METHODIST
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad
Abids, Hyderabad, Telangana, 500001
Estd : 2008

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

ELECTRONIC DESIGN AND AUTOMATION LAB

STUDENT LABORATORY MANUAL

(As per 2018-2019 Academic Regulations)

B.E VII SEMESTER E&CE

SUBJECT CODE: PC 752 EC

Name:_____

Roll No.:_____

METHODIST
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad
Abids, Hyderabad, Telangana, 500001
Estd : 2008

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

ELECTRONIC DESIGN AND AUTOMATION LAB

STUDENT LABORATORY MANUAL

(As per 2018-2019 Academic Regulations)

B.E VII SEMESTER E&CE

SUBJECT CODE: PC 752 EC

Prepared by

Mr. M. Mahesh Babu, Assistant Professor
Mr. I. Srikanth, Associate Professor

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Vision of the Institute:

To produce ethical, socially conscious and innovative professionals who would contribute to sustainable technological development of the society.

Mission of the Institute:

To impart quality engineering education with latest technological developments and interdisciplinary skills to make students succeed in professional practice

To encourage research culture among faculty and students by establishing state of art laboratories and exposing them to modern industrial and organizational practices

To inculcate humane qualities like environmental consciousness, leadership, social values, professional ethics and engage in independent and lifelong learning for sustainable contribution to the society.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Vision of the Department:

To strive to become centre of excellence in Education, Research with moral, ethical values and serve society

Mission of the Department:

M1: To provide Electronics & Communication Engineering knowledge for successful career either in industry or research

M2: To develop Industry-Interaction for innovation, product oriented research and development.

M3: To facilitate value added education combined with hands-on trainings.

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Program Educational Objectives:

PEO 1: Apply the knowledge of Basic sciences and Engineering in designing and implementing the solutions in emerging areas of Electronics and Communication Engineering.

PEO 2: Pursue the research or higher education and practise profession.

PEO 3: Adapt to the technological advancements for providing the sustainable Engineering solutions to meet organisation/society needs

PEO 4: Work as an individual or in a team with professional ethics and values.

Program Outcomes:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Program Specific Outcomes (PSOs):

PSO1: Professional Competence: Apply the knowledge of Electronics & Communication Engineering principles in different domains like VLSI, Signal processing, Communication, Embedded system & Control Engineering.

PSO2: **Technical Skills**: Able to design and implement products using the cutting- edge software and hardware tools and hence provide simple solutions to complex problems.

PSO3: Social consciousness: Graduates will be able to demonstrate the leadership qualities and strive for the betterment of organization, environment and society

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Laboratory Code of Conduct

1. Students should report to the concerned labs as per the time table schedule.
2. Students who turn up late to the labs will in no case be permitted to perform the experiment scheduled for the day.
3. Students should bring a note book of about 100 pages and should enter the readings/observations into the note book while performing the experiment.
4. After completion of the experiment, certification of the concerned staff in-charge in the observation book is necessary.
5. Staff member in-charge shall award 25 marks for each experiment based on continuous evaluation and will be entered in the continuous internal evaluation sheet.
6. The record of observations along with the detailed experimental procedure of the experiment performed in the immediate last session should be submitted and certified by the staff member in-charge.
7. Not more than three students in a group are permitted to perform the experiment on a set-up for equipment-based labs. Only one student is permitted per computer system for computer-based labs.
8. The group-wise division made in the beginning should be adhered to, and no student is allowed to mix up with different groups later.
9. The components required pertaining to the experiment should be collected from the stores in-charge, only after duly filling in the requisition form/log register.
10. When the experiment is completed, students should disconnect the setup made by them, and should return all the components/instruments taken for the purpose.
11. Any damage of the equipment or burn-out of components will be viewed seriously by either charging penalty or dismissing the total group of students from the lab for the semester/year.
12. Students should be present in the labs for the total scheduled duration.
13. Students are required to prepare thoroughly to perform the experiment before coming to Laboratory.
14. Procedure sheets/data sheets provided to the students, if any, should be maintained neatly and returned after the completion of the experiment.

METHODIST
COLLEGE OF ENGINEERING AND TECHNOLOGY
Approved by AICTE New Delhi | Affiliated to Osmania University, Hyderabad
Abids, Hyderabad, Telangana, 500001
Estd : 2008

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Course Code	Course Title						Core / Elective
PC 752 EC	**Electronic Design and Automation Lab**						**Core**
Prerequisite	Contact Hours per Week				CIE	SEE	Credits
	L	T	D	P			
-	-	-	-	**2**	**25**	**50**	**1**

Course Objectives

- Familiarize with the usage of IDE tools and execution of programs using ARM processor.
- Know about the usage of various devices like LCD, Temperature sensor, Buzzer, Stepper Motor by interfacing them to LPC2148.
- Understand the designing and implementation of combinational and sequential logic circuits using Verilog HDL.
- Study of Mentor Graphics Tools
- Implement basic gates at transistor level

Course Outcomes

After completing this course, the student will be able to

1. Familiarize with the usage of IDE tools and program using various on chip like LCD, Temperature sensor, Buzzer, Stepper Motor by interfacing them to ARM Processor
2. Design the digital logic circuits in various modelling styles using Verilog HDL
3. Familiarize with VLSI CAD tools like Mentor Graphics / Cadence
4. Implement basic gates at transistor level
5. Implement the digital circuits at transistor level.

PART-A

Interfacing Programs using embedded C on ARM Micro Controller Kit

1. Program to interface 8-Bit LED and switch interface
2. Program to implement Buzzer interface on IDE environment
3. Program to display message in a 2 line x 16 characters LCD display and verify the result in debug terminal
4. Stepper motor interface
5. ADC & Temperature sensor LM35 interface
6. Transmission from kit and reception from PC using serial port.

PART-B

Implementation of programs using Verilog HDL code on FPGA Board

1. Adders / Subtractors
2. Multiplexer / Demultiplexer
3. Flip flops
4. Counters/Registers

5. Vending Machine controller
6. Multipliers

PART C

Transistor Level implementation of CMOS circuits using VLSI CAD tool

1. Basic Logic Gates: Inverter, NAND and NOR
2. Half Adder and Full Adder
3. 4:1 Multiplexer
4. 2:4 Decoder

Note: A minimum of 10 experiments to be performed and at least 3 experiments from each part to be performed.

Methodist College of Engineering & Technology

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Experiment No.	Title of the Experiment	Date	Page No.	Marks				Remarks/ Signature
				E	O	R	T	
1								
2a.								
2b.								
3								
4								
5a.								
5b.								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								

E: Experiment (10 Marks) O: Observation (10 Marks) R: Record (5Marks) T: Total (25 Marks)

INTRODUCTION

Interfacing Programs using embedded C on ARM Micro Controller Kit

LPC2148 ARM7 Based Microcontroller

1. GENERAL DESCRIPTION:

The LPC2141/2/4/6/8 microcontrollers are based on a 32/16 bit ARM7TDMI-S CPU with Real time emulation and embedded trace support, that combines the microcontroller with embedded high speed flash memory ranging from 32 k B to 512 k B. A 128-bit wide memory interface and unique accelerator architecture enable 32-bit code execution at the maximum clock rate. For critical code size applications, the alternative I 6-bit Thumb mode reduces code by more than 30 % with minimal performance penalty.

Due to their tiny size and low power consumption, LPC2I41/2/4/6/8 are ideal for applications where miniaturization is a key requirement, such as access control and point-of-sale. A blend of serial communications interfaces ranging from a USB 2.0 Full Speed device, multiple UARTS, SPI, SSP to I2Cs and on-chip SRAM of 8 kB up to 40 kB, make these devices very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging, providing both large buffer size and high processing power. Various 32-bit timers, single or dual IO-bit ADC(s), IO-bit DAC, PWM channels and 45 fast GPIO lines with up to nine edge or level sensitive external interrupt pins make these microcontrollers particularly suitable for industrial control and medical systems.

2. FEATURES of LPC2148:

- 16/32-bit ARM7TDMI-S microcontroller in a tiny LQFP64 package.
- 8 to 40kB of on-chip static RAM and 32 to 512kB of on-chip flash program memory.
- 128 bit Wide interface/accelerator enables high speed 60 MHz operation.
- In-System/In-Application Programming (ISP/IAP) via on-chip boot-loader software. Single flash sector or full chip erase in 400 ms and programming of 256 bytes in 1ms.
- Embedded ICE RT and Embedded Trace interfaces offer real-time debugging with the on-chip
- Real Monitor software and high speed tracing of instruction execution.
- USB 2.0 Full Speed compliant Device Controller with 2 kB of endpoint RAM.
- In addition, the LPC2146/8 provide 8 kB of on-chip RAM accessible to USB by DMA.
- One or two (LPC2 I 41/2 vs. LPC2144/6/8) 10-bit AID converters provide a total of 6/14 analog inputs, with conversion times as low as 2.44 μs per channel.
- Single IO-bit DIA converter provides variable analog output.
- Two 32-bit timers/external event counters (with four captures and four compare Channels each), PWM unit (six outputs) and watchdog.
- Low power real-time clock with independent power and dedicated 32 Hz clock input.
- Multiple serial interfaces including two UARTs (I 6C550), two Fast I2C-b'us (400 k bit/s), SPI and SSP with buffering and variable data length capabilities.
- Vectored interrupt controller with configurable priorities and vector addresses.
- Up to 45 of 5 V tolerant fast general purpose I/0 pins in a tiny LQFP64 package.
- Up to nine edge cir level sensitive external interrupt pins available.

3. LPC-2148 BASED ARM-7 DEVELOPMENT KIT

Physitech electronics LPC2148 Education Board based on NXP's ARM7TDMI LPC2148 microcontroller.

The LPC2148 development kit is intended as a demonstration and evaluation of ARM7core Philips microcontroller. This document is a User's Guide that describes the *LPC2148 Education Board* hardware design and some basic software interface principles with the hardware.

Fig. 1: Physitech LPC-2148 BASED ARM-7 DEVELOPMENT KIT

4. FEATURES OF DEVELOPMENT kit:

Experimental board for Philips ARM7TDMI LPC2148 microcontroller

- 16x2 Alpha-numeric LCD connector (Graphical LCD which is optional)
- AT keyboard Interface.
- UART-to-serial Bridge interface on UART #0/1.
- USB 2.0 device interface (on LPC2148)).
- I2C interface bus connector for external devices.
- 256 Kbit I2C E2PROM for storing non-volatile parameters.
- All LPC2148 I/O pins are available on connectors.
- LED Interface with 8-GPIO Pins.
- 8- Push-On/OFF buttons
- 8- Toggle Switches.

- MMC/SD memory card interface.
- Temperature sensor (LM35).
- Piezoelectric buzzer.
- 4-channel 10-bit Analog to Digital Converter in which provision has been given to connect 2 Analog inputs.
- One analog output through 10-bit DAC.
- Audio Input with 3.5mm Jack
- Audio Output with 3.5mm Jack
- 12.0000 MHz crystal for maximum execution speed.
- 32.768 kHz RTC crystal.
- Reset Switch.
- Auto Boot Selection Switch, to enable switching between Run and Program mode.
- Power supply: USB and/or External Adapter
- Power supply LED.
- All GPIO Port Pins of LPC2148 are available for user to connect any daughter card interfacing.

5. Expansion Boards

The 64 pin expansion connector, with all LPC2148 I/O pins available, allows the *LPC2148 Education Board* to be expanded with new and exciting hardware. The experiments can be more advanced and complex by using the expansion boards. The list below presents the first set of available expansion boards. As seen, many of the boards are communication oriented.

- Ethernet board using SPI interface.
- RF module using SPI interface.
- Graphical LCD.
- Stepper motor.
- 4x4 keypad interface.
- Relay cards.
- GSM\GPS

6. Creating Project Using Keil uVision 4:

1. This is evaluation version. So lets start with new project.
   Thus tutorial demonstrates the LED blinking program for LPC2148. Do follow the steps.
2. Click the **"Keil uVision4"** from desktop Menu.
3. Click Project from menu bar and select new uVision project.
4. Under the Create New Project window, select the project folder in "Save in" and name the project under "File Name"and Click **Save**.
   Ex:- I have given file name as **Keil_Main** under **Sample** Folder.
5. This will create a new project with new work environment. This will give you a new window.
6. Select Target Device under "Target 1". I have selected LPC2148 as target device for Sample Programming.
   Note:- To select LPC2148, Expand "**NXP (founded by Philips)**", select LPC2148 and click OK.
7. You will get below window. Please select **"Yes"** in order to add the startup code to your project for LPC2148.

8. To see the Start Up code for LPC2148 Provided by Keil, Expand "**Source Group**" Folder under "**Target 1**" Folder.Double click the "**Startup.s**" for Project Startup File.
9. To Write the C Code in Keil, We Need Text Editor. Select "**New** " under "**File"** ToolBar in Keil, as shown in above Figure.
10. Save the Text Editor in Project Folder with "**filename.c**" Extension.
11. Add the Source File (**filename.c**) to Project using "**Source Group 1**" Options, As shown in Figures.
12. Add the Target header File by Right Clicking the Mouse.
13. Write the Code and **Save** the Cod using "**CTRL+S**" and to Compile the Written code using "**Build (F7)"** as shown in the Figure.
14. After compiling the code, You can get the **Errors** and **Warnings** at "**Build Output**"

7. Flash Magic Programmer

Flash Magic is an application developed by Embedded Systems Academy to allow you to easily access the features of a microcontroller device. With this program you can erase individual blocks or the entire Flash memory of the microcontroller.

Procedure:-

1. Go to the **"Flash Magic"** Folder in Your Computer and double click the **.EXE** File.
2. Click the "**NEXT**" option in Setup .
3. Accept the agreement in Flash Magic Setup.
4. Click the "**NEXT**" in Setup to Destination Folder for Flash magic.
5. Select the Location of **Start Menu Folder** for Flash Magic.
6. Before Installation, Create the Desktop Icon for Flash Magic and click "**Next**" Option to Install Flash Magic.
7. Install the Flash Magic Programmer
8. Click **"Finish"** option to launch Flash magic.
9. To run Flash magic click on the **Flash Magic** Icon on the Desktop.
10. Programming the HEX File to Target Microcontroller using Flash Magic.

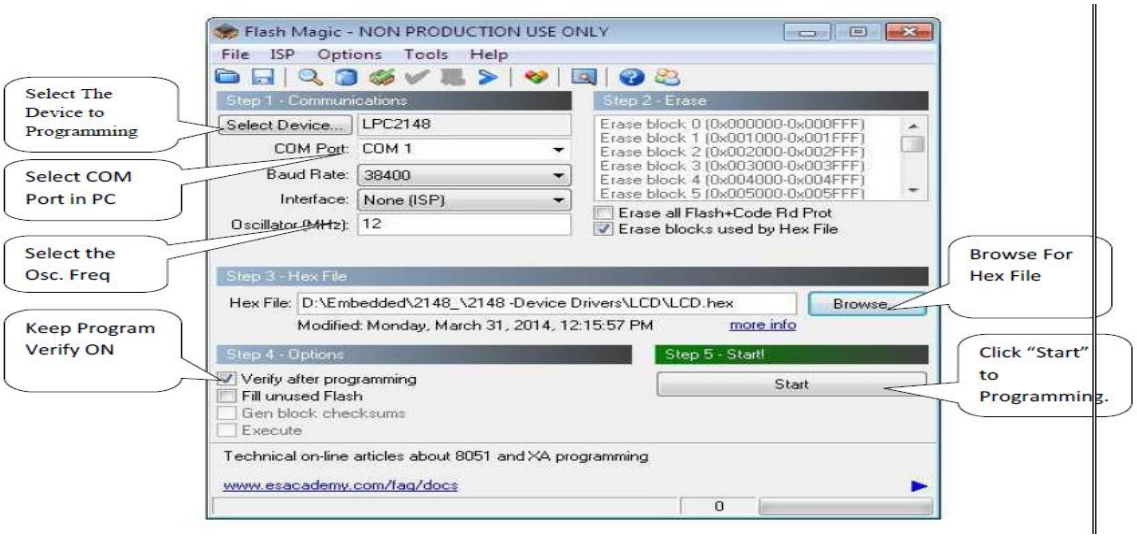


Fig. 2: Flash Magic Programming Tool

LEDs

The port pins P1.16 to P1.23 of the LPC2148 microcontrollers are connected to 8 red LED's(D8-D15) via a Buffer(U20). The buffers enable is given to P1.30 for enabling or disabling the LED operations. *Figure* illustrates the 8 LEDs in the design.

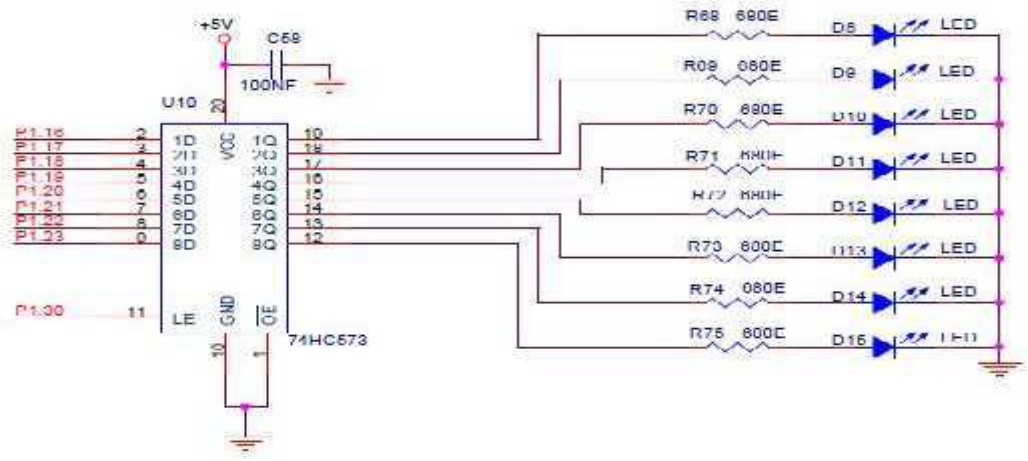


Fig. 3: LED's port pins of the LPC2148 MC

Push buttons:

The push buttons (SW13-SW20) are interfaced to port P0.16-P0.23 of the LPC2148, via a 74HC573 buffer (U14). The Push buttons are enabled using the jumper JP7. *Figure* illustrates the push buttons in the design.

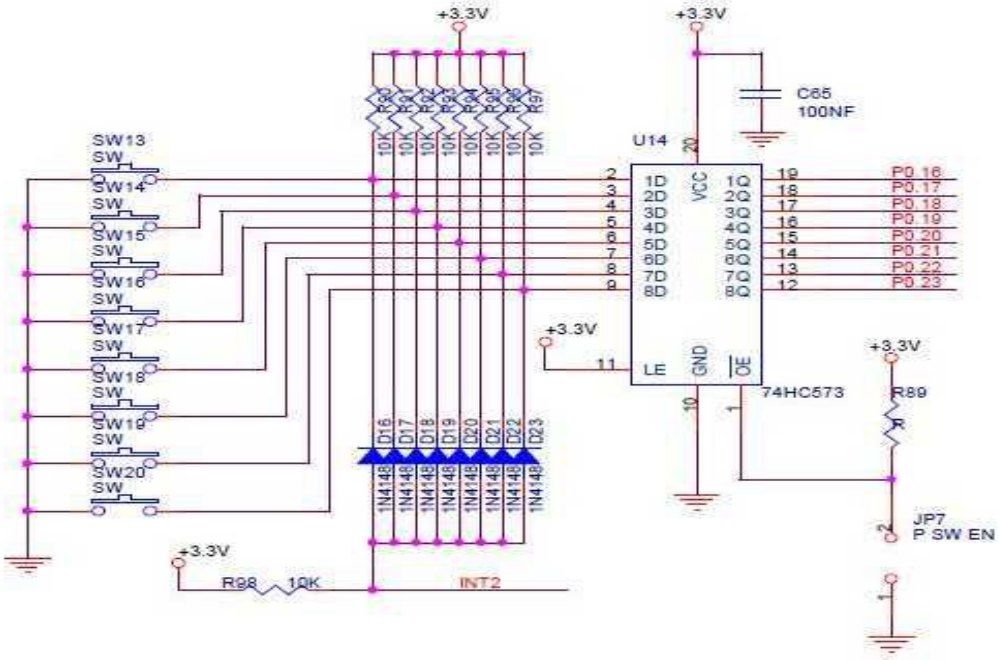


Fig. 4: Push buttons are interfaced to port pins of the LPC2148 MC

Toggle switches: The toggle switches (SW5-SW12) are interfaced to port P0.16-P0.23 of the LPC2148, via a 74HC573 buffer (U13). The toggle switches are enabled using the jumper JP6. The figure illustrates the toggle switches design.

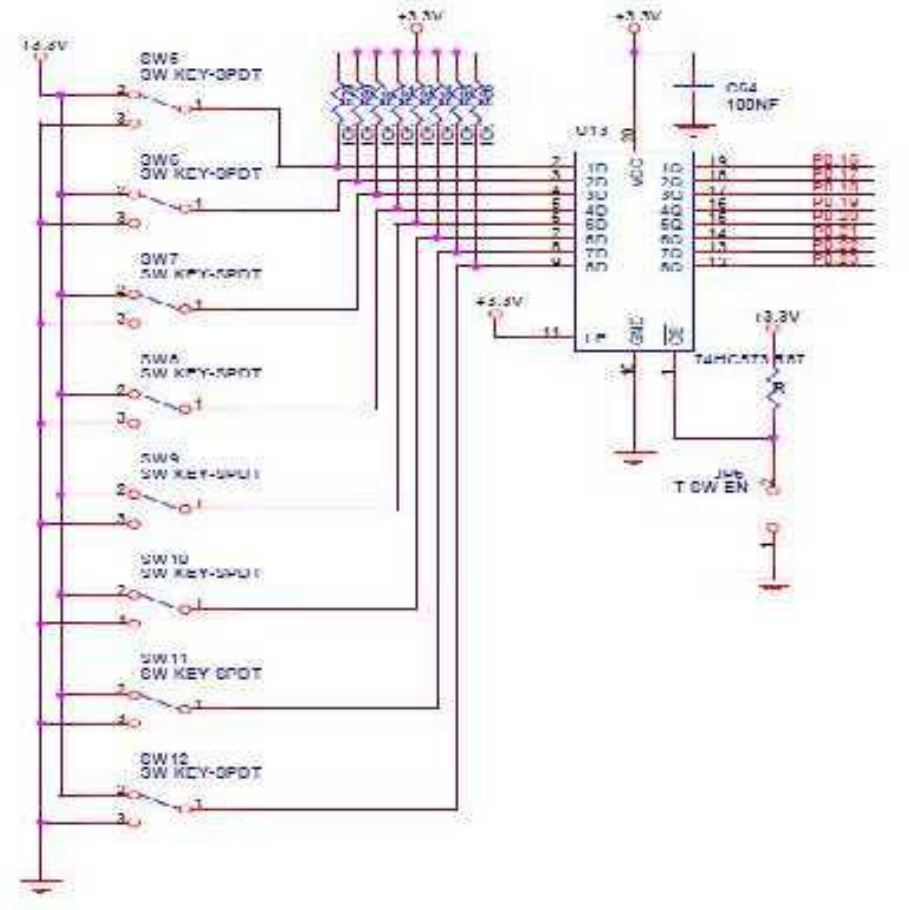


Fig. 5: Toggle Switches are interfaced to port pins of the LPC2148 MC

Buzzer/PWM out:

There is also a small piezoelectric buzzer on the *LPC2148 Education Board*. *Figure* below illustrate the buzzer part of the design. The alternative PWM feature of pin P0.7 (the PWM2 signal) can be used to modulate the buzzer to oscillate around different frequencies. It's not the pulse width feature that is used to change the frequency. Only the volume of the sound will be changed by alternating the pulse width. Instead, it's possible to change the frequency of the PWM signal, and this will also change the frequency of with the buzzer oscillate.

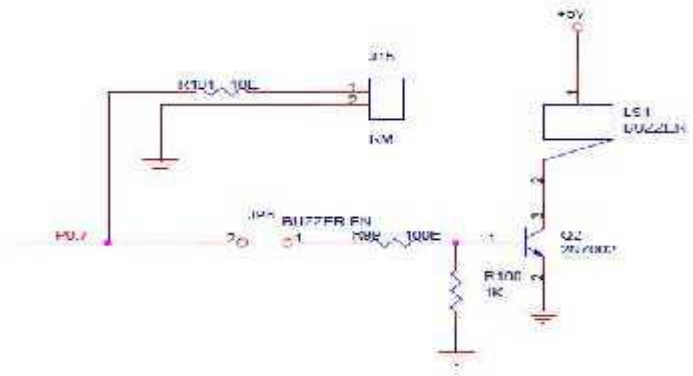


Fig. 6: PWM interfaced to port pins of the LPC2148 MC

The buzzer can be disconnected by removing jumper JP8, and this is also the default position for this jumper since the buzzer sound can be quite annoying if always left on. The relimate connector (J15) is connected to P0.7 for obtaining PWM out, to connect externally.

16x2 LCD

The part of the LPC2148 SDK Board is the 2x16 character LCD. The interface to the LCD is a simulated memory bus. 11 pins are needed to create an 8-bit interface; 8 data bits (D0-D7), 1 address bit (RS), 1 read/write bit (R/W), and one control signal (E) via 74HC573 buffers U11 and U12. *Figure* below illustrate the LCD part of the design and which pins are used for the interface.

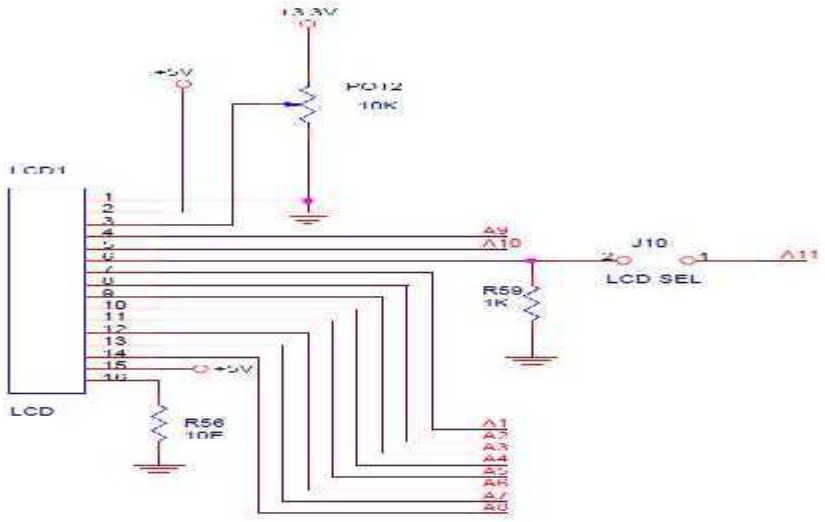


Fig. 7: LCD interfaced to port pins of the LPC2148 MC

The LCD can be selected using the jumper(J10). The LCD is powered from the +5V power supply. The display contrast adjustment (via POT2) is mounted for adjusting the LCD's contrast to the users need.

Experiment –1

Demonstration of Switch and LED Interface

1.1 Aim:

To Write a Program to demonstrate Switch and LED Interface using ARM kit.

1.2 Apparatus:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Serial USB cable
5. Power Adapter (for Physitech Kit)

1.3 Theory:

We are going to read the switches and turn ON/OFF the LEDs accordingly. In this program we are going to do both INPUT and OUTPUT operation. The port pin to which switch is connected is configured as Input and the pin to which LED is connected is configured as OUTPUT. Here the switch status is read and accordingly the LED will be turned ON/OFF.

a) What is a switch?

A switch is an electrical component that can break an electrical circuit, interrupting the current or diverting it from one conductor to another. A switch may be directly manipulated by a human as a control signal to a system, or to control power flow in a circuit.

b) Interfacing switch

Figure shows how to interface the switch to microcontroller. A simple switch has an open state and closed state. However, a microcontroller needs to see a definite high or low voltage level at a digital input. A switch requires a pull-up or pull-down resistor to produce a definite high or low voltage when it is open or closed. A resistor placed between a digital input and the supply voltage is called a "pull-up" resistor because it normally pulls the pin's voltage up to the supply.

The push buttons: The Push buttons (SW13-SW20) are interfaced to port P0.16-P0.23 of the LPC2148, via a 74HC573 buffer (U14). The Push buttons are enabled using the jumper JP7.

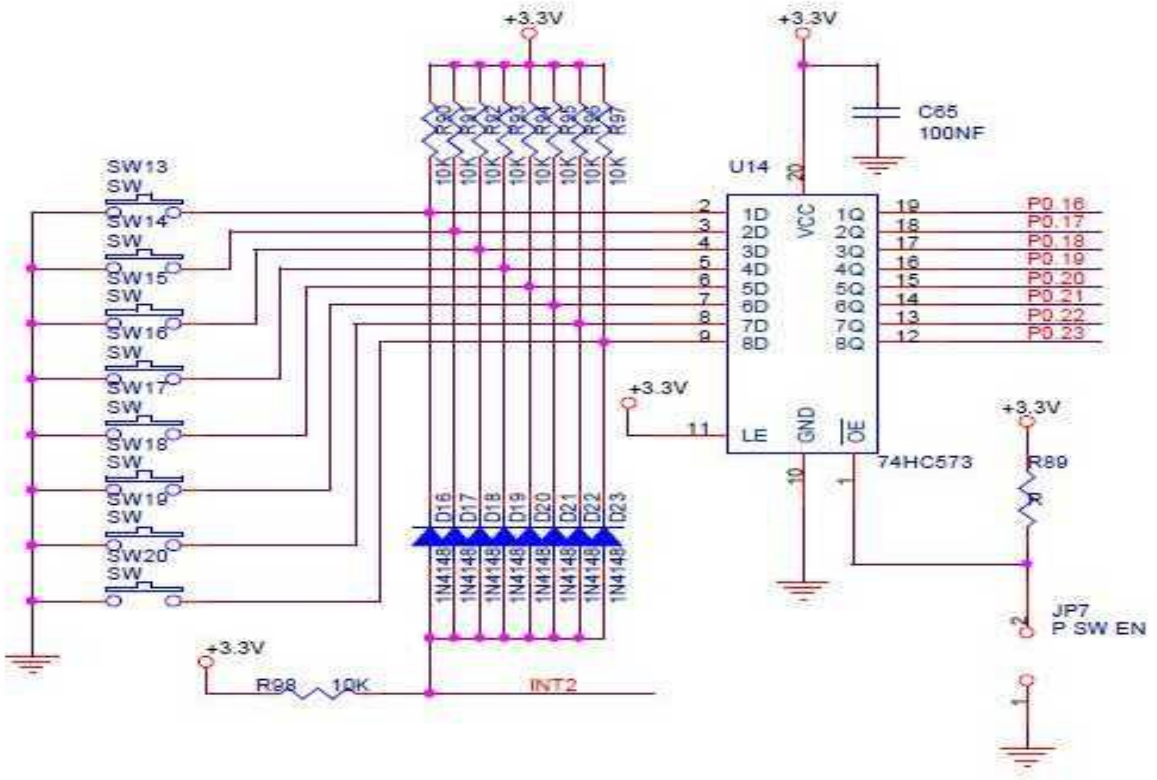


Fig. 1.1: illustrates the push buttons in the design.

Toggle switches: The toggle switches (SW5-SW12) are interfaced to port P0.16-P0.23 of the LPC2148, via a 74HC573 buffer (U13). The toggle switches are enabled using the jumper JP6.

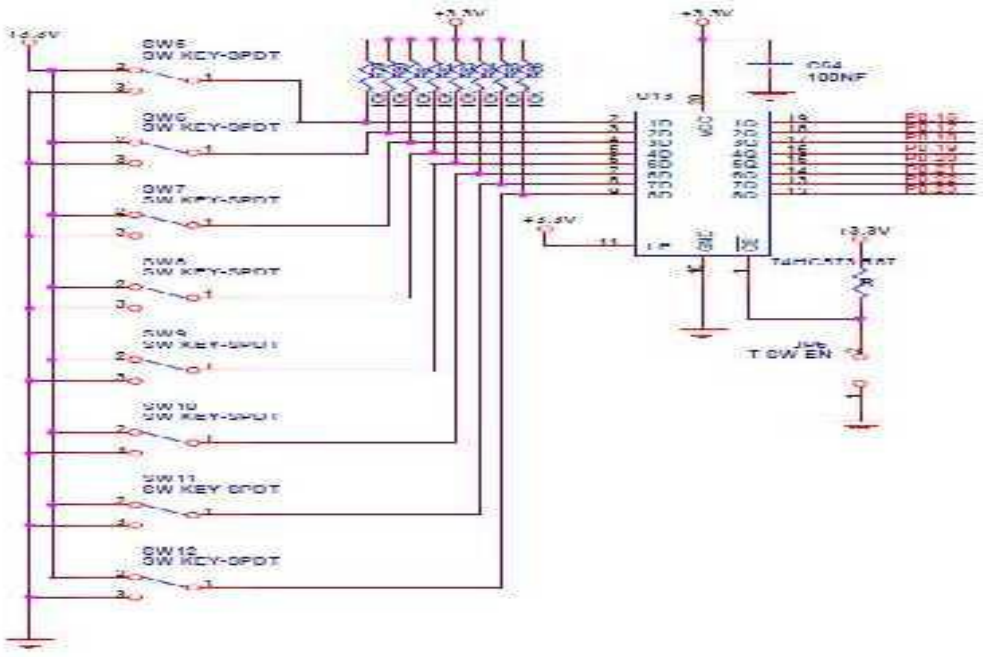


Fig. 1.2: illustrates the toggle switches design.

c) LED (LIGHT EMITTING DIODE):

Light Emitting Diodes (**LED**) is the most commonly used components, usually for displaying pins digital states. Typical uses of **LEDs** include alarm devices, timers and confirmation of user input such as a mouse click or keystroke.

d) Interfacing LED

Figure shows how to interface the **LED** to microcontroller. As you can see the Cathode is connected to GND & the Anode is connected through a resistor is connected to the Microcontroller pin. So when the Port Pin is HIGH the **LED** is ON & when the Port Pin is LOW the **LED is turned OFF.** The port pins P1.16 to P1.23 of the LPC2148 microcontrollers are connected to 8 red LED's(D8-D15) via a Buffer(U20). The buffers enable is given to P1.30 for enabling or disabling the LED operations.

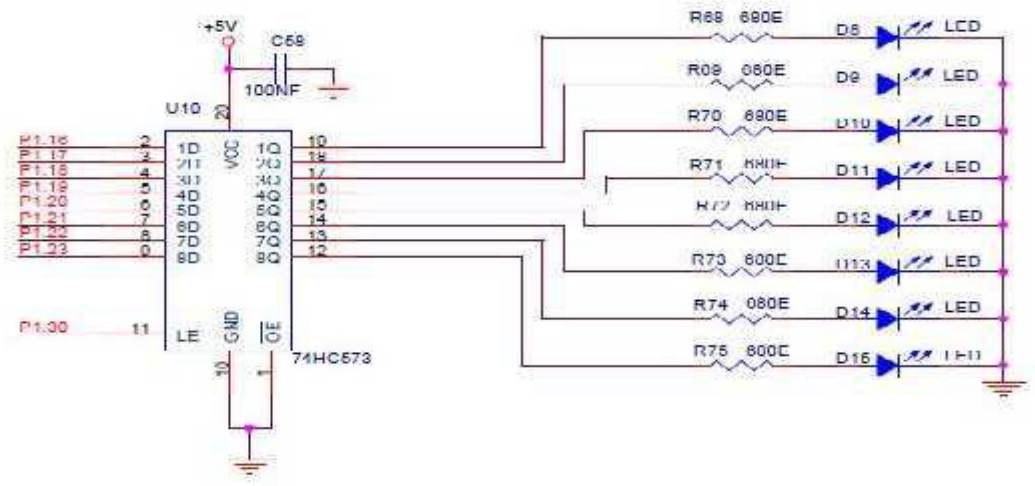


Fig. 1.3: illustrates the 8 LEDs in the design.

Register Configuration

The Below registers will be used for Configuring and using the GPIOs for sending and receiving the Digital signals.

PINSEL: GPIO Pins Select Register

Almost all the LPC2148 pins are multiplexed to support more than 1 function. Every GPIO pin has a minimum of one function and max of four functions. The required function can be selected by configuring the PINSEL register. This implies that we need two PINSEL registers to configure a PORT pins. By this, the first 16(P0.0-P0.16) pin functions of PORT0 can be selected by 32 bits of PINSEL0 register. The remaining 16 bits(P0.16-P0.32) are configured

using 32bits of PINSEL1 register. As mentioned earlier every pin has a max of four functions. Below table shows how to select the function for a particular pin using two bits of the PINSEL register.

Value	Function	Enumeration
00	Primary (default) function, typically GPIO port	PINSEL_FUNC_0
01	First alternate function	PINSEL_FUNC_1
10	Second alternate function	PINSEL_FUNC_2
11	Third alternate function	PINSEL_FUNC_3

IODIR: GPIO Direction Control Register.

This register individually controls the direction of each port pin.

Values	Direction
0	Input
1	Output

IOSET: Port Output Set Register.

This register controls the state of output pins. Writing 1s produces highs at the corresponding port pins. Writing 0s has no effect. Reading this register returns the current contents of the port output register, not the physical port value.

Values	IOSET
0	No Effect
1	Sets High on Pin

IOCLR: Port Output Clear Register.

This register controls the state of output pins. Writing 1s produces lows at the corresponding port pins. Writing 0s has no effect.

Values	IOCLR
0	No Effect
1	Sets Low on Pin

IOPIN: GPIO Port Pin Value Register.

This register is used for both reading and writing data from/to the PORT.

Output: Writing to this register places corresponding values in all bits of the particular PORT pins.

Input: The current state of digital port pins can be read from this register, regardless of pin direction or alternate function selection (as long as pins are not configured as an input to ADC).

Note: It is recommended to configure the PORT direction and pin function before using it.

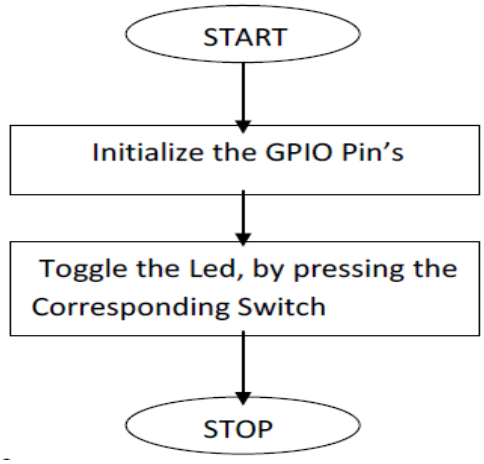
1.4 Algorithm:

STEP1: Initialize the GPIO Pin's

STEP2: Toggle the Led, to press the Corresponding Switch

STEP3: END

1.5 Flow Chart:-



1.6 Program:

```
#include <LPC214X.H>

#define LSB 16                  // The LSB of LED PORT and/or switch Port

#define MASK 0xFF               // Mask Value for 8-Bit Port

#define LED_LATCH_EN 30         // LED Latch Enable bit HIGH-> Enable, LOW-> Disable

/*      Name: delay function Definition

Description: Provides the Delay of Specified time in the order of mSec.
```

```
@Passing Arguments: count specifies the no of mSec.
@Returning Arguments: returns nothing.                                    */
void delay(unsigned int count)
{       unsigned int i,j;
        for(i=0;i<count;i++)
        for(j=0;j<109;j++);
}
int main(void)
{       unsigned int switchPortVal;          // swithPortVal will hold the value read from port.
        IODIR0 &= ~(MASK << LSB);            // Configure the Direction of Switch Port as Inputs
        IODIR1=0x40ff0000;                   // Configure the Direction of LED Port and it Latch
        IOSET1=1<<LED_LATCH_EN;              // Enable the LED Latch
        for(;;)                              // Infinite Loop: Read the values from Switch Port and
        {                                    // Write back to LED Port Indefinitely
           switchPortVal = IOPIN0 & (MASK << LSB);
// Read the P0 Value and Extract the Swith Data P0.16 to P023
           IOSET1 = switchPortVal | (1<<LED_LATCH_EN);// Set the LED Bits, where ever it is high
           IOCLR1 = ~( switchPortVal | (1<<LED_LATCH_EN)) ;
// Clear the LED Bits, where ever it is LOW
           delay (100);
        }
}
```

1.7 Result:

We have been interfaced the Switch and LED to ARM processor and observed the output on LED by glowing when switch is ON state, LED OFF when switch is OFF state.

Experiment-2

Demonstration of Buzzer Interface

2.1 Aim:-

To Write a Program to demonstrate Buzzer Interface using ARM kit.

2.2 Apparatus:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Serial USB cable
5. Power Adapter (for Physitech Kit)

2.3 Theory:

To make constant beep using Buzzer, we used Pulse Width Modulation (PWM) Concept. The Buzzer pin is given to PWM Channel 2. To enable Buzzer, use **JP8** Jumper. For buzzer, give 12V-DC power supply using adaptor.

Buzzer/PWM out:

There is also a small piezoelectric buzzer on the *LPC2148 Education Board*. *Figure* below illustrate the buzzer part of the design. The alternative PWM feature of pin P0.7 (the PWM2 signal) can be used to modulate the buzzer to oscillate around different frequencies. It's not the pulse width feature that is used to change the frequency. Only the volume of the sound will be changed by alternating the pulse width. Instead, it's possible to change the frequency of the PWM signal, and this will also change the frequency of with the buzzer oscillate.

The buzzer can be disconnected by removing jumper JP8, and this is also the default position for this jumper since the buzzer sound can be quite annoying if always left on. The relimate connector (J15) is connected to P0.7 for obtaining PWM out, to connect externally.

2.4 To Set the Clock:

ADPwmSetClock();

To Initialize PWM,

ADInitPWM();

To Set the Parameters,

ADPwmSetParameters();

2.5 ALGORITHM:

STEP1: Set the Clock

STEP2: Initialize PWM

STEP3: Select The Corresponding Channel

STEP4: Give The Duty cycle, Total period

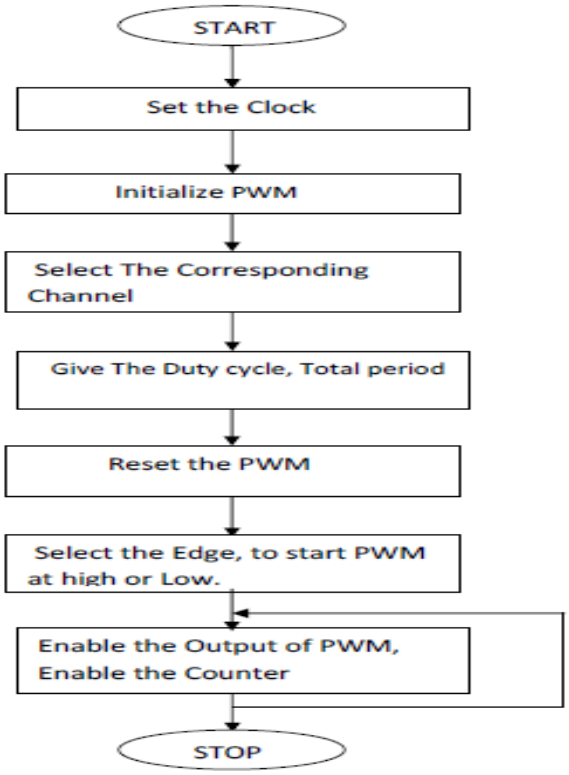
STEP5: Reset the PWM

STEP6: Select the Edge start with high or Low.

STEP7: Enable the Output Of PWM, Enable the Counter

STEP8: END

2.6 Flow Chart:-



2.7 Program:

```
#include <LPC214X.H>

#include "pwm.h"                /*User Defined header files for PWM_driver file. */

 /*Name: main Descripion: Top function of the Experiment, incldudes the functionality to beep the buzzer with pwm based delay.

@Passing  Arguments : void

@Returning Arguments: returns the success of funcion.                */

int main(void)

{

     ADPwmSetClock();                          // setting the clock for PWM as 60Mhz.

     ADInitPWM(PWM2,1,50000,200000);

//setting the PWM2, edge as 1,duty_Cycle-50mSec,total_Period-200mSec

    ADPwmSetParameters(PWM2,50000,200000);

//setting the parameters PWM2,duty_Cycle-50mSec,total_Period-200mSec

    while(1);


}
```

2.8 Result: we have been interfaced the Buzzer module to ARM processor and observed the beep sound as output.

Experiment 3

Demonstration of LCD Display Interface

3.1 Aim:

To Write a Program to demonstrate LCD Display Interface using ARM kit.

3.2 Apparatus:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Serial USB cable
5. Power Adapter (for Physitech Kit)

3.3 Theory:

For LCD, we are doing interfacing 2 x16 LCD with ARM7 Board by giving the some predefined commands to LCD using GPIO (P1.16 to P1.23) pins. In the Program, putStrL Function is used to print the data. In this function, we have two arguments. First one is what is the message have to print and second one is where should that message print in LCD. Use J10 Jumper to enable LCD in hardware. The same thing can observe in keil debugger by selecting PORT1.

16x2 LCD:

The part of the LPC2148 SDK Board is the 2x16 character LCD. The interface to the LCD is a simulated memory bus. 11 pins are needed to create an 8-bit interface; 8 data bits (D0-D7), 1 address bit (RS), 1 read/write bit (R/W), and one control signal (E) via 74HC573 buffers U11 and U12.

The LCD can be selected using the jumper(J10). The LCD is powered from the +5V power supply. The display contrast adjustment (via POT2) is mounted for adjusting the LCD's contrast to the users need.

To Set the Clock

setClock();

To Initialize LCD

lcdInit();

To Send The Command to LCD, Use

putComL();

To Send The Data to LCD, Use

putCharL();

putStrL();

3.4 Algorithm:

STEP1: Set the Clock

STEP2: Initialize LCD

STEP3: Initialize GPIO for LCD Pins

STEP4: Clear RS Bit

STEP5: Enable LCD

STEP6: Send Command

STEP7: Disable LCD

STEP8: if Command count not equal to Zero ,Repeat STEP4

STEP9: Set RS Bit

STEP10:- Send Data

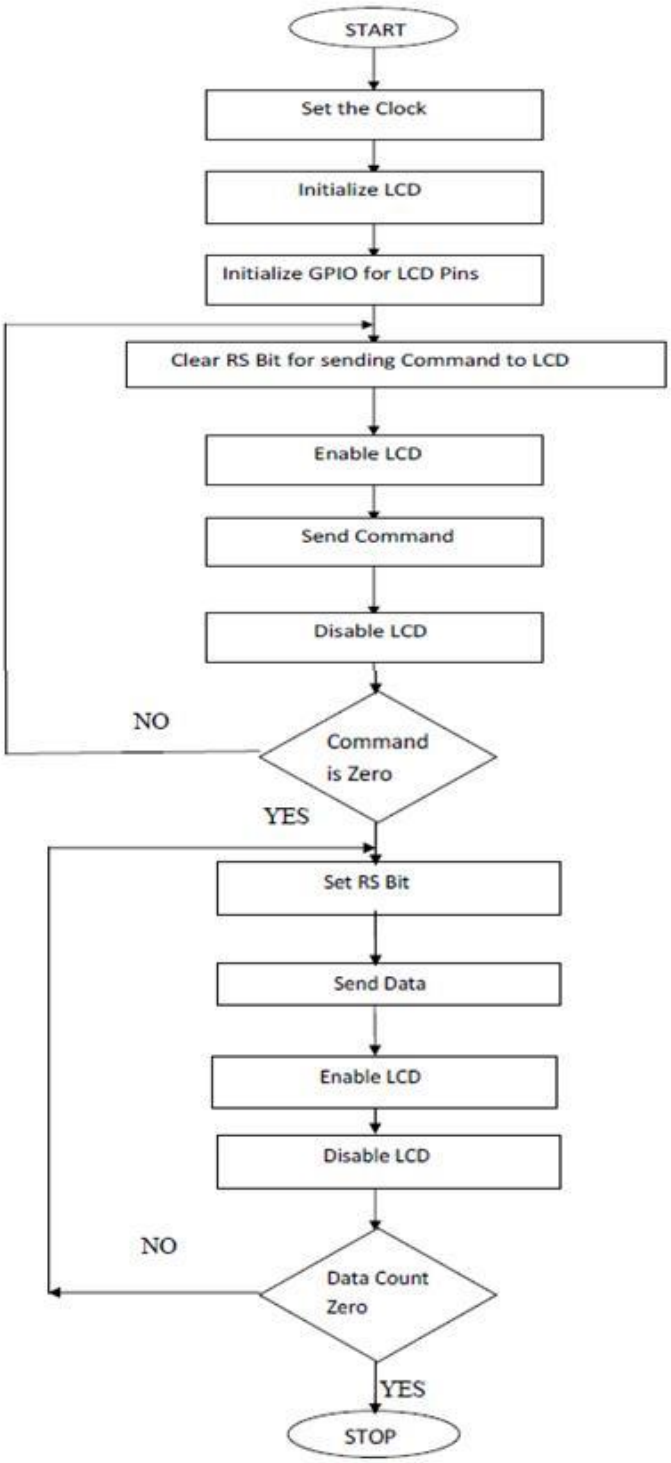
STEP11:- Enable LCD

STEP12:- Disable LCD

STEP13:- If Data Count is not equal to Zero repeat STEP9

STEP14:-END

3.5 **Flow Chart:-**



3.6 **Program:**

```
#include <RTL.h>                        /* RTX kernel functions & defines      */
#include <LPC214X.H>
#include "lcd.h"
extern void init_serial (void);         /* Initialize Serial Interface         */
OS_TID tsk1, tsk2;
OS_SEM semaphore1;
void delay(unsigned int count)
{
        unsigned int i,j;
        for(i=0;i<count;i++)
        for(j=0;j<8255;j++);
}
/*---------------------------------------------------------------------------
 *    Task 1 - High Priority - Active every 3 ticks
 *---------------------------------------------------------------------------*/
__task void task1 (void)
{
        OS_RESULT ret;
        static unsigned int count1 = 0;
        while (1)
{
                os_dly_wait(3);                 /* Pass control to other tasks for 3 OS ticks */
                ret = os_sem_wait (semaphore1, 1);  /* Wait 1 ticks for the free semaphore */
                if (ret != OS_R_TMO)
{                   /* If there was no time-out the semaphore was aquired */
                    delay(100);
                    putStrL("              ",0x80);
                    putStrL("Task 1",0x80+count1);
                    if(count1++ == 10)count1 = 0;
                    os_dly_wait(20);
                            os_sem_send  (semaphore1);    /* Return a token back to a
semaphore */
```

```
                }
        }
}

/*---------------------------------------------------------------------------
 *    Task 2 - Low Priority - looks for a free semaphore and uses the resource
 *                        whenever it is available
 *---------------------------------------------------------------------------*/

__task void task2 (void)
{
    static unsigned int count2 = 0;

    while (1)
    {                   /* Wait indefinitely for a free semaphore */
        os_sem_wait (semaphore1, 0xFFFF);    /* OK, the serial interface is free now, use it.
*/
        putStrL("               ",0xC0);
        putStrL("Task 2",0xCA-count2);
        if(count2++ == 10)count2 = 0;
        os_dly_wait(20);                          /* Return a token back to a semaphore. */
        os_sem_send (semaphore1);
     }
}

/*---------------------------------------------------------------------------
 *    Task 3 'init'
 *---------------------------------------------------------------------------*/
__task void init (void)
{
     lcdInit();
     os_sem_init (semaphore1, 1);     /* Initialize the Semaphore before the first use */
     tsk1 = os_tsk_create (task1, 1);   /* Create an instance of task1 with priority 10 */
     tsk2 = os_tsk_create (task2, 0);   /* Create an instance of task2 with default priority 1 */
```

```
    os_tsk_delete_self ();                /* Delete the init task */
}

/*----------------------------------------------------------------------------
 *    Main: Initialize and start RTX Kernel
 *----------------------------------------------------------------------------*/
int main (void)
{
   os_sys_init (init);              /* Initialize RTX and start init   */
}
```

3.7 Result:

We have been interfaced the 16X2 LCD display module to ARM processor and observed the output message “Task1” and “Task2” on LCD display module.

Experiment 4

Demonstration of Stepper Motor interface using ARM7

4.1 Aim:

To Write a Program to demonstrate stepper motor interface to ARM kit.

Apparatus:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Flash Magic
5. Serial USB cable
6. Stepper Motor
7. Power Adapter (for Physitech Kit)

4.2 Theory:

Stepper Motor:

A stepper motor is a brushless, synchronous electric motor that converts digital pulses into mechanical shaft rotation. Every revolution of the stepper motor is divided into a discrete number of steps, and the motor must be sent a separate pulse for each step.

Interfacing Stepper Motor

Fig. 1 shows how to interface the Stepper Motor to microcontroller. As you can see the stepper motor is connected with Microcontroller output port pins through a ULN2803A array. So when the microcontroller is giving pulses with particular frequency to ULN2803A, the motor is rotated in clockwise or anticlockwise.

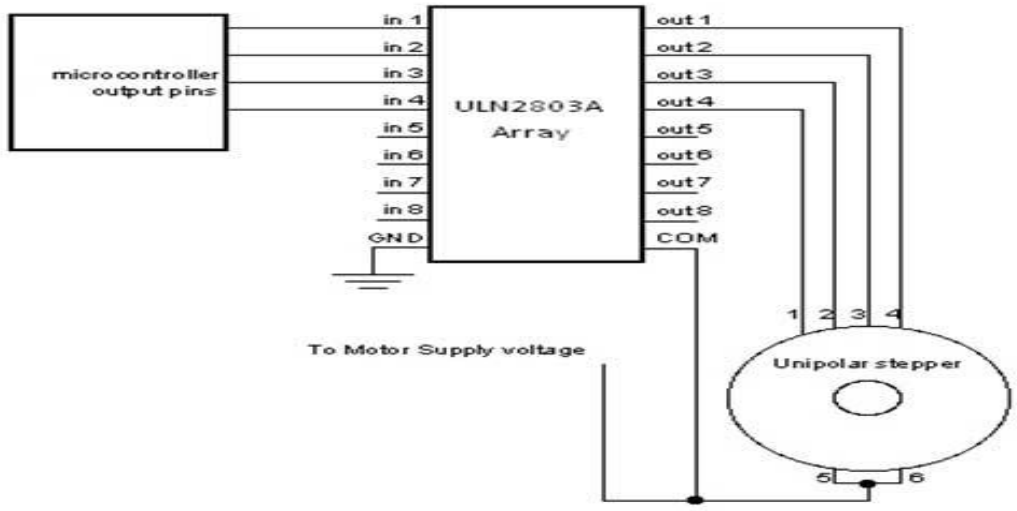


Fig. 4.1: Interfacing Stepper Motor to Microcontroller

Interfacing Stepper Motor with LPC2148:

Controlling a stepper motor using LPC2148 Development Board. It works by turning ON & OFF a four I/O port lines generating at a particular frequency.

The ARM7 LPC2148 Development Board has four numbers of I/O port lines, connected with I/O Port lines (P1.16 – P1.19) to rotate the stepper motor. ULN2803 is used as a driver for port I/O lines, drivers output connected to stepper motor, connector provided for external power supply if needed.

4.3 Circuit Diagram to Interface Stepper Motor with LPC2148

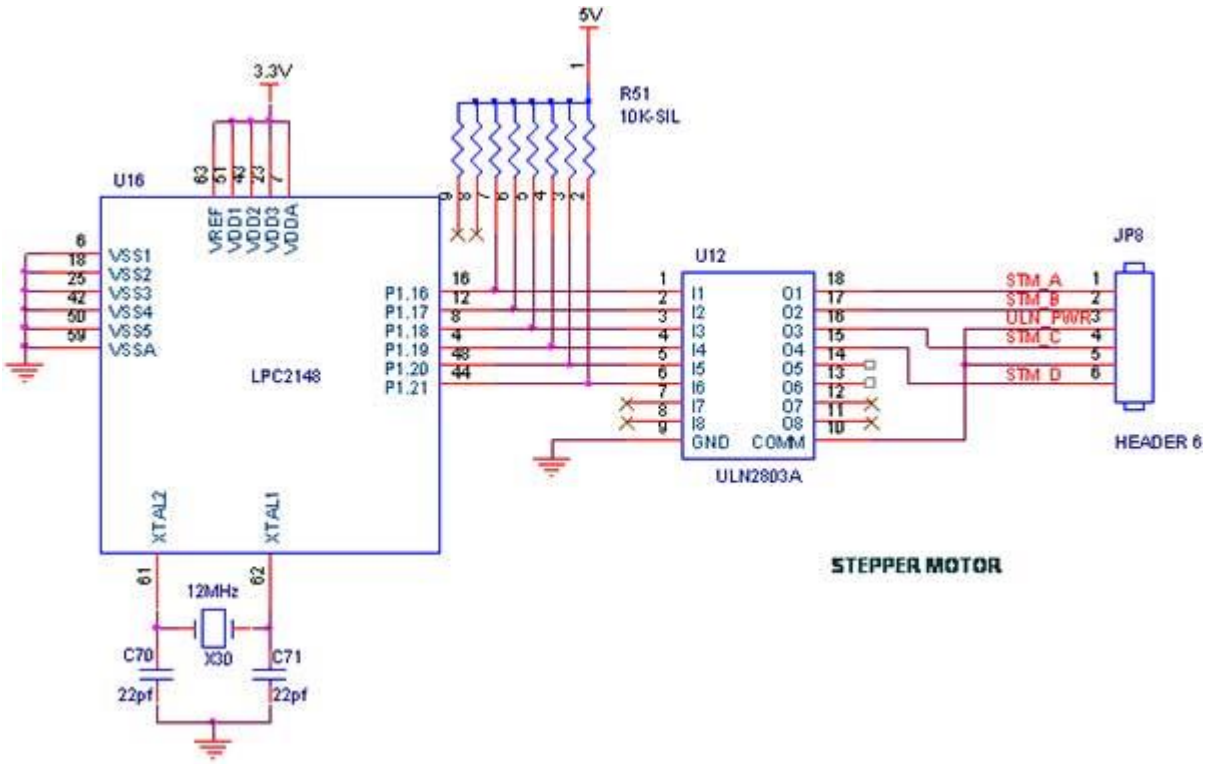


Fig. 4.2: Interface Stepper Motor with LPC2148

4.4 Program:

```
#include <lpc214x.h>
#define SW1 24 //SW1 (P1.24)
#define SW2 25 //SW2 (P1.25)
#define SW3 26 //SW3 (P1.26)
#define COIL_A 16
//change the Stepper Motor Port! void motor_cw(void);
void motor_ccw(void);
void delay(int);
```

```
unsigned char STEP[] = {0x09, 0x08, 0x0C, 0x04, 0x06, 0x02, 0x03, 0x01};
void main(void)
{
unsigned char i = 0;
PINSEL2 &= 0xFFFFFFF3;
// P1.16 - P1.31 as GPIO IODIR1 = 0x000F0000;
// P1.16 - P1.19 as Output while(1)
// Loop forever { if (!(IOPIN1 & (1<<SW1)))
// Switch SW1 ON/OFF { motor_cw();
// Stepper Motor clockwise } else if (!(IOPIN1 & (1<<SW2)))
// Switch SW2 ON/OFF { motor_ccw(); // Stepper Motor anticlockwise
}
else if (!(IOPIN1 & (1<<SW3)))// Switch SW3 ON/OFF
{
        while (i < 12)
        {
                motor_cw ();
                // clockwise for req. angle i++;
        }
}
else i = 0;
}
}
void delay(int n)
{
        int i,j;
        for(i=0;i<n;i++)
        {
                for(j=0;j<0x3FF0;j++)
                {
                ;
                }
        }
}
```

```
void motor_ccw(void)
{
        unsigned int i=0;
        while (STEP[i] != '\0')
        {
                IOSET1 = STEP[i] << COIL_A;
                delay(1);
                IOCLR1 = STEP[i] << COIL_A;
                delay(1); i++;
        }
}
void motor_cw(void)
{
        int i = 7; while (i >= 0)
        {
                IOSET1 = STEP[i] << COIL_A;
                delay(1);
                IOCLR1 = STEP[i] << COIL_A;
                delay(1);
        i--;
        }
}
```

4.5 Testing the Stepper Motor with LPC2148:

Give +3.3V power supply to LPC2148 Development Board; the Stepper Motor is connected with LPC2148 Development Board**.** When the program is downloading into LPC2148 in Primer Board, the LED output is working that the LED is ON some time period and the LED is OFF some other time period for a particular frequency. Now, the stepper motor is rotating.

☞If you are pressed switch sw1, then the stepper motor is rotating in clockwise direction.

☞If you are pressed switch sw2, then the stepper motor rotating in anti-clockwise direction.

☞If you are pressed switch sw3, then the motor is rotating the required angle which angle is set in code.

If you are not reading any output from LED, then you just check the jumper connections & check the LED is working.

4.6 Result:

We have been demonstrated the stepper motor interface to kit via serial cable and observed the output.

Experiment 5

Demonstration of ADC & Temperature sensor LM35 interface

5.1 Aim:

To Write a Program to demonstrate ADC & Temperature sensor LM35 interfaces using ARM7 kit.

5.2 Apparatus:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Flash Magic
5. Serial USB cable
6. RS-232 serial cable
7. Power Adapter (for Physitech Kit)

5.3 Theory:

Analog-to-digital conversion (ADC) is necessary because, while embedded systems deal with digital values (as we have deal with keypads and switches). Analog signals such as, temperature, speed and pressure are generated by peripheral devices such as microphones, analog cameras, sensors, and etc. They all need to be converted into digital data before being processed by the microcontroller.

Temperature sensor and ADC POT1:

This is a simple design using LM35 temperature sensor. The output of the LM35 is connected to the ADCIN2, via a 100E resistor and a jumper J6, of the LPC2148.

The POT1 is connected to port P0.28 ADC channel of the microcontroller. This function can be enabled using the jumper J5.

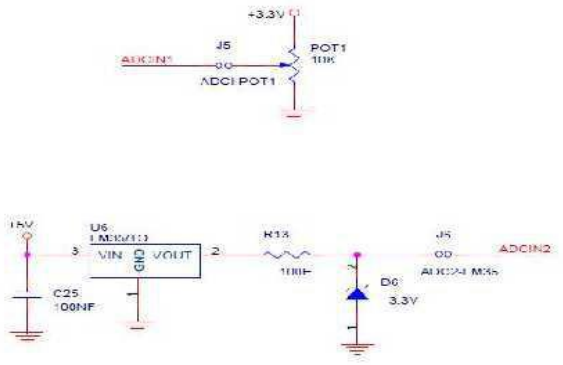


Fig. 5.1:illustates the ADC POT1 interfacing to port pins design.

16x2 LCD:

The part of the LPC2148 SDK Board is the 2x16 character LCD. The interface to the LCD is a simulated memory bus. 11 pins are needed to create an 8-bit interface; 8 data bits (D0-D7), 1 address bit (RS), 1 read/write bit (R/W), and one control signal (E) via 74HC573 buffers U11 and U12. *Figure* below illustrate the LCD part of the design and which pins are used for the interface.

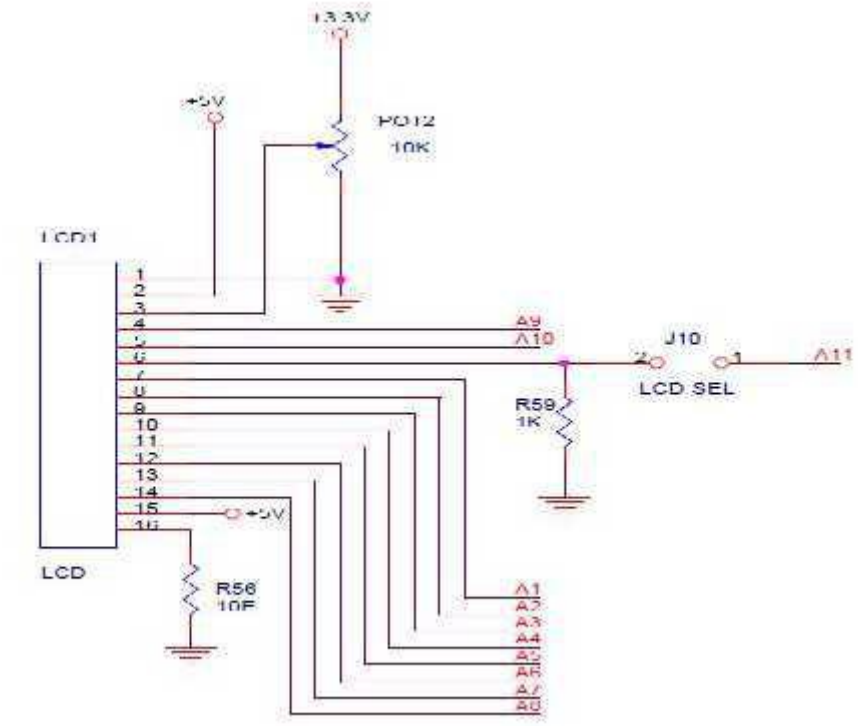


Fig. 5.2:illustates the LCD interfacing to pins design.

The LCD can be selected using the jumper(J10). The LCD is powered from the +5V power supply. The display contrast adjustment (via POT2) is mounted for adjusting the LCD's contrast to the users need.

5.4 Program:

```
#include<lpc214x.h>
#include<stdio.h>
#define adc_sel 27 //for pinsel
#define clkdiv 8
#define ch0 0
#define burst_mode 16
#define pdn 21
```

```
#define start 24
#define done 31
#define lcdport IO0SET
#define lcdportclr IOCLR0
#define rs 12
#define en 13
#define TEMP_PIN        (1<<29)     //P0.29 as Input to Temperature Sensor
#define TEMP_PIN_DIR    (1<<26)     //Bit 27:26 of PINSEL1 register
#define _PDN_BIT 1<<21
#define _ADCR_START_MASK 7<<24
#define _ADCR_SEL_MASK 0x000000FF
#define _ADC0_START 1<<24

void delay(int t)
{
        int i,j;
        for(i=0;i<t;i++)
        for(j=0;j<5000;j++);
}
void cmnd()
{
        lcdportclr=(1<<rs);
        //lcdportclr = (1<<rw);
        lcdport = (1<<en);
        delay(40);
        lcdportclr=(1<<en);
}
void lcdcmd(char ch)
{
    lcdport = ((ch&0xf0)<<13);
    cmnd();
    lcdportclr = ((ch&0xf0)<<13);

    lcdport = (((ch<<4)&0xf0)<<13);
```

```
        cmnd();
        lcdportclr = (((ch<<4)&0xf0)<<13);
}

void daten()
{
        lcdport=(1<<rs);
        //lcdportclr = (1<<rw);
        lcdport = (1<<en);
        delay(40);
        lcdportclr=(1<<en);
}

void lcddata(char ch)
{
        lcdport = ((ch&0xf0)<<13);
        daten();
        lcdportclr = ((ch&0xf0)<<13);

        lcdport = (((ch<<4)&0xf0)<<13);
        daten();
        lcdportclr = (((ch<<4)&0xf0)<<13);
}

void lcdstring(char *str)
{
        int j;
        for(j=0;str[j]!='\0';j++)
        {
                lcddata(str[j]);
        }
}
void lcd_init()
{
```

```
	lcdcmd(0x02);
	lcdcmd(0x28);
	lcdcmd(0x01);
	lcdcmd(0x0e);
}

void io_init()
{
	PINSEL0=0X000;
	IODIR0=0xffffff;
}
void Adc0Init(unsigned char clk)
{
	PCONP |= 0x00001000;//Power on the A/D converter 0
	//configure the A/D control register of A/D 0
	AD0CR =((unsigned long)(clk+1)<<8 ) | _PDN_BIT ;
}
unsigned int Adc0Read(unsigned char channel)
{
	static unsigned val;
	AD0CR &= ~(_ADCR_START_MASK|_ADCR_SEL_MASK);  //stop the A/D converter by masking the

//start bits and channel selection bit
	AD0CR |=((unsigned long)(1)<<channel);  //Select the A/D channel
	AD0CR |=_ADC0_START;
	while(!(AD0GDR & (0x80000000)));  //Wait for the conversion to get over

//by monitoring the 28th bit of A/D data register
	AD0CR &= ~(_ADCR_START_MASK|_ADCR_SEL_MASK);  //Stop the conversion by masking the start bits

	val = AD0GDR;
```

```
        val = ((val>>6 & 0x03FF));                              //Extract A/D result
        return(val);
}
int main()
{
        float temperature=0.0;
        char result[5];
        PINSEL1 |=    TEMP_PIN_DIR;      //Bit 27:26 of PINSEL1 register
        IODIR0        &=      ~(TEMP_PIN);          //Select the AD0.3 of P0.28 as Input

        io_init();
        lcd_init();
        lcdcmd(0x84);
        lcdstring("Temperature");
        Adc0Init(10);

        while(1)
        {

                        temperature = (((float)Adc0Read(2)/1023.0)*3.3*100);
                        lcdcmd(0xc5);
                        sprintf(result,"%f",temperature);
                        lcdstring(result);
                        lcddata(0xDF);          //for degree character
                        lcddata('C');
                        delay(100);

        }
}
```

5.5 Result: we have been demonstrated the temperature sensor (LM35), ADC to kit and observed the output.

Experiment 6

Demonstration of Serial Communication in ARM7

6.1 Aim:

To Write a Program to demonstrate Transmission from ARM kit and reception from PC using Serial port Communication.

6.2 Apparatus:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Serial USB cable
5. RS-232 serial cable
6. Power Adapter (for Physitech Kit)

6.3 Theory:

In ARM7, we have two Serial Communications (UART0 and UART1). Here, we are sending the some information to PC from target board (ARM7) by using UART0/1 (Universal Asynchronous Receiver/Transmitter) which works on the protocol of RS232. The Required message will send by using putStrS0/1 Function. If you want to send the Characters, use putCharS0/1 function. To see the sent message in PC, Use Hyper terminal (set baud rate as 9600).

UART#0(ISP) and UART#1:

There are two UART channels on the LPC2148 and only channel #1 has all control signals needed for a full modem implementation. Channel #0 consists receive and transmit signals. Channel #1 is free and used by some of the expansion boards.

Even though the CPU clock crystal is 12.000 MHz, any standard baud rate can be generated since both UART channels have a fractional baud rate generator. 'Fractional' means that the clock frequency does not have to be divided by an even integer value. It can also be divided by a fractional value, which means that a lot more baud rates can be achieved from basically any clock crystal frequency. There is even an *auto bauding* capability, meaning that basically any baud rate can be detected.

To Set the Clock, use:

setClock();

To Initialize UART0:

InitSerial0(9600);

To Write The Data We Can , Use:

putCharS0();

putStrS0();

To Read The Data We Can ,Use:

getCharS0();

6.4 Algorithm:

STEP1: Set the Clock

STEP2: Initialize UART0

STEP3: END

PutCharS0:-

STEP1: Initialization , Load LSRs with Respective values

STEP2: Copy data into Accumulator (or) memory location

STEP3: if THRE==1

STEP4: Copy into THR.

STEP5: END

GetCharS0:-

STEP1: Initialization, Load LSRs with Respective values

STEP2: RBR==1

STEP3: Copy into RBR Location

STEP4: END

6.5 Program:

```
#include <LPC214X.H>
#include "serial1.h"    /*User Defined Header Files for Common and serial_Driver files*/
#include "common.h"
/*  Name: main
Descripion: Top function of the Experiment, includes the functionality to write and read the
data using serial Communication.
@Passing  Arguments : void
@Returning Arguments: returns the success of funcion.      */
int main(void)
{
```

```
        setClock();                                 //setting clock for 60Mhz.
        InitSerial1(9600);                          // Setting Baudrate for 9600bps
        putStrS1("UART1 Test");                     // Writing the String into UART
        InitSerial1Int((unsigned)serial1_RxISR);    // Uart1 ISR
        while(1);                                   // End of Loop
}

/*  Name: serial1_RxISR
Descripion: This function helps to red the data from UART1 and display the red data.
@Passing  Arguments : void.
@Returning Arguments: none.                 */
void serial1_RxISR(void) __irq
{
        unsigned char ch;                           //Holds the red data
        ch = getCharS1();                           // getting the data from UART1
        putCharS1(ch);                              // Printing the red data
         VICVectAddr          = 0x00000000;                    //Dummy write to signal
end of interrupt
}
```

6.6 Result:

We have been demonstrated the serial communication from PC to kit via serial cable and observed the output on terminal window of PC.

Experiment 7

Demonstration of EEPROM interface in ARM7

7.1 Aim:

To Write a Program to demonstrate EEPROM interface in ARM kit and using I2C Communication.

7.2 Apparatus:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Serial USB cable
5. RS-232 serial cable
6. Power Adapter (for Physitech Kit)

7.3 Theory:

EEPROM:

The LPC2148 SDK also contains an E 2PROM accessible via the I2C interface. The LPC2148 microcontroller has two on-chip I2C communication channels. Channel #0 is used for communicating with the E2PROM. The other channel can optionally be used on an expansion board. More peripheral units are easily connected to the two-wire I2C bus, just as long as the addresses do not collide. The I2C interface to the EEPROM can be disconnected from the LPC2148 by removing jumpers on J1 and J2. Figure below shows the EEPROM section.

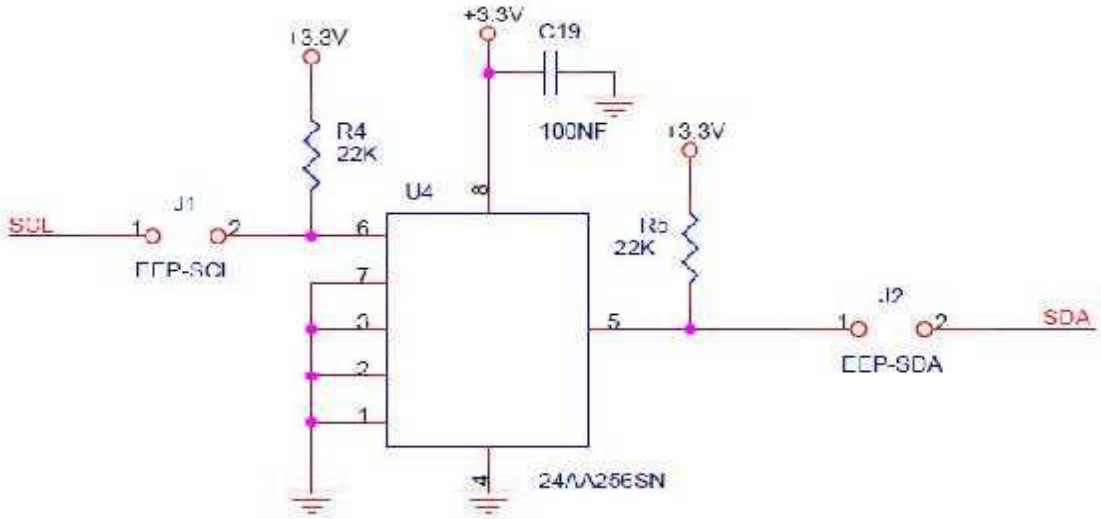


Fig. 7.1: the EEPROM section in ARM7 Based Development board.

I2C is a protocol for Synchronous serial Communication. This Communication will set up by setting SCK and SDA. ARM7 support I2C protocol and to enable I2C Protocol, use J1 and J2 Jumpers. For I2C Based EEPROM, in the code use putStrE and getStrE functions for write and read the data using I2C protocol. These both functions has two arguments. First one is what is the message have to store in EEPROM and second one is from where to read the message. LCD Interface will help to show the how communication will work.

To Set The Clock

setClock();

To Initialize LCD, I2C

lcdInit();

i2cInit();

To Write The Data into Eeprom ,use

putCharE();

putStrE();

To Read The Data into Eeprom, use

getCharE();

getStrE();

7.4 Algorithm:

STEP1: Set the Clock

STEP2: Initialize LCD

STEP3: Initialize I2C

STEP4: END

PutCharE:- To Write The Data into Eeprom ,use

STEP1: Assign Eeprom Address into AddrH and AddrL

STEP2: Enable I2C Interface

STEP3: If Start Address Condition is transmit

STEP4: Clear STA

STEP5: Clear I2C Interface

STEP6: Eeprom Write Address

STEP7:- Eeprom Write Address into Data Register

STEP8:- Clear I2C Interface

STEP9:- If write Address Condition transmit

STEP10:- Higher Byte Address into Data Register

STEP11:- Clear I2C Interface

STEP12: If Higher Data Byte In I2DAT transmit
STEP13: Lower Byte Address into Data Register
STEP14: Clear I2C Interface
STEP15: If lower Data Byte In I2DAT transmit
STEP16: Data Argument Into Data Register
STEP17: Clear I2C Interface
STEP18: If Data Byte In I2DAT transmit
STEP19: Clear I2C Interface
STEP20: Stop I2C Protocal
STEP21: END

GetCharE:- To Read The Data into Eeprom ,use

STEP1: Assign Eeprom Address into AddrH and AddrL
STEP2: Enable I2C Interface
STEP3: If Start Address Condition is transmit
STEP4: Clear STA
STEP5: Clear I2C Interface
STEP6: Eeprom Write Address
STEP7:- Eeprom Write Address into Data Register
STEP8:- Clear I2C Interface
STEP9:- If write Address Condition transmit
STEP10:- Higher Byte Address into Data Register
STEP11:- Clear I2C Interface
STEP12: If Higher Data Byte In I2DAT transmit
STEP13: Lower Byte Address into Data Register
STEP14: Clear I2C Interface
STEP15: If lower Data Byte In I2DAT transmit
STEP16: Enable the I2C Block to Acknowledge, its Own Slave Address
STEP17: Clear I2C Interface
STEP18: If Repeated Start Condition in I2C transmit
STEP19: Eeprom Write Address
STEP20: Eeprom Write Address into Data Register
STEP21: Clear I2C Interface
STEP22: Clear The Start Condition
STEP23: If Slave Read Condition transmit
STEP24: Clear I2C Interface
STEP25: If Data Byte Receiving is transmit
STEP26: Receiving Data into RxData
STEP27: Clear I2C Interface
STEP28: I2C stop
STEP29: Return the RxData
STEP30: END

PutstrE: To Write The Data into Eeprom ,use

STEP1: Assign Eeprom Address into AddrH and AddrL
STEP2: Enable I2C Interface

STEP3: If Start Address Condition is transmit
STEP4: Clear STA
STEP5: Clear I2C Interface
STEP6: Eeprom Write Address
STEP7:- Eeprom Write Address into Data Register
STEP8:- Clear I2C Interface
STEP9:- If write Address Condition transmit
STEP10:- Higher Byte Address into Data Register
STEP11:- Clear I2C Interface
STEP12: If Higher Data Byte In I2DAT transmit
STEP13: Lower Byte Address into Data Register
STEP14: Clear I2C Interface
STEP15: If lower Data Byte In I2DAT transmit
STEP16: Character By Character From String
STEP17: Clear I2C Interface
STEP18: If Higher Data Byte in I2DAT transmit
STEP19: Data is Null
STEP20: Clear I2C Interface
STEP21: If Higher Data Byte in I2DAT transmit
STEP22: Clear I2C Interface
STEP23:I2C Stop
STEP24: END

GetStrE:- To Read The Data into Eeprom ,use
STEP1: Read the Data from Eeprom and Storing into Read Array
STEP2: If Read the Data Untill Null Comes
STEP3: Return the Data
STEP4: END

7.5 Program:

```
#include <LPC214X.h>
/*User Defined header files for eeprom,lcd and common file.*/
#include "eeprom.h"
#include "lcd.h"
#include "common.h"
int main(void)
{
 unsigned char *ptr,x;                  //Holds the Red Data from EEPROM
 setClock();                                    //Clock for 60Mhz.
 lcdInit();                                     // LCD Intialization.
 i2cInit();                                     // I2C Intialization

 putStrL("EEPROM TEST",0x01);           // writing the String into LCD first Row.
 putStrE("OK",0x0001);      // writing the "OK" string to EEPROM at address of 0x0001
 putCharE('-',0x0000);      // writing the '-' character to EEPROM at address of 0x0000
 ptr=getStrE(0x0001);       // Reading the "OK" string from EEPROM  addr of 0x0001 and storing in *Ptr
 x=getCharE(0x0000);        // Reading the '-' Character from EEPROM  addr of 0x0001 and storing in 'x'.
 putStrL(ptr,0xC0);         //displaying the red data into LCD second line.
 putCharL(x);               //displaying the red character into LCD second line
 while(1);                                              //End of loop
}
```

7.6 Result:

We have been demonstrated the EEPROM interface kit via I2C.

PART-B

Implementation of programs using Verilog HDL code on FPGA Board

EXP 1	STUDY OF SIMULATION AND IMPLEMENTATION OF XILINX TOOL

1.1 AIM:

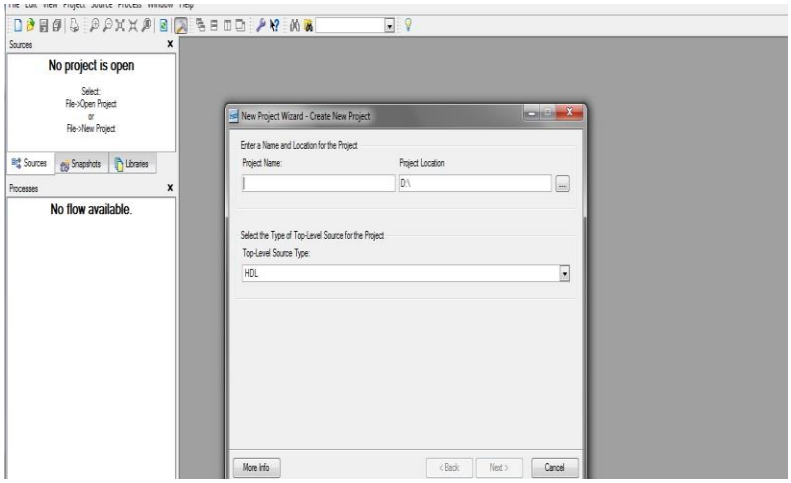
To study the simulation and implementation procedures of Xilinx tool and FPGA

STEP1:

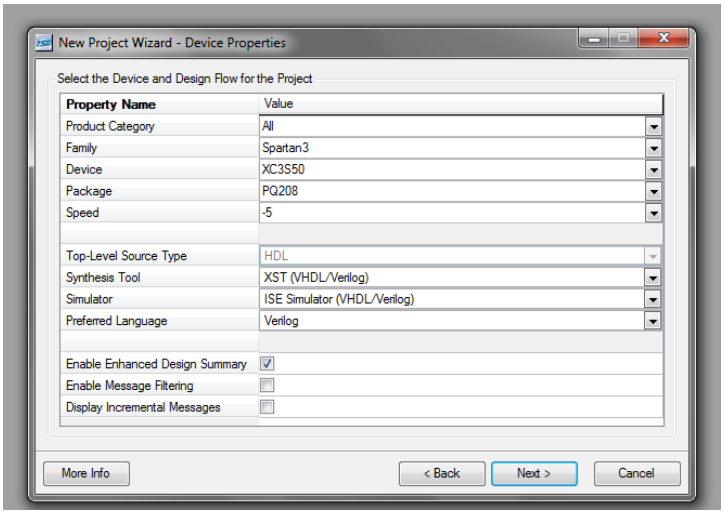
Click Xilinx ISE9.1

STEP2:

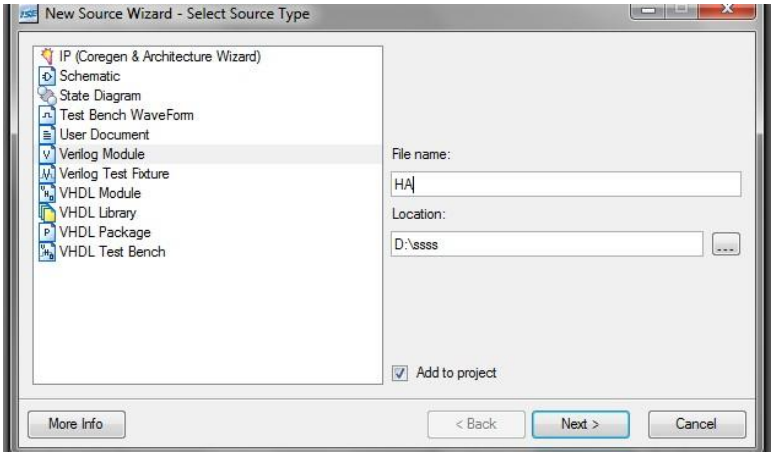
File ->New project and type the project name and check the top level source type as HDL



STEP3: Check the device properties and click Next

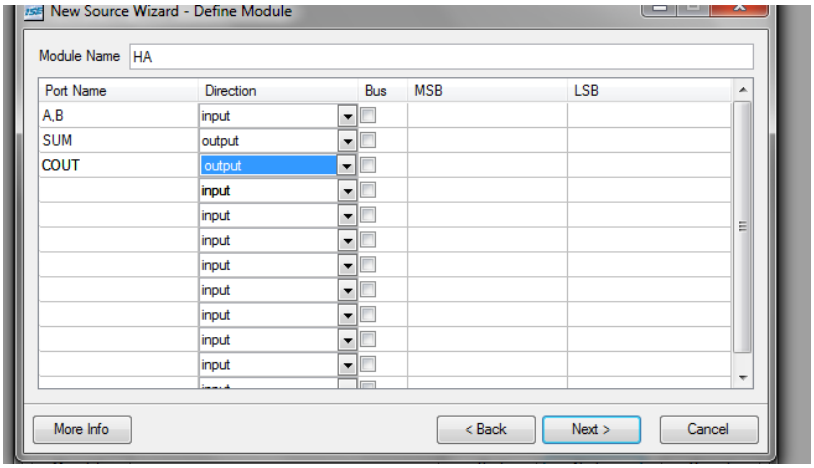


STEP4:Click New Source And Select the Verilog Module and then give the file name



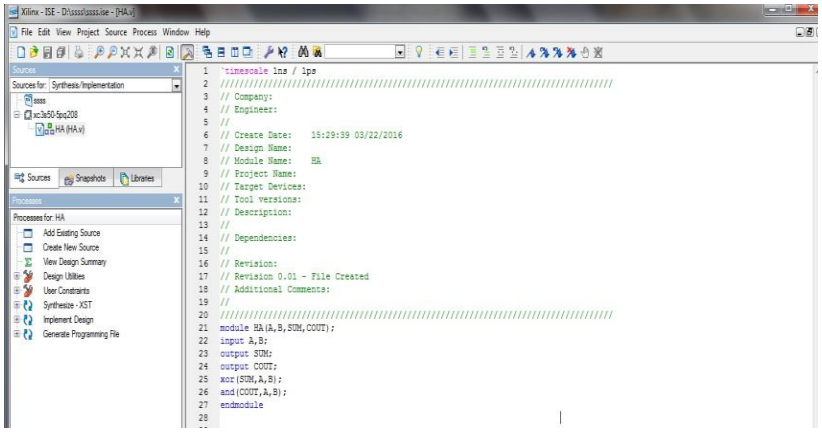
STEP5:

Select the Input,Output port names and click finish.

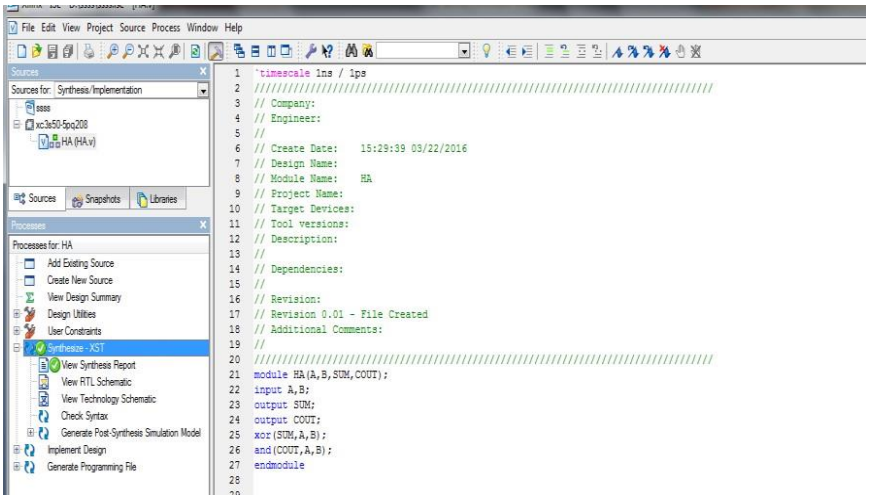


STEP6:

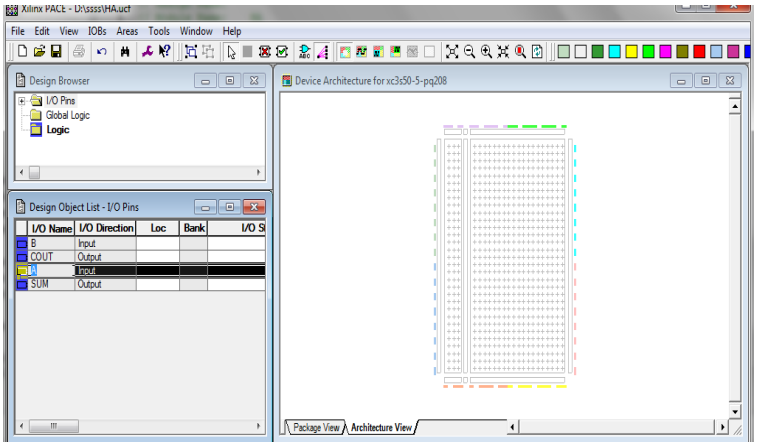
Type the program and save it



STEP7: Check the synthesize XST and check syntax

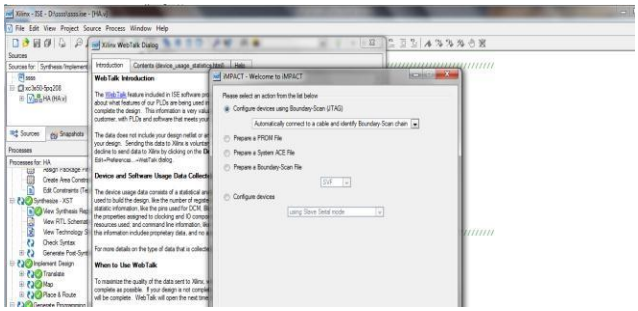


STEP8: Select user constraints-> assign package pins,set port numbers and save it then select IO Bus delimiter as XST default<>->click ok



STEP9:

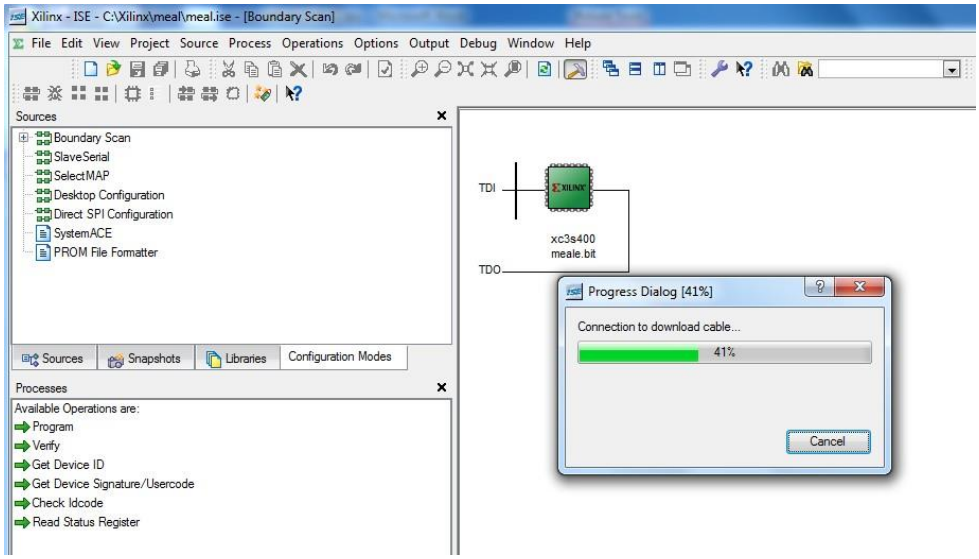
Double click implement design and click generate programming file->configure device(impact)->finish then select bit file



STEP10:

Right click on the xc3s400 figure->program->filename then click finish

and Finally check the functionality in hardware



1.2 Result:

Thus the simulation and implementation procedure of Xilinx and FPGA is studied.

Exp. No.: 2(A)	DESIGN & FPGA IMPLEMENTATION OF LOGIC GATES

2.1 AIM:

To design, simulate and implement basic logic gates using Verilog HDL

2.2 APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

2.3 THEORY:

AND GATE:

The AND gate performs logical multiplication which is most commonly known as the AND junction. The operation of AND gate is such that the output is high only when all its inputs are high and when any one of the inputs is low the output is low.

$$Y = a \& b$$

OR GATE:

The OR gate performs logical addition which is most commonly known as the OR junction. The operation of OR gate is such that the output is high only when any one of its input is high and when both the inputs are low the output is low.

$$Y = a \mid b$$

NOT GATE:

The Inverter performs a basic logic gate function called Inversion or Complementation. The purpose of an inverter is to change one logic level to opposite level. When a high level is applied top an inverter, the low level will appear at the output and vice versa.

$$Y = \sim a$$

NAND GATE:

The term NAND is derived from the complement of AND. It implies the AND junction with an inverted output. The operation of NAND gate is such that the output

is low only when all its inputs are high and when any one of the inputs is low the output is high.

Y = ~(a & b)

NOR GATE:

The term NOR is derived from the complement of OR. It implies the OR junction with an inverted output. The operation of NOR gate is such that the output is high only when all its inputs are low and when any one of the inputs is high the output is low.

Y = ~(a | b)

EX-OR GATE:

The output is high only when the inputs are at opposite level.

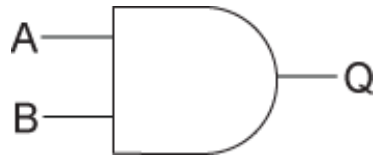
Y = a ^ b

EX-NOR GATE:

The output is high only when the inputs are at same level.

Y = ~(a ^ b)

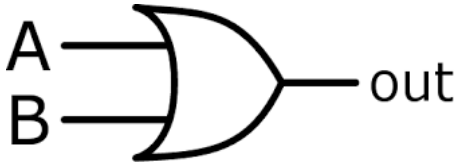
AND Gate:



Truth table:

AND Gate

Input1	Input2	Output
0	0	0
0	1	0
1	0	0
1	1	1



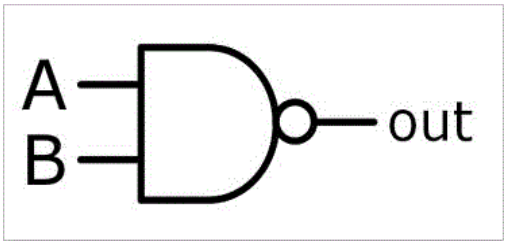
OR Gate:

Truth table:

OR Gate

Input1	Input2	Output
0	0	0
0	1	1
1	0	1
1	1	1

NAND Gate:

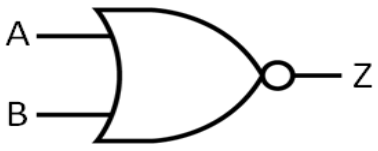


Truth table:

NAND Gate

Input1	Input2	Output
0	0	1
0	1	1
1	0	1
1	1	0

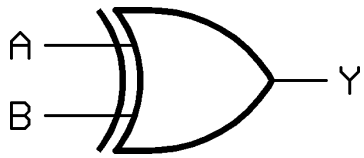
NOR Gate:



Truth table:

NOR Gate

Input1	Input2	Output
0	0	1
0	1	0
1	0	0
1	1	0



XOR Gate:

Truth table: **XOR Gate**

Input1	Input2	Output
0	0	0
0	1	1
1	0	1
1	1	0

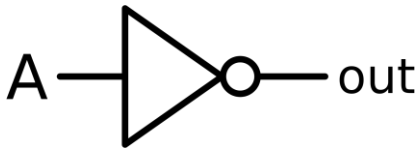
XNOR Gate:

Truth table:

XNOR Gate

Input1	Input2	Output
0	0	1
0	1	0
1	0	0
1	1	1

Not Gate:



Truth table:

NOT Gate

Input	Output
0	1
1	0

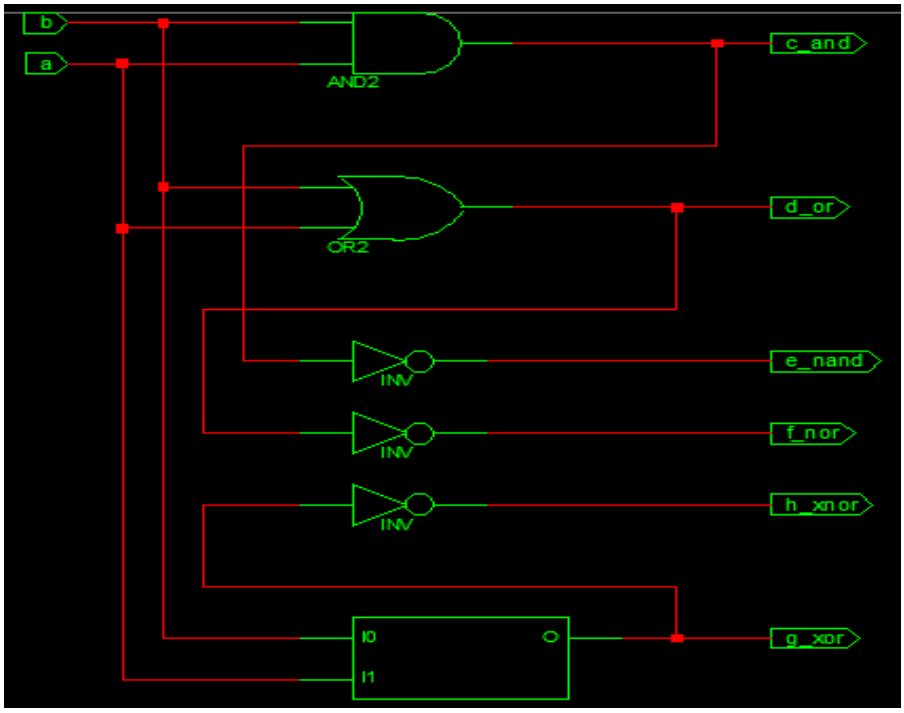
2.4 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

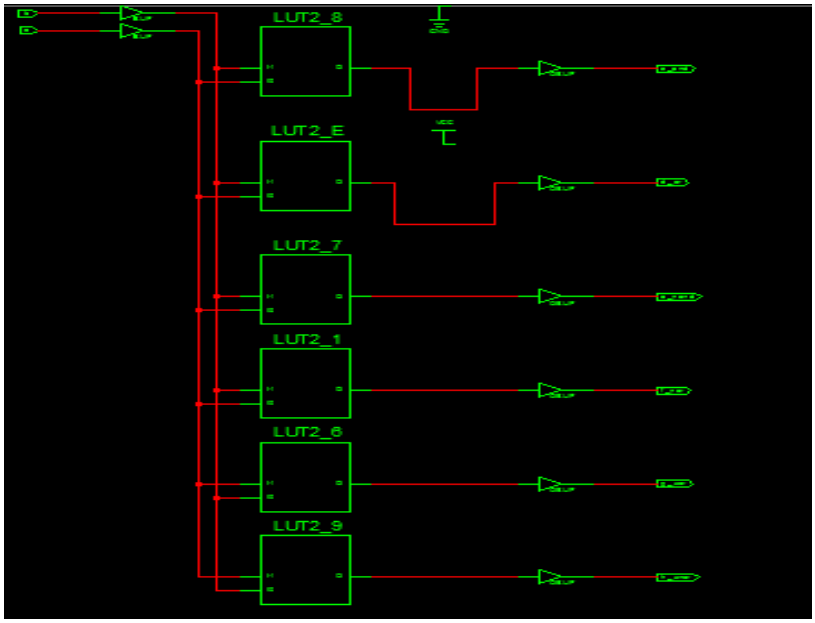
2.5 Program:

```
module gate(a, b, c_and, d_or, e_nand, f_nor, g_xor, h_xnor);
   input a;
   input b;
   output c_and;
   output  d_or;
   output e_nand;
   output f_nor;
   output g_xor;
   output h_xnor;
   and (c_and,a,b);
         or (d_or,a,b);
         nand (e_nand,a,b);
         nor (f_nor,a,b);
         xor (g_xor,a,b);
         xnor (h_xnor,a,b);
         endmodule
```

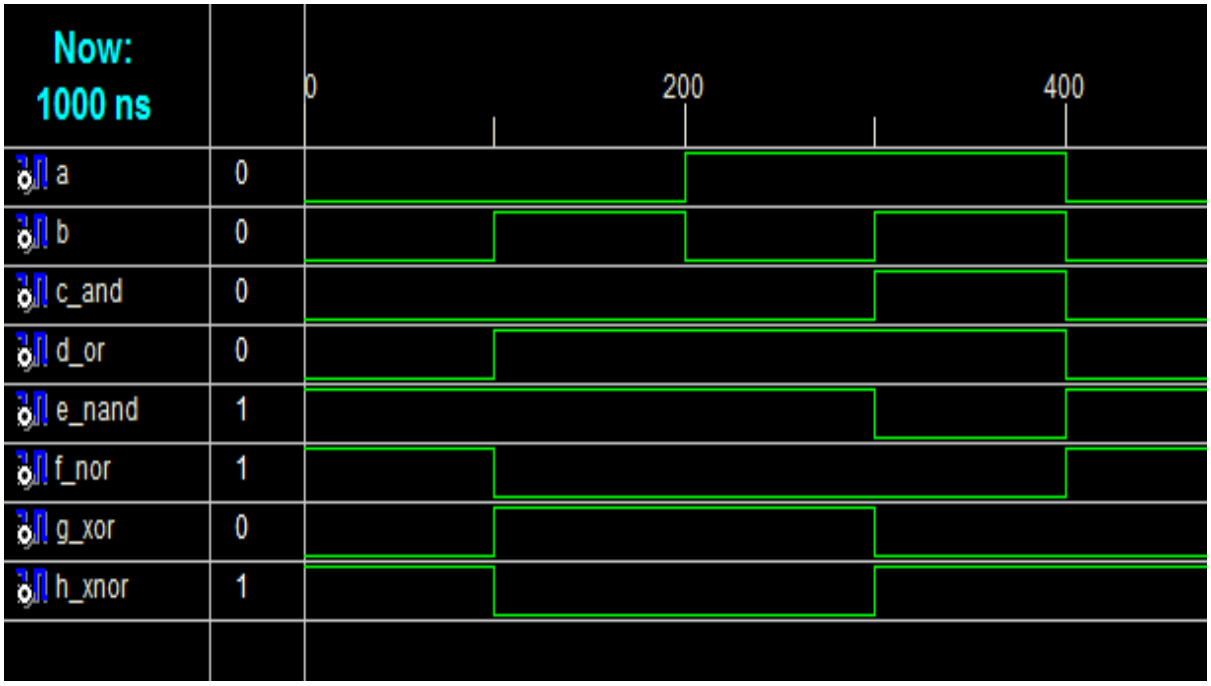
2.6 RTL Schematic:



2.7 Technological Schematic:



2.8 Output Waveform:



2.9 Result:

Thus the basic gates are designed, simulated and implemented using Verilog HDL.

Exp. No.: 2(B)	DESIGN & FPGA IMPLEMENTATION OF HALF ADDER AND FULL ADDER

2.1AIM:

To design, simulate and implement basic half adder and full adder using Verilog HDL

2.2 APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

2.3 THEORY:

HALF ADDER:

The half adder consists of two input variables designated as Augends and Addend bits. Output variables produce the Sum and Carry. The 'carry' output is 1 only when both inputs are 1 and ,sum' is 1 if any one input is 1. The Boolean expression is given by,

sum = x ^ y carry = x & y

FULL ADDER:

A Full adder is a combinational circuit that focuses the arithmetic sum of three bits. It consists of 3 inputs and 2 outputs. The third input is the carry from the previous Lower Significant Position. The two outputs are designated as Sum (S) and Carry (C). The binary variable S gives the value of the LSB of the Sum. The output S=1 only if odd number of 1's are present in the input and the output C=1 if two or three inputs are 1.

sum = x ^ y ^ z

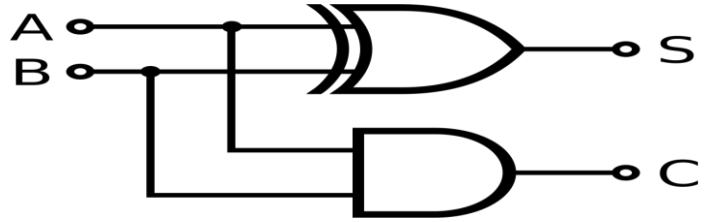
carry= (x & y) | (y & z) | (x & z)

2.4 ALGORITHM

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation

- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

Half Adder:



Program :

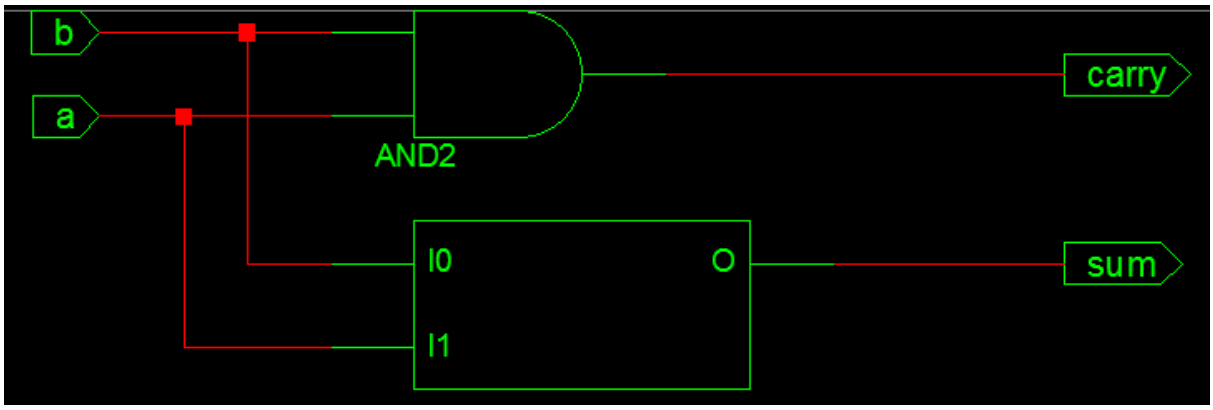
```
Module Half add(a,b,sum,carry);
input a,b;
output sum,carry;
 xor (sum,a,b);
and (carry,a,b);
endmodule
```

Truth table:

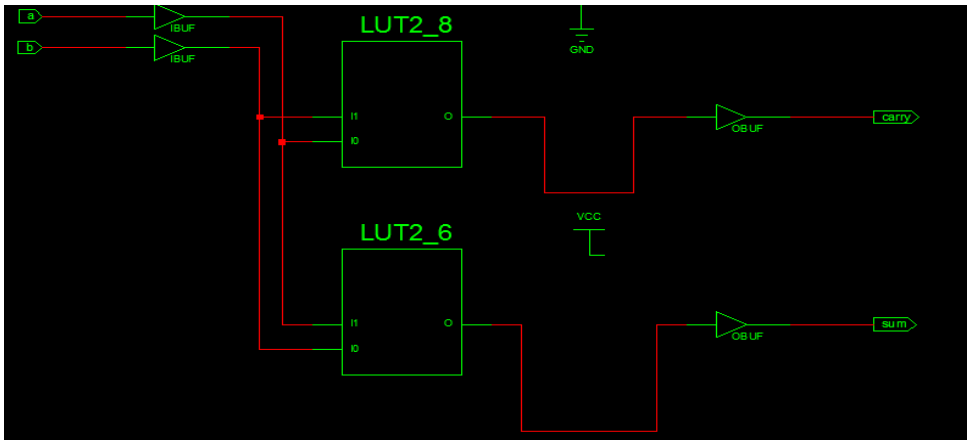
Half Adder

Input1	Input2	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

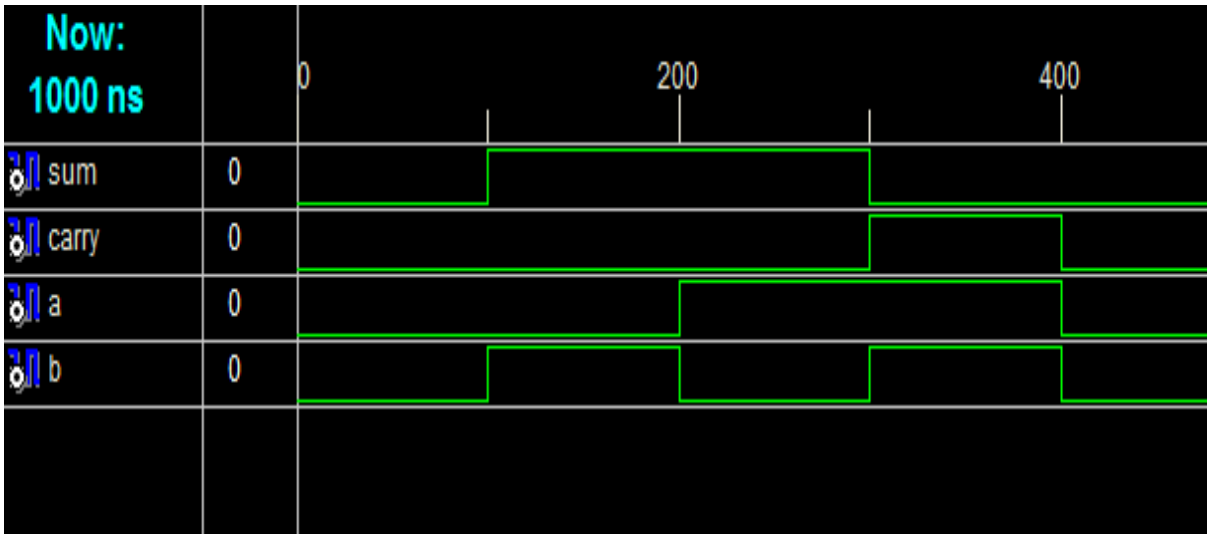
2.4 RTL SCHEMATIC:



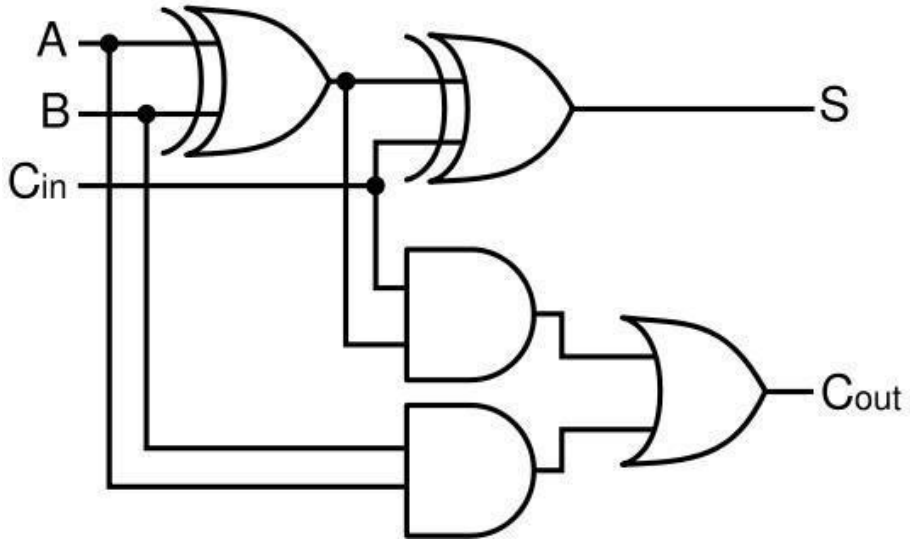
2.5 TECHNOLOGIC SCHEMATIC:



2.6 OUTPUT WAVEFORM:



2.7 Full Adder:



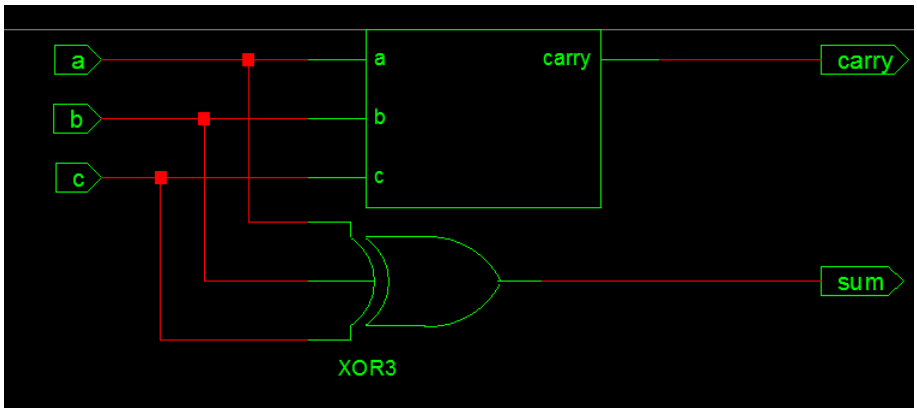
2.8 Program:

```
module fulladd (sum, carry, a, b, c);
input a, b, c;
output sum, carry;
wire w1, w2, w3;
xor (sum,a,b,c);
and (w1,a,b);
and(w2,b,c);
and(w3,c,a);
or(carry,w1,w2,w3);
endmodule
```

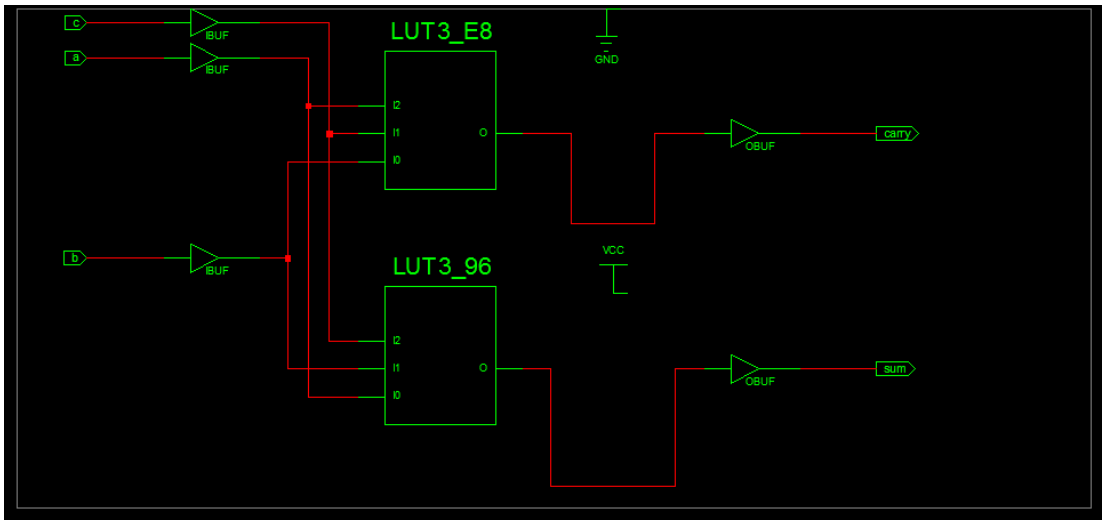
Truth Table:

a	b	c	carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

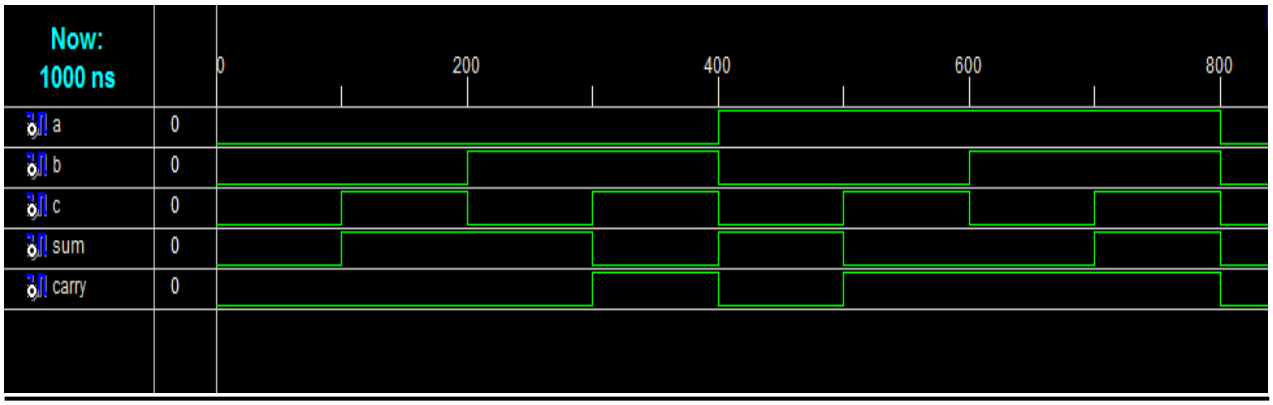
RTL SCHEMATIC:



TECHNOLOGIC SCHEMATIC:



OUTPUT WAVEFORM:



2.9 Result: Thus the half adder and full adder was simulated and implemented successfully.

Exp. No.: 2(C)	DESIGN & FPGA IMPLEMENTATION OF HALF SUBTRACTOR AND FULL SUBTRACTOR

2.C.1AIM:

To design, simulate and implement half subtractor and full subtractor using Verilog HDL.

2.C.2APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

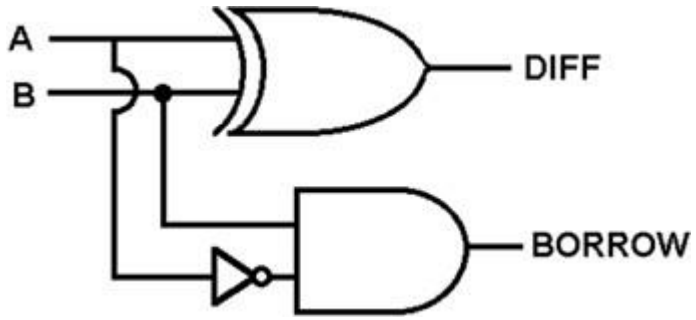
2.C.3Theory:

In electronics, a subtractor can be designed using the same approach as that of an adder. The binary subtraction process is summarized below. As with an adder, in the general case of calculations on multi-bit numbers, three bits are involved in performing the subtraction for each bit of the difference: the minuend (X_{i}), subtrahend (Y_{i}), and a borrow in from the previous (less significant) bit order position (B_{i}). The outputs are the difference bit (D_{i}) and borrow bit B_{i+1}. The subtractor is best understood by considering that the subtrahend and both borrow bits have negative weights, whereas the X and D bits are positive.

2.C.4 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

HALF SUBTRACTOR:



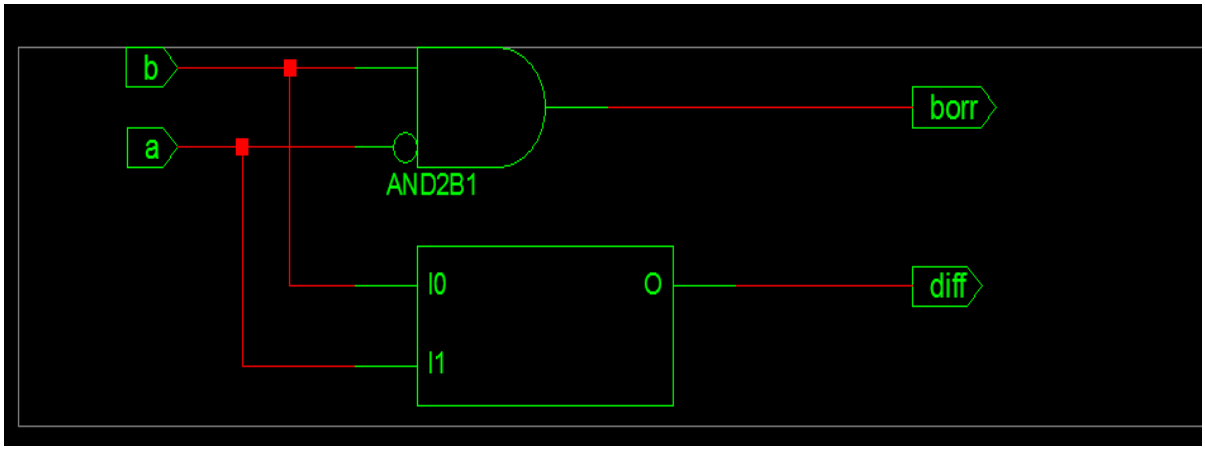
Program:

```
module halfSub(a, b, diff, borr);
input a, b;
output diff, borr;
 wire s;
 not (s, a);
 xor (diff, a, b);
 and (borr, s, b);
 endmodule
```

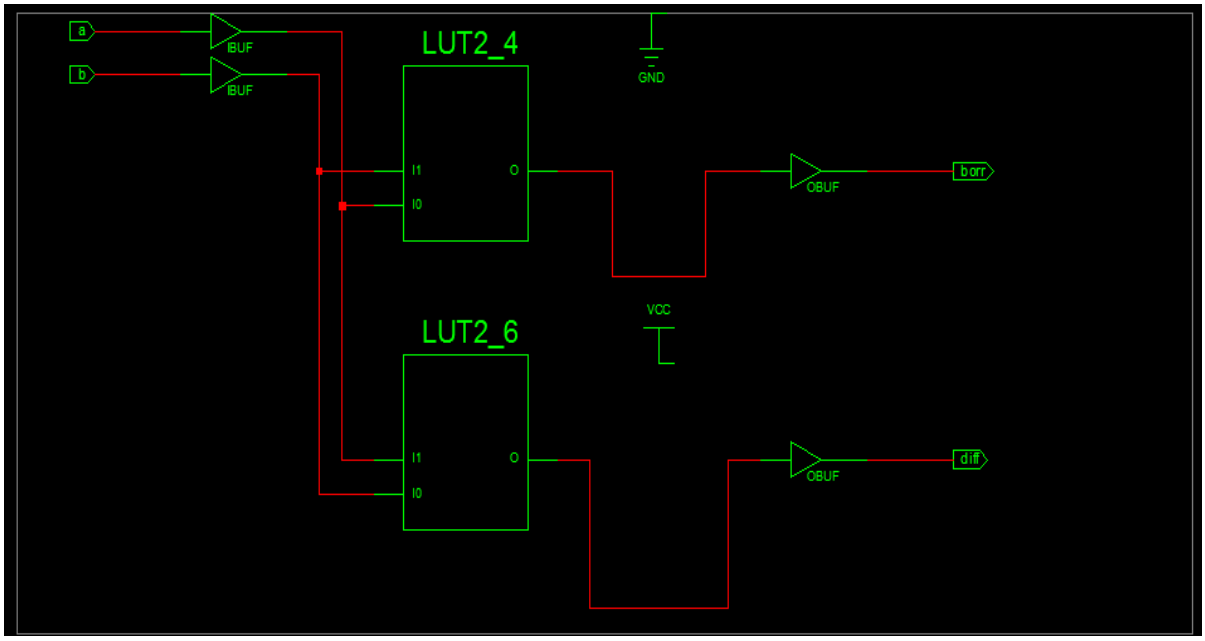
Truth Table:

Input1	Input2	Borrow	Difference
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

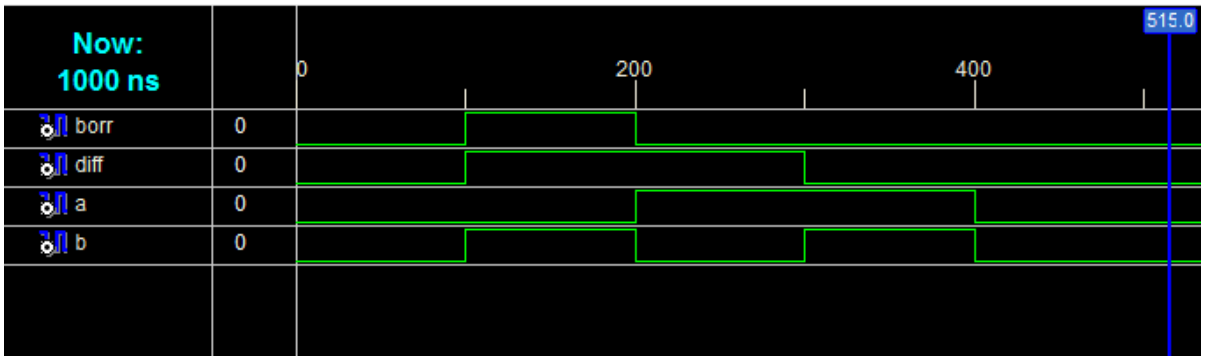
RTL SCHEMATIC



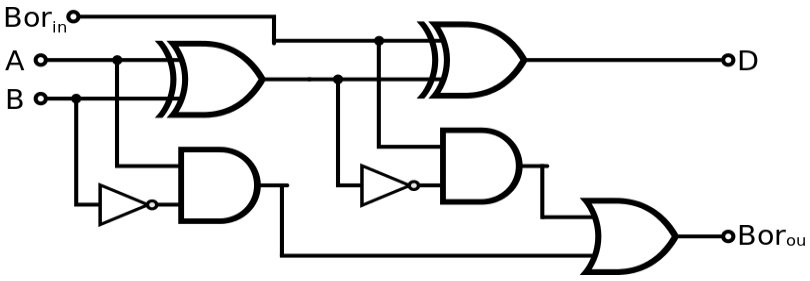
TECHNOLOGIC SCHEMATIC



Output Wave:



FULL SUBTRACTOR:



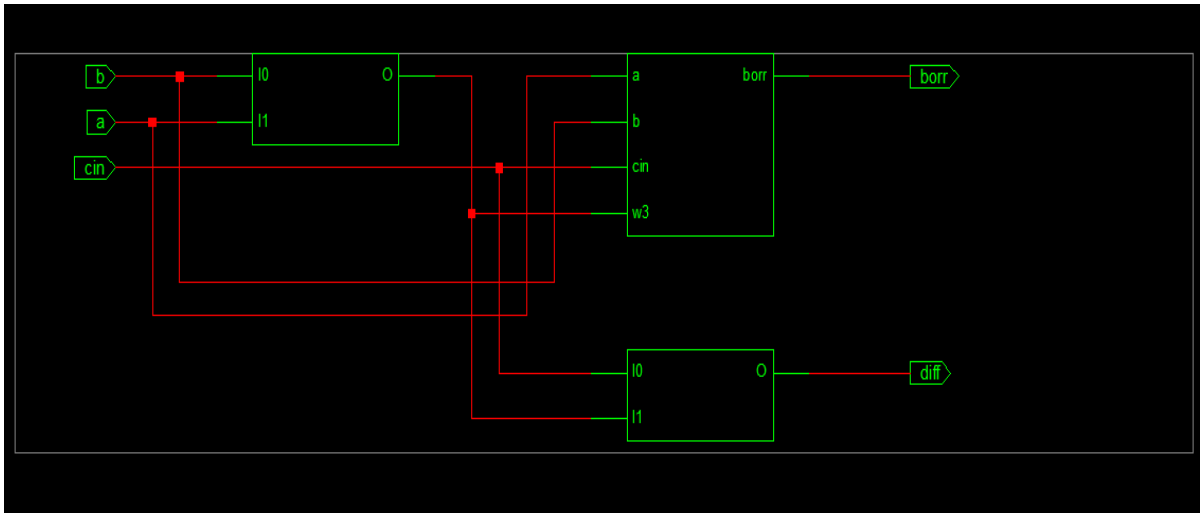
2.C.5 Program:

```
module fullsub (a, b, cin, diff, borr);
    input a, b, cin;
    output diff, borr;
    wire w1, w2, w3, w4, w5;
    not n1(w1, a);
    not n2(w4, w3);
    xor x1(w3, a, b);
    xor x2(diff, w3, cin);
    and a1(w2, w1, b);
    and a2(w5,w4,cin);
    or g1(borr, w5, w2);
    endmodule
```

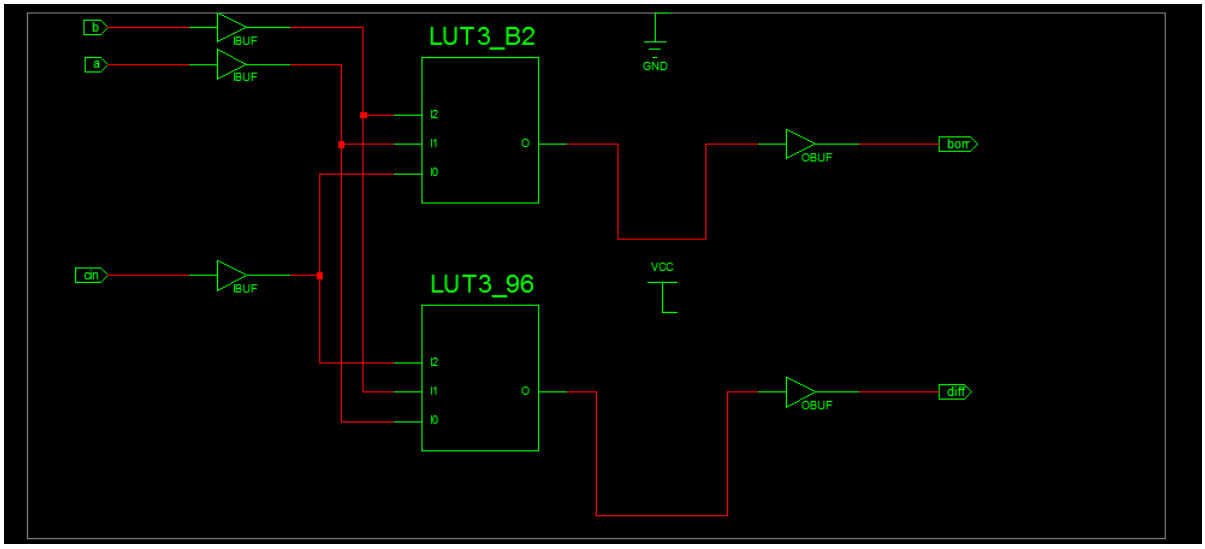
2.C.6 Truth Table:

A	*B*	*Cin*	*D*	B_{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

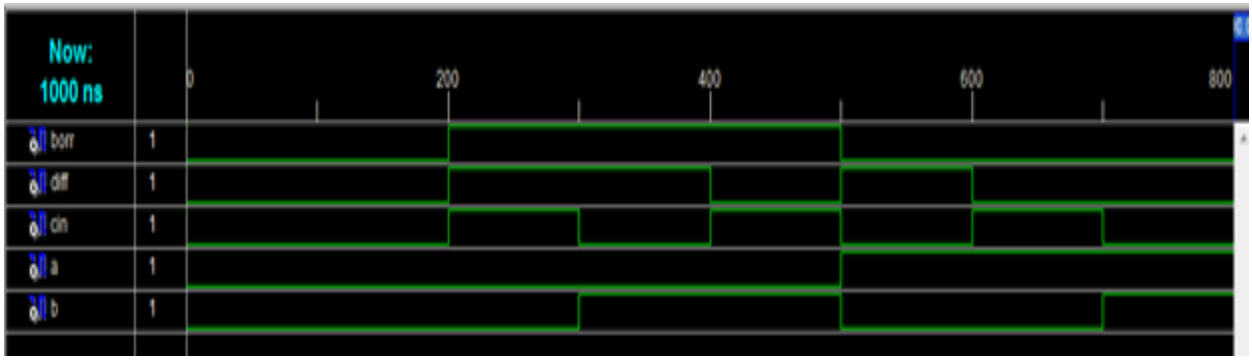
RTL Schematic



TEHNOLOGICAL SCHEMATIC



Output Wave:



2.C.7 Result: Thus the half subtractor and full subtractor was designed, simulated and implemented successfully.

Exp. No.: 3	DESIGN & FPGA IMPLEMENTATION OF FLIPFLOPS (D & T FLIPFLOPS)

3.1 AIM:

To implement Flipflops using Verilog HDL.

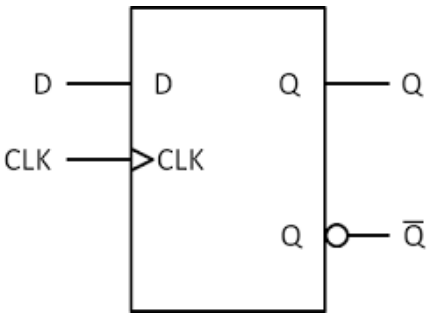
3.2 APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

3.3 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

3.4 D-FLIPFLOP:



3.5 PROGRAM:

```
Module DFF(Clock, Reset, d, q);
input Clock;
input Reset;
input d;
output q;
reg q;
always@(posedge Clock or negedge Reset)
if (~Reset)
q=1'b0;
else
q=d;
endmodule
```

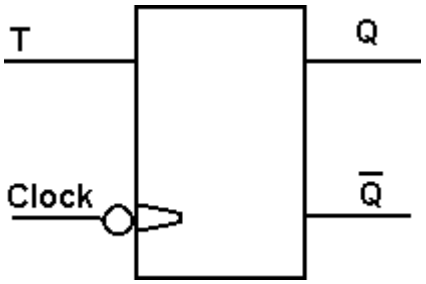
3.6 Truth Table:

D FlipFlop

Clock	Reset	Input (d)	Output q(~q)
0	0	0	0(1)
1	0	0	0(1)
0	0	1	0(1)
1	0	1	0(1)
0	0	0	0(1)
1	0	0	0(1)
0	1	1	0(1)
1	1	1	1(0)
0	1	0	1(0)
1	1	0	0(1)
0	1	1	0(1)
1	1	1	1(0)
0	0	0	0(1)
1	0	0	0(1)
0	0	0	0(1)

3.7 OUTPUT:

T-FLIPFLOP:



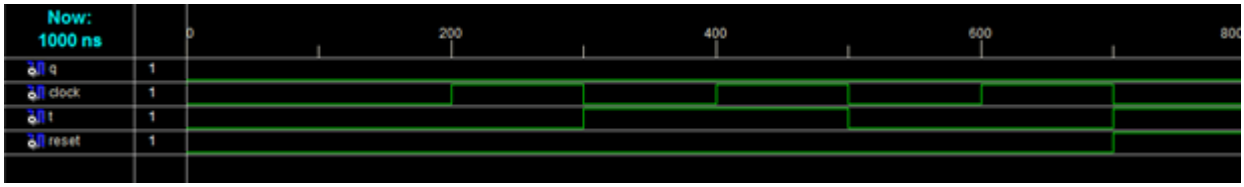
3.8 PROGRAM:

```
Module TFF(Clock, Reset, t, q);
input Clock;
input Reset;
input t;
output  q;
reg q;
always@(posedge Clock , negedge Reset)
if(~Reset) q=0;
else if (t) q=~q;
else q=q;
endmodule
```

3.9 TRUTH TABLE:

Clock	Reset	Input (t)	Output q(~q)
0	0	0	0(1)
1	0	0	0(1)
0	0	1	0(1)
1	0	1	0(1)
0	0	0	0(1)
1	0	0	0(1)
0	1	1	0(1)
1	1	1	1(0)
0	1	0	1(0)
1	1	0	1(0)
0	1	1	1(0)
1	1	1	0(1)
0	0	0	0(1)
1	0	0	0(1)
0	0	0	0(1)

OUTPUT WAVEFORM:



3.10 RESULT:

Thus the D & T flipflops are designed, simulated and implemented successfully.

Exp. No.: 4(A)	DESIGN & FPGA IMPLEMENTATION OF UP COUNTER AND DOWN COUNTER

4.A.1 AIM:

To implement Counters using Verilog HDL

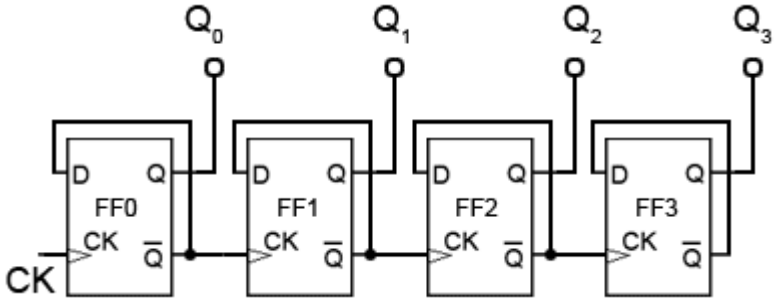
4.A.2 APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

4.A.3 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

4.A.4 UP COUNTER:

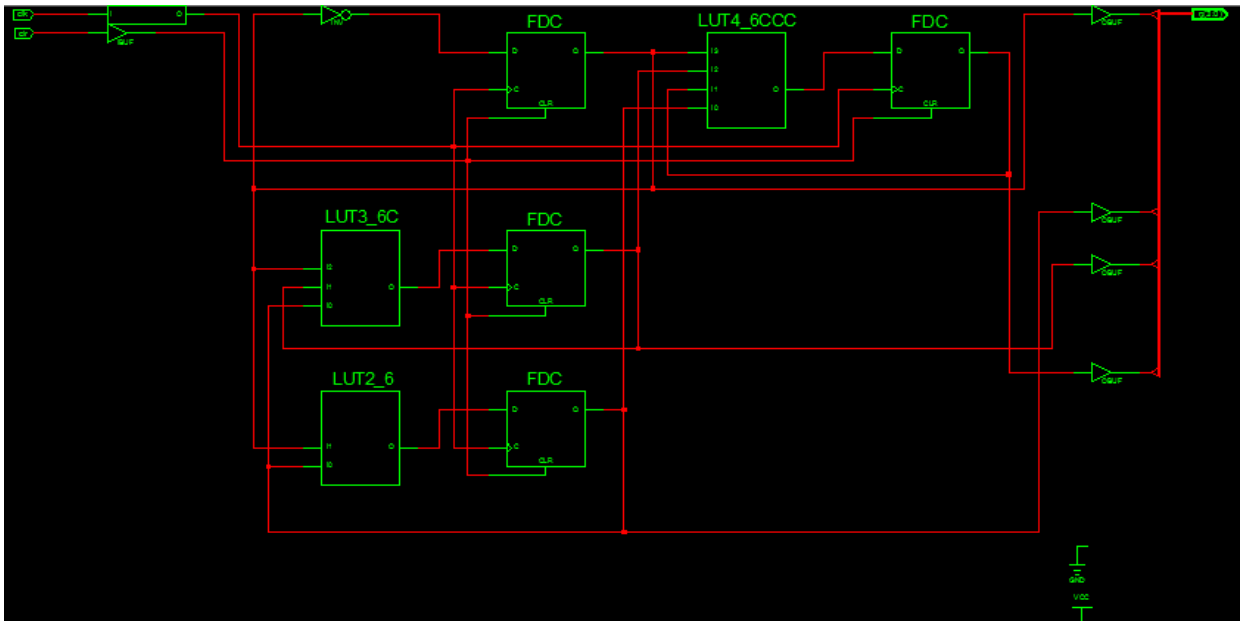


4.A.5 PROGRAM:

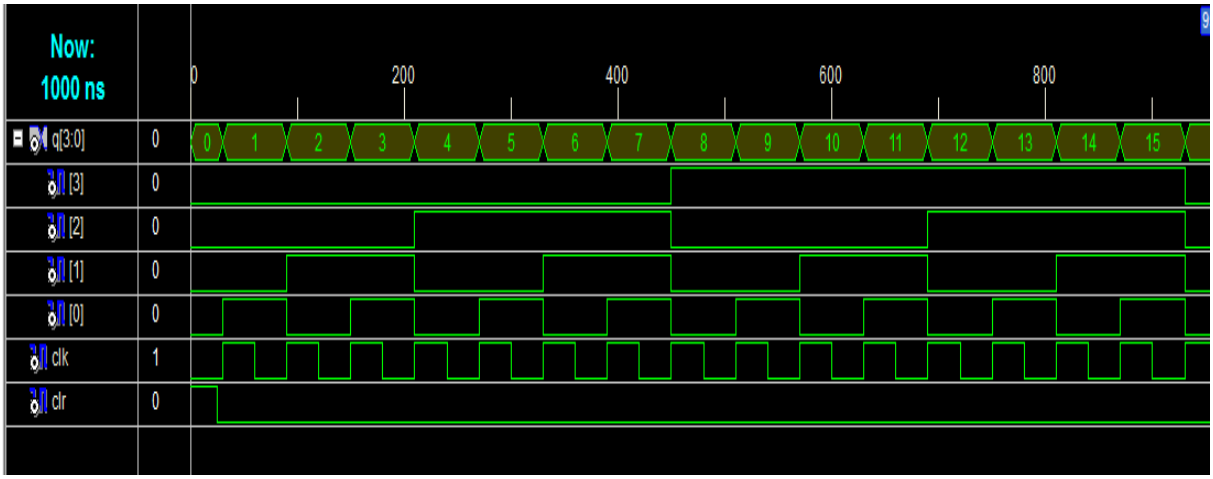
```
module upcounterr(clk,clr,q);
 input    clk, clr;
 output [3:0]q;
 reg    [3:0]tmp;
 always@(posedge clk or posedge clr)
 begin
 if (clr)
 tmp <= 4'b0000;
 else
 tmp <= tmp + 1'b1;
 end
 assign q = tmp;
 endmodule
```

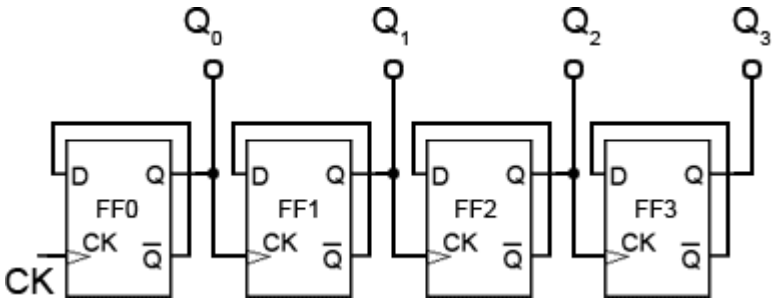
4.A.6 RTL SCHEMATIC:

4.A.7 TECHNOLOGICAL SCHEMATIC:



4.A.8 OUTPUT WAVEFORM:





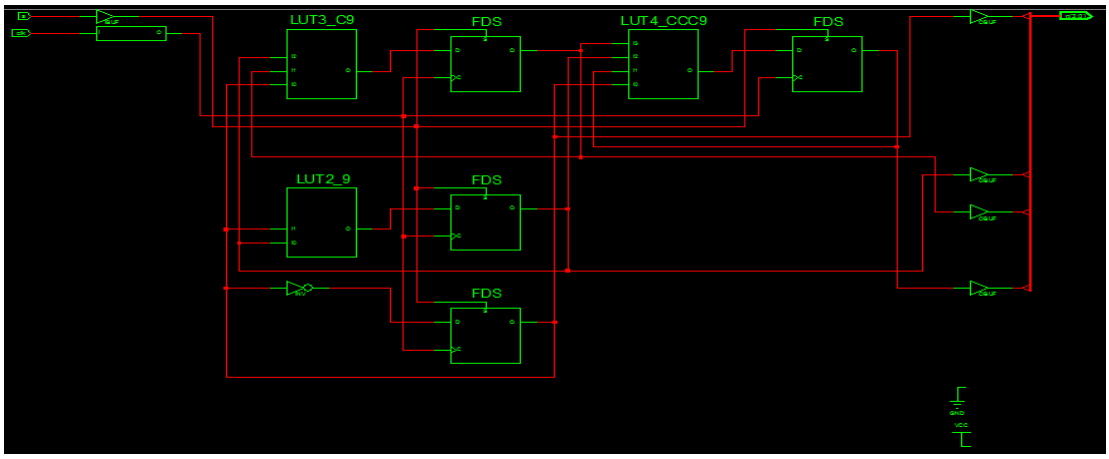
DOWN COUNTER:

4.A.9 PROGRAM:

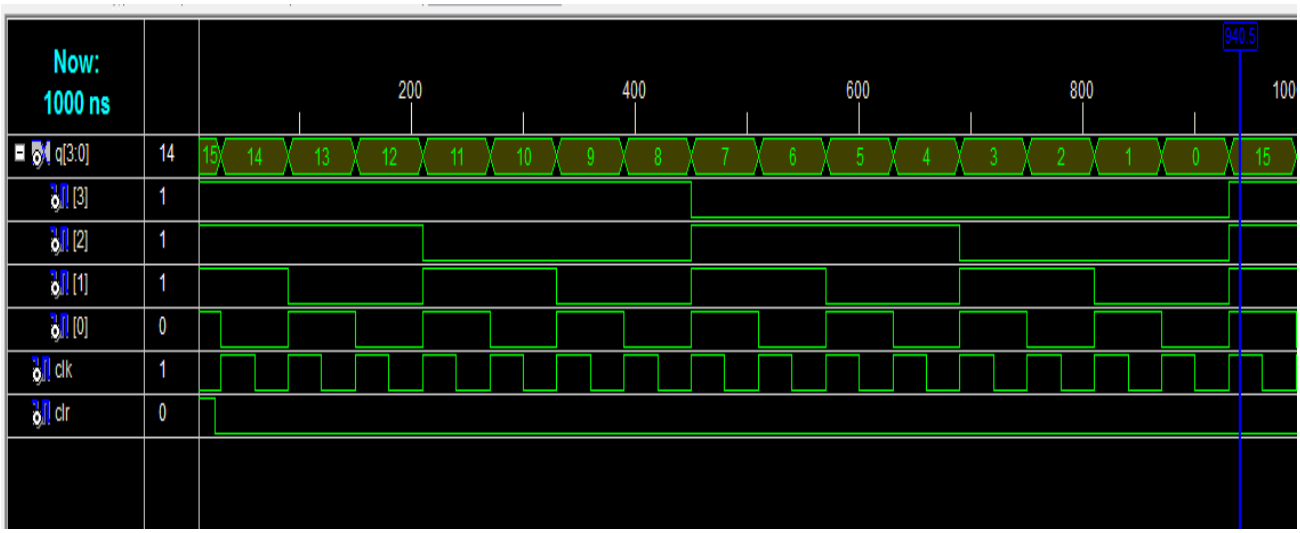
```
module downcounterr(clk,clr,q);
input    clk, clr;
output [3:0]q;
reg    [3:0]tmp;
always@(posedge clk or posedge clr)
begin
if (clr)
tmp <= 4'b1111;
else
tmp <= tmp - 1'b1;
end
assign q = tmp;
endmodule
```

4.A.10 RTL SCHEMATIC:

TECHNOLOGICAL SCHEMATIC:



4.A.1 OUTPUT WAVEFORM:



4.A.12 RESULT: Thus the up counter and down counter has been simulated and verified and implemented.

Exp. No.: 5(A)	DESIGN & FPGA IMPLEMENTATION OF FINITE STATE MACHINE (MOORE MACHINE)

5.A.1 AIM:

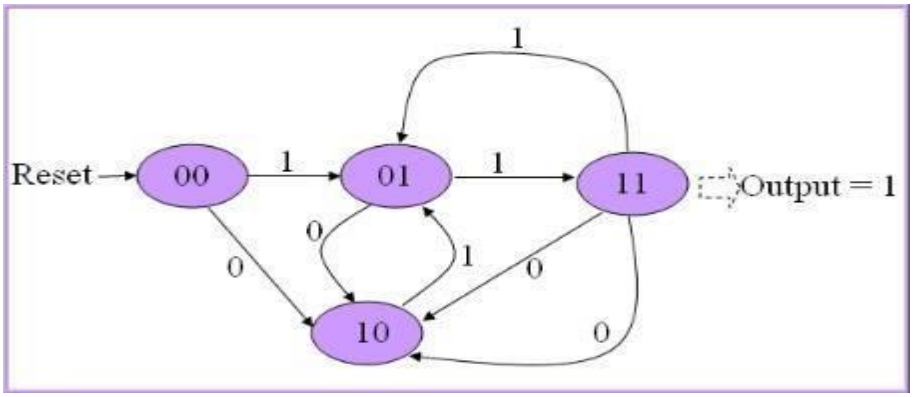
To implement finite state machine (moore machine) using Verilog HDL

5.A.2 SOFTWARE REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

5.A.3 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

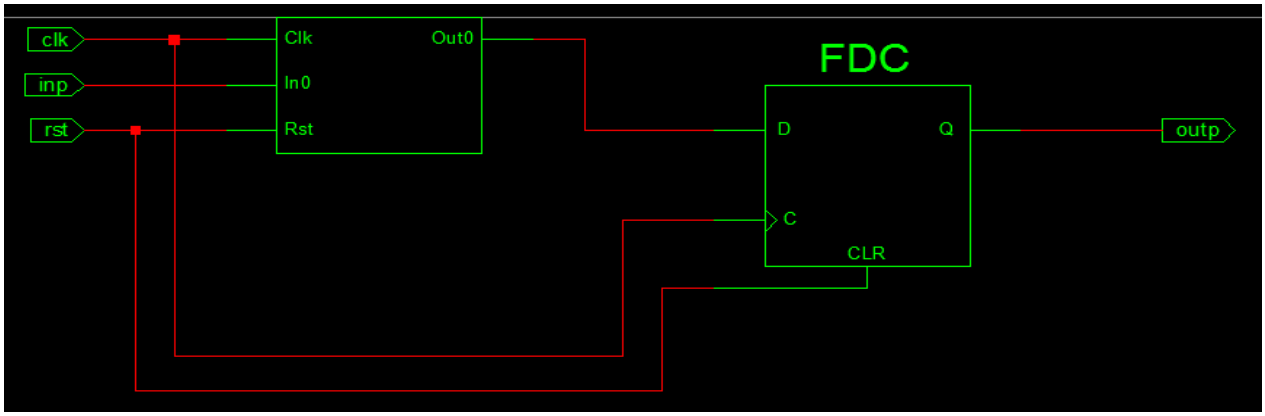


5.A.4 MOORE MACHINE:

5.A.5 PROGRAM:

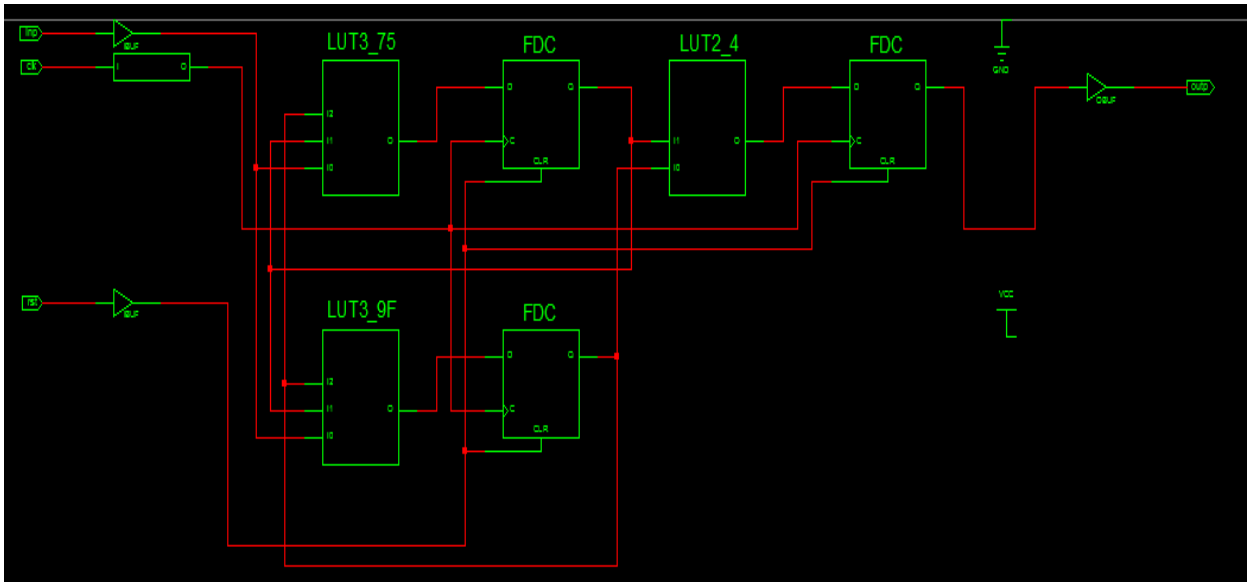
```
module moore( clk, rst, inp, outp);

  input clk, rst, inp;
  output outp;

  reg [1:0] state;
  reg outp;

  always @( posedge clk, posedge rst )
  begin
  if( rst )
     state <= 2'b00;
  else
  begin

     case( state )

     2'b00:
     begin
         if( inp ) state <= 2'b01;
         else state <= 2'b10;
     end

     2'b01:
     begin
         if( inp ) state <= 2'b11;
         else state <= 2'b10;
     end

     2'b10:
     begin
         if( inp ) state <= 2'b01;
         else state <= 2'b11;
     end

     2'b11:
     begin
         if( inp ) state <= 2'b01;
         else state <= 2'b10;
     end
     endcase
  end
end
always @(posedge clk, posedge rst)
begin
   if( rst )
     outp <= 0;
```

```
    else if( state == 2'b11 )
      outp <= 1;
    else outp <= 0;

end

endmodule
```

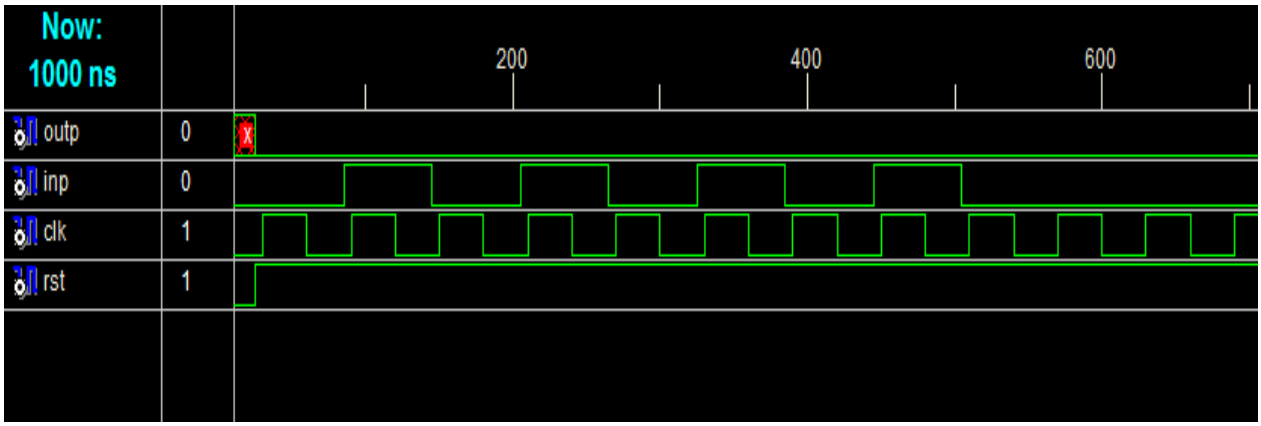


5.A.6 RTL SCHEMATIC:

TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



5.A.6 RESULT:

Thus the finite state machine (moore machine) has been simulated and verified and implemented.

Exp. No.: 5(B)	DESIGN & FPGA IMPLEMENTATION OF FINITE STATE MACHINE (MEALY MACHINE)

5.B.1 AIM:

To implement finite state machine (mealy machine) using Verilog HDL

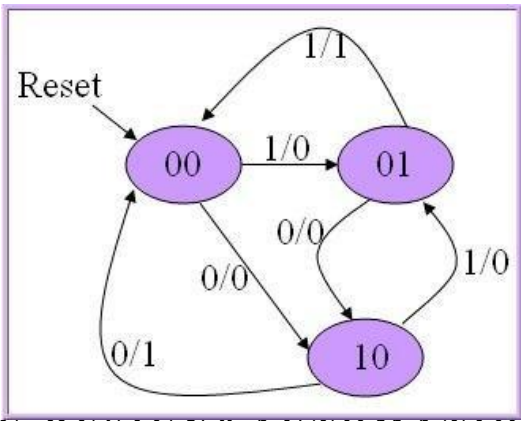
5.B.2 SOFTWARE REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

5.B.3 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

MEALY MACHINE: State diagram

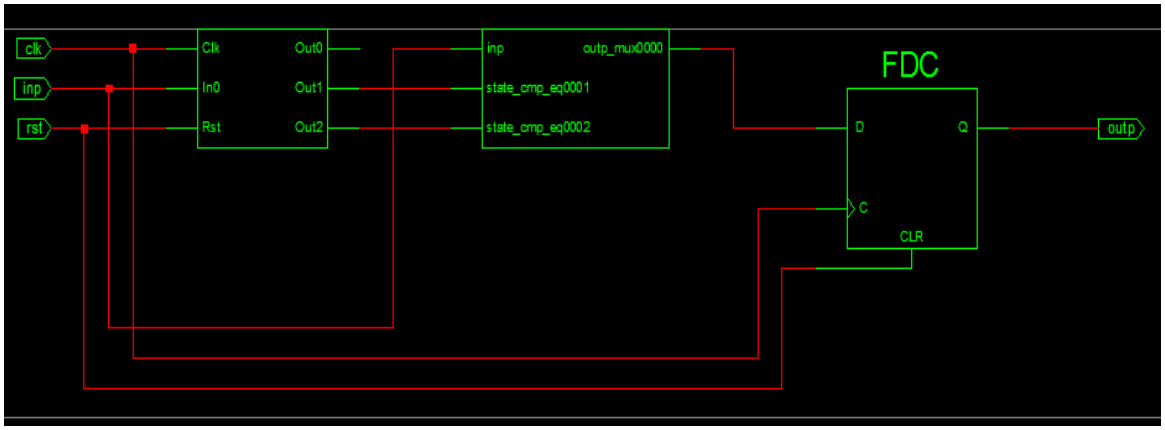


5.B.4 PROGRAM:

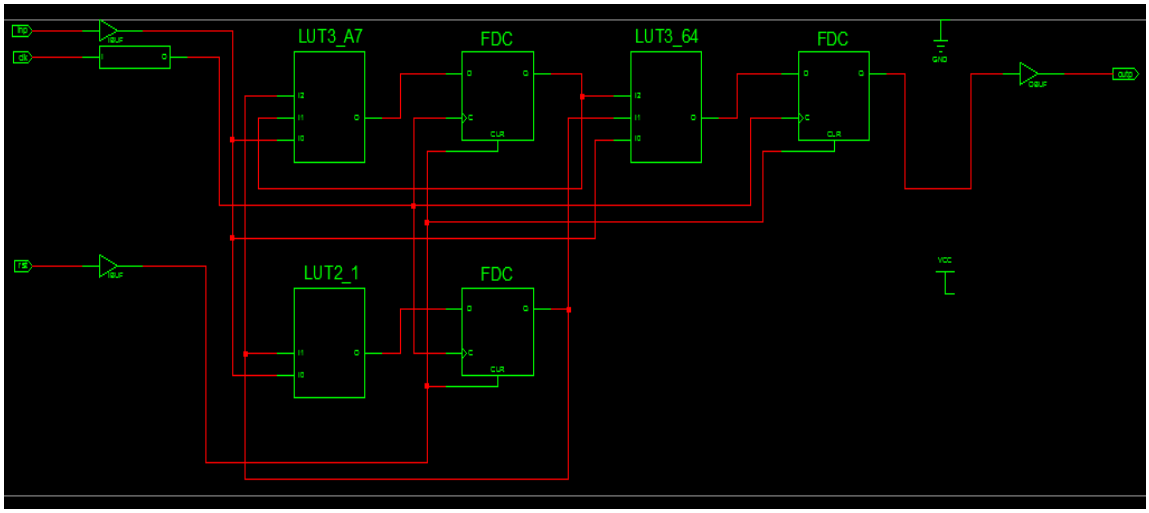
```
module mealy( clk, rst, inp, outp);

  input clk, rst, inp;
  output outp;

  reg [1:0] state;
  reg outp;

  always @( posedge clk, posedge rst ) begin
  if( rst ) begin
     state <= 2'b00;
     outp <= 0;
  end

  else begin

     case( state )
     2'b00: begin
         if( inp ) begin
           state <= 2'b01;
           outp <= 0;
         end
         else begin
            state <= 2'b10;
            outp <= 0;
         end
     end

     2'b01: begin
         if( inp ) begin
            state <= 2'b00;
            outp <= 1;
         end
         else begin
           state <= 2'b10;
           outp <= 0;
         end
     end

     2'b10: begin
         if( inp ) begin
            state <= 2'b01;
            outp <= 0;
```

```
            end
            else begin
              state <= 2'b00;
              outp <= 1;
            end

        end

        default: begin
            state <= 2'b00;
            outp <= 0;
        end
      endcase
      end
    end

    endmodule
```

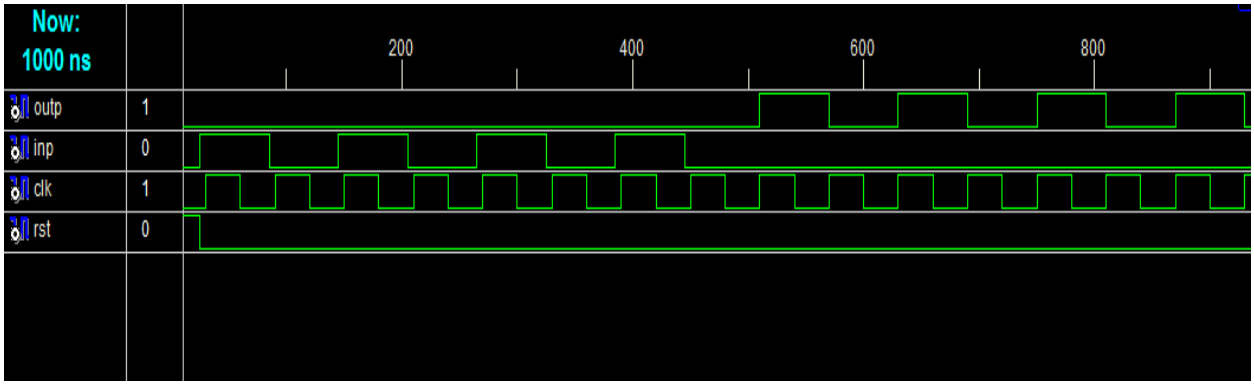
5.B.5 RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



5.B.6 OUTPUT WAVEFORM:



5.B.7 RESULT:

Thus the finite state machine (Mealy machine) has been simulated and verified and implemented.

Exp. No.: 6(A)	DESIGN & FPGA IMPLEMENTATION OF 4-BIT MULTIPLIER (SIMPLE & ARRAY MULTIPLIER)

6.A.1 AIM:

To implement Multiplexer & Demultiplexer using Verilog HDL.

6.A.2 APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

6.A.3 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

6.A. 4 BIT MULTIPLIER:

```
module unsignedmult (out, a, b);
Output [7:0] out;
 Input [3:0] a;
 Input [3:0] b;
 Assign out=a*b;
 endmodule
```

6.A.5 TRUTH TABLE:

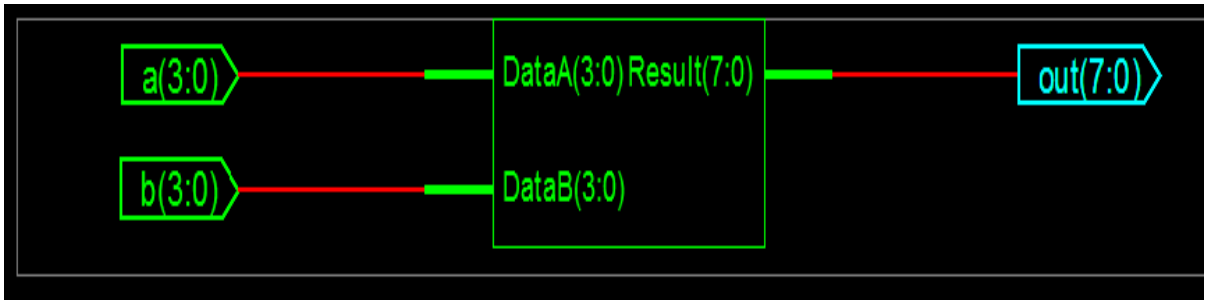
---------- A

B RES

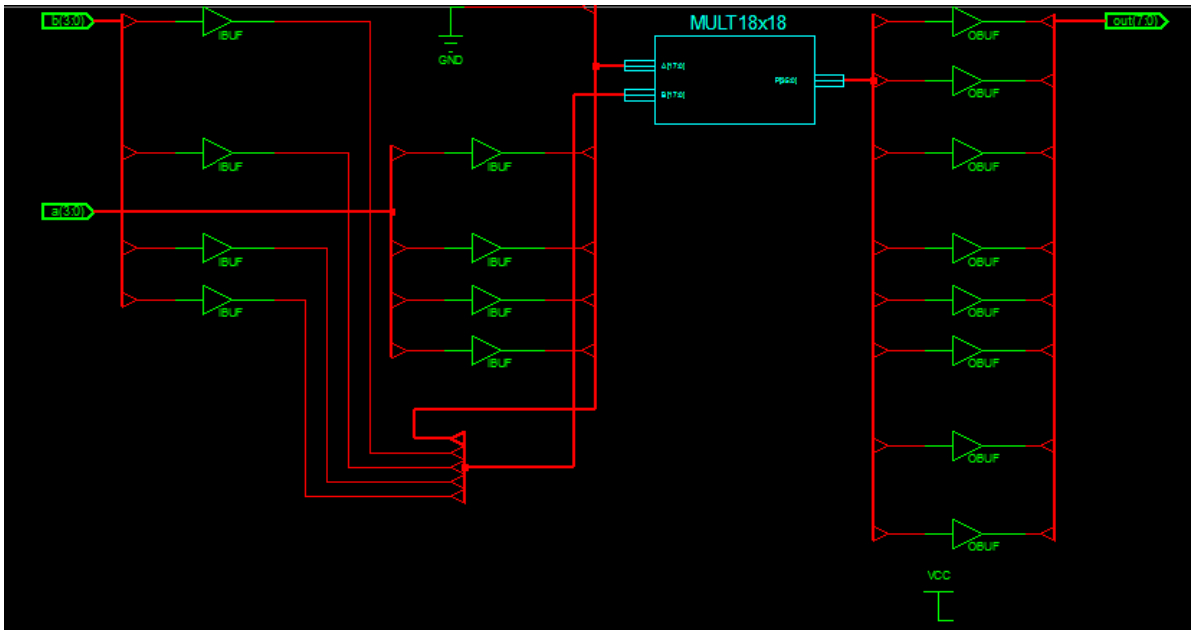
1000 1001 1000

6.A.6

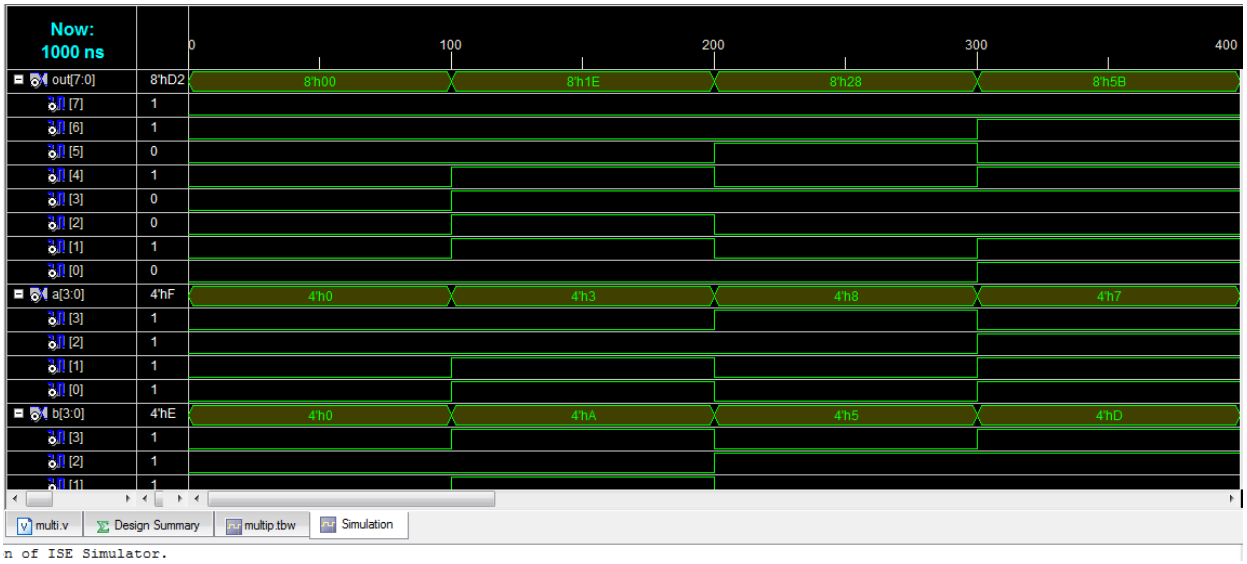
RTL SCHEMATIC:



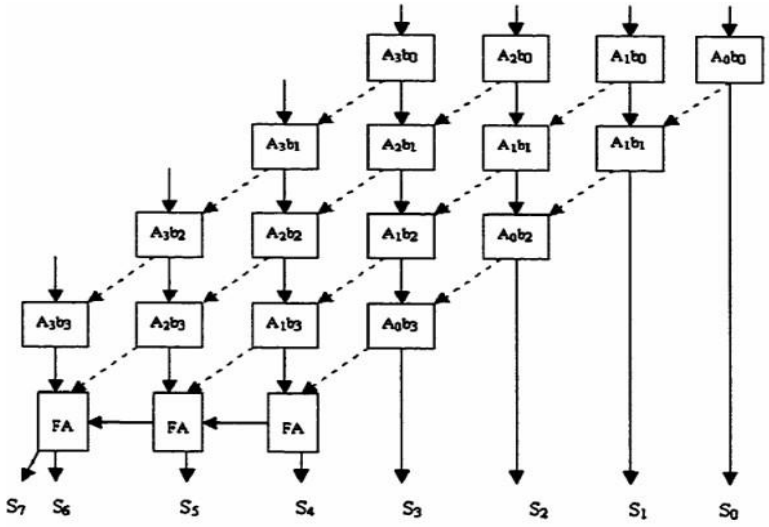
TECHNOLOGICAL SCHEMATIC:



6.A.6 OUTPUT WAVE:



6.A.7 ARRAY MULTIPLIER:

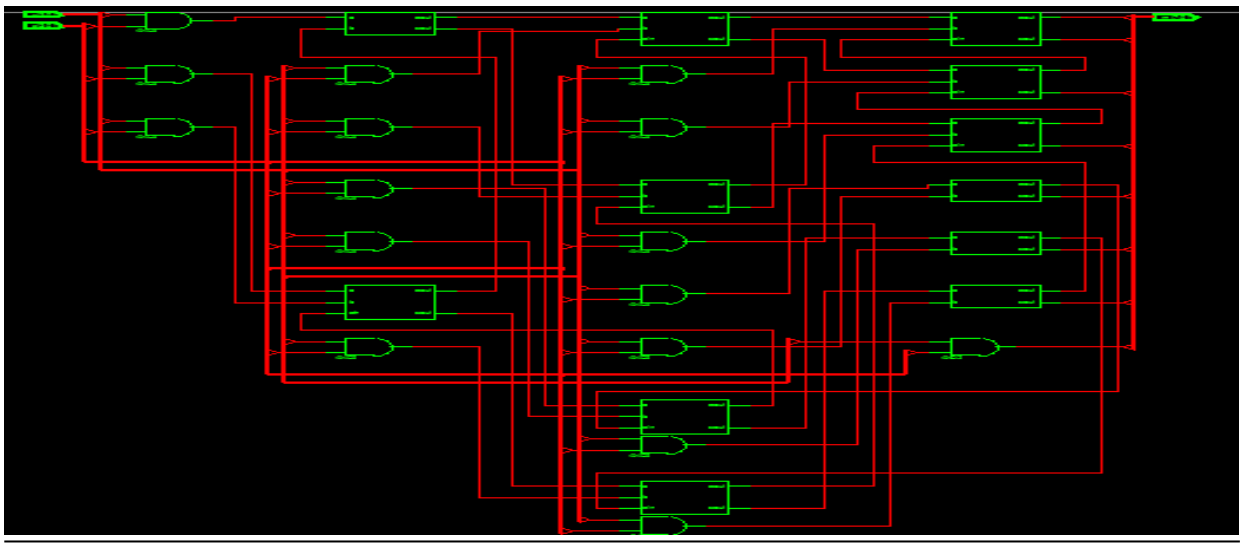


6.A.8 Program:

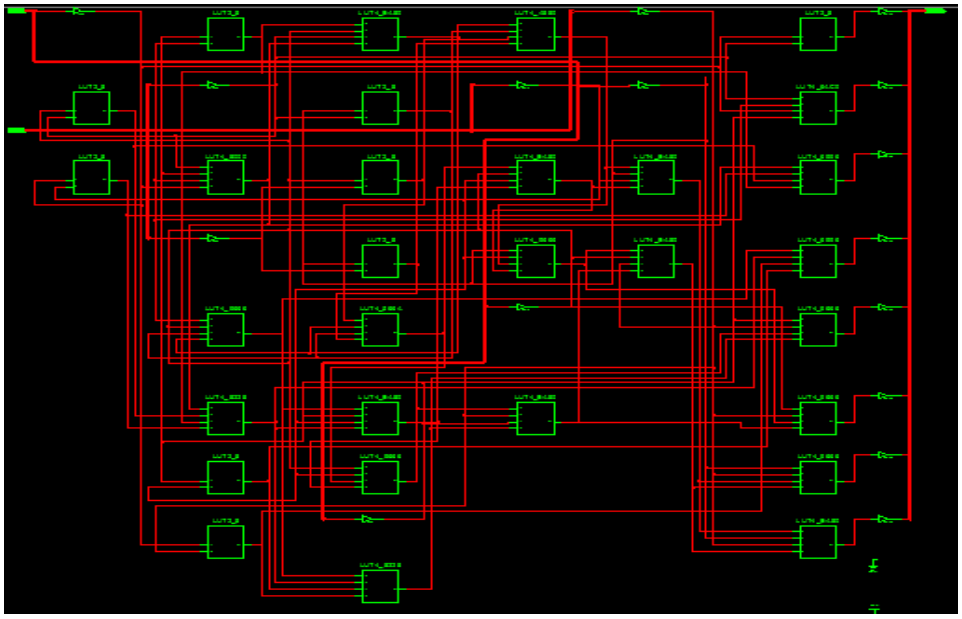
```
module HA (sout,cout,a,b);
input a,b;
output sout,cout;
assign sout=(a^b);
assign cout=(a&b);
endmodule
```

```
module FA (sout,cout,a,b,cin);
input a,b,cin;
output sout,cout;
assign sout=(a^b^cin);
assign cout=((a&b)|(b&cin)|(cin&a));
endmodule
module bitmul (m,x,y);
output [7:0]m;
input  [3:0]x;
input  [3:0]y;
assign m[0]=(x[0]&y[0]);
wire x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17;
HA HA1  (m[1],x1,(x[1]&y[0]),(x[0]&y[1]));
FA FA1  (x2,x3,(x[1]&y[1]),(x[0]&y[2]),x1);
FA FA2 (x4,x5,(x[1]&y[2]),(x[0]&y[3]),x3);
HA HA2 (x6,x7,(x[1]&y[3]),x5);
HA HA3 (m[2],x15,x2,(x[2]&y[0]));
FA FA5  (x14,x16,x4,(x[2]&y[1]),x15);
FA FA4  (x13,x17,x6,(x[2]&y[2]),x16);
FA FA3 (x9,x8,x7,(x[2]&y[3]),x17);
HA HA4 (m[3],x12,x14,(x[3]&y[0]));
FA FA8 (m[4],x11,x13,(x[3]&y[1]),x12);
FA FA7 (m[5],x10,x9,(x[3]&y[2]),x11);
FA FA6 (m[6],m[7],x8,(x[3]&y[3]),x10);
endmodule
```

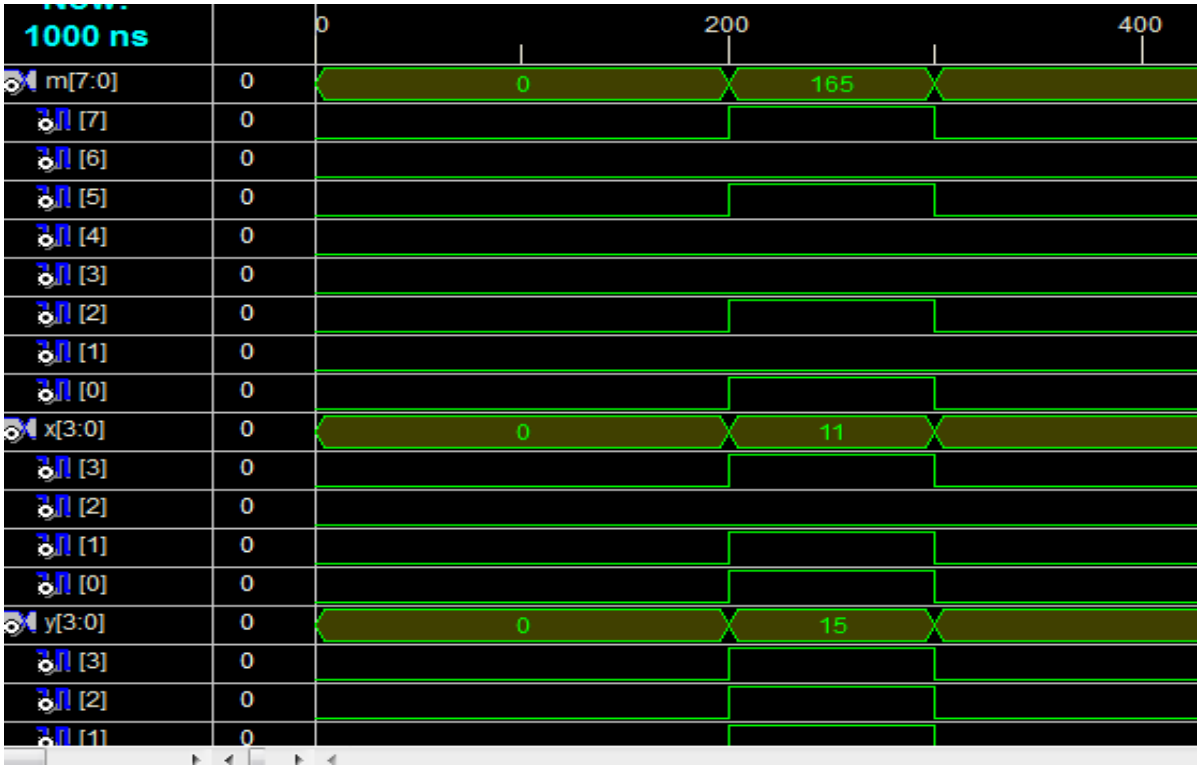
RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



OUTPUT WAVEFORM:



6.A.9 Result:

Thus 4-bit multiplier (simple and array multiplier) was implemented successfully.

PART-C

Transistor Level implementation of CMOS circuits using VLSI CAD tool

Exp. No.: 1	LAYOUT EXTRACTION AND SIMULATION OF C-MOS INVERTOR

1.1 AIM:

To draw the layout of an CMOS inverter

1.2 SOFTWARE USED:

- **Microwind**
- **DSCH**

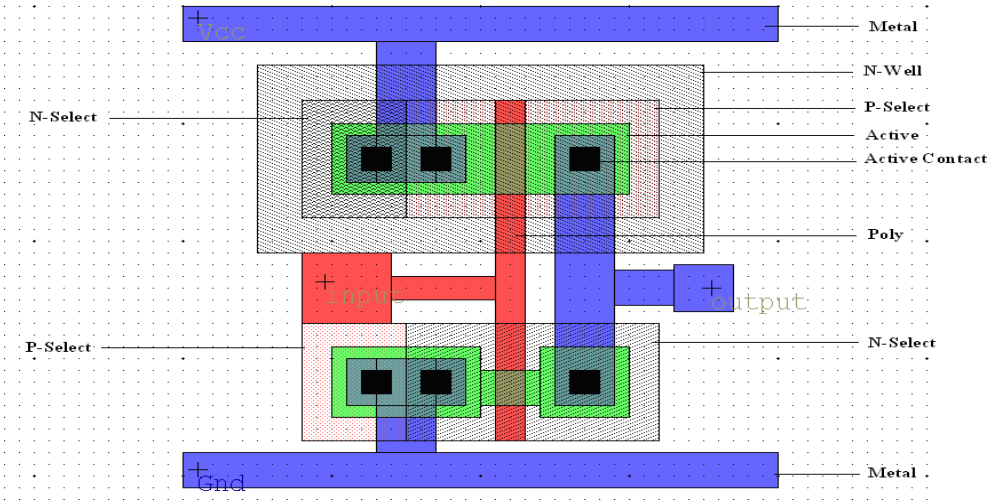
1.3DESCRIPTION:

CMOS INVERTER:

The NMOS transistor and the PMOS transistor form a typical complementary MOS (CMOS) device. When a low voltage (0 V) is applied at the input, the top transistor (P-type) is conducting (switch closed) while the bottom transistor behaves like an open circuit.

Therefore, the supply voltage (5 V) appears at the output. Conversely, when a high voltage (5 V) is applied at the input, the bottom transistor (N-type) is conducting (switch closed) while the top transistor behaves like an open circuit. Hence, the output voltage is low (0 V).

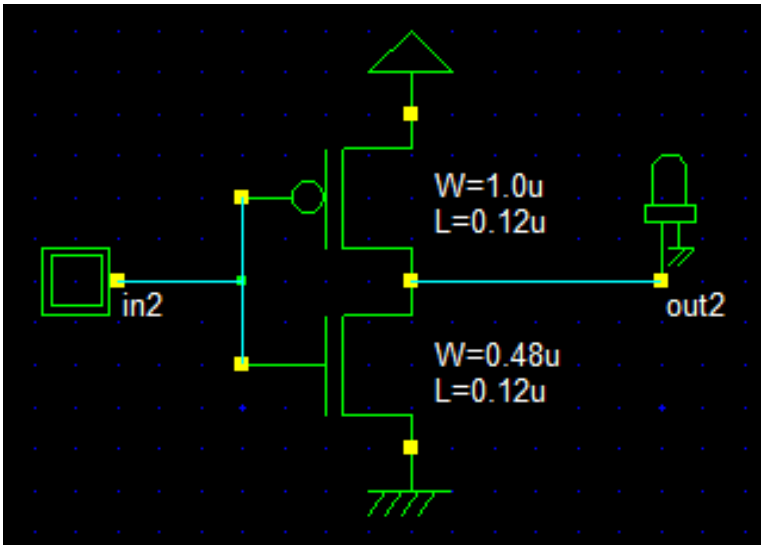
1.4 LAYOUT DIAGRAM



1.6 ALGORITHM:

- Open the DSCH2
- Drag the components like pmos,nmos,voltage source, ground, and LED from the symbol library.
- Connect the circuit as in the circuit diagram.
- Save the circuit & run the simulation
- Make verilog file go to Microwind and compile the verilog file saved in DSCH2
- Compile it and obtain the layourt diagram & draw the waveform

1.7 CIRCUIT DIAGRAM:



MOS LAYOUT

We use MICROWIND2 to draw the MOS layout and simulate its behavior. Go to the directory in which the software has been copied (By default MICROWIND2). Double-click on the MicroWind2 icon. The MICROWIND2 display window includes four main windows: the main menu, the layout display window,the icon menu and the layer palette. The layout window features a grid, scaled in lambda (□) units. The lambda unit is fixed to half of the minimum available lithography of the technology. The default technology is a CMOS 6-metal layers 0.25µm technology, consequently lambda is 0.125 µm.

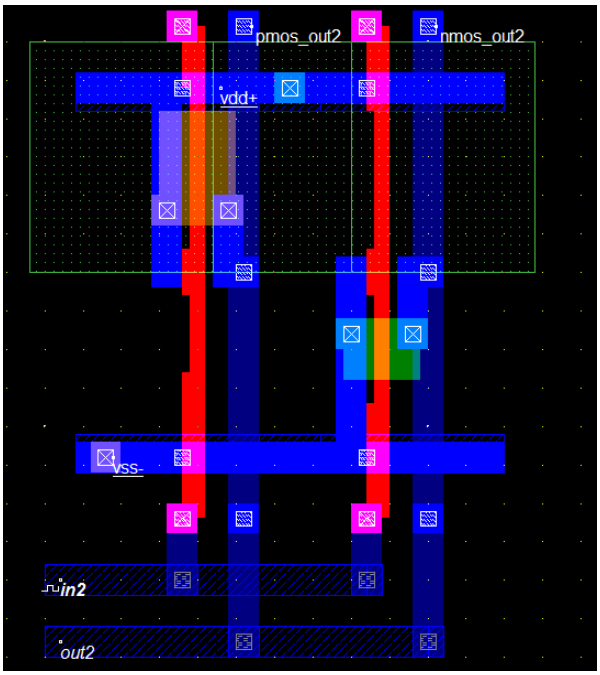
1.8 Verilog code:

module cmosInv(in2,out2) input in2;

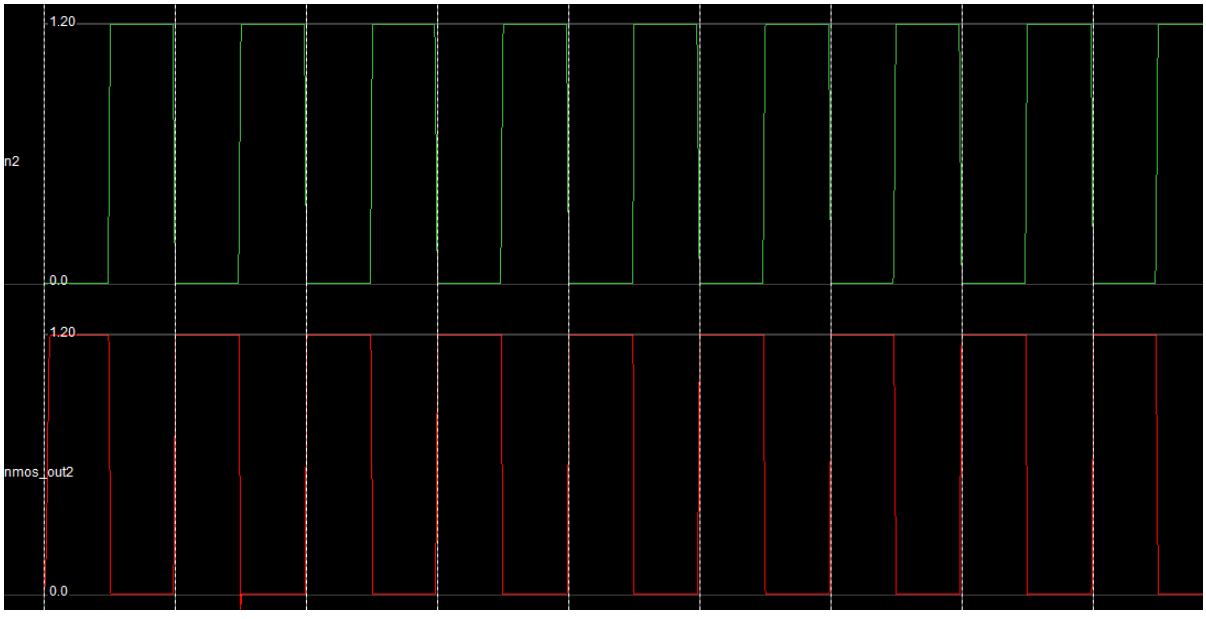
output out2;

```
 pmos #(17) pmos(out2,vdd,in2); // 1.0u 0.12u
nmos #(114) nmos(out2,vss,in2); // 0.48u 0.12u
endmodule
```

1.9 LAYOUT FOR C-MOS INVERTOR:



1.10 ANALOG SIMULATION:



Click on **Simulate à Start Simulation**. The timing diagrams will appears as follows

1.11 RESULT:

Thus the Layout design of a CMOS inverter has been drawn, verified and timing analysis perform.

Exp. No.: 2	LAYOUT EXTRACTION AND SIMULATION OF C-MOS NAND AND NOR GATE

2.1 AIM:

To design and simulate the cmos NAND and NOIR gate circuit.

2.2 SOFTWARE USED

- **Microwind**
- **DSCH**

2.3 THEORY:

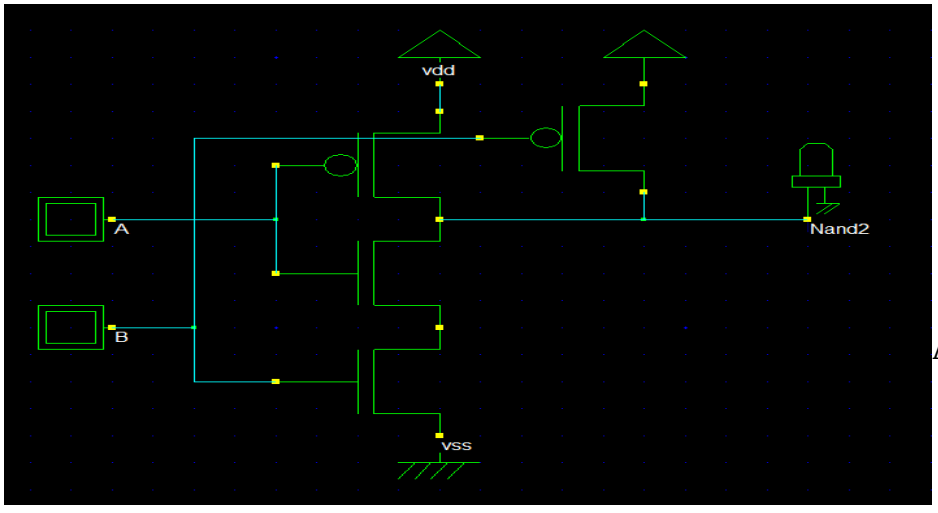
NAND and NOR gates are known as universal gates as any function can be implemented with them NAND functionality can be implemented by parallel combination of PMOS and series combination of NMOS transistor. When any one of the inputs is zero, then the output will be one and when both the inputs are one the output will be low. NOR functionality can be implemented by parallel combination of NMOS and series combination of PMOS transistor. When any one of the inputs is one, then the output will be one and when both the inputs are zero the output will be low.

2.4 ALGORITHM:

- Open the DSCH2
- Drag the components like pmos,nmos,voltage source, ground, and LED from the symbol library.
- Connect the circuit as in the circuit diagram.
- Save the circuit & run the simulation
- Make verilog file go to Microwind and compile the verilog file saved in DSCH2
- Compile it and obtain the layourt diagram & draw the waveform

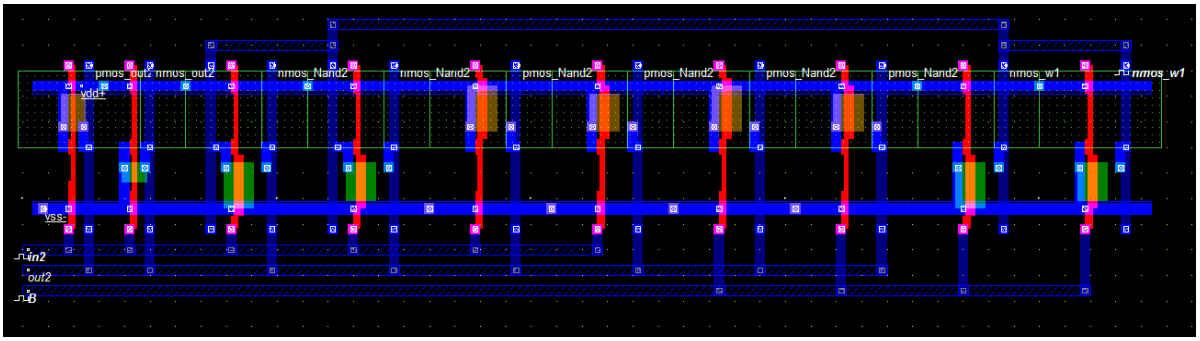
2.5 CIRCUIT DIAGRAM:

NAND gate:



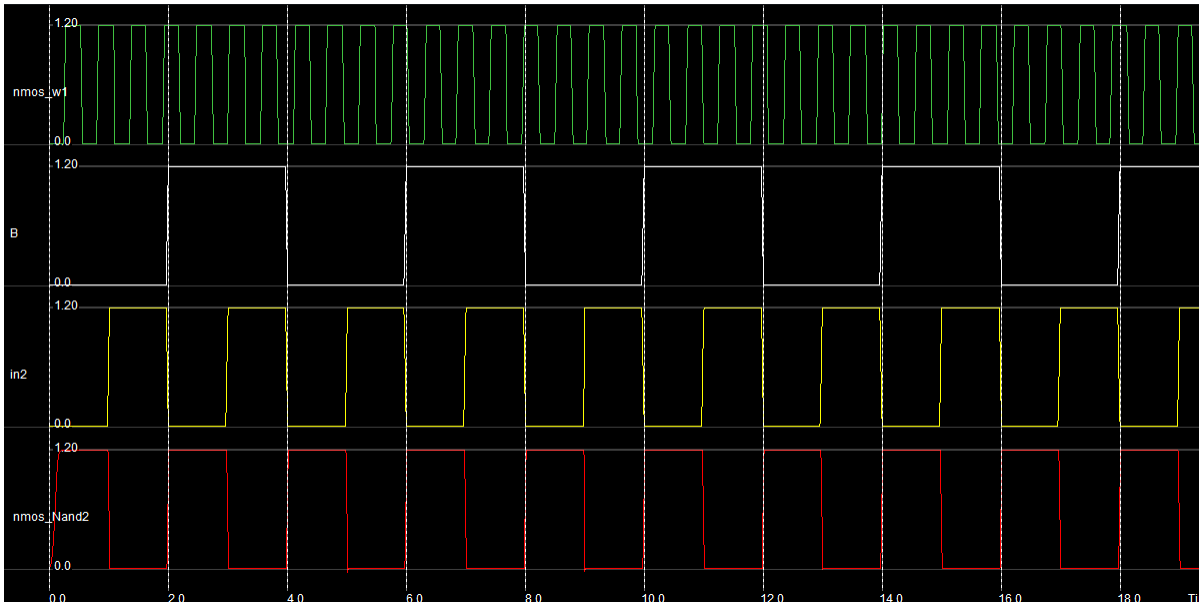
2.6 Verilog code:

```
module cmosNand2( A,B,Nand2);
 input A,B;
 output Nand2;
 nmos #(121) nmos(Nand2,w1,A); // 2.0u 0.25u
pmos #(121) pmos(Nand2,vdd,A); // 2.0u 0.25u
pmos #(121) pmos(Nand2,vdd,B);  // 2.0u 0.25u
nmos #(107) nmos(w1,vss,B); // 2.0u 0.25u
endmodule
```

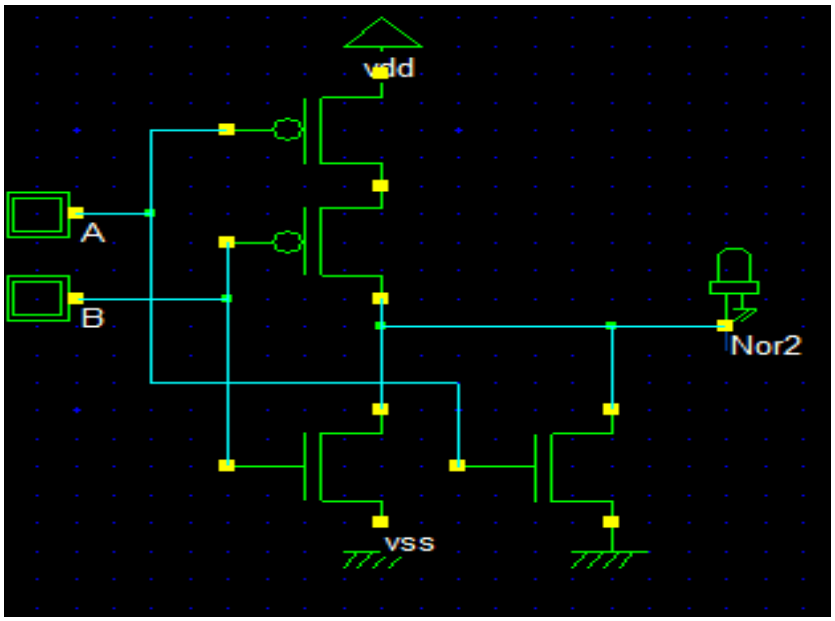


2.7 Layout:

Waveform:



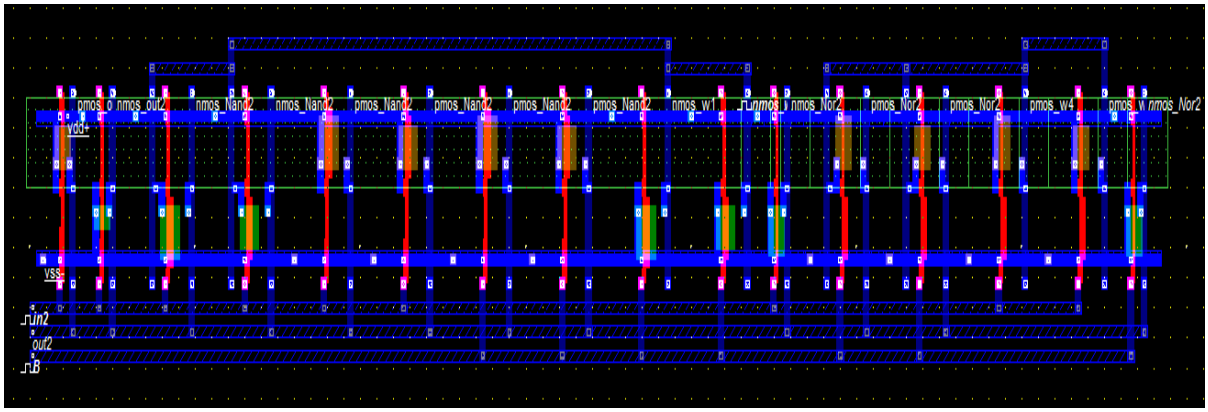
NOR gate:



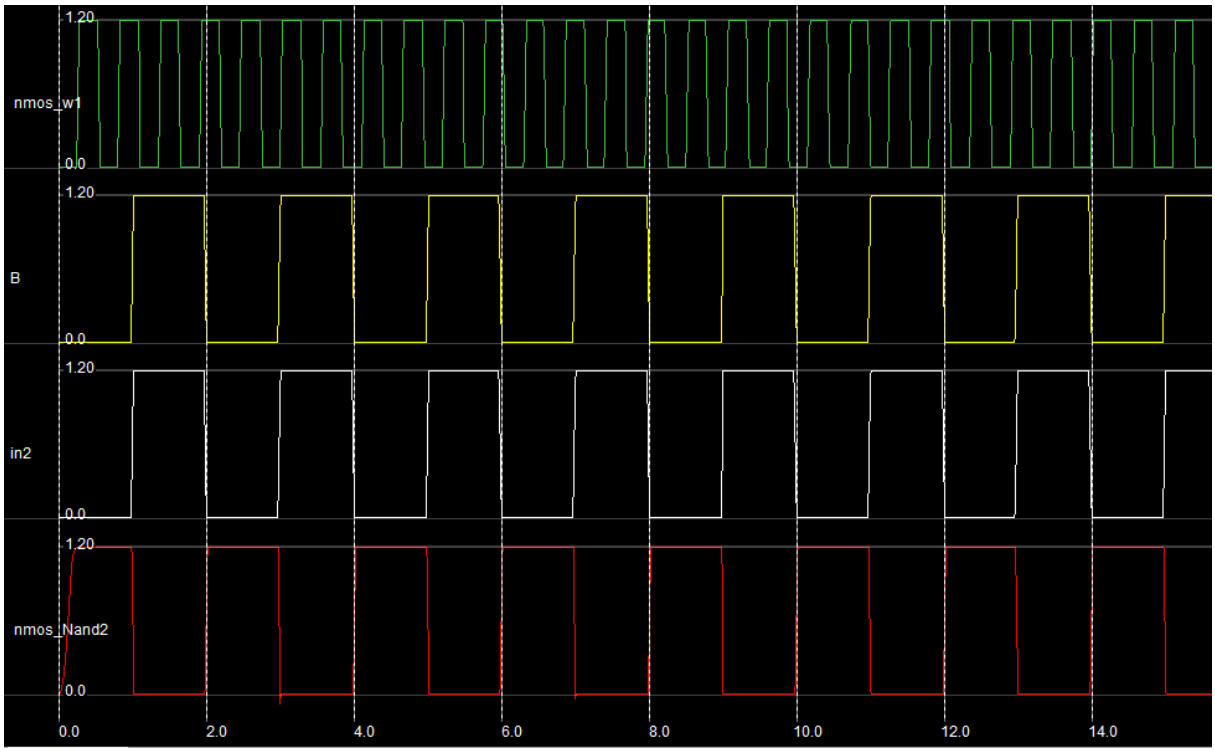
2.8 Verilog code:

```
module nor2Cmos( B,A,Nor2);
 input B,A;

 output Nor2;

 nmos #(121) nmos(Nor2,vss,A); // 1.0u 0.12u
pmos #(121) pmos(Nor2,w4,B); // 2.0u 0.12u
pmos #(107) pmos(w4,vdd,A); // 2.0u 0.12u
nmos #(121) nmos(Nor2,vss,B); // 1.0u 0.12u
endmodule
```

2. 9 Layout:



Waveform:



2.10 RESULT:

Thus the Layout design of a CMOS NAND and NOR gate has been drawn, verified and timing analysis performed

Exp. No.: 3	LAYOUT EXTRACTION AND SIMULATION OF C-MOS DIFFERENTIAL AMPLIFIER

3.1 AIM:

To design and simulate the emitter follower and differential amplifier circuit.

3.2 SOFTWARE USED

- **Microwind**
- **DSCH**

3.3 THEORY:

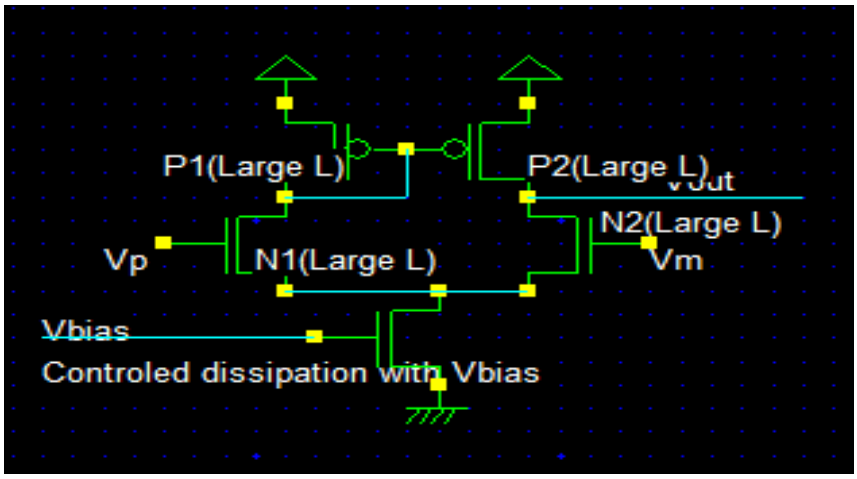
Differential amplifier:Differential Amplifier amplifies the current with very little voltage gain. It consists of two FETs connected so that the FET sources are connected together. The common source is connected to a large voltage source through a large resistor Re, forming the "long tail" of the name, the long tail providing an approximate constant current source. The higher the resistance of the current source Re, the lower Ac is, and the better the CMRR. In more sophisticated designs, a true (active) constant current source may be substituted for the long tail. The output from a differential amplifier is itself often differential.

3.4 ALGORITHM:

1. Open the DSCH2
2. Drag the components like pmos,nmos,voltage source, ground, and LED from the symbol library.
3. Connect the circuit as in the circuit diagram.
4. Save the circuit & run the simulation
5. Make verilog file go to Microwind and compile the verilog file saved in DSCH2
6. Compile it and obtain the layourt diagram & draw the waveform

3.5 CIRCUIT DIAGRAM:

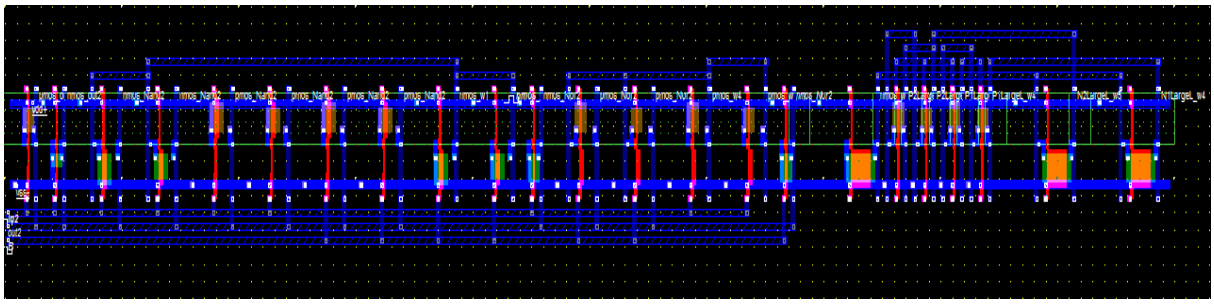
DIFFERENTIAL 3.6 AMPLIFIER:



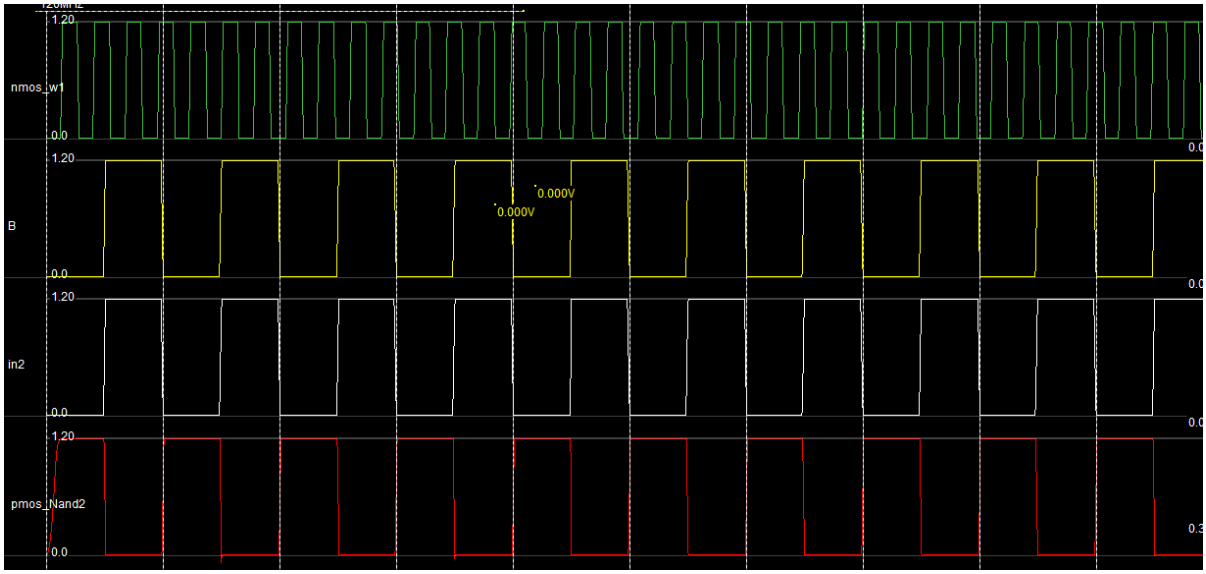
Verilog code:

```
module diff amp( );

 nmos #(17) nmos(w2,vss,w1); // 1.0u 1u

 pmos #(10) P2LargeL(w5,w3,w4); // 2.0u 0.12u
pmos #(24) P1LargeL(w4,w6,w4); // 2.0u 0.12u
nmos #(10) N2LargeL(w5,w2,w7); // 1.0u 1u
nmos #(24) N1LargeL(w4,w2,w8); // 1.0u 1u
endmodule
```

3.7 LAYOUT:



3.8 WAVEFORM:



3.9 RESULT: Thus the Layout design of a differential amplifier has been drawn, verified and timing analysis performed.

Beyond Syllabus Experiments

1. Demonstration of EEPROM interface in ARM7

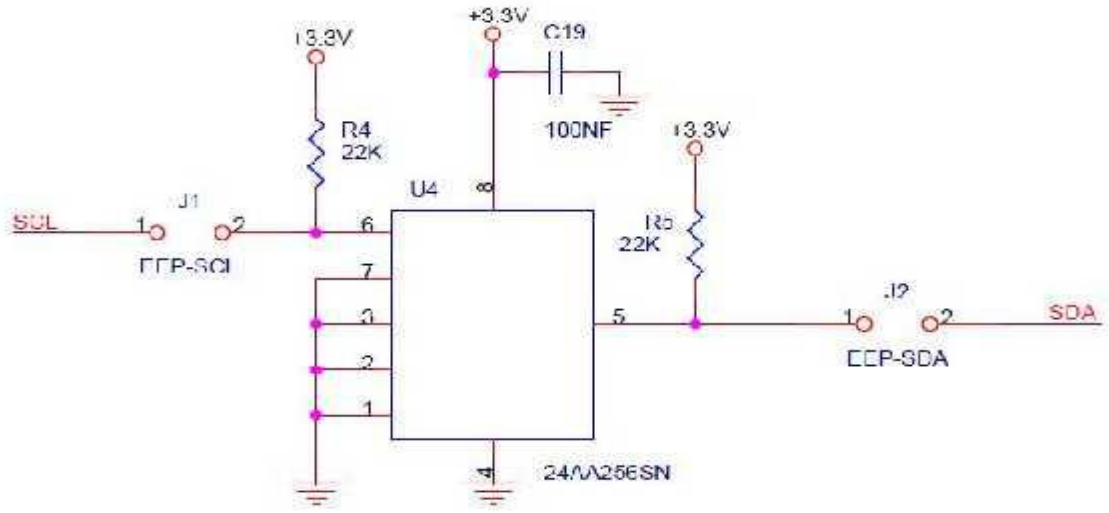
1.1 AIM: To Write a Program to demonstrate EEPROM interface in ARM kit and using I2C Communication.

1.2 APPARATUS:

1. Physitech electronics LPC2148 Education Board
2. PC with Windows 7/8/10 OS
3. KEIL µVISION 4 (MDK-ARM) Tool
4. Serial USB cable
5. RS-232 serial cable
6. Power Adapter (for Physitech Kit)

1.3 Description:

EEPROM: The LPC2148 SDK also contains an E 2PROM accessible via the I2C interface. The LPC2148 microcontroller has two on-chip I2C communication channels. Channel #0 is used for communicating with the E2PROM. The other channel can optionally be used on an expansion board. More peripheral units are easily connected to the two-wire I2C bus, just as long as the addresses do not collide. The I2C interface to the EEPROM can be disconnected from the LPC2148 by removing jumpers on J1 and J2. Figure below shows the EEPROM section.



I2C is a protocol for Synchronous serial Communication. This Communication will set up by setting **SCK** and **SDA**. ARM7 support I2C protocol and to enable I2C Protocol, use **J1** and

J2 Jumpers. For I2C Based EEPROM, in the code use **putStrE** and **getStrE** functions for write and read the data using I2C protocol. These both functions has two arguments. First one is what is the message have to store in EEPROM and second one is from where to read the message. LCD Interface will help to show the how communication will work.

To Set The Clock

setClock();

To Initialize LCD, I2C

lcdInit();

i2cInit();

To Write The Data into Eeprom ,use

putCharE();

putStrE();

To Read The Data into Eeprom, use

getCharE();

getStrE();

1.5 ALGORITHM:

STEP1: Set the Clock

STEP2: Initialize LCD

STEP3: Initialize I2C

STEP4: END

PutCharE:- To Write The Data into Eeprom ,use

STEP1: Assign Eeprom Address into AddrH and AddrL

STEP2: Enable I2C Interface **STEP3:** If Start Address Condition is transmit

STEP4: Clear STA

STEP5: Clear I2C Interface

STEP6: Eeprom Write Address

STEP7:- Eeprom Write Address into Data Register

STEP8:- Clear I2C Interface

STEP9:- If write Address Condition transmit

STEP10:- Higher Byte Address into Data Register

STEP11:- Clear I2C Interface

STEP12: If Higher Data Byte In I2DAT transmit

STEP13: Lower Byte Address into Data Register

STEP14: Clear I2C Interface

STEP15: If lower Data Byte In I2DAT transmit

STEP16: Data Argument Into Data Register

STEP17: Clear I2C Interface

STEP18: If Data Byte In I2DAT transmit

STEP19: Clear I2C Interface

STEP20: Stop I2C Protocal

STEP21: END

GetCharE:- To Read The Data into Eeprom ,use

STEP1: Assign Eeprom Address into AddrH and AddrL

STEP2: Enable I2C Interface

STEP3: If Start Address Condition is transmit

STEP4: Clear STA

STEP5: Clear I2C Interface

STEP6: Eeprom Write Address

STEP7:- Eeprom Write Address into Data Register

STEP8:- Clear I2C Interface

STEP9:- If write Address Condition transmit

STEP10:- Higher Byte Address into Data Register

STEP11:- Clear I2C Interface

STEP12: If Higher Data Byte In I2DAT transmit

STEP13: Lower Byte Address into Data Register

STEP14: Clear I2C Interface

STEP15: If lower Data Byte In I2DAT transmit

STEP16: Enable the I2C Block to Acknowledge, its Own Slave Address

STEP17: Clear I2C Interface

STEP18: If Repeated Start Condition in I2C transmit

STEP19: Eeprom Write Address

STEP20: Eeprom Write Address into Data Register

STEP21: Clear I2C Interface

STEP22: Clear The Start Condition

STEP23: If Slave Read Condition transmit

STEP24: Clear I2C Interface

STEP25: If Data Byte Receiving is transmit

STEP26: Receiving Data into RxData

STEP27: Clear I2C Interface

STEP28: I2C stop

STEP29: Return the RxData

STEP30: END

PutstrE: To Write The Data into Eeprom ,use

STEP1: Assign Eeprom Address into AddrH and AddrL

STEP2: Enable I2C Interface

STEP3: If Start Address Condition is transmit

STEP4: Clear STA

STEP5: Clear I2C Interface

STEP6: Eeprom Write Address

STEP7:- Eeprom Write Address into Data Register

STEP8:- Clear I2C Interface

STEP9:- If write Address Condition transmit

STEP10:- Higher Byte Address into Data Register

STEP11:- Clear I2C Interface

STEP12: If Higher Data Byte In I2DAT transmit

STEP13: Lower Byte Address into Data Register

STEP14: Clear I2C Interface

STEP15: If lower Data Byte In I2DAT transmit

STEP16: Character By Character From String

STEP17: Clear I2C Interface

STEP18: If Higher Data Byte in I2DAT transmit

STEP19: Data is Null

STEP20: Clear I2C Interface

STEP21: If Higher Data Byte in I2DAT transmit

STEP22: Clear I2C Interface

STEP23:I2C Stop

STEP24: END

GetStrE:- To Read The Data into Eeprom ,use

STEP1: Read the Data from Eeprom and Storing into Read Array

STEP2: If Read the Data Untill Null Comes

STEP3: Return the Data

STEP4: END

1.7 Program:

```
#include <LPC214X.h>
/*User Defined header files for eeprom,lcd and common file.*/
#include "eeprom.h"
#include "lcd.h"
#include "common.h"
int main(void)
{
 unsigned char *ptr,x;                    //Holds the Red Data from EEPROM
 setClock();                                    //Clock for 60Mhz.
```

```
 lcdInit();                                  // LCD Intialization.
 i2cInit();                                  // I2C Intialization
 putStrL("EEPROM TEST",0x01);                // writing the String into LCD first Row.
 putStrE("OK",0x0001);       // writing the "OK" string to EEPROM at address of 0x0001
 putCharE('-',0x0000);       // writing the '-' character to EEPROM at address of 0x0000
 ptr=getStrE(0x0001);        // Reading the "OK" string from EEPROM  addr of 0x0001 and storing in *Ptr
 x=getCharE(0x0000);         // Reading the '-' Character from EEPROM  addr of 0x0001 and storing in 'x'.
 putStrL(ptr,0xC0);          //displaying the red data into LCD second line.
 putCharL(x);                //displaying the red character into LCD second line
 while(1);                                                   //End of loop
}
```

1.7 Result: we have been demonstrated the EEPROM interface kit via I2C.

2. DESIGN &FPGA IMPLEMENTATIONOF 8-BIT ADDERS

(SIMPLE & RIPPLE CARRY ADDER)

2.1 AIM: To design and to implement 8-bit adders (simple adder and ripple carry adder using Verilog HDL.

2.2 APPARATUS REQUIRED:

- PC with Windows XP.
- XILINX 9.2i
- FPGA-SPARTAN-3 KIT
- PARALLEL TO JTAG CABLE

2.3 ALGORITHM:

- New project and type the project name and check the top level source type as HDL
- Enter the device properties and click Next
- Click New Source And Select the Verilog Module and then give the file name
- Give the Input and Output port names and click finish.
- Type the Verilog program and save it
- Double click the synthesize XST and check syntax
- Simulate the waveform by behavioral simulation
- For implementation Select User constraints and give input and output port pin number
- Click Implement design for Translate, map and place & route
- Generate .bit file using programming file
- Implement in FPGA through parallel-JTAG cable
- Check the behavior of design in FPGA by giving inputs

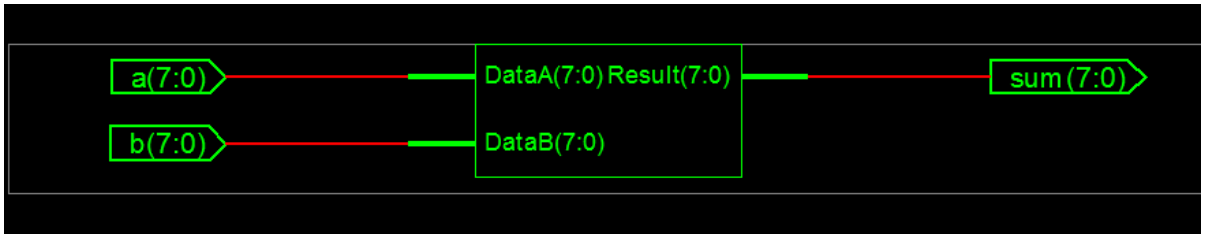
2.4 Program:

```
module ad(a,b,sum);
input [7:0]a,b;
output  [7:0]sum;
assign  sum=a+b;
endmodule
```

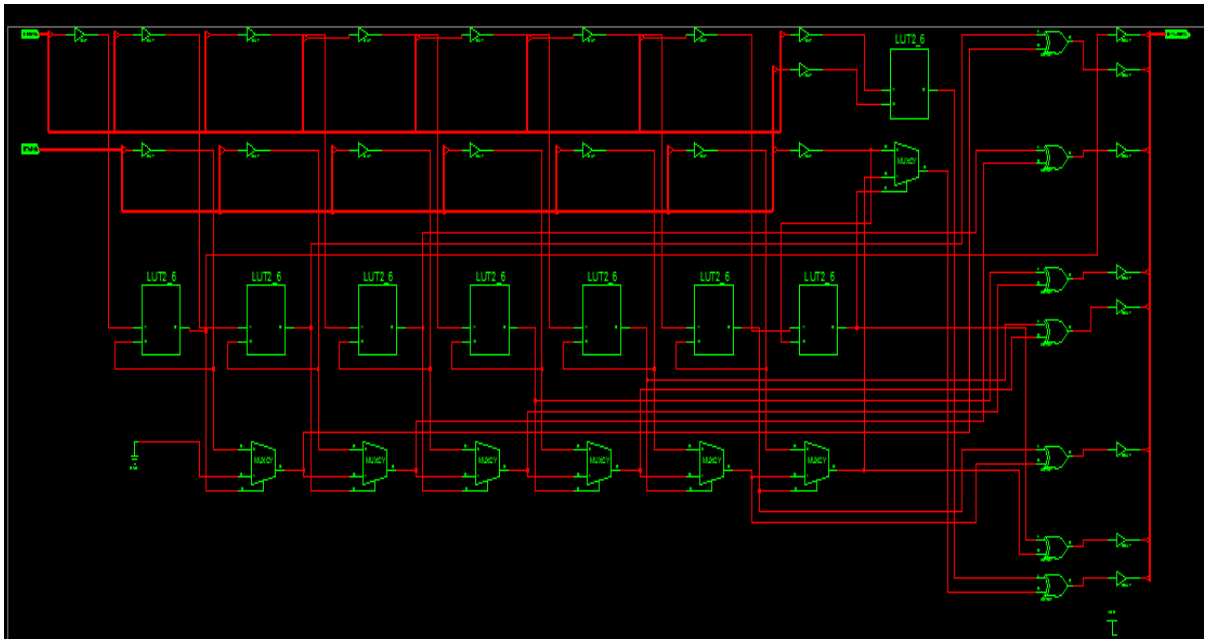
2.5 Truth Table:

A	B	RES
1111111	0000000	1111111

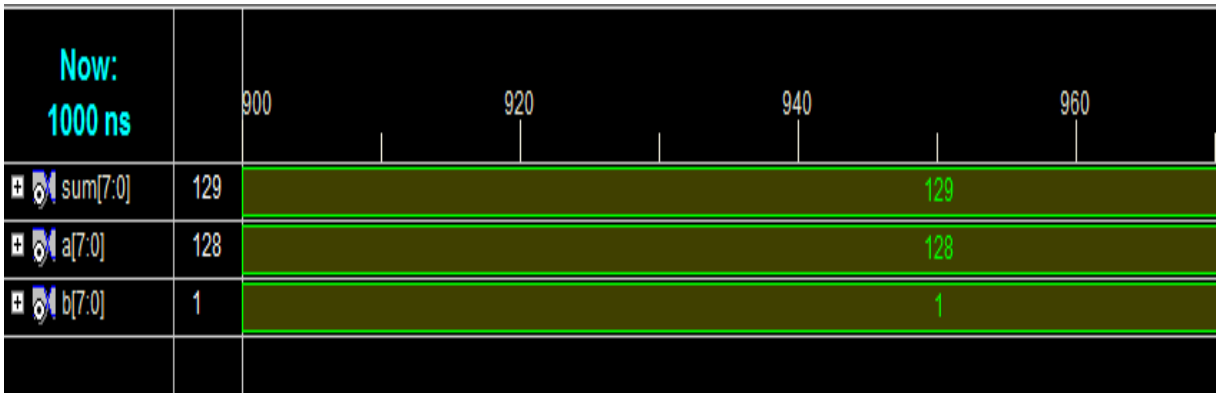
2.6 RTL SCHEMATIC:



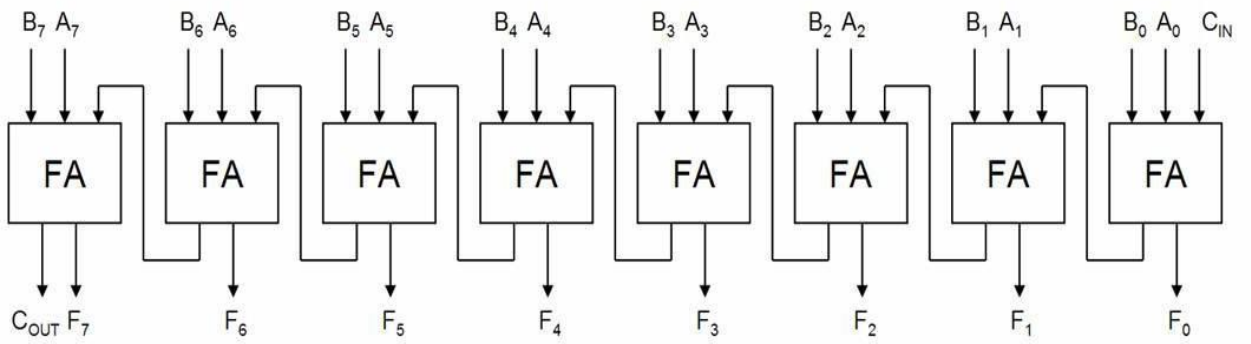
TECHNOLOGIC SCHEMATIC



2.7 OUTPUT WAVEFORM:



2.9 RIPPLE CARRY ADDER:



2.10 PROGRAM:

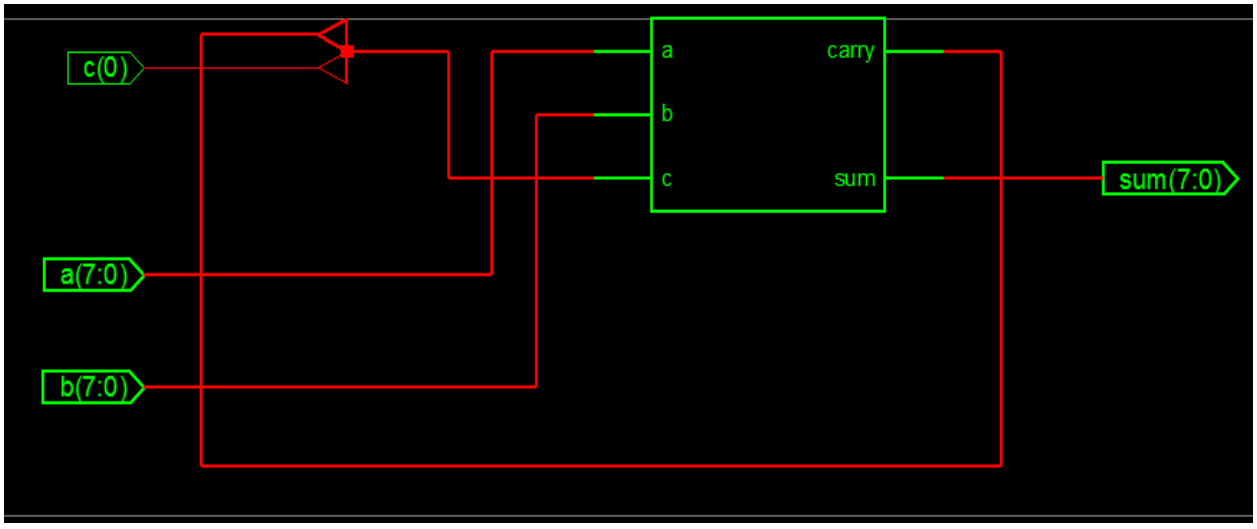
Main program:

```
module sw(a, b, c, sum, carry);
input [7:0]a;
input [7:0]b;
input c;
output [7:0]sum;
output carry;
wire [6:0]c;
gk fa0(a[0],b[0],c,sum[0],c1);
gk fa1(a[1],b[1],c1,sum[1],c2);
gk fa2(a[2],b[2],c2,sum[2],c3);
gk fa3(a[3],b[3],c3,sum[3],c4);
gk fa4(a[4],b[4],c4,sum[4],c5);
gk fa5(a[5],b[5],c5,sum[5],c6);
gk fa6(a[6],b[6],c6,sum[6],c7);
gk fa7(a[7],b[7],c7,sum[7],carry);
endmodule
```

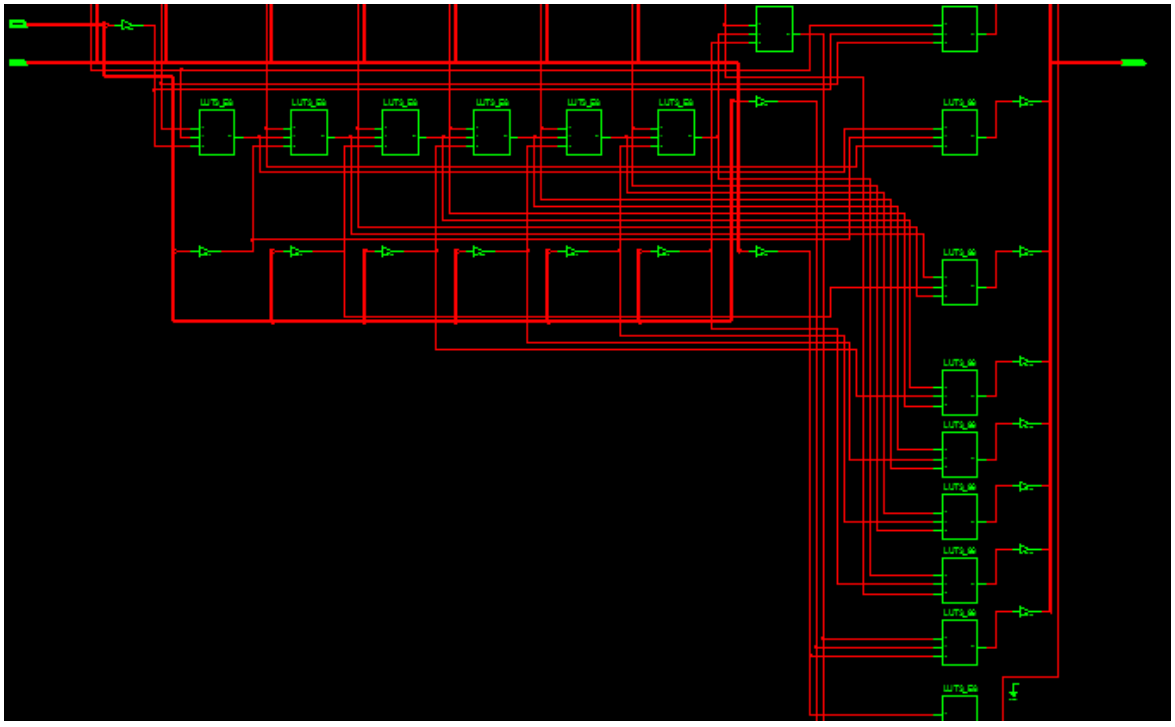
Sub-Program

```
module fulladd (sum, carry, a, b, c);
input a, b, c;
output sum, carry;
wire w1, w2, w3;
xor (sum,a,b,c);
and (w1,a,b);
and(w2,b,c,);
and(w3,c,a);
or(carry,w1,w2,w3);
endmodule
```

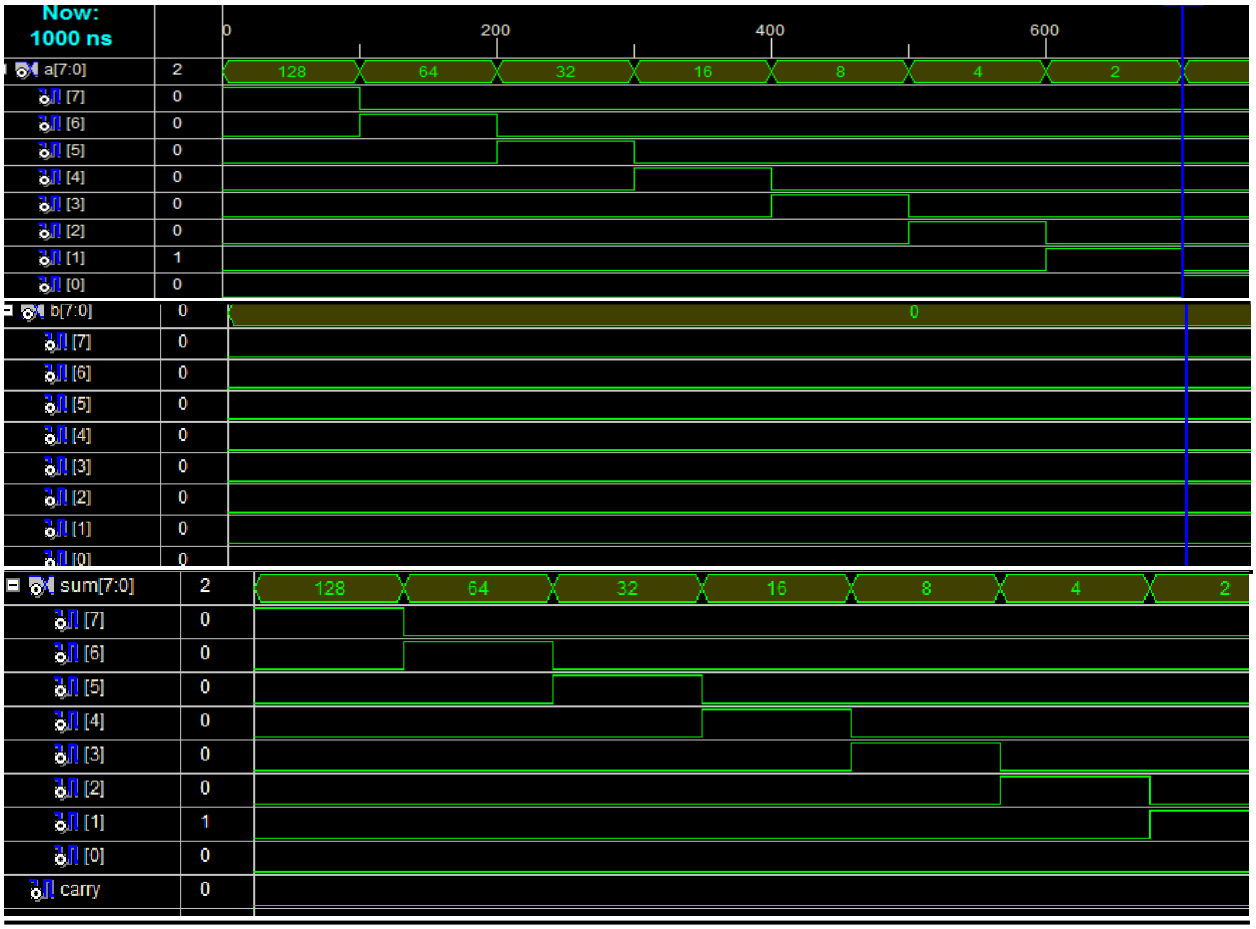
2.11 RTL SCHEMATIC:



TECHNOLOGICAL SCHEMATIC:



2.12 OUTPUT WAVEFORM:



2.13 RESULT: Thus simple adder and ripple carry adder was designed and implemented successfully.

3. DESIGN OF CMOS DIFFERENTIAL AMPLIFIER USING TANNER

3.1 AIM:

To design a CMOS Differential Amplifier using the Schematic entry tool, Tanner and verify its functioning.

3.2 APPARATUS REQUIRED:

1. Tanner tool
2. PC

3.3 THEORY:

A differential amplifier is a type of electronic amplifier that multiplies the difference between two inputs by some constant factor (the differential gain). Many electronic devices use differential amplifiers internally. The output of an ideal differential amplifier is given by:

$$V_{\text{out}} = A_{\text{d}}(V_{\text{in}}^{+} - V_{\text{in}}^{-})$$

Where Vin+ and Vin- are the input voltages and Ac is the differential gain. In practice, however, the gain is not quite equal for the two inputs. This means that if Vin+ and Vin-are equal, the output will not be zero, as it would be in the ideal case. A more realistic expression for the output of a differential amplifier thus i n c l u d e s a second term.

$$V_{\text{out}} = A_{\text{d}}(V_{\text{in}}^{+} - V_{\text{in}}^{-}) + A_{\text{c}}\left(\frac{V_{\text{in}}^{+} + V_{\text{in}}^{-}}{2}\right)$$

*A*c is called the common-mode gain of the amplifier. As differential amplifiers are often used when it is desired to null out noise or bias-voltages that appear at both inputs, a low common-mode gain is usually considered good.

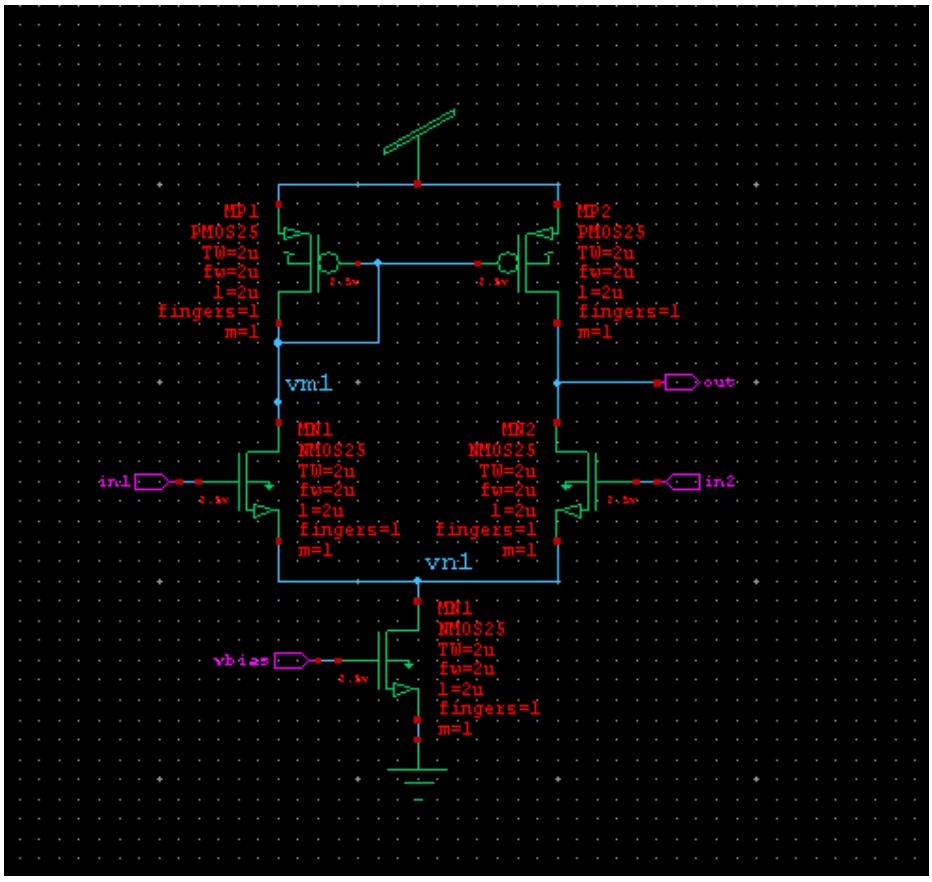
The common-mode rejection ratio, usually defined as the ratio between differential- mode gain and common-mode gain, indicates the ability of the amplifier to accurately cancel voltages that are common to both inputs. Common-mode rejection ratio (CMRR):

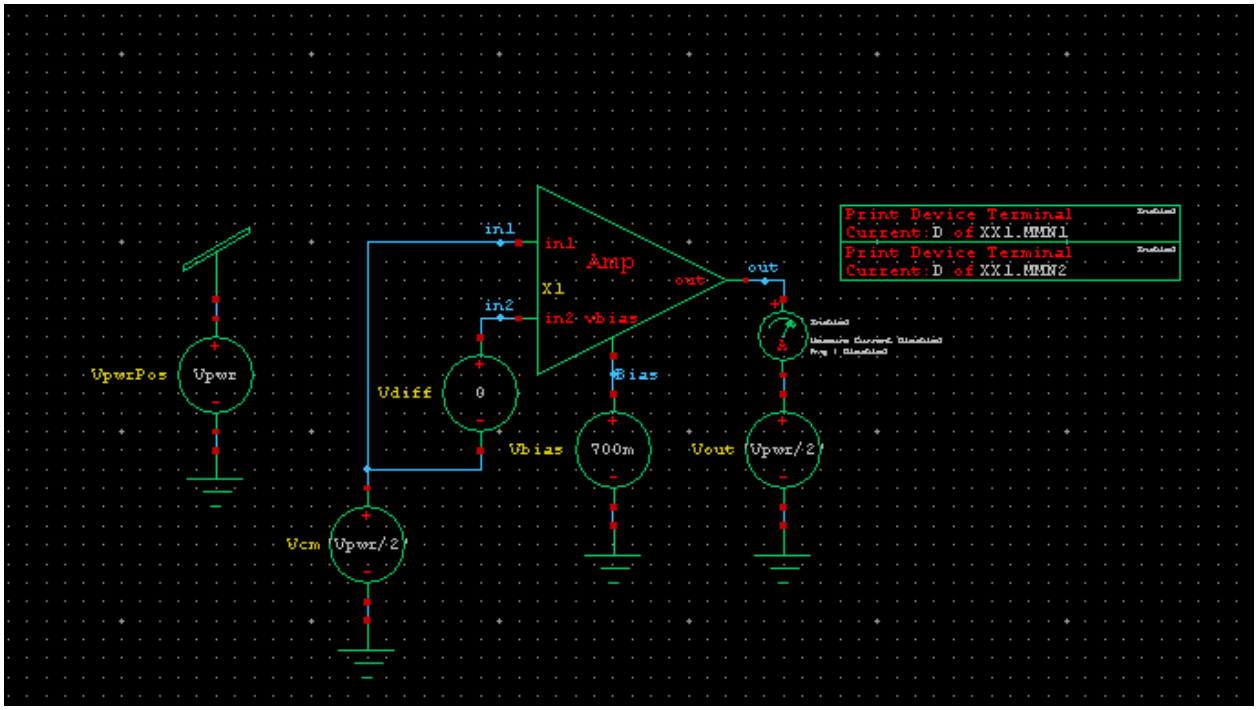
$$\text{CMRR} \triangleq \frac{A_{\text{d}}}{A_{\text{c}}}$$

3.4 ALGORITHM

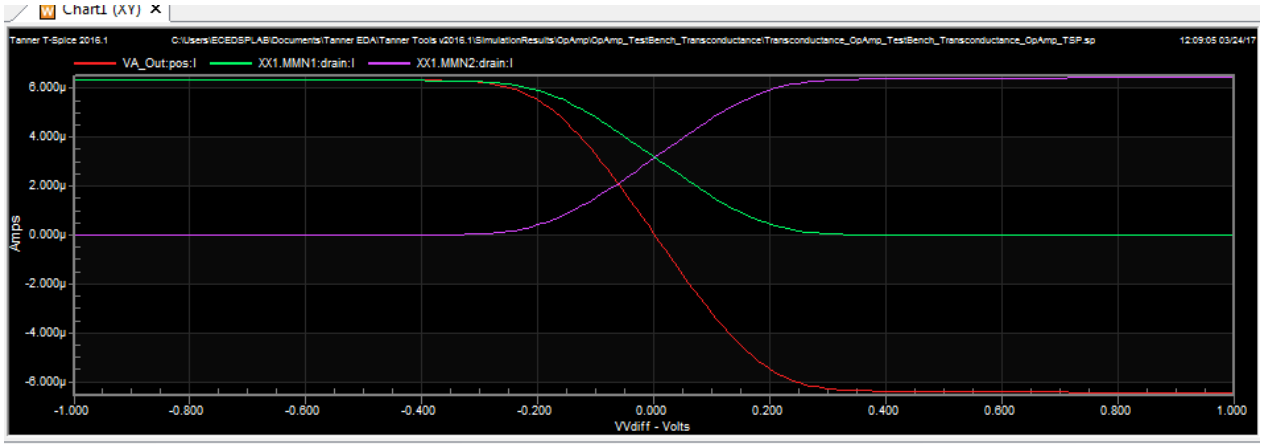
- Draw the schematic of CMOS differential amplifier using S-edit.
- Perform Transient Analysis of the CMOS Inverter.
- Obtain the output waveform from W-edit.
- Obtain the spice code using T-edit.

DIFFERENTIAL AMPLIFIER CIRCUIT:





3.5WAVEFORM:



3.6 RESULT: The design and simulation of Differential Amplifier has been performed using Tanner EDA Tools.