments

Q1. Design a FSM to detect the sequence '1010'.

Q2. Design a state flow diagram for the sequence detector FSM '10010'.

Q3. Design the state table for the sequence detector FSM '10010'.

Lab 13 – HDL CODE TO DETECT A SEQUENCE

14.1 Introduction

A sequence detector is a sequential state machine that takes an input string of bits and generates an output 1 whenever the target sequence has been detected.

14.2 Objective

To perform the design flow to generate state machines in VHDL code to detect the given sequence of bits

14.3 Prelab Preparation:

Read Appendix B and Appendix C of this manual, paying particular attention to the methods of using measurement instruments. Prior to coming to lab class, complete Part 0 of the Procedure.

14.4 Equipment needed

Xilinx software.
Personal computer.

14.5 Background

As an illustrative example a sequence detector for bit sequence '1011' is described. Every clock-cycle a value will be sampled, if the sequence '1011' is detected a '1' will be produced at the output for 1 clock-cycle. There are two methods to design state machines, first is Mealy and second is Moore style. We will give you an example for both styles. Following is the behavior description of the sequencer for a Mealy style implementation and the state diagram is shown in Figure 14.1.

When in initial state (S0) the machine gets the input of '1' it jumps to the next statewith the output equal to '0'. If the input is '0' it stays in the same state. When in 2nd state (S1) the machine gets an input of '0' it jumps to the 3rd state withthe output equal to '0'. If it gets an input of '1' it stays in the same state. When in the 3rd state (S2) the machine gets an input of '1' it jumps to the 4th statewith the output equal to '0'. If the input received is '0' it goes back to the initial state. When in the 4th state (S3) the machine gets an input of '1' it jumps back to the 2ndstate, with the output equal to '1'. If the input received is '0' it goes back to the 3rd state.

Following is the behavior description of the sequencer for a Moore style implementation and the state diagram is shown in figure 14.2:

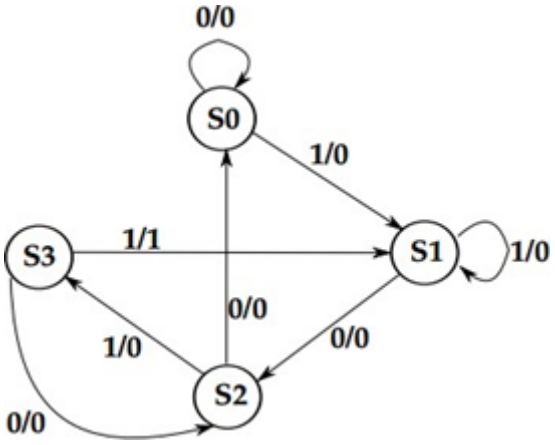


Figure 14.1: Mealy State Machine for Detecting a Sequence of '1011'

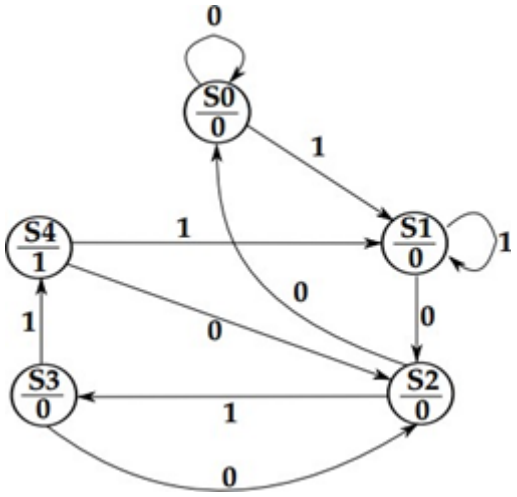


Figure 14.2: Moore State Machine for Detecting a Sequence of '1011'

In initial state (S0) the output of the detector is '0'. When machine gets the input of'1' it jumps to the next state. If the input is '0' it stays in the same state.
In 2nd state (S1) the output of the detector is '0'. When machine gets an input of '0'it jumps to the 3rd state. If it gets an input of '1' it stays in the same state.
In the 3rd state (S2) the output of the detector is '0'. When machine gets an input of'1' it jumps to the 4th state. If the input received is '0' it goes back to the initial state.
In the 4th state (S3) the output of the detector is '0'. When machine gets an input of'1' it jumps to the 5th state. If the input received is '0' it goes back to the 3rd state.
In the 5th state the output of the detector is '1'. When machine gets an input of '0'it jumps to the 3rd state, otherwise it jumps to the 2nd state.
After designing the state machines the models have to be transformed into VHDL code describing the architecture. Therefore, it is helpful to get an understanding about thebuilding blocks. Figure 11.3 shows the entity for the sequence detector to be developed.The two blocks inside, i.e., the combinational and the register block is build out ofthe two processes used within the architecture in VHDL. The combinational block decidesthe next state of the FSM according to the current state and the input as well as drives theoutput according to the state (and input for Mealy implementation). The register blocksaves the current state of the FSM. This structure can be used to write the VHDL code.

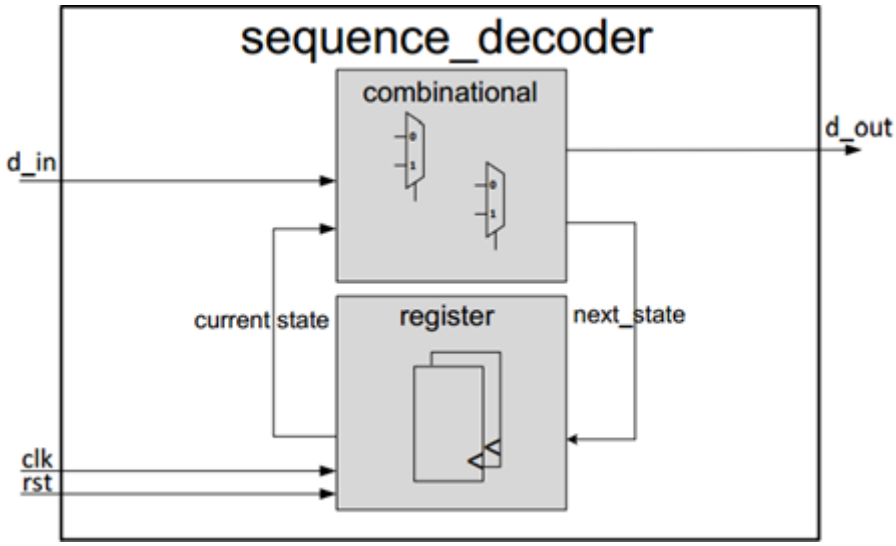


Figure 14.3: Block diagram clarifying the basic building blocks of an FSM

14.6 Procedure

Part (0) After answering the following questions, review the laboratory exercise procedures and plan how you will use the experience gained in these calculations to find the values sought.

1. Create a module with required number of variables and mention it's input/output.
2. Write the description of given Boolean function using operators or by using the built in primitive gates.
3. Create another module referred as test bench to verify the functionality.
4. Follow the steps required to simulate the design and compare the obtained output with the corresponding truth table.

14.7 VHDL Program

```
// sequence detector

library ieee;
use ieee.std_logic_1164.all;
entity mealy is
port ( clk : in std_logic;
din : in std_logic;
rst : in std_logic;
dout : out std_logic);
end mealy;

architecture behavioral of mealy is
type state is (st0, st1, st2, st3);
signal present_state, next_state : state;
begin
```

```
syncronous_process : process (clk)
begin
if rising_edge(clk) then
if (rst = '1') then
present_state <= st0;
else
present_state <= next_state;
end if;
end if;
end process;

next_state_and_output_decoder : process(present_state, din)
begin
dout <= '0'; case (present_state) is when st0 =>
if (din = '1') then
next_state <= st1;
dout <= '0';
else
next_state <= st0;
dout <= '0'; end if; when st1 =>
if (din = '1') then
next_state <= st1;
dout <= '0';
else
next_state <= st2;
dout <= '0'; end if; when st2 =>
if (din = '1') then
next_state <= st1;
dout <= '1';
else
next_state <= st0;
dout <= '0'; end if; when others =>
next_state <= st0;
dout <= '0';
end case;
end process;

end behavioral;
```

14.8 Further Probing Experiments

Q1. Design a FSM to detect the sequence '1010'.

Q2. Design a state flow diagram for the sequence detector FSM '10010'.

Q3. Design a state flow diagram for the sequence detector FSM '1010101.

Appendix A - Safety, Do's and Don'ts

Electricity, when improperly used, is very dangerous to people and to equipment. This is especially true in an industrial environment where large amounts of power is used, and where high voltages are present [1]; in environments where people are especially susceptible to electric shock such as maintenance of a high voltage system (while in operation) or in hospitals where electrical equipment is used to test or control physiological functions [2, 3]; and in an experimental or teaching laboratory where inexperienced personnel may use electrical equipment in experimental or nonstandard configuration.

Engineers play a vital role in eliminating or alleviating the danger in all three types of environments mentioned above. For conditions where standard equipment is used in standard configurations, govern- mental agencies and insurance underwriters impose strict laws and regulations on the operation and use of electrical equipment including switchgear, power lines, safety devices, etc. As a result, corporations and other organizations in turn impose strict rules and methods of operation on their employees and contractors. Engineers who are involved in using electrical equipment, in supervising others who use it, and in designing such systems, have a great responsibility to learn safety rules and practices, to observe them, and to see that a safe environment is maintained for those they supervise. In any working environment there is always pressure to "get the job done" and take short cuts. The engineer, as one who is capable of recognizing hazardous conditions, is in a responsible position both as an engineer and as a supervisor or manager and must maintain conditions to protect personnel and avoid damage to equipment.

Because of their non-standard activities, experimental laboratories are exempt from many of these rules and regulations. This puts more responsibility on the engineer in this environment to know and enforce the safest working procedures.

The knowledge and habit-forming experience to work safely around electrical equipment and the ability to design safe electrical equipment begins with the first student laboratory experience and con- tinues through life. This includes learning the types of electrical injuries and damage, how they can be prevented, the physiology of electrical injuries, and steps to take when accidents.

Physiology of Electrical Injuries

There are three main types of electrical injuries: electrical shock, electrical burns, and falls caused by electrical shock. A fourth type, 'sunburned' eyes from looking at electric arcs, such as arc-welding, is very painful and may cause loss of work time but is usually of a temporary nature. Other injuries may be indirectly caused by electrical accidents, e.g., burns from exploding oil-immersed switch gear or transformers.

Although electric shock is normally associated with high-voltage AC contact, under some circumstances death can occur from voltages from substantially less than the nominal 120 Volts AC found in residential systems. Electric shock is caused by an electric current passing through a part of the human body. The human body normally has a high resistance to electric currents so that a high voltage is usually required to cause lethal currents. This resistance is almost all in the skin, but when the skin is wet its resistance is much lower. When a person is hot and sweaty or is standing in water, contact with 120 Volts or less is likely to cause a fatal shock.

Electric shock is not a single phenomenon but is a disturbance of the nerves that is caused by electric current. A current through a part of the body such as the arm or leg will cause pain and muscle contraction. If a victim receives an electric shock from grasping a live conductor, a current of greater than 15 to 30 mA through the arm will cause muscle contractions so severe that the victim cannot let go. Similar currents through leg muscles may cause sudden contractions causing the victim to jump or fall, resulting in possible injuries or death. It is also possible for a prolonged period of contact of more than a minute or so to cause chest muscles to be contracted, preventing breathing and resulting in suffocation or brain damage from lack of oxygen.

The predominant cause of death by electric shock is generally attributed to ventricular fibrillation, which is an uncontrolled twitching or beating of the heart that produces no pumping action and therefore no blood circulation. Unless corrective action is taken, death follows quickly from lack of oxygen to the brain. While the amount of current that will cause fibrillation depends on several variables, 0.5 to 5A through the body will normally cause the very small current through the heart that causes fibrillation in most people. Larger currents than this through the heart causes contraction or clamping of the heart muscle and resulting death unless corrective action is taken. Prolonged contact of more than a minute or so may cause chest muscles to contract, preventing breathing and resulting in suffocation or brain damage from lack of oxygen.

Death by electric shock is most often attributed to ventricular fibrillation, which is an uncontrolled twitching or beating of the heart that produces no pumping action and therefore no blood circulation. Unless corrective action is taken, death follows quickly from lack of oxygen to the brain. While the amount of current that will cause fibrillation depends on several variables, 0.5 to 5 amperes through the body will normally cause the very small current (approximately 1 mA) through the heart that is sufficient to cause fibrillation in most people. Larger currents than this through the heart cause contraction or clamping of the heart muscle, resulting in death unless corrective action is taken.

Electric burns may be caused by electric currents flowing in or near parts of the body. Such burns are similar to burns from ordinary heat sources, except that those caused by high-frequency currents are generally deeper and take longer to heal the other burns. Electrocution will often leave severe burns at the points where the current entered and left the body.

Source of Electric Shock

Since electric shock is caused by an electric current through a part of the body, it is prevented by not allowing the body to become part of any electric circuit. From this viewpoint, electric circuits may be classified as either grounded or ungrounded.

Electric circuits may be classified as either grounded or ungrounded. Grounded circuits are safer for most conditions, since they result in known voltages at other points in the circuit and provide easier and better protection against faulty conditions in the circuit. The disadvantage is that a person standing on a non-insulated floor can receive a shock by touching only one conductor.

Almost all electric power generation, transmission, and distribution systems are grounded to protect people and equipment against fall conditions caused by windstorms, lightning, etc. Residential, commercial, and industrial systems such as lighting and heating are always grounded for greater safety. Communication, computer, and similar systems are grounded for safety reasons and to prevent or reduce noise, crosstalk, static, etc. Many electronic equipment or instruments are grounded for safety and noise prevention, also. Common examples are DC power supplies, oscilloscopes, oscillators, and analog and digital multimeters.

Ungrounded circuits are used in systems where isolation from other systems is necessary, where low voltages and low power are used, and in other instances where obtaining a ground connection is difficult or impractical. In the ungrounded circuit, contact with two points in the circuit that are at different potentials is required to produce an electrical shock. The hazard is that with no known ground, a hidden fault can occur, causing some unknown point to be grounded, in which case, touching a supposedly safe conductor while standing on the ground could result in an electric shock.

Protecting People and Equipment in the Laboratory

Prevention of electric shock to individuals and damage to equipment in the laboratory can be done by strict adherence to several common-sense rules summarized below:

Protecting People

1. When hooking up a circuit, connect to the power source last, while power is off.
2. Before making changes in a circuit, turn off or disconnect the power first, if possible.
3. Never work alone where the potential of electric shock exists.
4. When changing an energized connection, use only one hand. Never touch two points in the circuit that are at different potentials.
5. Know that the circuit and connections are correct before applying power to the circuit.
6. Avoid touching capacitors that may have a residual charge. The stored energy can cause a severe shock even after a long period of time.
7. Insulate yourself from ground by standing on an insulating mat where available.

The above rules and the additional rules given below also serve to protect instruments and other circuits from damage.

Protecting Equipment

1. Set the scales of measurement instrument to the highest range before applying power.
2. Before making changes in a circuit, turn off or disconnect the power first, if possible.
3. When using an oscilloscope, do not leave a bright spot or trace on the screen for long periods of time. Doing so can burn the image into the screen.
4. Be sure instrument grounds are connected properly. Avoid ground loops and accidental grounding of "hot" leads.
5. Check polarity markings and connections of instruments carefully before connecting power.
6. Never connect an ammeter across a voltage source, but only in series with a load.
7. Do not exceed the voltage or current ratings of circuit elements or instruments. This particularly applies to wattmeters, since the current or voltage rating may be exceeded with the needle still reading on the scale.
8. Be sure any fuses and circuit breakers are of suitable value.

When connecting electrical elements to make up a network in the laboratory, it easy to lose track of various points in the network and accidentally connect a wire to the wrong place. One procedure to help avoid this problem is to connect first the main series loop of the circuit, then go back and add the elements in parallel.

Types of Equipment Damage Excessive currents and voltages can damage instruments and other circuit elements. A large over- current for a short time or a smaller over-current for a longer time will cause overheating, resulting in insulation scorching and equipment failure.

Blown fuses are the most common equipment failure mode in this laboratory. The principal causes for these failures include:
• incorrectly wired circuits;
• accidental shorts;
• switching resistance settings while power is applied to the circuit;
• changing the circuit while power is applied;
• using the wrong scale on ammeter;
• connecting an ammeter across a voltage source;
• using a low-power resistor box (limit 1/2 amp) when high power is required;
• turning on an auto-transformer at too high a setting.

All of these causes are the result of carelessness by the experimenter.

Some type of insulating material, such as paper, cloth, plastic, or ceramic, separates conductors that are at different potentials in electrical devices. The voltage difference that this material can withstand is determined by design (type, thickness, moisture content, temperature, etc.). Exceeding the voltage rating of a device by an appreciable amount can cause arcing or corona, resulting insulation breakdown, and failure.

Some electrical devices can also be damaged mechanically by excessive currents. An example is the D'Arsonval meter, the indicator in most analog metering instruments. A large pulse of over current will provide mechanical torque that can cause the needle to wrap around the pin at the top of the scale, thereby causing permanent damage even though the current may not have been on long enough to cause failure due to overheating.

After Accident Action

Since accidents do happen despite all efforts to prevent them, plans for appropriate reaction to an accident can save time and lives. Such a plan should include immediate availability of first aid material suitable for minor injuries or for injuries that are likely because of the nature of the work. Knowledge of how to obtain trained assistance such as Emergency Medical Services (EMS) should be readily available for everyone.

Treating victims for electrical shock includes four basic steps that should be taken immediately. Step two requires qualification in CPR and step three requires knowledge of mouth-to-mouth resuscitation. Everyone who works around voltages that can cause dangerous electrical shock should take advantage of the many opportunities available to become qualified in CPR and artificial respiration.

Immediate Steps After Electric Shock

1. Shut off all power and remove victim from the electric circuit. If the power cannot be shut off immediately,use an insulator of some sort, such as a wooden pole, to remove victim from the circuit. Attempts to pull the victim from the circuit with your hands will almost always result in your joining the victim in the electric shock.

2. If you are qualified in CPR, check for ventricular fibrillation or cardiac arrest. If either is detected, external cardiac massage should be started at once. Whether you are qualified in CPR or not, notify EMS and the ECE Department at once, using the telephone numbers listed below.

3. Check for respiratory failure and take appropriate action. This may have resulted from physical paralysis of respiratory muscles or from a head injury. Sometimes many hours pass before normal respiration returns. Artificial respiration should be continued until trained EMS assistance arrives.

4. Check for and treat other injuries such as fractures from a fall or burns from current entry and exit sites. Investigations are always after accidents. As an engineer you will be involved as a part of the investigating team or in providing information to an investigator. Information obtained and notes written immediately after the emergency will aid this investigation and assist in preventing future accidents of a similar nature.

Investigations are always made after accidents. As an engineer, you will be involved as a part of the investigating team or in providing information to an investigator. Information obtained and notes written immediately after the emergency will aid the investigation and assist in preventing future accidents of a similar nature.

Emergency Numbers

Fire: 101
Ambulance: 102 or 108

Appendix A References

1. W.F. Cooper, Electrical Safety Engineering, Newnes-Butterworth, London-Boston, 1978.

2. W.H. Buschsbaum and B. Goldsmith, Electrical Safety in the Hospital, Medical Economics Company, Oradel, NJ, 1975.

3. J.G. Wester, editor, Medical Instrumentation Application and Design, Houghton Mifflin Company, Boston, 1978.

Do's and Don'ts

Do's	
1	Program to blink all the alternative LEDs of Port1 with 100 ms in 8051 Microcontroller.
2	Design and write the common cathode display codes to display even numbers from 0 to 9 on seven segment display.
3	Embedded C program for 8051 microcontroller to rotate the stepper motor 360° clockwise by half step sequence and 360° anticlockwise by full step sequence.
4	Embedded C program to display the sine wave at the output of digital to analog converter (DAC).

Don'ts	
1	Program to blink all the alternative LEDs of Port1 with 100 ms in 8051 Microcontroller.
2	Design and write the common cathode display codes to display even numbers from 0 to 9 on seven segment display.
3	Embedded C program for 8051 microcontroller to rotate the stepper motor 360° clockwise by half step sequence and 360° anticlockwise by full step sequence.
4	Embedded C program to display the sine wave at the output of digital to analog converter (DAC).
SAFETY NORMS	
1	Program to blink all the alternative LEDs of Port1 with 100 ms in 8051 Microcontroller.
2	Design and write the common cathode display codes to display even numbers from 0 to 9 on seven segment display.
3	Embedded C program for 8051 microcontroller to rotate the stepper motor 360° clockwise by half step sequence and 360° anticlockwise by full step sequence.

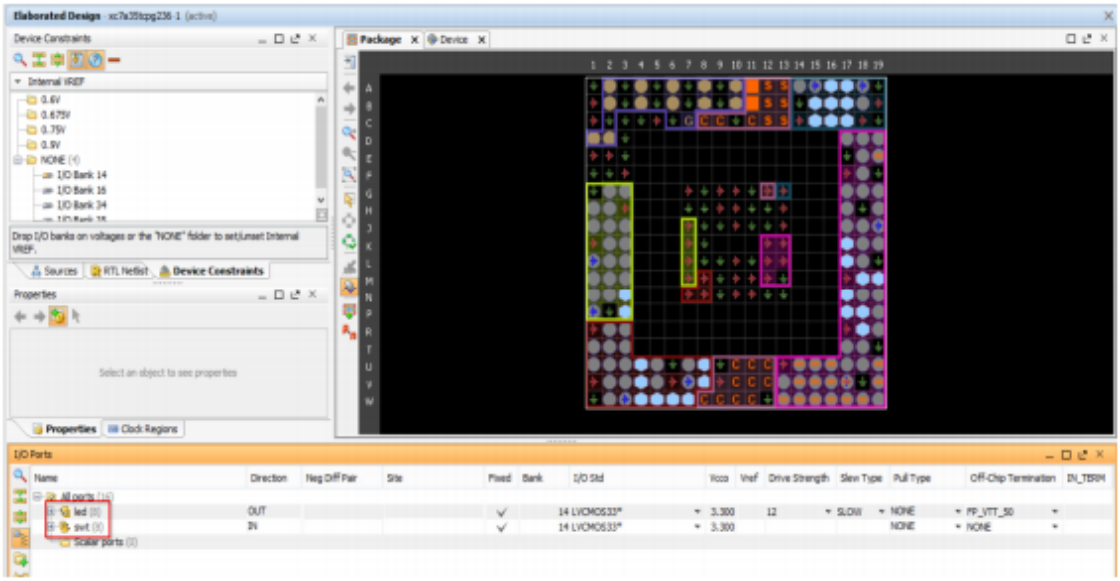
Appendix B - Simulate the Design using the XSim Simulator

The VHDL testbench has the same structure as any VHDL design source code. There are a few exceptions that need some explanation. After the library declarations, note that the Entity declaration is left empty on Lines 16 and 17. The Unit Under Test (UUT; or the VHDL code being simulated) is instantiated as a component declaration from Lines 20 to 25.To generate the expected results during simulation, Lines 38 through 48 emulate the behavior of the UUT. Lines 49 to 52 is the port declaration for the UUT. Lines 56 through 86 define the stimuli generation and compares the expected output against the UUT output. Line 87 ends the testbench. To provide feedback to the user via the Vivado simulator console window, examine Lines 74 through 80. Note multiple lines have been concatenated into one line, separated by the VHDL end of line character ";".

Let us now discuss the architecture of 8051 Microcontroller. In the following diagram, the system bus connects all the support devices to the CPU. The system bus consists of an 8-bit data bus, a 16-bit address bus and bus control signals. All other devices like program memory, ports, data memory, serial interface, interrupt control, timers, and the CPU are all interfaced together through the system bus.

I/O Planning layout view for Basys3

The testbench and source files will be compiled and the Vivado simulator will be run (assuming no errors). You will see a simulator output similar to the one shown below. You will see matching and mismatching results from the UUT and the procedure in the testbench. The mismatches were deliberately inserted to demonstrate the TEXTIO functionality of the VHDL testbench.



Central Processor Unit (CPU)
As we know that the CPU is the brain of any processing device of the microcontroller. It monitors and controls all operations that are performed on the Microcontroller units. The User has no control over the work of the CPU directly . It reads program written in ROM memory and executes them and do the expected task of that application.

The Microcontroller 8051 can be configured in such a way that it temporarily terminates or pause the main program at the occurrence of interrupts. When a subroutine is completed, Then the execution of main program starts. Generally five interrupt sources are there in 8051 Microcontroller. There are 5 vectored interrupts are shown in below
• INTO
• TFO
• INT1
• TF1
• R1/T1

Out of these, (INT0) and (INT1) are external interrupts that could be negative edge triggered or low level triggered. When All these interrupts are activated, set the corresponding flogs except for serial interrupt,.The interrupt flags are cleared when the processor branches to the interrupt service routine (ISR). The external interrupt flags are cleared when the processor branches to the interrupt service routine, provides the interrupt is a negative edge triggered whereas the timers and serial port interrupts two of them are external interrupts, two of them are timer interrupts and one serial port interrupt terminal in general.

Memory
Microcontroller requires a program which is a collection of instructions. This program tells microcontroller to do specific tasks. These programs require a memory on which these can be saved and read by Microcontroller to perform specific operations of a particular task. The memory which is used to store the program of the microcontroller is known as code memory or Program memory of applications. It is known as ROM memory of microcontroller also requires a memory to store data or operands temporarily of the micro controller. The data memory of the 8051 is used to store data temporarily for operation is known RAM memory. 8051 microcontroller has 4K of code memory or program memory, that has 4KB ROM and also 128 bytes of data memory of RAM.

BUS Basically Bus is a collection of wires which work as a communication channel or medium for transfer of Data. These buses consists of 8, 16 or more wires of the microcontroller. Thus, these can carry 8 bits,16 bits simultaneously. Hire two types of buses that are shown in below
• Address Bus
• Data Bus

Address Bus: Microcontroller 8051 has a 16 bit address bus for transferring the data. It is used to address memory locations and to transfer the address from CPU to Memory of the microcontroller. It has four addressing modes that are
• Immediate addressing modes.
• Bank address (or) Register addressing mode.
• Direct Addressing mode.
• Register indirect addressing mode.

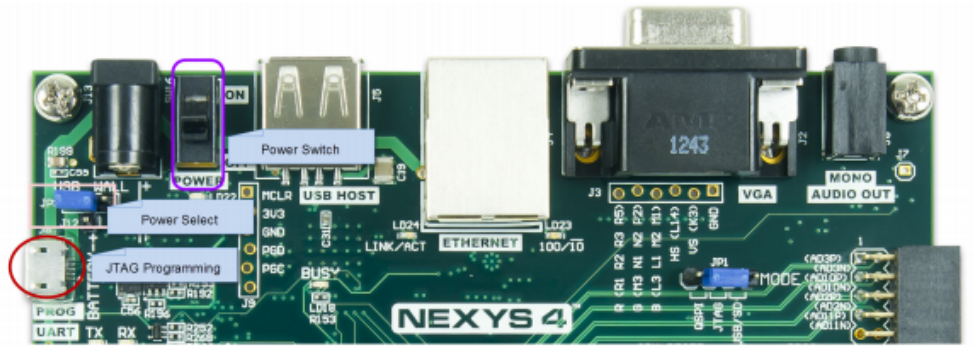
Data Bus: Microcontroller 8051 has 8 bits of the data bus, which is used to carry data of particular applications.

Oscillator
Generally, we know that the microcontroller is a device, therefore it requires clock pulses for its operation of microcontroller applications. For this purpose, microcontroller 8051 has an on-chip oscillator which works as a clock source for Central Processing Unit of the microcontroller. The output pulses of oscillator are stable. Therefore, it enables synchronized work of all parts of the 8051 Microcontroller.

Input/Output Port
Normally microcontroller is used in embedded systems to control the operation of machines in the microcontroller. Therefore, to connect it to other machines, devices or peripherals we require I/O interfacing ports in the microcontroller interface. For this purpose microcontroller 8051 has 4 input, output ports to connect it to the other peripherals.

Timers/Counters
8051 microcontroller has two 16 bit timers and counters. These counters are again divided into a 8 bit register. The timers are used for measurement of intervals to determine the pulse width of pulses.

Simulate the design for 1000 ns using the XSIM simulator



Pins 1 to 8: These pins are known as Port 1. This port doesn't serve any other functions. It is internally pulled up, bi-directional I/O port.
Pin 9: It is a RESET pin, which is used to reset the microcontroller to its initial values.
Pins 10 to 17: These pins are known as Port 3. This port serves some functions like interrupts, timer input, control signals, serial communication signals RxD and TxD, etc.
Pins 18 & 19: These pins are used for interfacing an external crystal to get the system clock.
Pin 20: This pin provides the power supply to the circuit.
Pins 21 to 28: These pins are known as Port 2. It serves as I/O port. Higher order address bus signals are also multiplexed using this port.
Pin 29: This is PSEN pin which stands for Program Store Enable. It is used to read a signal from the external program memory.
Pin 30: This is EA pin which stands for External Access input. It is used to enable/disable the external memory interfacing.
Pin 31: This is ALE pin which stands for Address Latch Enable. It is used to demultiplex the address-data signal of port.
Pins 32 to 39: These pins are known as Port 0. It serves as I/O port. Lower order address and data bus signals are multiplexed using this port.
Pin 40: This pin is used to provide power supply to the circuit.

Appendix C -Vivado Tutorial

Introduction:

This tutorial guides you through the design flow using Xilinx Vivado software to create a simple digital circuit using VHDL. A typical design flow consists of creating model(s), creating user constraint file(s), creating a Vivado project, importing the created models, assigning created constraint file(s), optionally running behavioral simulation, synthesizing the design, implementing the design, generating the bitstream, and finally verifying the functionality in the hardware by downloading the generated bitstream file. You will go through the typical design flow targeting the Artix-7 based Basys3 and Nexys4 DDR boards. The typical design flow is shown below. The circled number indicates the corresponding step in this tutorial.

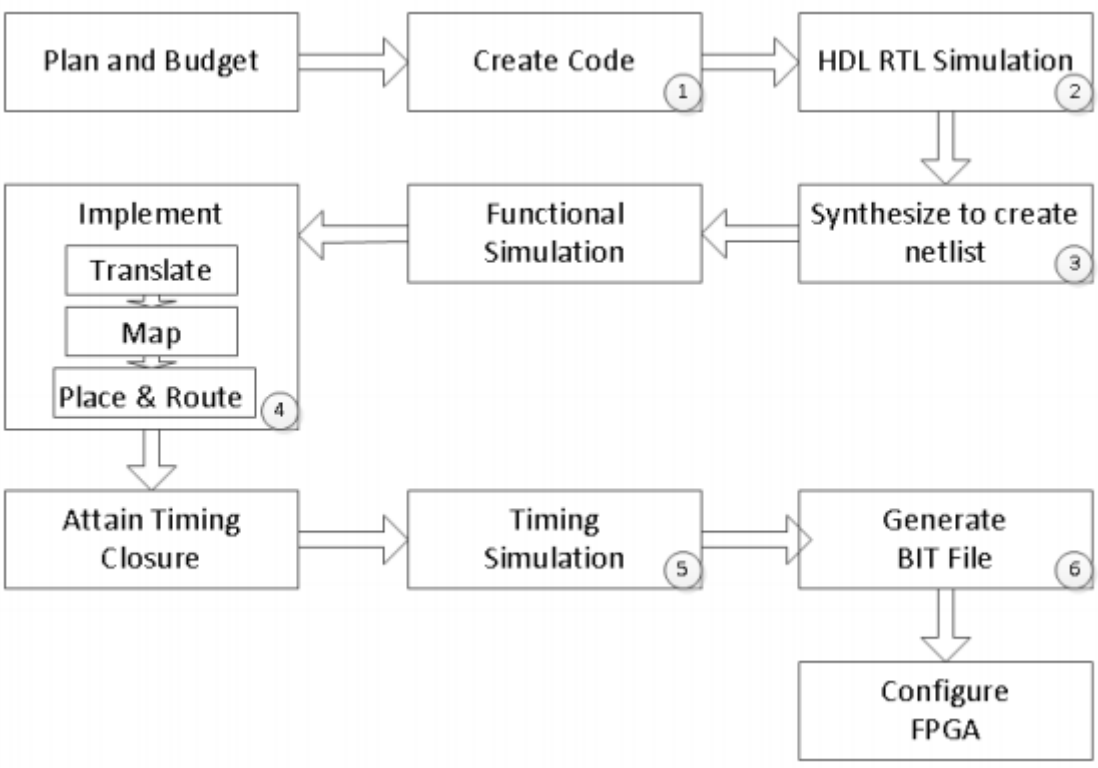


Figure C.1: A typical design flow

Objectives:

After completing this tutorial, you will be able to:
• Create a Vivado project sourcing HDL model(s) and targeting a specific FPGA device located

on the Basys3 or Nexys4 DDR boards
- Use the provided user constraint file (XDC) to constrain pin locations
- Simulate the design using the XSIM simulator
- Synthesize and implement the design
- Generate the bitstream
- Download the design and verify the functionality

Procedure:

This tutorial is broken into steps that consist of general overview statements providing information on detailed instructions that follow. Follow these detailed instructions to progress through the tutorial.

Design Description:

The design consists of some inputs directly connected to the corresponding output LEDs. Other inputs are logically operated on before the results are output on the remaining LEDs as shown in Figure 2.

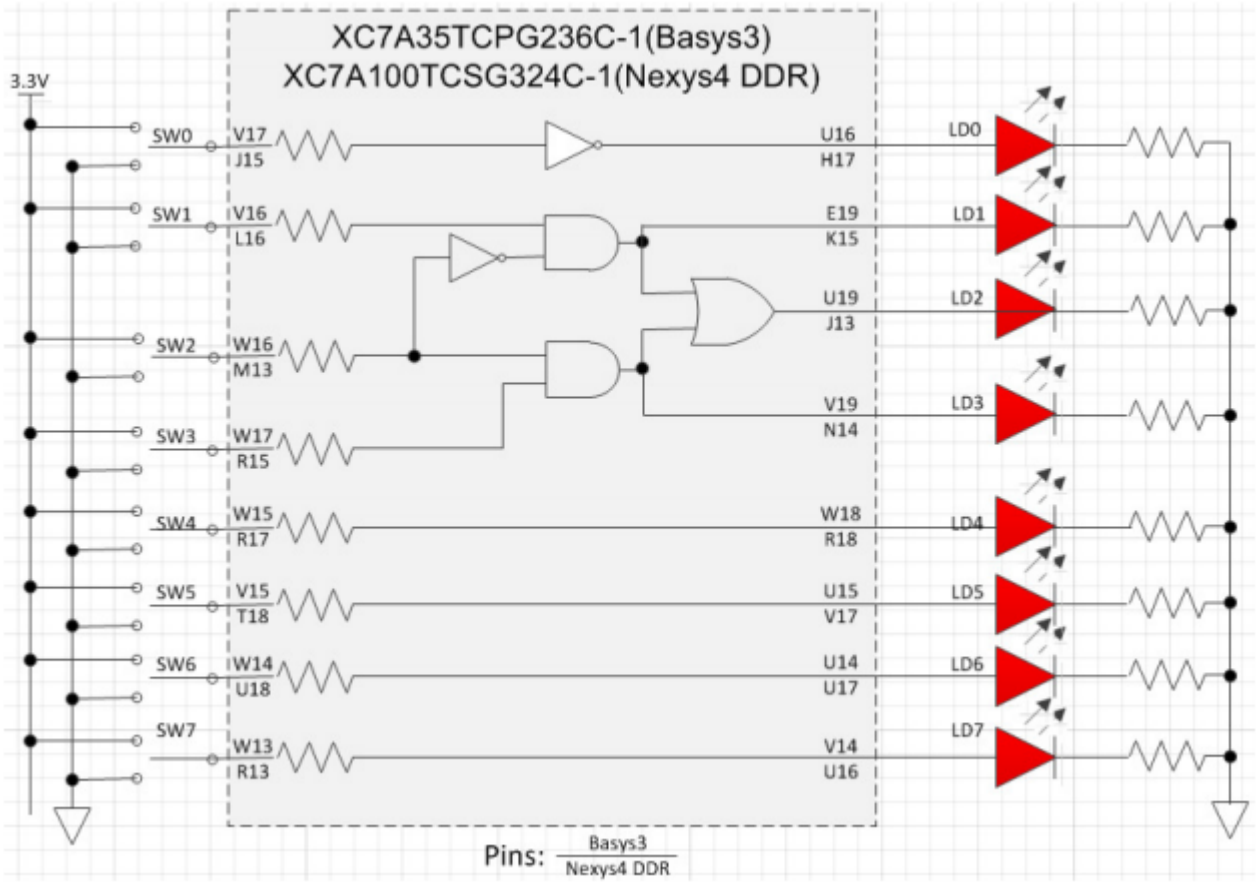


Figure C.2: Completed Design

General Flow for this tutorial:

- Create a Vivado project and analyze source files
- Simulate the design using XSIM simulator
- Synthesize the design
- Implement the design
- Perform the timing simulation
- Verify the functionality in hardware using Basys3 or Nexys4 DDR board

Vivado Tutorial:

1. Launch Vivado and create a project targeting the xc7a35tcpg236-1 (Basys3) or xc7a100tcsg324-1 (Nexys4 DDR) device and using the VHDL. Use the provided tutorial.vhd and Nexys 4DDRMaster.xdc or Basys 3Master.xdc files from the sources ORtutorial directory.

2. Open Vivado by selecting Start > All Programs > Xilinx Design Tools > Vivado 2015.1 > Vivado 2015.1.

3. Click Create New Project to start the wizard. You will see Create A New Vivado Project dialog box. Click Next.

4. Click the Browse button of the Project location field of the New Project form, browse to *c* :, and click Select

5. Enter tutorial in the Project name field. Make sure that the Create Project Subdirectory box is checked. Click Next.

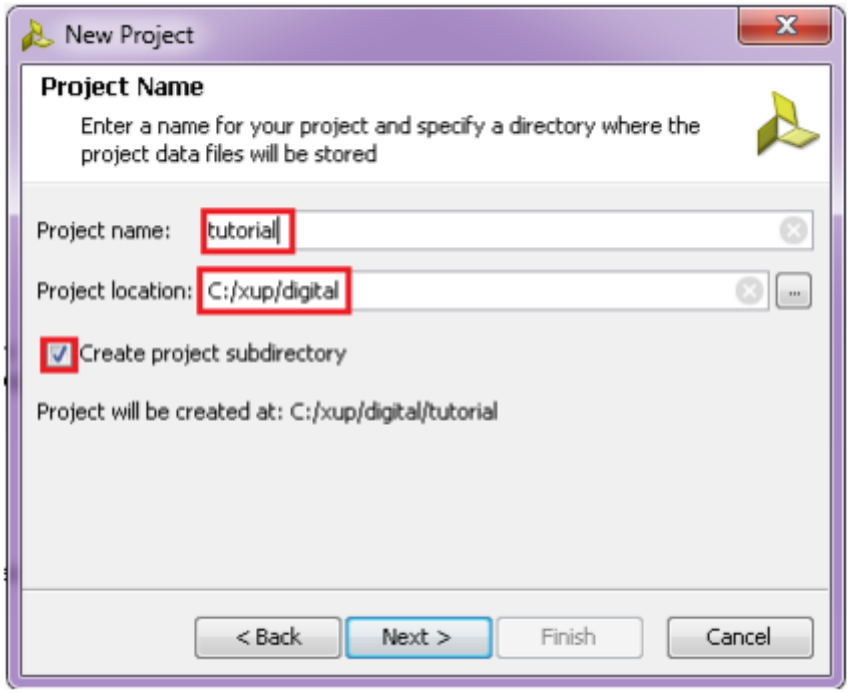


Figure C.3: Part Selection for Basys3

6. Select VHDL as the Target Language and as the Simulator language in the Add Sources form. on the Green Plus button, then click on the Add Files... button, browse to the

c :directory, select tutorial.vhd, click Open, and verify the Copy constraints files into projects box is check. Then click Next.

7. Click Next at the Add Existing IP form.

8. In the Add Constraints form, click on the Green Plus button, then the Add Files... button, browse to the $c : directory, selectBasys3Master.xdc$), click Open, and then click Next. The XDC constraint file assigns the physical IO locations on FPGA to the switches and LEDs located on the board. This information can be obtained either through a board's schematic or board's user guide.

9. In the Default Part form, using the Parts option and various drop-down fields of the Filter section, select the xc7a35tcpg236-1 part (for Basy3) or xc7a100tcsg324-1 part (for Nexys4 DDR). Click Next.

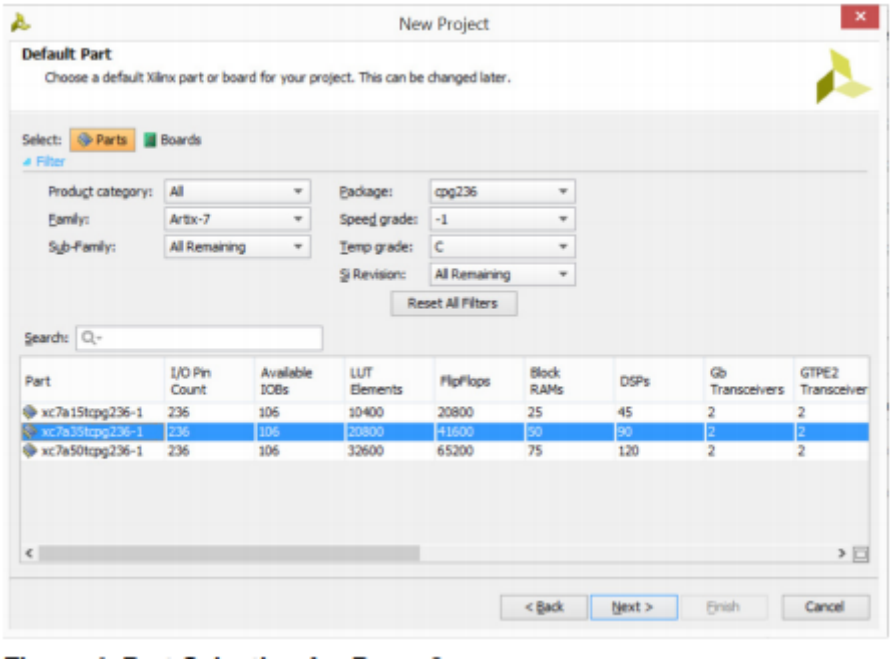


Figure C.4: Part Selection for Basys3

10. Click Finish to create the Vivado project.

11. Use the Windows Explorer and look at the c : directory. You will find that the tutorial.srcs and other directories, and the tutorial.xpr (Vivado) project file have been created. Two sub-directories, constrs 1 and sources, are created under the tutorial.srcs directory; deep down under them, the copied Nexys 4DDR Master.xdc or Basys3 Master.xdc (constraint) and tutorial. vhd (source) files respectively are placed.

12. Open the tutorial.vhd source and analyze the content.

13. In the Sources pane, double-click the tutorial.vhd entry to open the file in text mode. The design takes input from slide switches 0 to 7 of the board and toggles the LEDs on the board. Since combinatorial logic is inserted between some switches, the LEDs will turn on/off depending on the pattern of the switches. This is a very basic combinatorial logic demo.

14. Open the Basys3 Master.xdc or Nexys4 DDR Master. xdc source, analyze the content and edit the file.

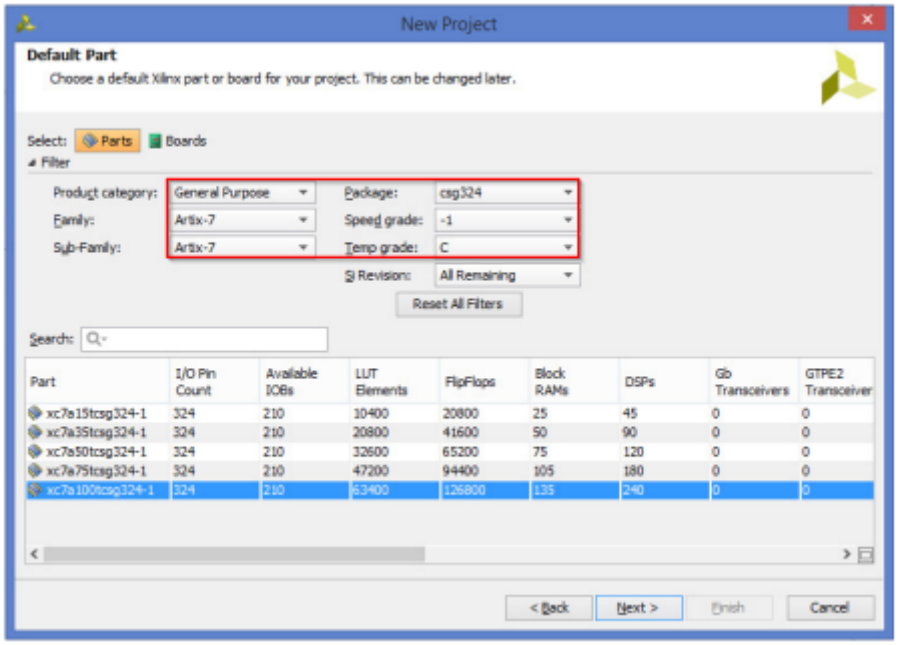


Figure C.5: Part Selection for Nexys4 DDR

15. In the Sources pane, expand the Constraints folder and double-click the Basys3 Master.xdc (Basys3) or Nexys 4DDR Master.xdc (Nexys4 DDR) entry to open the file in text mode.