POORNIMA
COLLEGE OF ENGINEERING

Department of Electrical Engineering

Embedded System Lab Manual

Year: -4thYr. /VII SEM

Lab Code: - 7EE4-21

INDEX

POORNIMA COLLEGE OF ENGINEERING, JAIPUR

DEPARTMENT OF ELECTRICAL ENGINEERING

VISION

To be a model of excellence in Professional Education and Research by creating electrical engineers who are prepared for lifelong engagement in the rapidly changing fields and technologies with the ability to work in team.

MISSION

- ✓ To provide a dynamic environment of technical education wherein students learn in collaboration with others to develop knowledge of basic and engineering sciences.
- ✓ To identify and strengthen current thrust areas based upon informed perception of global societal issues in the electrical and allied branches.
- ✓ To develop human potential with intellectual capability who can become a good professional, researcher and lifelong learner.

PROGRAM EDUCATIONAL OBJECTIVES (PEO's)

PEO 1: Graduates will have the ability to formulate, analyze and apply design process using the basic knowledge of engineering and sciences to solve complex electrical engineering problems.

PEO 2: Graduates will exhibit quality of leadership, teamwork, time management, with a commitment towards addressing societal issues of equity, public and environmental safety using modern engineering tools.

PEO 3: Graduates will possess dynamic communication and have successful transition into a broad range of multi-disciplinary career options in industry, government and research as lifelong learner.

PROGRAM OUTCOMES (PO's)

Engineering Graduates will be able to:

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 (PSO's)

PSO1: Graduate possesses the ability to apply fundamental knowledge of basic sciences, mathematics and computation to solve the problems in the field of electrical engineering for the benefit of society.

PSO2: Graduate possesses the ability to professionally communicate and ethically solve complex electrical engineering problems using modern engineering tools.

PSO3: Graduate possesses sound fundamental knowledge to be either employable or develop entrepreneurship in the emerging areas of renewable and green energy, electric and hybrid vehicles and smart grids and shall be susceptive to life- long learning.

LAB OUTCOMES

LO1: Explain the fundamentals of embedded system and sensor integration. **[Understand]**

LO2: Practice the programming knowledge for controlling a real time process using hardware in loop system. **[Apply]**

LO3: Investigate the type of sensor required in a particular control process. **[Analyze]**

LO4: Criticize the processing time requirements for conversion of real time data into digital domain and vice versa. **[Analyze]**

LO5: Check the complex real world embedded system processes. **[Evaluate]**

MAPPING OF LO WITH PO

LO	LAB OUTCOME	PO											
		1	2	3	4	5	6	7	8	9	10	11	12
1	Student will be able to **Explain** the fundamentals of embedded system and sensor integration. **[Understand]**	2	-	-	-	3	-	-	3	3	2	1	3
2	Student will be able to **Practice** the programming knowledge for controlling a real time process using hardware in loop system. **[Apply]**	2	-	-	-	3	-	-	3	3	2	1	3
3	Student will be able to **Investigate** the type of sensor required in a particular control process. **[Analyze]**	2	-	-	2	3	-	-	3	3	2	1	3
4	Student will be able to **Criticize** the processing time requirements for conversion of real time data into digital domain and vice versa. **[Analyze]**												
5	Student will be able to **Check** the complex real world embedded system processes. **[Evaluate]**	2	-	-	3	3	-	-	3	3	2	1	3

MAPPING OF LO WITH PSO

LO	LAB OUTCOME	PSO1	PSO2	PSO3
1	Student will be able to **Explain** the fundamentals of embedded system and sensor integration. **[Understand]**	1	-	-
2	Student will be able to **Practice** the programming knowledge for controlling a real time process using hardware in loop system. **[Apply]**	-	-	-
3	Student will be able to **Investigate** the type of sensor required in a particular control process. **[Analyze]**	-	-	-
4	Student will be able to **Criticize** the processing time requirements for conversion of real time data into digital domain and vice versa. **[Analyze]**	-	-	-

LAB RULES

DO'S:

- Enter the lab on time and leave at proper time.
- Wait for the previous class to leave before the next class enters.
- Keep the bag outside in the respective racks.
- Utilize lab hours in the corresponding.
- Before switching on the power supply, get it checked by the lecturer/Technical assistant.
- Switch off or silent your mobile before enter the lab.
- Maintaining discipline.
- Proper handling of equipment must be done.

DONT'S:

- Don't abuse the equipment.
- Don't bring any external material in the lab, except your lab record, copy and books.
- Don't bring the mobile phones in the lab. If necessary then keep them in silence mode.
- Please be considerate of those around you, especially in terms of noise level. While labs are a natural place for conversations of all types, kindly keep the volume turned down.
- Do not touch any any power supply wire or main supply.
- Do not attempt experiment without permission.
- Do not overcrowd on a table.
- Do not manipulate the experiment result.

SAFETY MEASURES

- Specific Safety Rules like Do's and Don'ts are displayed and instructed for all students.
- First aid box and fire extinguishers are kept in each laboratory.
- Insulation carpet is available in machine lab and Measurement and Instrumentation Lab.
- Well trained technical supporting staff monitor the labs at all times.
- Damaged equipment's are identified and serviced at the earliest.
- Periodical calibration of the lab equipment's are regularly done
- A clean and organized laboratories are maintained
- The use of cell phones is prohibited.
- Appropriate storage areas are available.
- In order to create more space in the laboratories, a separate section has racks to store the belongings of the students.
- Proper earthing is provided in the labs.

LIST OF EXPERIMENTS

Max. Marks=100

SN	Contents
1	Introduction to Embedded Systems and their working.
2	Data transfer instructions using different addressing modes and block trans- fer.
3	Write a program for Arithmetic operations in binary and BCD-addition, subtraction, multiplication and division and display.
4	Interfacing D/A converter & Write a program for generation of simple waveforms such as triangular, ramp, Square etc.
5	Write a program to interfacing IR sensor to realize obstacle detector.
6	Write a program to implement temperature measurement and displaying the same on an LCD display.
7	Write a program for interfacing GAS sensor and perform GAS leakage detection.
8	Write a program to design the Traffic Light System and implement the same using suitable hardware.
9	Write a program for interfacing finger print sensor.
10	Write a program for Master Slave Communication between using suitable hardware and using SPI
11	Write a program for variable frequency square wave generation using with suitable hardware.
12	Write a program to implement a PWM based speed controller for 12 V/24V DC Motor incorporating a suitable potentiometer to provide the set point.

EVALUATION SCHEME

Name Of Exam	Conducted By	Experiment Marks	Viva Marks	Total
I Mid Term	PCE	30	10	40
II Mid Term	PCE	30	10	40
End Term	RTU	30	10	40

Name Of Exam	Conducted By	Performance Marks	Attendance Marks	Total
Sessional	PCE	30	10	40

DISTRIBUTION OF LAB RECORD MARKS PER EXPERIMENT

Attendance	Record	Performance	Total
2	3	5	10

LAB PLAN

Total number of experiment: 12

Total number of turns required: 12

NUMBER OF TURNS REQUIRED FOR

Experiment Number	Turns	Scheduled Day
Zero Lab	1	Turn 1
Exp. 1	1	Turn 2
Exp. 2	1	Turn 3
Exp. 3	1	Turn 4
Exp. 4	1	Turn 5
Exp. 5	1	Turn 6
Exp. 6	1	Turn 7
Exp. 7	1	Turn 8
Exp. 8	1	Turn 9
Exp. 9	1	Turn 10
Exp. 10	1	Turn 11
Exp. 11	1	Turn 12
Exp. 12	1	Turn 13

DISTRIBUTION OF LAB HOURS

- Explanation of Experiment & Logic : 20 Minutes
- Performing the Experiment : 40 Minutes
- File Checking : 30 Minutes
- Viva/Quiz : 20 Minutes
- Solving of Queries : 10 Minutes

ROTOR PLAN

ROTOR I

1. Introduction to Embedded Systems and their working
2. Data transfer instructions using different addressing modes and block transfer.
3. Write a program for Arithmetic operations in binary and BCD-addition, sub- traction, multiplication and division and display.
4. Interfacing D/A converter & Write a program for generation of simple wave- forms such as triangular, ramp, Square etc.
5. Write a program to interfacing IR sensor to realize obstacle detector.
6. Write a program to implement temperature measurement and displaying the same on an LCD display.

ROTOR II

7. Write a program for interfacing GAS sensor and perform GAS leakage detection.

8. Write a program to design the Traffic Light System and implement the same using suitable hardware.

9. Write a program for interfacing finger print sensor..

10. Write a program for Master Slave Communication between using suitable hardware and using SPI

11. Write a program for variable frequency square wave generation using with suitable hardware.

12. Write a program to implement a PWM based speed controller for 12 V/24V DC Motor incorporating a suitable potentiometer to provide the set point.

ZERO LAB

Introduction to Lab:

a). **Relevance to Branch**: - Embedded system are used in industrial control applications which is a very important field for Electrical Engineers as the devices that requires timing and control use microprocessors. Electrical Engineers basically deal with the power electronics and control panels to control the amount of load to be delivered to the different sections. Computer Architecture deals with the memory elements of computer and its organization which is again very useful and relevant for electrical engineering branch.

b). **Relevance to Society**: - Looking into the present arena of faster and faster devices most of which are based on the microprocessors, it becomes very important for Engineers to develop newer and portable devices for the society and that can be done by including the subject in the curriculum as a basis for that .

c). **Relevance to Self**: - It helps the students to acquire knowledge of both the hardware and the software. It is a very important subject for Competitive exams like GATE, PSUs and many other and most important for securing marks in university exams. It is very useful in building the projects.

d). **Relation with laboratory**: - :- Lab is important to get practical knowledge of what we study in theory. This Lab is related to the Microprocessor Programming Lab.

e) Pre- Requisites (Connection with previous year): -

1. Electronic Devices Lab (3EE7A)
2. Analog Electronics Lab (4EE7A)
3. Microprocessor Lab (5EE4-23)
4. Power Electronics Lab (5EE7A)
5. Advanced Power electronics Lab (6EE9A)

DEPARTMENT OF ELECTRICAL ENGINEERING

EMBEDDED SYSTEM LAB

Experiment No : 1

Introduction to Embedded Systems and their working.

TITLE : Introduction to Embedded Systems and their working.

AIM

To study 8051Microcontroller.

THEORY

The 8051 Microcontroller includes a whole family of microcontrollers that have numbers ranging from 8031to 8051 and are available in N channel metal oxide silicon (NMOS) and complementary metal oxide silicon (CMOS) construction in a variety of package types. There exits a enhanced version of 8051 known as 8052 with its own family of variations and even includes one member that can be programmed in BASIC.

Architecture

From internal block diagram of 8051 we see the unique features of the microcontroller are :

i) Internal ROM AND RAM.
ii) I/O port with programmable pins.
iii) Timers and counters.
iv) Serial data communication.

The figure also shows usual CPU component which are as follows:
The program counter, Arithmetic Logic Unit, working registers and clock circuits.
The 8051 architecture consists of these specific features:

i) Eight bit CPU with registers A (accumulator) & B.
ii) Sixteen bit program counter (PC) and data pointer (DPTR)
iii) Eight bit program status word(PSW)
iv) Internal ROM or EPROM (8751) of 0 (8031) to 4k (8051)
v) Eight bit stack pointer (SP).
vi) Internal RAM of 128 bytes.
    a) Four register banks, each containing 8 registers.
    b) Sixteen bytes may be addressed at bit level.
    c) Eighty bytes of general purpose data memory.
vii) Thirty two input output pins arranged as four, eight bit ports Po-P3
viii) Two 16 bits timer/counter To-T1
ix) The duplex serial data receiver /transmitter.(SBUF)
x) Control registers: TCON, TMOD, SCON, PCON, IP& IE.
xi) Two external and three internal interrupt sources.
xii) Oscillator and clock circuits.

The hardware of 8051 consists of following main parts.

1) **The 8051 oscillator and clock**.

The heart of 8051 is the circuitry that generates the clock pulses by which all internal operations are synchronized. Pins XTAL1 & XTAL2 are provided to connect a resonant network to form an oscillator.

Typically a quartz crystal and capacitor are employed. The crystal frequency is basic internal clock frequency of microcontroller.

The 8051 is normally available with certain maximum and minimum frequency typically 1 MHz to 16 MHz. Minimum frequency imply that certain internal memories are dynamic and must always operate above a minimum frequency otherwise all data will be lost.

Ceramic resonators may be used as low cost alternative to crystal resonators. However, a decrease in frequency stability and accuracy make the ceramic resonator a poor choice of high speed serial data communication with other systems, or, critical timing, is to be done. The ALE pulse which is primarily used as a timing pulse for external memory access, in dictates when every instruction byte is fetched.

2) Program counter and data pointer (PC & DPTR)

The 8051 contains two 16 bit registers, program counter and data pointer. Each is used to hold the address of a byte in memory.

Program instruction bytes are fetched from locations in memory that are addressed by PC. PC is automatically incremented after every instruction byte is fetched and may be altered by certain instructions. The PC is only register that does not have any internal address.

The DPTR register is made up of two 8 bit registers named DPH &DPL, which are used to furnish memory addresses for internal and external code access and external data access. The DPTR is under the control of program instructions and can be specified by its 16 bit name, DPTR or by each byte individual byte name DPH &DPL .DPTR does not have a single internal address; DPH &DPL are each assigned an address.

3) A & B CPU register

The 8051 contains 34 general purpose or working registers. Two of these registers A& B hold results of many instructions, particularly mathematical and logical operations of 8051 central processing unit,(CPU).The other 32 are arranged as part of internal RAM in four bank B0-B3 of eight registers and comprise the mathematical core.

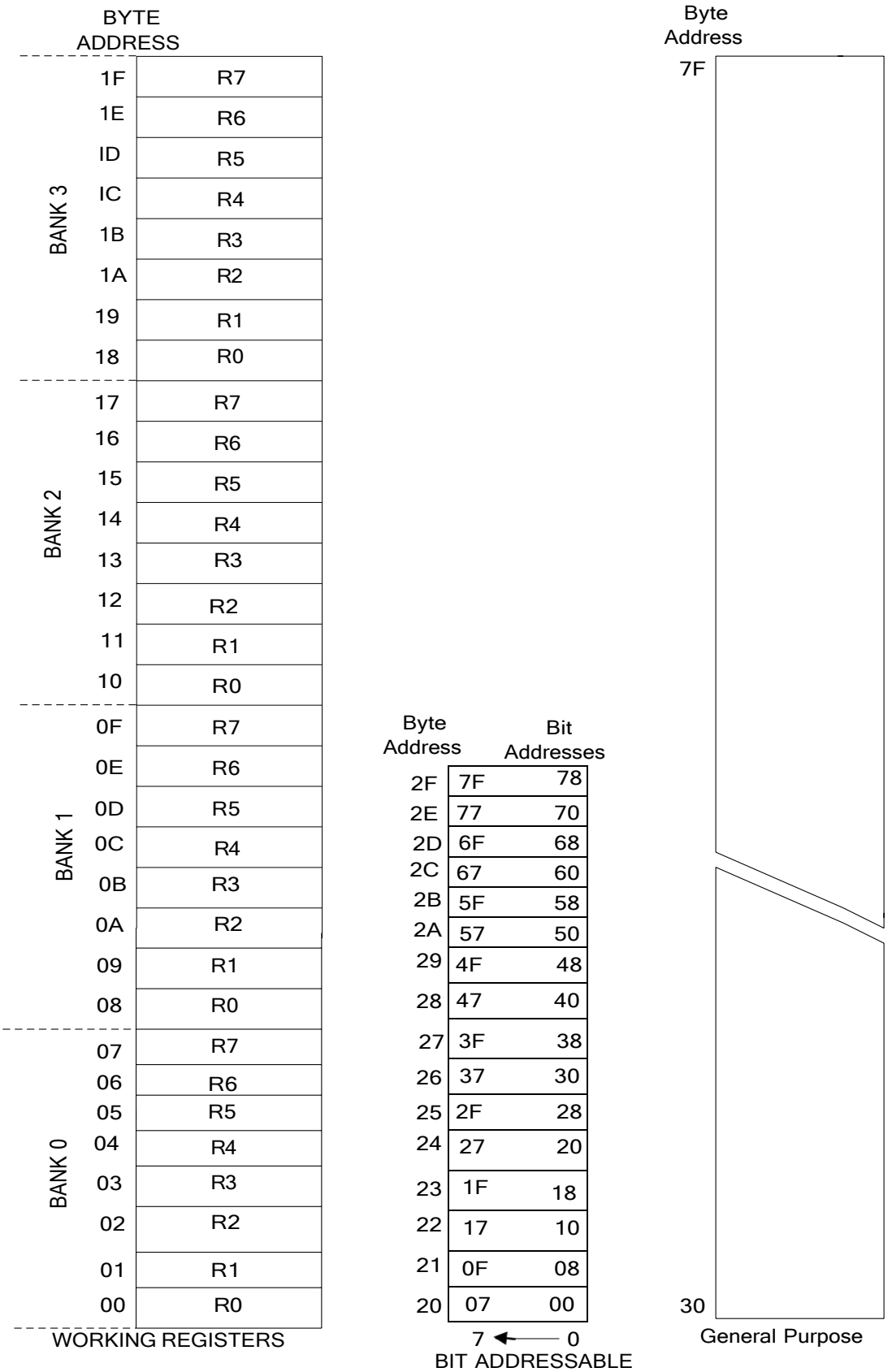
The accumulator is the most versatile of the two CPU registers and used for many operations including addition, subtraction, multiplication and division and boolean bit manipulation .It is also used for all data transfer between the 8051 and any external memory. The register B is used for multiplication and division operations together with A. This has no other function other than as a location where data may be stored.

4) Flags and Program status word (PSW)

Flags are 1 bit registers provided to store the result of certain program instructions. Other instructions can test the conditions of the flags and make decisions based upon the flag status. In order that flag may be conveniently addresses as they are grouped inside the program status word (PSW) &power control (PCON) register.

The 8051 has four math flags that respond automatically to the outcomes of mathematical operations and three general purpose user flags that can be set to 1 or cleared to zero by programmers as desired. The math flag includes carry(C) ,auxiliary carry(AC),overflow(O) and parity(P) .User flags are named Fo,GFO,GF1 they are general purpose flag that may be used by programmer to record some event in the program. Note that all of the flags can be set and cleared by the programmer at will. The math flag, However are also affected by math operations.

Internal RAM Organization



5) **Internal RAM**.

The 128 byte Internal RAM which is shown in figure is organized into three distinct are as

a. Thirty two bytes from address 00h TO 1F H that make up 32 working register organized as four banks of eight registers named R0 to R7. Each register can be addressed by a name (when its bank is selected) or by its RAM address. Thus RO of bank 3 is RO (if bank 3 is currently selected) or address 18 h (whether banks 3 is selected or not). Bits RS0 &RS1 in the PSW determine which bank of registers is currently in use at any time when the program is running. Register banks not selected can be used as general purpose RAM. Bank 0 is selected n RESET.

b. A bit addressable area of 16 bytes occupies RAM byte address 20H to 2FH from a total of 128 addressable bits. An addressable bit may be specified by its bit address of 00H to 75 H or 8 bits may form any byte address from 20H to 2FH Thus, for example bit address 4F H is also bit 7 of byte address, 29 H.

c. A general purpose RAM area above the bit area from 30 H to 7FH is addressable as byte.

6) Internal Memory

The functioning computer must have memory for program code bytes. Commonly in ROM& RAM memory for variable data that can be altered as the program runs. The 8051 has internal RAM &ROM memory for these functions .Additional memory can be added externally using suitable circuits.

7) The stack and the stack pointer

The stack refers to an area of internal RAM that is used in conjunction with certain op codes to store and retrieve data quickly. The 8 bit stack pointer SP register is used by the 8051 to hold an internal RAM address that is called top of the stack. The address hold in SP register is the location in internal RAM where the last byte of data was stored by a stack operation.

When data is to be placed on stack , SP increments before storing the data on the stack so that the stack grows up as data is stored. As data is retrieved from the stack, the byte is read from the stack and then the SP decrements to point to next available byte of stored data.

The stack of is limited in height to the size of internal RAM .The stack has a potential to overwrite a valuable data in the register banks, bit addressable RAM and scratch pad RAM areas.

8) Special function register

The 8051 operations that do not use the internal 128 byte RAM addresses from 00H to 7F H are done by a group of specific internal registers, each called a special function register (SFR),which may be addressed much like internal RAM, using addresses 80 H to FF H. Some SFR's are also bit addressable as incase of bit area of RAM .This feature allows the programmer to change only what needs to be altered, leaving the remaining bits in that SFR unchanged.

Not all of addresses from 890 H to FF H is used for SFR's and attempting to use an address that is not defined, or empty, results in unpredictable results. The PC is not part of SFR and has no internal RAM address.

SFR's are named in certain op codes by their functional names, such as A or TH0 and are referred by other op codes by their addresses a such as OEOH or 8C H. Note that any address used in program must start with a number; thus address EOH for the A SFR begins with 0.Failure to use this number convention will result in an assembler error when the program is assembled.

9) Internal ROM

A corresponding block of internal program code, contained in an internal ROM, occupies code address space 0000H to 0FFFH.

10) I/O Port

All ports are 8 bit bidirectional I/O ports. All ports except port 1 have alternate function.

i) Port 0- Bi-directional. Lower order address and data bus for external memory
ii) Port 1- Only used as input and output ports.
iii) Port 2- Input /output ports also to supply a high order address byte in conjunction with port 0 to address external memory.
iv) Port 3- Port 3 pins have special serial port functions along with their I/P-O/P purpose.

Counter and Timer

Two 16 bits up counter T0 &T1 are provided for general use. Each counter can count internal clock pulses, acting as timer or it can count external pulses as counter. All counters is controlled by bit status in timer mode control (TMOD) and timer counter control register (TCON).

TMOD consists of two duplicate four bit register each of which controls the action of one of the timer. TCON has control bits and flags for the timer in the upper nibble, and control bits and flags for the external interrupts in the lower nibble.

Interrupts:

There are two external timer interrupts and one serial interrupt**.**

PROGRAM STATUS WORD

7	6	5	4	3	2	1	0
CY	AC	FO	RS1	RS0	OV	-	P

Bit	Symbol	Function
7	CY	Carry flag used in arithmetic jump, rotate and Boolean instructions
6	AC	Auxiliary carry flag is used for BCD arithmetic.
5	FO	User flag 0
4	RS1	Register bank select bit 1
3	RS0	Register bank select bit 0
2	OV	Overflow flag used in arithmetic instructions.
1	-	Reserved for future use
0	P	Parity flag shows parity of register A: 1=odd parity.

RS1	RS0	BANK SELECTED
0	0	0
0	1	1

1	0	2
1	1	3

CONCLUSION

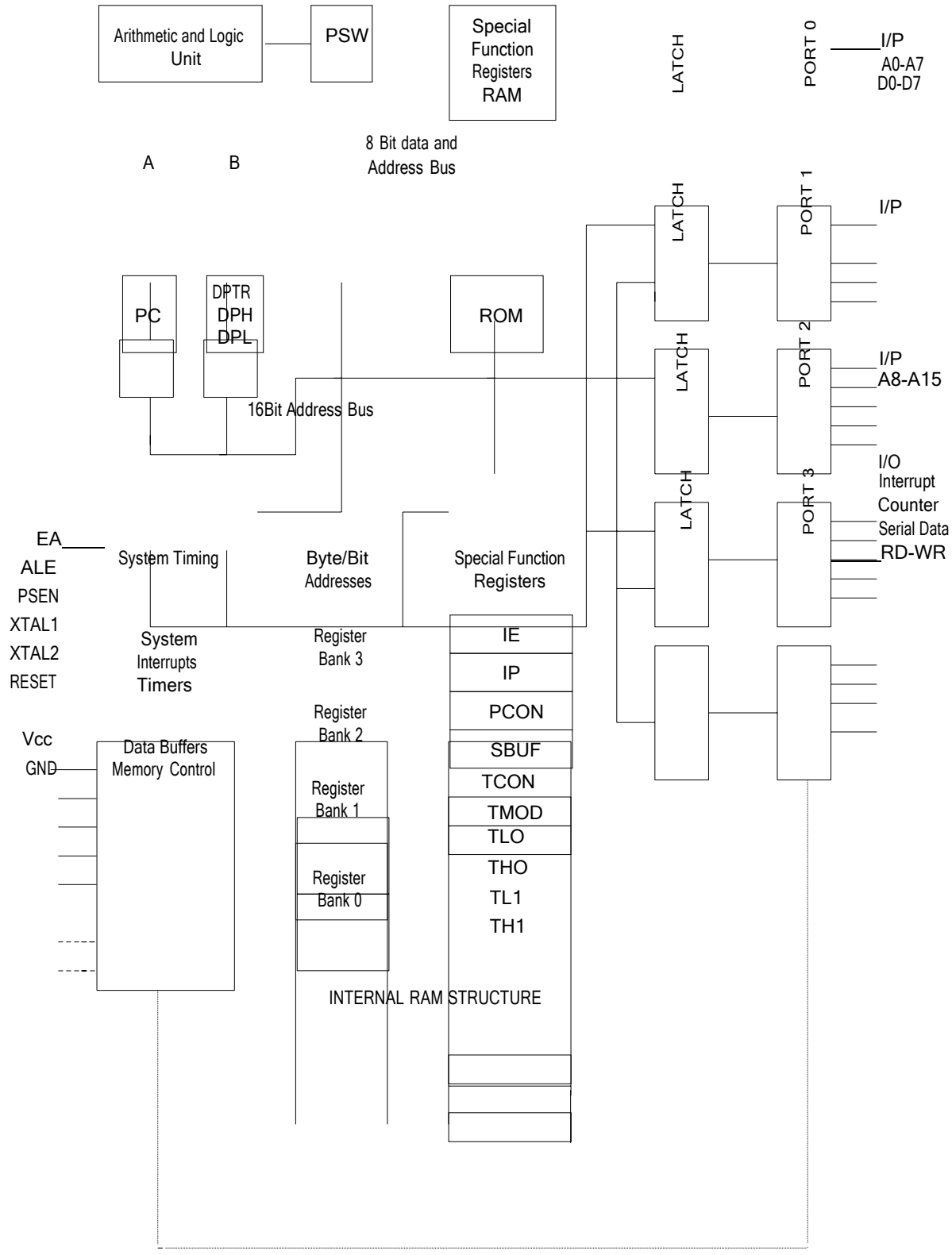
Successfully Study the Introduction to Embedded Systems and their working.

VIVA QUESTIONS:

1. Can you define what an embedded system is and provide examples of where they are commonly used?
2. What are the key components of an embedded system and how do they interact with each other?
3. Explain the difference between microcontrollers and microprocessors in the context of embedded systems.
4. What are the advantages and disadvantages of using an embedded system compared to a general-purpose computer?
5. Can you explain the role of sensors and actuators in embedded systems? Provide examples of each.
6. How does real-time processing play a crucial role in embedded systems? Can you give examples of real-time applications?
7. What are the various programming languages used for programming embedded systems, and what factors might influence the choice of language?
8. Discuss the importance of power consumption and efficiency in embedded systems design.
9. What are some common communication protocols used in embedded systems, and when would you choose one over the other?
10. Explain the concept of interrupts in embedded systems and how they are used to handle time-critical tasks.
11. What are the challenges involved in debugging and testing embedded systems? How can they be overcome?
12. Discuss the concept of firmware and its significance in embedded systems.
13. Can you explain the process of designing and implementing a simple embedded system project from conception to completion?
14. How do embedded systems contribute to the development of IoT (Internet of Things) devices?
15. What are some emerging trends and advancements in the field of embedded systems that you find particularly interesting or promising?

DIAGRAM

8051 BLOCK DIAGRAM



PIN DIAGRAM OF 8051 8-BIT MICROCONTROLLER

Port 1.0	1 P1.0	Vcc 40	+5V
Port 1.1	2 P1.1	(AD0)P0.0 39	Port 0.0
Port 1.2	3 P1.2	(AD1)P0.1 38	Port 0.1
Port 1.3	4 P1.3	(AD2)P0.2 37	Port 0.2
Port 1.4	5 P1.4	(AD3)P0.3 36	Port 0.3
Port 1.5	6 P1.5	(AD4)P0.4 35	Port 0.4
Port 1.6	7 P1.6	(AD5)P0.5 34	Port 0.5
Port 1.7	8 P1.7	(AD6)P0.6 33	Port 0.6
Reset Input	9 RST	(AD7)P0.7 32	Port 0.7
Port 3.0	10 P3.0(RXD)	(VPP)/EA 31	External Enable
Port 3.1	11 P3.1(TXD)	(PROG)ALE 30	Address Latch Enable
Port 3.2	12 P3.2(INTO)	PSEN 29	Program store Enable
Port 3.3	13 P3.3(INT1)	(A15)P2.7 28	Port 2.7 Address 15
Port 3.4	14 P3.4(TO)	(A14)P2.6 27	Port 2.6 Address 14
Port 3.5	15 P3.5(T1)	(A13)P2.5 26	Port 2.5 Address 13
Port 3.6	16 P3.6(WR)	(A12)P2.4 25	Port 2.4 Address 12
Port 3.7	17 P3.7(RD)	(A11)P2.3 24	Port 2.3 Address 11
Crystal Input 2	18 XTAL2	(A10)P2.2 23	Port 2.2 Address 10
Crystal Input 1	19 XTAL1	(A9)P2.1 22	Port 2.1 Address 9
Ground	20 Vss	(A8)P2.0 21	Port 2.0 Address 8

PIN DIAGRAM OF 8051 8-BIT MICROCONTROLLER

Port 1.0	1	P1.0	Vcc 40		+5V
Port 1.1	2	P1.1	(AD0)P0.0 39		Port 0.0 Address/Data 0
Port 1.2	3	P1.2	(AD1)P0.1 38		Port 0.1Address/Data 1
Port 1.3	4	P1.3	(AD2)P0.2 37		Port 0.2 Address/ Data 2
Port 1.4	5	P1.4	(AD3)P0.3 36		Port 0.3Address/ Data 3
Port 1.5	6	P1.5	(AD4)P0.4 35		Port 0.4 Address/ Data 4
Port 1.6	7	P1.6	(AD5)P0.5 34		Port 0.5 Address/ Data 5
Port 1.7	8	P1.7	(AD6)P0.6 33		Port 0.6 Address/ Data 6
Reset Input	9	RST	(AD7)P0.7 32		Port 0.7 Address/ Data 7
Port 3.0	10	P3.0(RXD)	(VPP)/EA 31		External Enable EPROM Program Pulse
Port 3.1	11	P3.1(TXD)	(PROG)ALE 30		Address Latch Enable
Port 3.2	12	P3.2(INTO)	PSEN 29		Program store Enable
Port 3.3	13	P3.3(INT1)	(A15)P2.7 28		Port 2.7 Address 15
Port 3.4	14	P3.4(TO)	(A14)P2.6 27		Port 2.6 Address 14
Port 3.5	15	P3.5(T1)	(A13)P2.5 26		Port 2.5 Address 13
Port 3.6	16	P3.6(WR)	(A12)P2.4 25		Port 2.4 Address 12
Port 3.7	17	P3.7(RD)	(A11)P2.3 24		Port 2.3 Address 11
Crystal Input 2	18	XTAL2	(A10)P2.2 23		Port 2.2 Address 10
Crystal Input 1	19	XTAL1	(A9)P2.1 22		Port 2.1 Address 9
Ground		Vss	(A8)P2.0 21		Port 2.0 Address 8

DEPARTMENT OF ELECTRICAL ENGINEERING

EMBEDDED SYSTEM LAB

Experiment No : 2

Data transfer instructions using different addressing modes and block transfer

TITLE : To arrange numbers in ascending order.

AIM

Data transfer instructions using different addressing modes and block transfer.

PROGRAME

To write a Assembly program for data transfer using different addressing modes.

Immediate Addressing Mode Register Addressing Mode Direct Addressing Mode
Indirect Addressing Mode Base Index Addressing Mode

Immediate Addressing Mode:
In this addressing mode, the source must be a value that can be followed by the '#' and destination must be SFR registers, general purpose registers and address. It is used for immediately storing the value in the memory registers.

Memory address	Opcode	Label	Mnemonic	Comment
C000	74,20		MOV A, #20h	//A is an accumulator register
C002	78,20		MOV R0,#20h	// R0 is a general purpose register; 20 is stored in the R0 register//
C004	80,07		MOV P0, #07h	//P0 is a SFR register;07 is stored in the P0//
C006	75,05		MOV 20h, #05h	//20h is the address of the RAM; 05 stored in the 20h//

RESULT

Memory Location/Reg	Content
A	
R0	
P0	
20H	

Register Addressing Mode:

In this addressing mode, the source and destination must be a register, but not general purpose registers. So the data is not moved within the general purpose bank registers.

Syntax:

Memory address	Opcode	Label	Mnemonic	Comment
0000	74,23		MOV A,#23H	// A is a SFR register
0002	FB		MOV R3,A	//MOVING ACCUMULATOR CONTENT
0003	74,63		MOV A,#63H	IMMIGIATE DATA TO ACCUM
0005	CB		XCH A, R3	EXCHANGE
0006	EB		MOV A, R3	STORING R3 IN ACCUM

RESULT

Memory Location/Reg	Content
A	
R3	
A	
A AND R3	
A	

Direct Addressing Mode

In this addressing mode, the source or destination (or both source and destination) must be an address, but not value.

Memory address	Opcode	Label	Mnemonic	Comment
0000	75,97,01		MOV 07h,#01H	Moving immediate data (01h) in address location (07h)
0003	E5,07		MOV A, 07H	Moving data available at location/ address (07h) in Accumulator
0005	85,00,07		MOV 00h,07H	Moving data available at address location (07h) in to another address location (00h)
0008	E6		MOV A,@R0	Moving data available at address location which is identified by Ro in to an Accumulator
0009	A6,07		MOV @R0, 07H	Moving data present at address 07h in to address identified by R0
000B	75,01,35		MOV 01,#35H	Moving immediate data (35h) into address location 01h
000E	77,44		MOV @R1,#44H	Moving immediate data (44h) into address identified by memory pointer R1

RESULT

Memory Location/Reg	Content
07	
A	
00	
A	
07	
01	
35	

Indirect Addressing Mode:

In this addressing mode, the source or destination (or destination or source) must be a indirect address, but not a value. This addressing mode supports the pointer concept. The pointer is a variable that is used to store the address of the other variable. This pointer concept is only used for R0 and R1 registers.

In above example @R0 and @R1 is used as indirect addressing mode. Students must practice with different address and values.

Base Index Addressing Mode:

This addressing mode is used to read the data from the external memory or ROM memory. All addressing modes cannot read the data from the code memory. The code must read through the DPTR register. The DPTR is used to point the data in the code or external memory.

Syntax:

Memory address	Opcode	Label	Mnemonic	Comment
C000	93		MOVC A, @A+DPTR	C indicates code memory
C001	E0		MOVX A, @DPTR	X indicate external memory
C002	74,00		MOV A, #00H	00H is stored in the A register
C004	90,05,00		MOV DPTR, #0500H	DPTR points 0500h address in the memory
C007	93		MOVC A, @A+DPTR	send the value to the A register
C008	F5		MOV P0, A	data of A send to the PO registrar

RESULT

Memory Location/Reg	Content
A	
A	
A	
DPTR	
A	
PO	

VIVA QUESTIONS:

1. What is the purpose of data transfer instructions in computer architecture?
2. Can you explain the concept of addressing modes in the context of data transfer instructions?
3. What are the common addressing modes used in data transfer instructions?
4. How do immediate addressing mode and direct addressing mode differ from each other?
5. Describe the register addressing mode and its significance in data transfer operations.
6. What is indirect addressing mode, and how is it utilized in data transfer instructions?
7. Explain indexed addressing mode and its application in data transfer operations.
8. What are the benefits of using different addressing modes in data transfer instructions?
9. What is block transfer, and why is it important in data processing?
10. How does block transfer differ from regular data transfer instructions?
11. Describe the process of block transfer using assembly language instructions.
12. What are the advantages of employing block transfer over individual data transfer operations?
13. Can you explain the role of memory management in block transfer operations?
14. How do you optimize block transfer operations for performance efficiency?
15. Discuss any potential challenges or limitations associated with block transfer in data processing tasks.

■■■

List of data transfer instructions

Data transfer instructions are responsible for transferring data between various memory storing elements like registers, RAM, and ROM. The execution time of these instructions varies based on how complex an operation they have to perform. In the table given below, we have listed all the data transfer instruction. In the table [A]= Accumulator; [Rn]=Register in RAM; DPTR=Data Pointer; PC=Program Counter

Operation	Mnemonics	Description
Register to register	MOV A, Rn	[A]<-[Rn]
	MOV Rn, A	[Rn]<-[A]
	XCH A, Rn	[A]<-[Rn]
Memory to register	MOV A, @Rn	[A]<-[Address in register]
	MOV A, address	[A]<-[Address]
	MOV Rn, address	[Rn]<-[Address]
	MOVX A, @Rn	[A]<-[Address in External ROM]
	MOVC A, @A+DPTR	[A]<-[Address in Internal ROM]
	MOVC A, @A+PC	[A]<-[Address in Internal ROM]
	MOVX A, @DPTR	[A]<-[Address in External ROM]
	XCH A, @Rn	[A]<-[Address]
	XCHD A, @Rn	[A]<-[Address]
	XCH A, address	[A]<-[Address]
Register to memory	MOVX @Ri, A	[Address]<-[A]
	MOV a8, A	[Address]<-[A]
	MOV a8, Rn	[Address]<-[Rn]
	MOVX @DPTR, A	[Address]<-[A]
	MOV @Rn, A	[Address]<-[A]
Data to Register	MOV A, #data	[A]<-[Data]
	MOV Rn, #data	[Rn]<-[Data]
	MOV DPTR, #data	[DPTR]<-[Data]
Address to Address	MOV a8, @Rn	[Address]<-[Address]
	MOV address, address	[Address]<-[Address]
	MOV @Rn, address	[Address]<-[Address]
Data to address	MOV address, #data	[Address]<-[Data]
	MOV @Rn, #d8	[Address]<-[Data]
Stack	PUSH a8	Data added to stack
	POP a8	Data removed from the stack

DEPARTMENT OF ELECTRICAL ENGINEERING

EMBEDDED SYSTEMS

Experiment No : 03

Write a program for Arithmetic operations in binary and BCD- addition, subtraction, multiplication and division and display.

TITLE: Write a program for Arithmetic operations in binary and BCD-addition, subtraction, multiplication and division and display.

AIM

Write a program for Arithmetic operations in binary and BCD-addition, subtraction, multiplication and division and display.

THEORY

A decimal number contains 10 digits (0-9). Now the equivalent binary numbers can be found out of these 10 decimal numbers. In case of **BCD** the binary number formed by four binary digits, will be the equivalent code for the given decimal digits. In **BCD** we can use the binary number from 0000-1001 only, which are the decimal equivalent from 0-9 respectively. Suppose if a number have single decimal digit then it's equivalent **Binary Coded Decimal** will be the respective four binary digits of that decimal number and if the number contains two decimal digits then it's equivalent **BCD** will be the respective eight binary of the given decimal number. Four for the first decimal digit and next four for the second decimal digit. It may be cleared from an example. Let, (12)10 be the decimal number whose equivalent **Binary coded decimal** will be 00010010. Four bits from L.S.B is binary equivalent of 2 and next four is the binary equivalent of 1.

Table given below shows the binary and BCD codes for the decimal numbers 0 to 15. From the table below, we can conclude that after 9 the decimal equivalent binary number is of four bit but in case of BCD it is an eight bit number. This is the main difference between Binary number and binary coded decimal. For 0 to 9 decimal numbers both binary and BCD is equal but when decimal number is more than one bit BCD differs from binary.

Decimal number	Binary number	Binary Coded Decimal(BCD)
0	0000	0000
1	0001	0001
2	0010	0010
3	0011	0011
4	0100	0100
5	0101	0101
6	0110	0110

7	0111	0111
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

BCD Addition

Like other number system in BCD arithmetical operation may be required. BCD is a numerical code which has several rules for addition. The rules are given below in three steps with an example to make the idea of **BCD Addition** clear.

1. At first the given number are to be added using the rule of binary. For example,

Case 1:

$$\begin{array}{r} 1010 \\ +\ 0101 \\ \hline 1111 \end{array}$$

Case 2:

$$\begin{array}{r} 0001 \\ +\ 0101 \\ \hline 0110 \end{array}$$

2. In second step we have to judge the result of addition. Here two cases are shown to describe the rules of **BCD Addition**. In case 1 the result of addition of two binary number is greater than 9, which is not valid for BCD number. But the result of addition in case 2 is less than 9, which is valid for BCD numbers.
3. If the four bit result of addition is greater than 9 and if a carry bit is present in the result then it is invalid and we have to add 6 whose binary equivalent is $(0110)_2$ to the result of addition. Then the resultant that we would get will be a valid binary coded number. In case 1 the result was $(1111)_2$, which is greater than 9 so we have to add 6 or $(0110)_2$ to it.

$$(1111)_2 + (0110)_2 = 0001\ 0101 = 15$$

As you can see the result is valid in BCD.

But in case 2 the result was already valid BCD, so there is no need to add 6. This is how BCD Addition could be.

Now a question may arrive that why 6 is being added to the addition result in case BCD Addition instead of any other numbers. It is done to skip the six invalid states of binary coded decimal i.e from 10 to 15 and again return to the BCD codes.

Now the idea of BCD Addition can be cleared from two more examples.

Example:1

Let, 0101 is added with 0110.

```
   0101
 + 0110
   1011  → Invalid BCD number
 + 0110  → Add 6
---------
0001 0001 → Valid BCD number
```

Check your self.

$(0101)_2 \rightarrow (5)_{10}$ *and* $(0110)_2 \rightarrow (6)_{10}(5)_{10} + (6)_{10} = (11)_{10}$

Example:2

Now let 0001 0011 is added to 0010 0110.

```
  0001 0001
+ 0010 0110
-----------
  0011 0111  ⟶ Valid BCD number
```

$(0001\ 0001)_{BCD} \rightarrow (11)_{10}$, $(0010\ 0110)_{BCD} \rightarrow (26)_{10}$ *and* $(0011\ 0111)_{BCD} \rightarrow (37)_{10}(11)_{10} + (26)_{10} = (37)_{10}$

So no need to add 6 as because both $(0011)_2 = (3)_{10}$ *and* $(0111)_2 = (7)_{10}$ are less than $(9)_{10}$. This is the process of BCD Addition.

BCD Subtraction

There are several methods of **BCD Subtraction**. BCD subtraction can be done by 1's compliment method and 9's compliment method or 10's compliment method. Among all these methods 9's compliment method or 10's compliment method is the most easiest. We will clear our idea on both the methods of **BCD Subtraction**.

In 1st method we will do **BCD Subtraction** by 1"s compliment method. There are several steps for this method shown below. They are:-

1. At first 1"s compliment of the subtrahend is done.
2. Then the complimented subtrahend is added to the other number from which the subtraction is to be done. This is called adder 1.
3. Now in BCD Subtraction there is a term „EAC(end-around-carry)". If there is a carry i.e if EAC = 1 the result of the subtraction is +ve and if EAC = 0 then the result is –ve. A table shown below gives the rules of EAC.

carry of individual groups	EAC = 1	EAC = 0
1	Transfer real result of adder 1 and add 0000 in adder 2	Transfer 1"s compliment result of adder 1 and add 1010 in adder 2
0	Transfer real result of adder 1 and add 1010 in adder 2	Transfer 1"s compliment result of adder 1 and add 0000 to adder 2

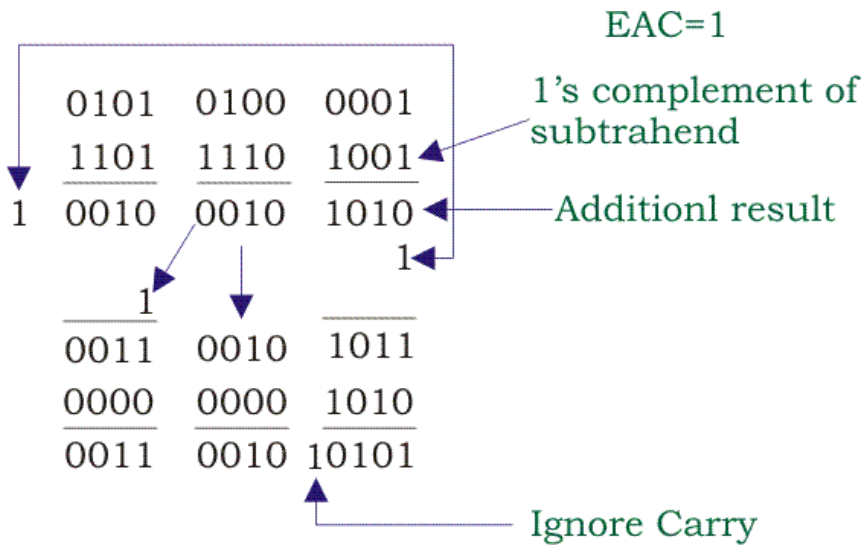
4. In the final result if any carry bit occurs the it will be ignored.

Examples given below would make the idea clear of BCD Subtraction.

Example: – 1

In this example 0010 0001 0110 is subtracted from 0101 0100 0001.

- At first 1"s compliment of the subtrahend is done, which is 1101 1110 1001 and is added to 0101 0100 0001. This step is called adder 1.
- Now after addition if any carry occurs then it will be added to the next group of numbers towards MSB. Then EAC will be examined. Here, EAC = 1. So the result of addition is positive and true result of adder 1 will be transferred to adder 2.
- Now notice from LSB. There are three groups of four bit numbers. 1010 is added 1011 which is the first group of numbers because it do not have any carry. The result of the addition is the final answer.
- Carry 1 will be ignored as it is from the rule.
- Now move to the next group of numbers. 0000 is added to 0010 and gives the result 0010. It is the final result again.
- Now again move to the next group here 0000 is also added to 0011 to give the final result 0011.
- You may have noticed that in this two groups 0000 is added, because result of first adder do not contain any carry. Thus the results of the adder 2 is the final result of BCD Subtraction.



Therefore,

$$(0101\ 0100\ 0001) - (0010\ 0001\ 0110) = (0011\ 0010\ 0101)$$

Now you can check yourself.

$$(0101\ 0100\ 0001) = (541)_{10}$$

$$(0010\ 0001\ 0110) = 216_{10}$$

$$(0011\ 0010\ 0101) = 325_{10}$$

We know that 541 − 216 = 325, Thus we can say that our result of **BCD Subtraction** is correct.

PROGRAM

When it comes to adding BCD numbers in microcontrollers, normal binary addition will give the wrong results. So to solve this issue, microcontrollers use the DA command, which converts the binary results to BCD. The DA command can take only one operand; A.

Example

```
MOV A,#47H; A=47H first BCD operand(0100 0111 BCD)
MOV B, #25H ;B=25 second BCD operand(0010 0101 BCD)
ADD A,B ;hex addition (A=6CH)
DA A; adjusts for BCD addition (A=72H)
```

Program

```
MOV R0, #20H;set source address 20H to R0
MOV R1, #30H;set destination address 30H to R1
MOV A, @R0;take the first operand from source to register A
INC R0; Point to the next location
MOV B, @R0; take the second operand from source to register B
DIV AB ; Divide A by B
MOV @R1, A; Store Quotient to 30H
INC R1; Increase R1 to point to the next location
MOV @R1, B; Store Remainder to 31H
HALT:  SJMP HALT ;Stop the program
```

8051 provides **DI VAB** instruction. By using this instruction, the division can be done. In some other microprocessors like 8085, there was no DIV instruction. In that microprocessor, we need to use repetitive Subtraction operations to get the result of the division.

When the denominator is00H, the overflow flag OV will be 1. otherwise it is 0 for the division.

Output

Address	Value
20H	0EH

Address	Value
21H	03H
30H	04H
31H	02H

Packed BCD

In packed BCD, a single byte has two BCD numbers in it, one in the lower 4 bits, and one in the upper 4 bits. For example, "0101 1001″ is packed BCD for 59H. It takes only 1 byte of memory to store the packed BCD operands. And so one reason to use packed BCD is that it is twice as efficient in storing data.

There is a problem with adding BCD numbers, which must be corrected. The problem is that after adding packed BCD numbers, the result is no longer BCD. Look at the following.

```
MOV A,#17H
ADD A,#28H
```

Adding these two numbers gives 0011 111 IB (3FH), which is not BCD! A BCD number can only have digits from 0000 to 1001 (or 0 to 9). In other words, adding two BCD numbers must give a BCD result. The result above should have been 17 + 28 = 45 (0100 0101). To correct this problem, the programmer must add 6 (0110) to the low digit: 3F + 06 = 45H. The same problem could have happened in the upper digit (for example, in 52H + 87H = D9H). Again to solve this problem, 6 must be added to the upper digit (D9H + 60H = 139H) to ensure that the result is BCD (52 + 87 = 139). This problem is so pervasive that most microprocessors such as the 8051 have an instruction to deal with it. In the 8051 the instruction "DA A" is designed/to correct the BCD addition problem. This is discussed next.

DA instruction

The DA (decimal adjust for addition) instruction in the 8051 is provided to correct the aforementioned problem associated with BCD addition. The mnemonic "DA" has as its only operand the accumulator "A". The DA instruction will add 6 to the lower nibble or higher nibble if needed; otherwise, it will leave the result alone. The following example will clarify these points.

```
MOV  A,#47H    ;A=47H first BCD operand
MOV  B,#25H    ;B=25 second BCD operand
ADD  A,B       ;hex(binary) addition (A=6CH)
DA   A         ;adjust for BCD addition (A=72H)
```

After the program is executed, register A wili contain 72H (47 + 25 = 72). The "DA" instruction works only on A. In other words, while the source can be an operand of any addressing mode, the destination must be in register A in order for DA to work. It also needs to be emphasized that DA must be used after the addition of BCD operands and that BCD operands can never have any digit greater than 9. In other words, A – F digits are not allowed. It is also important to note that DA works only after an ADD instruction; it will not work after the INC instruction.

Summary of DA action

After an ADD or ADDC instruction,

1. If the lower nibble (4 bits) is greater than 9, or if AC = 1, add 0110 to the lower 4 bits.
1. If the upper nibble is greater than 9, or if CY = 1, add 0110 to the upper 4 bits.

In reality there is no other use for the AC (auxiliary carry) flag bit except for BCD addition and correction. For example, adding 29H and 18H will result in 41H, which is incorrect as far as BCD is concerned.

```
     Hex          BCD
     29           0010 1001
+    18      +    0001 1000
     41           0100 0001      AC=1
+     6      +         0110
     47           0100 0111
```

Since AC = 1 after the addition, "DA A" will add 6 to the lower nibble. The final result is in BCD format.

Example 6-4

Assume that 5 BCD data items are stored in RAM locations starting at 40H, as shown below. Write a program to find the sum of all the numbers. The result must be in BCD.

40= (71) 41=(11) 42=(65) 43=(59) 44=(37)

Solution:

```
          MOV  R0,#40H    ;load pointer
          MOV  R2,#5      ;load counter
          CLR  A          ;A=0
          MOV  R7,A       ;clear R7
AGAIN:    ADD  A,@R0      ;add the byte pointer to A by R0
          DA   A          ;adjust for BCD
          JNC  NEXT       ;if CY=0 don't accumulate carry
          INC  R7         ;keep track of carries
NEXT:     INC  R0         ;increment pointer
          DJNZ R2,AGAIN   ;repeat until R2 is zero
```

Subtraction of unsigned numbers

```
SUBB A, source ;A = A - source - CY
```

In many microprocessors there a^e two different instructions for subtraction: SUB and SUBB (subtract with borrow). In the 8051 we have only SUBB. To make SUB out of SUBB, we have to

make CY = 0 prior to the execution of the instruction. Therefore, there are two cases for the SUBB instruction: (1) with CY = 0, and (2) with CY = 1. First we examine the case where CY = 0 prior to the execution of SUBB. Notice that we use the CY flag for the borrow.

SUBB (subtract with borrow) when CY=0

In subtraction, the 8051 microprocessors (indeed, all modern CPUs) use the 2′s complement method. Although every CPU contains adder circuitry, it would be too cumbersome (and take too many transistors) to design separate subtracter circuitry. For this reason, the 8051 uses adder circuitry to perform the subtraction command. Assuming that the 8051 is executing a simple subtract instruction and that CY = 0 prior to the execution of the instruction, one can summarize the steps of the hardware of the CPU in executing the SUBB instruction for unsigned numbers, as follows.

1. Take the 2′s complement of the subtrahend (source operand).
2. Add it to the minuend (A).
3. Invert the carry.

These three steps are performed for every SUBB instruction by the internal hardware of the 8051 CPU, regardless of the source of the operands, provided that the addressing mode is supported. After these three steps the result is obtained and the flags are set. Example 6-5 illustrates the three steps.

Example 6-5

```
CLR  C          ;make CY=0
MOV  A,#3FH     ;load 3FH into A (A = 3FH)
MOV  R3,#23H    ;load 23H into R3 (R3 = 23H)
SUBB A,R3       ;subtract A - R3, place result in A
```

Solution:

```
A  =  3F   0011 1111      0011 1111
R3 =  23   0010 0011    + 1101 1101 (2's complement)
      1C                1 0001 1100
                        0 CF=0 (step 3)
```

The flags would be set as follows: CY = 0, AC – 0, and the programmer must look at the carry flag to determine if the result is positive or negative.

Show the steps involved in the following.

If the C Y = 0 after the execution of SUBB, the result is positive; if C Y = 1, the result is negative and the destination has the 2′s complement of the result. Normally, the result is left in 2′s

complement, but the CPL (complement) and INC instructions can be used to change it. The CPL instruction performs the 1 „s complement of the operand; then the operand is incremented (INC) to get the 2′s complement. See Example 6-6.

Example 6-6

Analyze the following program:

```
      CLR   C
      MOV   A,#4CH      ;load A with value 4CH (A=4CH)
      SUBB  A,#6EH      ;subtract 6E from A
      JNC   NEXT        ;if CY=0 jump to NEXT target
      CPL   A           ;if CY=1 then take 1's complement
      INC   A           ;and increment to get 2's complement
NEXT: MOV   R1,A        ;save A in R1
```

Solution:

Following are the steps for "SUBB A, #6EH":

```
      4C      0100 1100                0100 1100
   -  6E      0110 1110   2's comp =   1001 0010
     -22                              0 1101 1110
CY = 1, the result is negative, in 2's complement.
```

SUBB (subtract with borrow) when CY= 1

This instruction is used for multibyte numbers and will take care of the borrow of the lower operand. If CY = 1 prior to executing the SUBB instruction, it also subtracts 1 from the result. See Example 6-7.

Example 6-7

Analyze the following program:

```
      CLR   C               ;CY = 0
      MOV   A,#62H          ;A = 62H
      SUBB  A,#96H          ;62H - 96H = CCH with CY = 1
      MOV   R7,A            ;save the result
      MOV   A,#27H          ;A=27H
      SUBB  A,#12H          ;27H - 12H - 1 = 14H
      MOV   R6,A            ;save the result
```

Solution:

After the SUBB, A = 62H – 96H = CCH and the carry flag is set high indicating there is a borrow. Since CY = 1, when SUBB is executed the second time A = 27H – 12H -1 = 14H. Therefore, we have 2762H – 1296H = 14CCH.

UNSIGNED MULTIPLICATION AND DIVISION

In multiplying or dividing two numbers in the 8051, the use of registers A and B is required since the multiplication and division instructions work only with these two registers. We first discuss multiplication.

Multiplication of unsigned numbers

The 8051 supports byte-by-byte multiplication only. The bytes are assumed to be unsigned data. The syntax is as follows:

```
MUL AB    ;A x B, place 16-bit result in B and A
```

In byte-by-byte multiplication, one of the operands must be in register A, and the second operand must be in register B. After multiplication, the result is in the A and B registers; the lower byte is in A, and the upper byte is in B. The following example multiplies 25H by 65H. The result is a 16-bit data that is held by the A and B registers.

```
MOV  A,#25H     ;load 25H to reg. A
MOV  B,#65H     ;load 65H in reg. B
MUL  AB         ;25H * 65H = E99 where
                ;B = 0EH and A = 99H
```

Multiplication	Operand 1	Operand 2	Result
byte × byte	A	B	A = low byte, B = high byte

Unsigned Multiplication Summary (MUL AB)

Note: Multiplication of operands larger than 8 bits takes some manipulation. It is left to the reader to experiment with.

Division of unsigned numbers

In the division of unsigned numbers, the 8051 supports byte over byte only. The syntax is as follows.

```
DIV AB    ;divide A by B
```

When dividing a byte by a byte, the numerator must be in register A and the denominator must be in B. After the DIV instruction is performed, the quotient is in A and the remainder is in B. See the following example.

```
MOV  A,#95      ;load 95 into A
MOV  B,#10      ;load 10 into B
DIV  AB         ;now A = 09 (quotient) and
                ;B = 05(remainder)
```

Notice the following points for instruction “DIV AB”.

1. This instruction always makes CY = 0 and OV = 0 if the denominator is not 0.
   1. If the denominator is 0 (B = 0), OV = 1 indicates an error, and CY = 0. The standard practice in all microprocessors when dividing a number by 0 is to indicate in some way the invalid result of infinity. In the 8051, the OV flag is set to 1.

Unsigned Division Summary (DIV AB)

Division	Numerator	Denominator	Quotient	Remainder
byte / byte	A	B	A	B

(If B = 0, then OV = 1 indicating an error)

An application for DIV instructions

There are times when an ADC (analog-to-digital converter) is connected to a port and the ADC represents some quantity such as temperature or pressure. The 8-bit ADC provides data in hex in the range of 00 – FFH. This hex data must be converted to decimal. We do that by dividing it by 10 repeatedly, saving the remainders as shown in Example 6-8.

Example 6-8

Write a program (a) to make PI an input port, (b) to get a byte of hex data in the range of 00 – FFH from PI and convert it to decimal. Save the digits in R7, R6, and R5, where the least significant digit is in R7.

Solution:

```
MOV  A,#0FFH
MOV  P1,A       ;make P1 an input port
MOV  A,P1       ;read data from P1
MOV  B,#10      ;B=0A hex (10 dec)
DIV  AB         ;divide by 10
MOV  R7,B       ;save lower digit
MOV  B,#10      ;
DIV  AB         ;divide by 10 once more
MOV  R6,B       ;save the next digit
MOV  R5,A       ;save the last digit
```

The input value from PI is in the hex range of 00 – FFH or in binary 00000000 to 11111111. This program will not work if the input data is in BCD. In other words, this program converts from binary to decimal. To convert a single decimal digit to ASCII format, we OR it with 30H as shown in Sections 6.4 and 6.5.

Example 6-9

Analyze the program in Example 6-8, assuming that PI has a value of FDH for data. **Solution:**

To convert a binary (hex) value to decimal, we divide it by 10 repeatedly until the quotient is less than 10. After each division the remainder is saved. In the case of an 8-bit binary such as FDH we have 253 decimal as shown below (all in hex).

```
            Quotient   Remainder
FD/0A=      19         3 (low digit)
19/0A=      2          5 (middle digit)
                       2 (high digit)
```

Therefore, we have FDH = 253. In order to display this data it must be converted to ASCII, which is described in a later section in this chapter.

RESULT

Successfully study the programming Arithmetic operations in binary and BCD-addition, subtraction, multiplication and division in microcontroller kit.

VIVA QUESTIONS:

1. Can you explain the difference between binary and BCD (Binary Coded Decimal) representation?
2. How do you convert a decimal number into its binary equivalent?
3. What are the basic arithmetic operations involved in binary addition?
4. Could you outline the algorithm for performing binary addition in a program?
5. How do you handle carry-over in binary addition?
6. What are the key considerations when implementing binary subtraction in a program?
7. How is binary subtraction different from binary addition algorithmically?
8. Can you discuss the process of performing binary multiplication?
9. What are the key challenges in binary multiplication, especially when implemented in a program?
10. How do you handle overflow in binary multiplication?
11. Explain the steps involved in binary division.
12. What are some key differences between binary division and other arithmetic operations?
13. How do you handle remainders in binary division?
14. What is the significance of BCD representation in arithmetic operations?
15. Can you discuss any potential limitations or drawbacks of using BCD in arithmetic operations compared to binary representation?

DEPARTMENT OF ELECTRICAL ENGINEERING

Embedded System

Experiment No: 4

Interfacing of 8 bit DAC 0800 with 89C51 Microcontroller.

TITLE: Interfacing of 8 bit DAC 0800 with 80C51 micro-controller.

Interfacing of 8 bit DAC 0800 with 80C51 micro-controller.

APPRATUS

1) 89C51 Kit.
2) DAC interfacing cord.
3) CRO.
4) 26 pin FRC cable.
5) Kit power supply.
6) Dual power supply with +/- 12volt/ 200mA.
7) Software.

DEFINATION

The digital to analog converter (DAC) is a device widely used to convert digital pulses to analog signals.

THEORY

Digital to analog converter

Digital to analog converter using 150ns DAC 0808/1408. OPAMP is used to convert current output from DAC to voltage output & as an amplifier.

DAC 0808 a monolithic DAC featuring full scale output current setting time of 150 ns which dissipating only 38-40MW with 5Volt D.C. supply. No reference current timing is required for most of the applications.

Triangular Wave Form:-

If 8bit binary number is connected to DAC input & incremented from 0 to 255 steps continuously & again decremented up to zero. This process is repeated. Then you will get triangular wave consisting of 256 with positive and negative.

Square Wave:-

A square wave can be generated by sending FFH that is high signal to DAC. Then apply some delay & afterwords send OOH , some delay with repeatability. We can form square wave.

Interfacing Diagram Description:-

Fig. 1 shows interface of DAC 0809 with 89C51. DAC 0800 is an 8-bit DAC. The output of 89C51 is a digital signal which may further converted into analog signal using DAC. The

output of DAC is a current & it is necessary to convert this into voltage signal in some cases. Therefore, current to voltage converter is needed.

In fig 1 operational amplifier base current to voltage converter is shown. It is only necessary to output digital data byte & analog output will be available till the time digital data at input of DAC remains the same. Port data of 8051are internally latched & this makes again the DAC interfacing very simple.

Connector details:-

For interfacing of LED logic, port O, port 2,port 3 of microcontroller has been used. On the board of LED logic 26 pin FRC connector is provided for connection purpose. The details of this connector are given to understand each & every pin connection of 8051 part although it is for reference.

PIN NUMBERS	DETAILS
01	PC.4
02	PC.5
03	PC.2
04	PC.3
05	PC.0
06	PC.1
07	PB.6
08	PB.7
09	PB.4
10	PB.5
11	PB.2
12	PB.3
13	PB.0
14	PB.1
15	PA.6
16	PA.7
17	PA.4
18	PA.5
19	PA.2
20	PA.3

21	PA.0
22	PA.1
23	PC.6
24	PC.7
25	GND
26	Vcc

PROGRAM

Square Wave

Memory address	Opcode	Label	Mnemonic	Comment
9400			ORG 9400H	
9400	75 81 90		MOV SP,# 90H	
9403	79 00		MOV R1, # 00HA	
9405	90 94 2D		MOV DPTR, # LINE 1	Initialize DPTR
9408	12 00 5A		LCALL MSG OUT	
940B	90 00 03		MOV DPTR, # CW 55	
940E	74 80		MOV A, # 80H	Copy content of port 0 to A
9410	F0		MOVX @DPTR, A	Copy content of A to DPTR
9411	74 FF	LOOP	MOV A, # OFFH	
9413	90 00 00		MOV DPTR, # PORTA	
9416	F0		MOVX @ DPTR,A	
9417	90 00 01		MOV DPTR, # PORTB	
941A	FO		MOVX @DPTR, A	
941B	14	HERE	DEC A	
941C	B4 00 FC		CJNE A, # 00H, HERE	Compare A with 00 H and jump to

				LOOP if not equal
941F	90 00 00	LOOP 1	MOV DPTR, #PORTA	
9422	F0		MOVX @DPTR,A	
9423	90 00 01		MOV DPTR, # PORTB	
9426	F0		MOVX @DPTR,A	
9427	04	HERE 1	INC A	
9428	B4 FF FC		CJNE A, # OFFH, HERE 1	Compare A with FF H and jump to LOOP if not equal
942B	81 11		AJMP LOOP	Absolute jump to LOOP
942D	3A 53 51 55	LINE1	DB	
9431	41 52 45 20		SQUARE WAVE	
9435	57 41 56 45		03H	
9439	03			

Triangular Wave

Memory address	**Opcode**	**Label**	**Mnemonic**	**Comment**
9400			ORG 9400H	
9400	75 81 90		MOV SP,# 90H	
9403	79 00		MOV R1, # 00HA	
9405	90 94 35		MOV DPTR, # LINE 1	Initialize DPTR
9408	12 00 5A		LCALL MSG OUT	
940B	90 00 03		MOV DPTR, # CW 55	
940E	74 80		MOV A, # 80H	Copy content of port 0 to A
9410	F0		MOVX @DPTR, A	Copy content of A to DPTR
9411	75 35 FF		MOV MN, # OFFH	
9414	15 35	LOOP	DEC MN	
9416	90 00 00		MOV DPTR # PORTA	

9419	E5 35		MOV A, MN	
941B	FO		MOVX @DPTR, A	
941C	90 00 01		MOV DPTR, # PORTB	
941F	F0		MOVX @DPTR,A	
9420	B4 00 F1		CJNE A, # OFFH, LOOP	Compare A with FF H and jump to LOOP if not equal
9423	05 35	LOOP1:	INC MN	
9425	E5 35		MOV A, MN	
9427	90 00 00		MOV DPTR, # PORT A	
942A	F0		MOVX @DPTR,A	
942B	90 00 01		MOV DPTR, # PORTB	
942E	F0		MOVX @DPTR,A	
942F	B4 FF F1		CJNE A, # OFFH, LOOP1	Compare A with FF H and jump to LOOP1 if not equal
9432	02 94 14		LJMP LOOP	
9435	54 52 41 49	LINE1:	DB TRIANGULAR WAVE 03H	
9439	4E 47 55 4C			
943D	41 52 20 57			
9441	41 56 45 03			

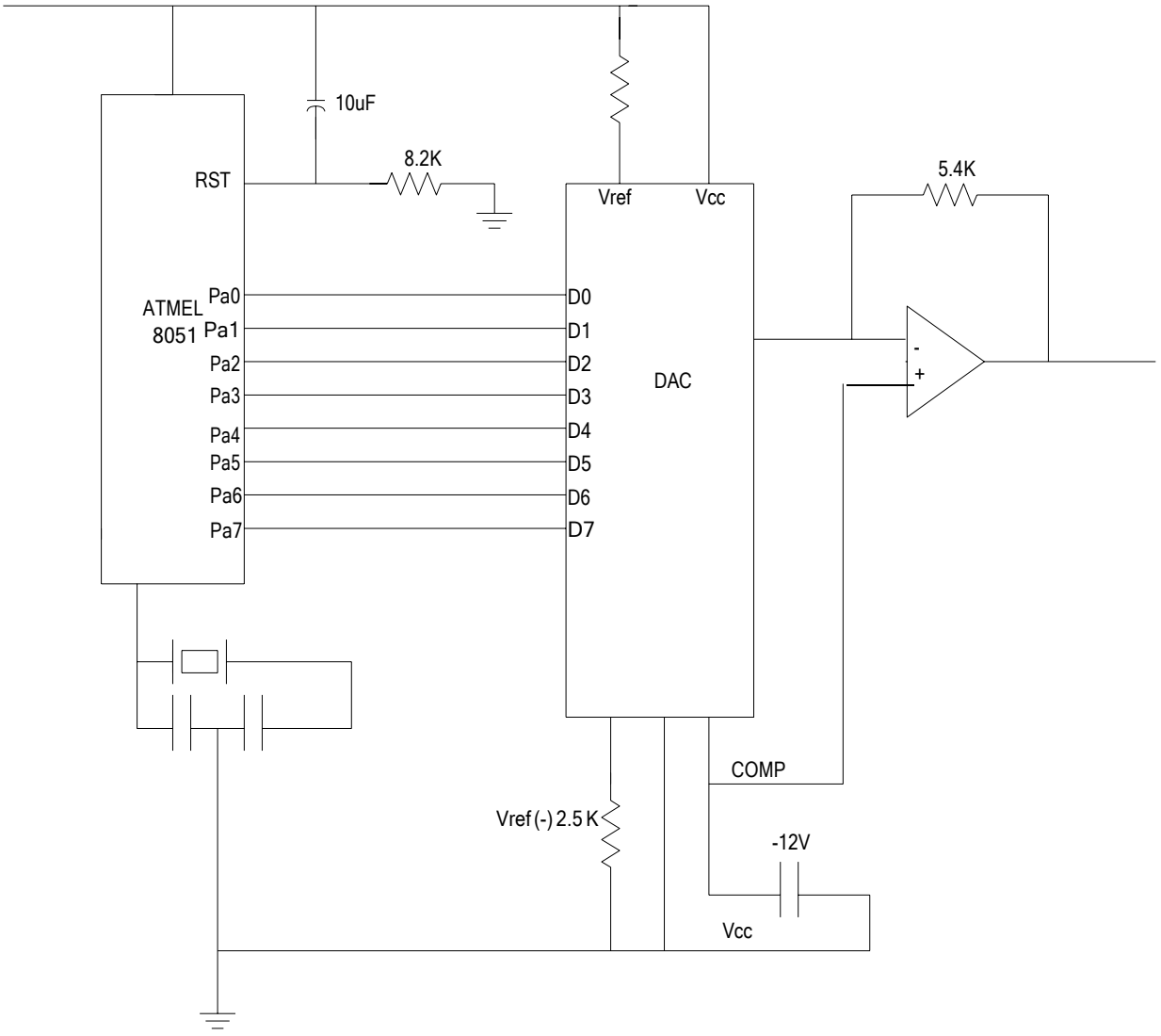
VIVA QUESTIONS:

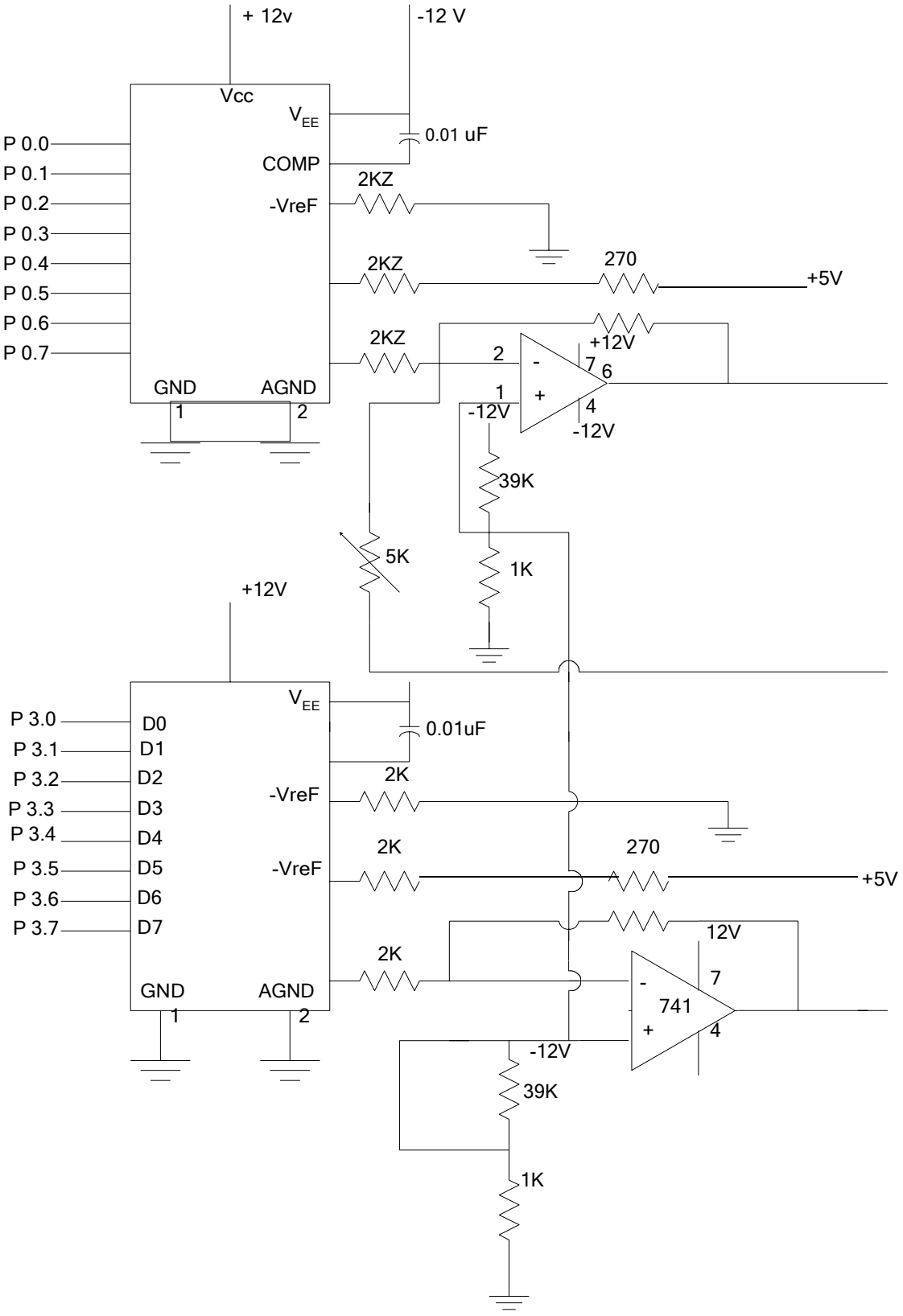
1. What is the purpose of interfacing a DAC (Digital-to-Analog Converter) with a microcontroller?
2. Can you explain the basic working principle of the DAC 0800?
3. How does the 89C51 Microcontroller communicate with the DAC 0800?
4. What are the key features of the 89C51 Microcontroller that make it suitable for interfacing with external devices like a DAC?
5. Explain the significance of the 8-bit resolution of the DAC 0800.
6. What is the role of the DAC 0800 in converting digital signals from the microcontroller into analog voltage levels?
7. How do you configure the 89C51 Microcontroller to send digital data to the DAC 0800?
8. Can you describe the hardware connections required for interfacing the 89C51 Microcontroller with the DAC 0800?

9. What is the maximum output voltage range of the DAC 0800, and how is it determined?
10. How do you calibrate the output voltage of the DAC 0800 to achieve precise analog voltage levels?
11. What are the potential applications of interfacing a DAC with a microcontroller in real-world scenarios?
12. Discuss any potential challenges or limitations associated with interfacing the DAC 0800 with the 89C51 Microcontroller.
13. How do you verify the proper functionality of the interfaced system in the experiment?
14. What are the advantages of using an 8-bit DAC over a DAC with lower or higher resolution?
15. Can you suggest any improvements or enhancements to the experimental setup for better performance or functionality?

■■■

DIAGRAM


10uF
8.2K
RST
Vref
Vcc
5.4K
ATMEL
8051
Pa0
Pa1
Pa2
Pa3
Pa4
Pa5
Pa6
Pa7
D0
D1
D2
D3
D4
D5
D6
D7
DAC
-
+
COMP
Vref (-) 2.5 K
-12V
Vcc


+ 12v
-12 V
Vcc
V_{EE}
0.01 uF
COMP
2KZ
-VreF
P 0.0
P 0.1
P 0.2
P 0.3
P 0.4
P 0.5
P 0.6
P 0.7
2KZ
270
+5V
2KZ
+12V
2
7
6
1
4
-12V
-12V
GND
AGND
1
2
39K
5K
1K
+12V
V_{EE}
P 3.0
P 3.1
P 3.2
P 3.3
P 3.4
P 3.5
P 3.6
P 3.7
D0
D1
D2
D3
D4
D5
D6
D7
0.01uF
2K
-VreF
2K
270
-VreF
+5V
12V
2K
7
741
4
GND
AGND
1
2
-12V
39K
1K

DEPARTMENT OF ELECTRICAL ENGINEERING

Embedded System Lab

Experiment No: 5

Write a program to interfacing IR sensor to realize obstacle detector

TITLE: Write a program to interfacing IR sensor to realize obstacle detector

AIM

Write a program to interfacing IR sensor to realize obstacle detector.

APPRATUS

1) Ardunio kit
2) DSO
3) Variable Resistor (10K)
4) Resistor
5) IR Sensor
6) Kit power supply or Dual power supply with +/- 12volt/ 200mA.
7) Proteus Software
8) Connecting code and DSO Probe

THEORY

A) Introduction

Infrared technology addresses a wide variety of wireless applications. The main areas are sensing and remote controls. In the electromagnetic spectrum, the infrared portion is divided into three regions: near infrared region, mid infrared region and far infrared region.The wavelengths of these regions and their applications are shown below.

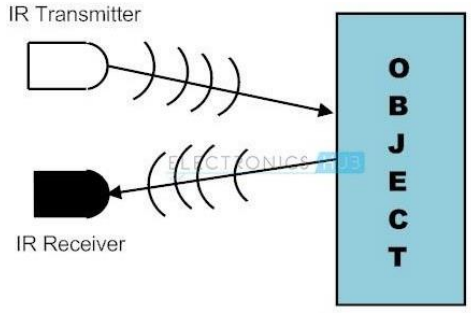
- Near infrared region — 700 nm to 1400 nm — IR sensors, fiber optic
- Mid infrared region — 1400 nm to 3000 nm — Heat sensing
- Far infrared region — 3000 nm to 1 mm — Thermal imaging

The frequency range of infrared is higher than microwave and lesser than visible light. For optical sensing and optical communication, photo optics technologies are used in the near infrared region as the light is less complex than RF when implemented as a source of signal. Optical wireless communication is done with IR data transmission for short range applications. An infrared sensor emits and/or detects infrared radiation to sense its surroundings. The working of any Infrared sensor is governed by three laws: Planck's Radiation law, Stephen – Boltzmann law and Wien's Displacement law. Planck's law states that "every object emits radiation at a temperature not equal to 00K". Stephen – Boltzmann law states that "at all wavelengths, the total energy emitted by a black body is proportional to the fourth power of the absolute temperature". According to Wien's Displacement law, "the radiation curve of a black body for different temperatures will reach its peak at a wavelength inversely proportional to the temperature".

The basic concept of an Infrared Sensor which is used as Obstacle detector is to transmit an infrared signal, this infrared signal bounces from the surface of an object and the signal is received at the infrared receiver.

B) Types of IR Sensors

Infrared sensors can be passive or active. Passive infrared sensors are basically Infrared detectors. Passive infrared sensors do not use any infrared source and detects energy emitted by obstacles in the field of view. They are of two types: quantum and thermal. Thermal infrared sensors use infrared energy as the source of heat and are independent of wavelength. Thermocouples, pyroelectric detectors and bolometers are the common types of thermal infrared detectors.
Quantum type infrared detectors offer higher detection performance and are faster than thermal type infrared detectors. The photosensitivity of quantum type detectors is wavelength dependent. Quantum type detectors are further classified into two types: intrinsic and extrinsic types. Intrinsic type quantum detectors are photoconductive cells and photovoltaic cells.
Active infrared sensors consist of two elements: infrared source and infrared detector. Infrared sources include an LED or infrared laser diode. Infrared detectors include photodiodes or phototransistors. The energy emitted by the infrared source is reflected by an object and falls on the infrared detector.



C) IR Transmitter

Infrared Transmitter is a light emitting diode (LED) which emits infrared radiations. Hence, they are called IR LED's. Even though an IR LED looks like a normal LED, the radiation emitted by it is invisible to the human eye.
The picture of a typical Infrared LED is shown below.

There are different types of infrared transmitters depending on their wavelengths, output power and response time. A simple infrared transmitter can be constructed using an infrared LED, a current limiting resistor and a power supply. The schematic of a typical IR transmitter is shown below.

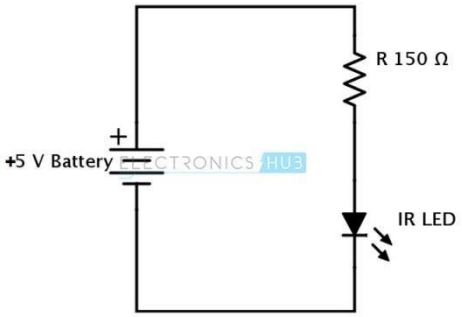


Figure 1: IR Transmitter circuit

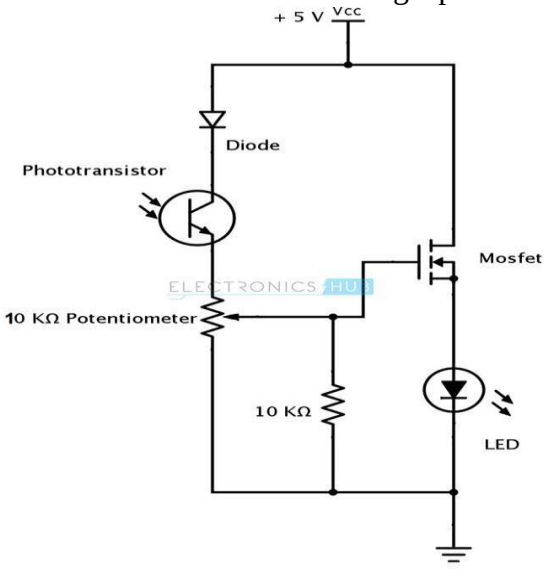
When operated at a supply of 5V, the IR transmitter consumes about 3 to 5 mA of current. Infrared transmitters can be modulated to produce a particular frequency of infrared light. The most commonly used modulation is OOK (ON – OFF – KEYING) modulation.

IR transmitters can be found in several applications. Some applications require infrared heat and the best infrared source is infrared transmitter. When infrared emitters are used with Quartz, solar cells can be made.

D) IR Receiver

Infrared receivers are also called as infrared sensors as they detect the radiation from an IR transmitter. IR receivers come in the form of photodiodes and phototransistors. Infrared Photodiodes are different from normal photo diodes as they detect only infrared radiation. The picture of a typical IR receiver or a photodiode is shown below.

Different types of IR receivers exist based on the wavelength, voltage, package, etc. When used in an infrared transmitter – receiver combination, the wavelength of the receiver should match with that of the transmitter.A typical infrared receiver circuit using a phototransistor is shown below.

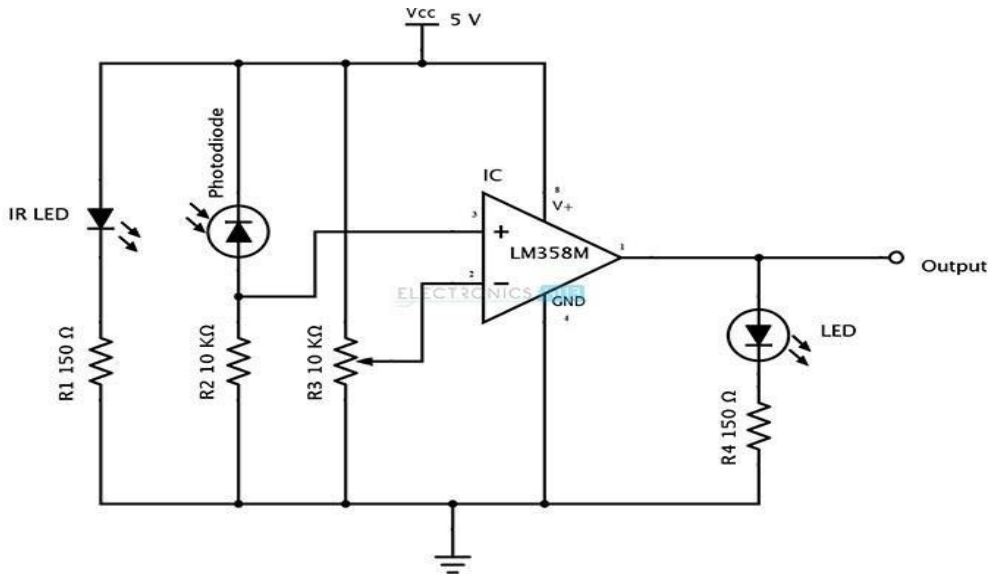


It consists of an IR phototransistor, a diode, a MOSFET, a potentiometer and an LED. When the phototransistor receives any infrared radiation, current flows through it and MOSFET turns on. This in turn lights up the LED which acts as a load. The potentiometer is used to control the sensitivity of the phototransistor.

WORKING PRINCIPLE

The principle of an IR sensor working as an Object Detection Sensor can be explained using the following figure. An IR sensor consists of an IR LED and an IR Photodiode; together they are called as Photo – Coupler or Opto – Coupler. When the IR transmitter emits radiation, it reaches the object and some of the radiation reflects back to the IR receiver. Based on the intensity of the reception by the IR receiver, the output of the sensor is defined.

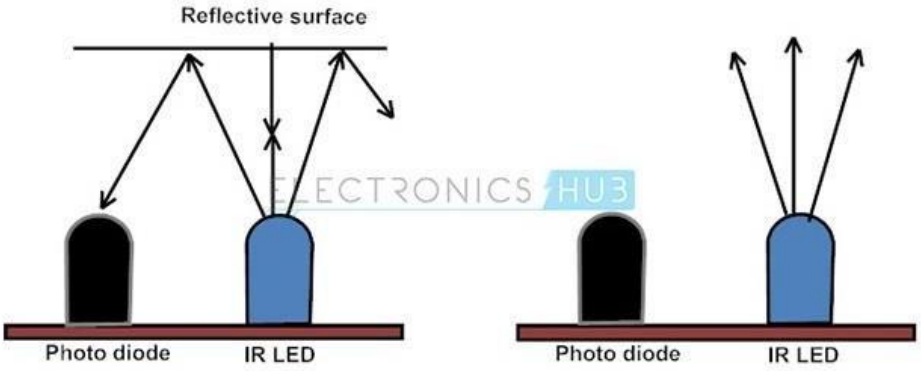
Obstacle Sensing Circuit or IR Sensor Circuit



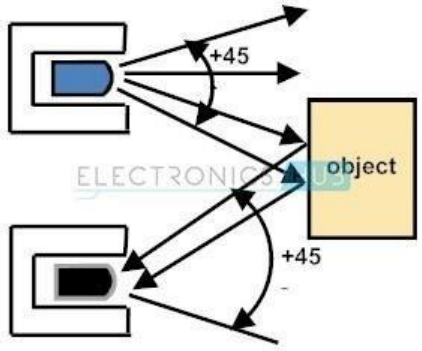
It consists of an IR LED, a photodiode, a potentiometer, an IC Operational amplifier and an LED. IR LED emits infrared light. The Photodiode detects the infrared light. An IC Op – Amp is used as a voltage comparator. The potentiometer is used to calibrate the output of the sensor according to the requirement. When the light emitted by the IR LED is incident on the photodiode after hitting an object, the resistance of the photodiode falls down from a huge value. One of the input of the op – amp is at threshold value set by the potentiometer. The other input to the op-amp is from the photodiode's series resistor. When the incident radiation is more on the photodiode, the voltage drop across the series resistor will be high. In the IC, both the threshold voltage and the voltage across the series resistor are compared. If the voltage across the resistor series to photodiode is greater than that of the threshold voltage, the output of the IC Op – Amp is high. As the output of the IC is connected to an LED, it lightens up. The threshold voltage can be adjusted by adjusting the potentiometer depending on the environmental conditions. The positioning of the IR LED and the IR Receiver is an important factor. When the IR LED is held directly in front of the IR receiver, this setup is called Direct Incidence. In this case, almost the entire radiation from the IR LED will fall on the IR receiver. Hence there is a line of sight communication between the infrared transmitter and the receiver. If an object falls in this line, it obstructs the radiation from reaching the receiver either by reflecting the radiation or absorbing the radiation.

Distinguishing Between Black and White Colors

It is universal that black color absorbs the entire radiation incident on it and white color reflects the entire radiation incident on it. Based on this principle, the second positioning of the sensor couple can be made. The IR LED and the photodiode are placed side by side. When the IR transmitter emits infrared radiation, since there is no direct line of contact between the transmitter and receiver, the emitted radiation must reflect back to the photodiode after hitting any object. The surface of the object can be divided into two types: reflective surface and non-reflective surface. If the surface of the object is reflective in nature i.e. it is white or other light color, most of the radiation incident on it will get reflected back and reaches the photodiode. Depending on the intensity of the radiation reflected back, current flows in the photodiode.If the surface of the object is non-reflective in nature i.e. it is black or other dark color, it absorbs almost all the radiation incident on it. As there is no reflected radiation, there is no radiation incident on the photodiode and the resistance of the photodiode remains higher allowing no current to flow. This situation is similar to there being no object at all. The pictorial representation of the above scenarios is shown below.

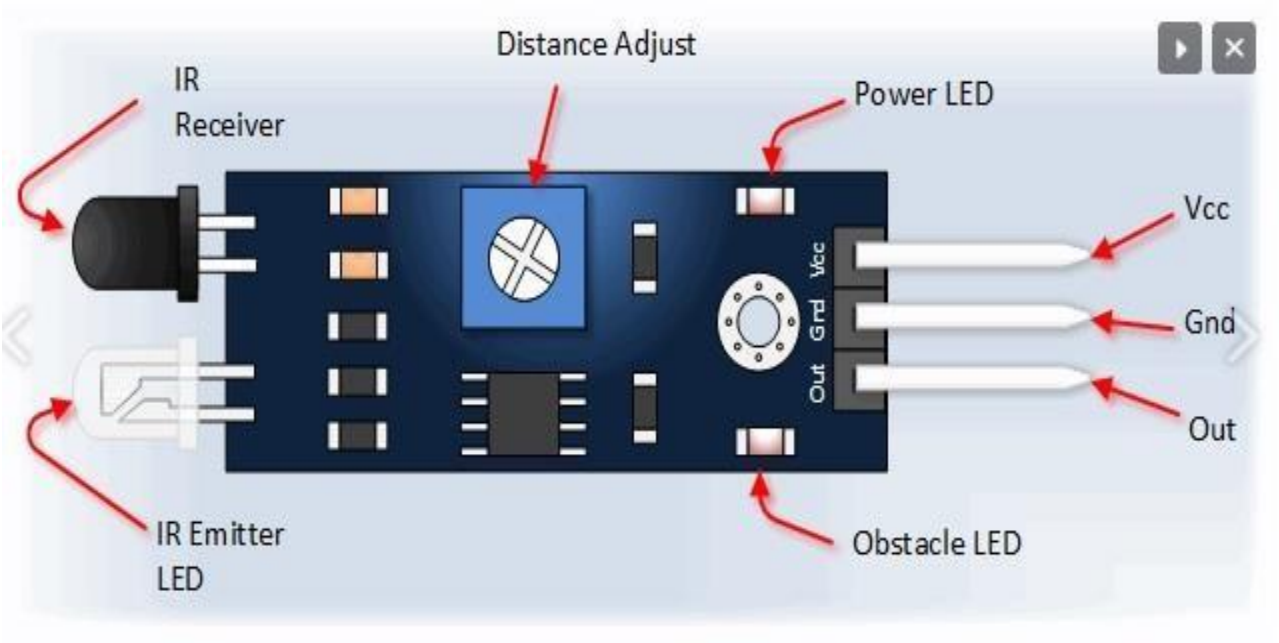


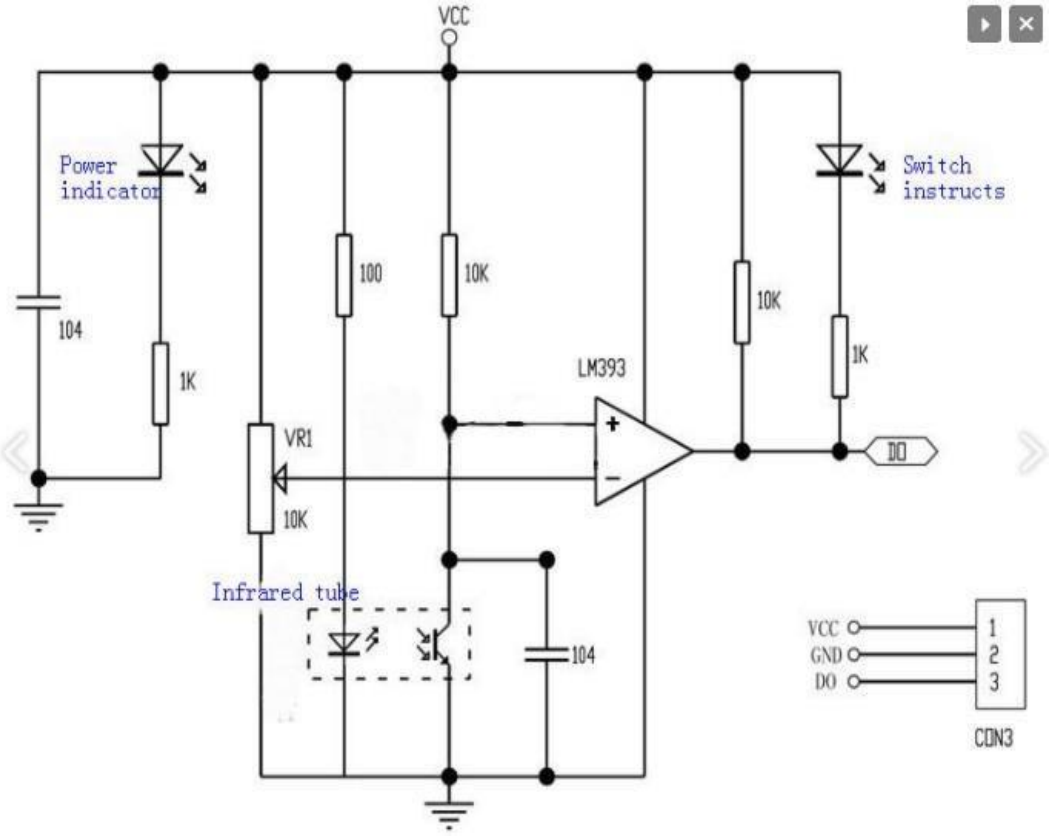
The positioning and enclosing of the IR transmitter and Receiver is very important. Both the transmitter and the receiver must be placed at a certain angle, so that the detection of an object happens properly. This angle is the directivity of the sensor which is +/- 45 degrees.

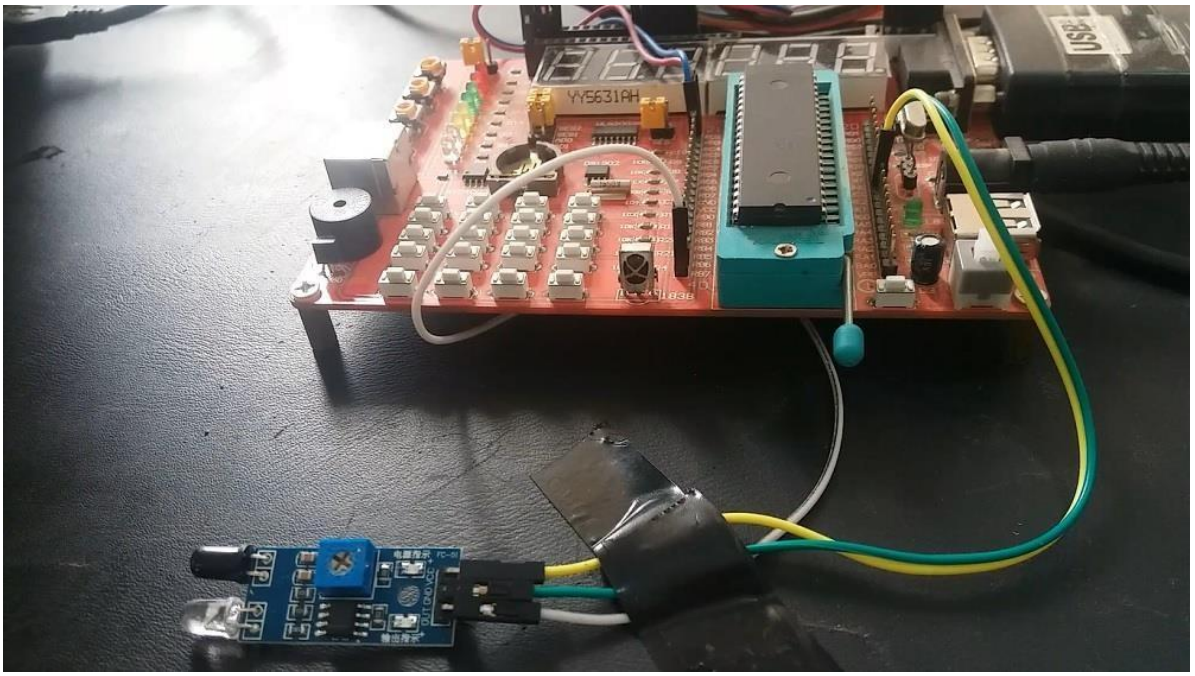


In order to avoid reflections from surrounding objects other than the object, both the IR transmitter and the IR receiver must be enclosed properly. Generally the enclosure is made of plastic and is painted with black color.

CIRCUIT DIAGRAM







.

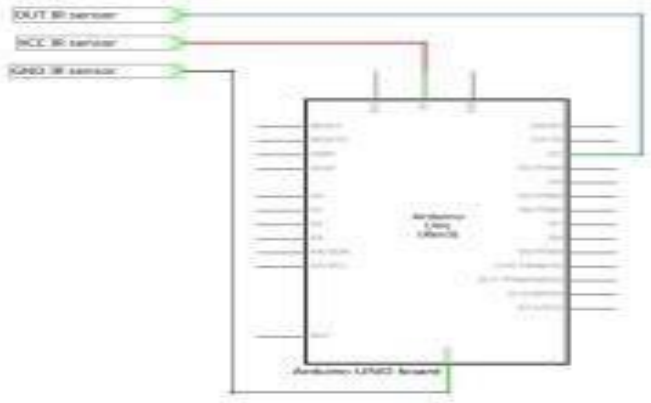
PROGRAM & PROCEDURE

Test IR sensor FC-51 with serial terminal (Test 01)

In the first demo, through the connection between the Arduino serial port and the PC, we will read about the detection of the object.

Lets take a look to steps required by this 1 test:

i. We connect the OUT pin of the sensor to digital pin 2 of Arduino called IR.
ii. The setup() function is performed only once before the main loop. We insert here the initialization code which enables serial port Arduino and sets the digital pin 2 as input.
iii. loop() is the main function and is cyclically repeated until you turn off the Arduino board. We convert in C language the operation of the electronic circuit analyzed before. We save in the variable detection the value taken from the pin IR with the specific function digitalRead, if the value is low there is an object otherwise there isn't.



```
#define IR 2
int detection = HIGH; // no obstacle
void setup() {
Serial.begin(9600);
pinMode(IR, INPUT);
}
void loop() {
detection = digitalRead(IR);
if(detection == LOW){
Serial.print("There is an obstacle!\n");
}
else{
Serial.print("No obstacle!\n");
}
delay(500); // in ms
}
```

IR sensor FC-51 and LED (Test 02)

In this test we associate an input to each operation state of IR sensor. The required components are:

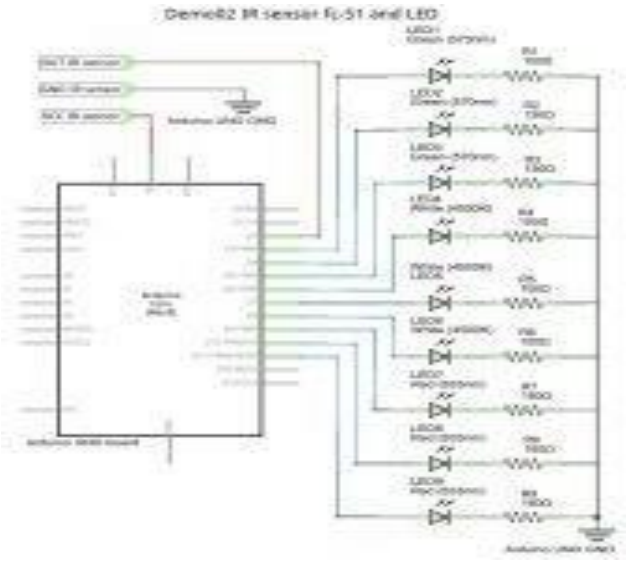
- **IR sensor FC-51**;
- **3 x Green LEDs**;
- **3 x R=150Ω**;
- **3 x White LEDs**;
- **3 x R=100Ω**;
- **3 x Red LEDs**;
- **3 x R=160Ω**.

Remember that the I/O pins can absorb/disburse up to 40mA max, total maximum 200mA (see ATmega328P datasheet at page 313).
Lets take a look to steps required by this demo:

1. We connect the OUT pin of the sensor to digital pin 2 of the Arduino. We define the digital pins of LEDs as an array of pins, from 3 to 11 called **LedPIN**.

2. The **setup()** function is executed only once before the main loop. In addition to the initialization already seen, we call the 9 LED as output using a for loop.

3. The **loop()** function is the main function and is cyclically repeated until you turn off the Arduino board. We save in the variable **detection** the value taken from pin **IR** with the specific function **digitalRead()**. This value can be low, if there is an object, or high if there is no object. We do this loop every millisecond.



Lets take a look to our code:

```
  #define IR 2 // digital pin input for ir sensor
int detection = HIGH; // no obstacle
int i = 0;
// array digital pin for: green led(3,4,5) - white led (6,7,8)- red led (9,10,11)
int LedPIN[] = {3, 4, 5, 6, 7, 8, 9, 10, 11};
void setup() {
pinMode(IR, INPUT);
for(i = 0; i < 9; i++){
pinMode(LedPIN[i], OUTPUT);
}
}
void loop() {
detection = digitalRead(IR);
if(detection == LOW){
BlinkLED();
}
else{
LedOFF();
}
delay(1);
}
```

CONCLUSION

Successfully design the circuit of interfacing IR sensor to realize obstacle detector in Proteussoftware and implement on hardware at Adriano microcontroller kit.

VIVA QUESTIONS:

1. Can you explain the basic principle behind an IR sensor?
2. What components are typically found in an IR sensor module?
3. What role does the comparator play in the IR sensor circuit?
4. How does the IR sensor detect obstacles?
5. Describe the process of interfacing an IR sensor with a microcontroller.
6. What are the key considerations when choosing a microcontroller for interfacing with an IR sensor?
7. How can you calibrate an IR sensor for optimal obstacle detection?
8. What are the potential challenges in interfacing multiple IR sensors in a single project?
9. How can you differentiate between different types of obstacles detected by the IR sensor?
10. Can you explain the significance of using interrupts in an IR sensor-based obstacle detection system?
11. How would you handle noise and interference in the output of an IR sensor?
12. Discuss the importance of feedback mechanisms in improving the accuracy of obstacle detection.
13. How can you optimize the power consumption of an IR sensor-based obstacle detection system?
14. What are some alternative sensor technologies that can be used alongside or instead of IR sensors for obstacle detection?
15. Can you outline some real-world applications where IR sensor-based obstacle detection systems are used?

DEPARTMENT OF ELECTRICAL ENGINEERING

EMBEDDED SYSTEMS

Experiment No : 06

Write a program to implement temperature measurement and displaying the same on an LCD display.

TITLE: Write a program to implement temperature measurement and displaying the same on an LCD display.

AIM

Write a program to implement temperature measurement and displaying the same on an LCD display. To interface an LM35 (or LM34) temperature sensor to the 8051 trainer.

COMPONENT

- 8051 trainer
- 8051 assembler
- LM35 (or LM34)
- ADC804
- LM336-2.5
- 10K POT
- 1K, 1.5K, and 10K resistors
- 8051 Trainer
- 20x2 LCD DMC20261 from Optrex DMC series, or a compatible one.
- Dot Matrix LCD Module: Character-type DMC Series User's Manual by Optrex Corp.

WORKING

M35 is an analog, linear temperature sensor whose output voltage varies linearly with change in temperature. LM35 is three terminal linear temperature sensor from National semiconductors. It can measure temperature from-***55 degree celsius to +150 degree celsius.*** The voltage output of the LM35 increases 10mV per degree Celsius rise in temperature. LM35 can be operated from a 5V supply and the stand by current is less than 60uA. So we need about LM35 for this particular temperature display project using arduino uno.

LM35 is an analog temperature sensor. This means the output of LM35 is an analog signal. Microcontrollers don't accept analog signals as their input directly. We need to convert this analog output signal to digital before we can feed it to a microcontroller's input. For this purpose, we can use an ADC(Analog to Digital Converter).If we are using a basic microcontroller like 8051, we need to use an external ADC to convert analog output from LM35 to digital. We then feed the output of ADC (converted digital value) to input of 8051. But modern day boards like Arduino and most modern day micro controllers come with inbuilt ADC. Our arduino uno has an in built 10 bit ADC (6 channel). We can make use of this in built ADC of arduino to convert the analog output of LM35 to digital output. Since Arduino uno has a 6 channel inbuilt ADC, there are 6 analog input pins numbered from A0 to A5. Connect analog out of LM35 to any of these analog input pins of arduino.

LM35 is available in the market in 3 series variations – LM35A, LM35C and LM35D series. The main difference between these 3 versions of LM35 IC are in their range of temperature measurements. The LM35D series is designed to measure from 0 degree Celsius to 100 degree Celsius, whereas the LM35A series is designed to measure a wider range of -55 degree Celsius to 155 degree Celsius. The LM35C series is designed to measure from -40 degree Celsius to 110 degree Celsius.

CIRCUIT

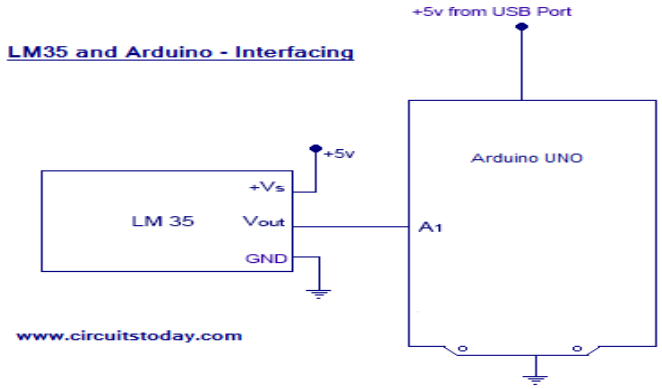


Fig 1. LM35 and Arduino – Circuit Diagram

Connect LM35 to Arduino uno as shown in circuit diagram. The +5v for LM35 can be taken from the +5v out pin of arduino uno. Also the ground pin of LM35 can be connected to GND pin of arduino uno. Connect Vout (the analog out of LM35) to any of the analog input pin of arduino uno. In this circuit diagram, we have connected Vout of LM35 to A1 of arduino.

Temperature Display on 16×2 LCD Module – using Arduino and lM35

Now lets go on to add a 16×2 LCD display with LM35 and Arduino – interface and lets display the temperature values on this LCD display (instead of serial monitor). So we are going to build none other than a standalone temperature display using arduino. Circuit Diagram – LM35 and Arduino – Temperature Display on 16×2 LCD

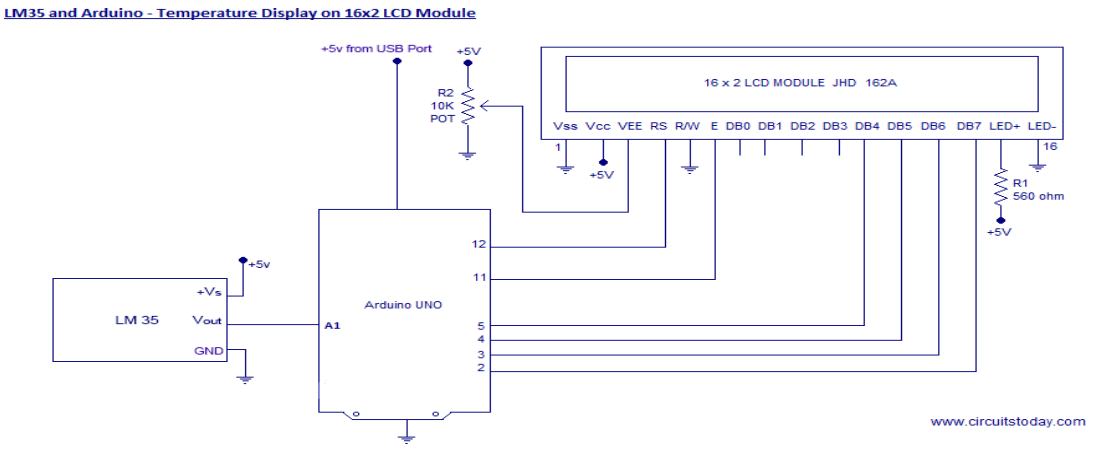
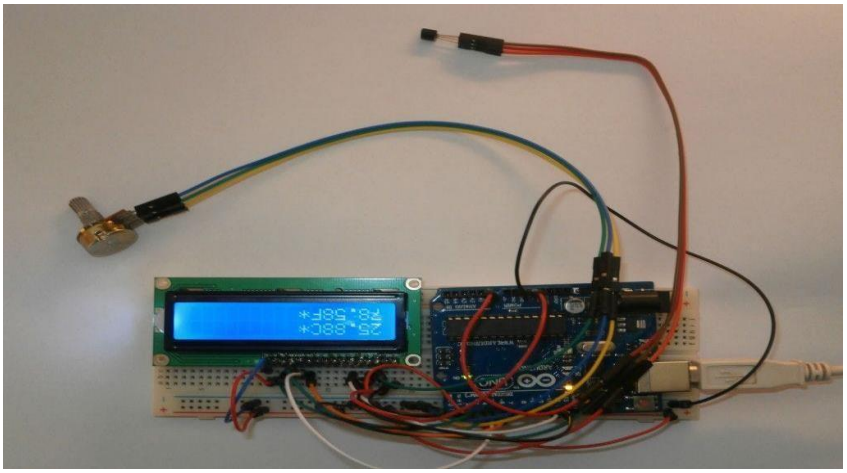
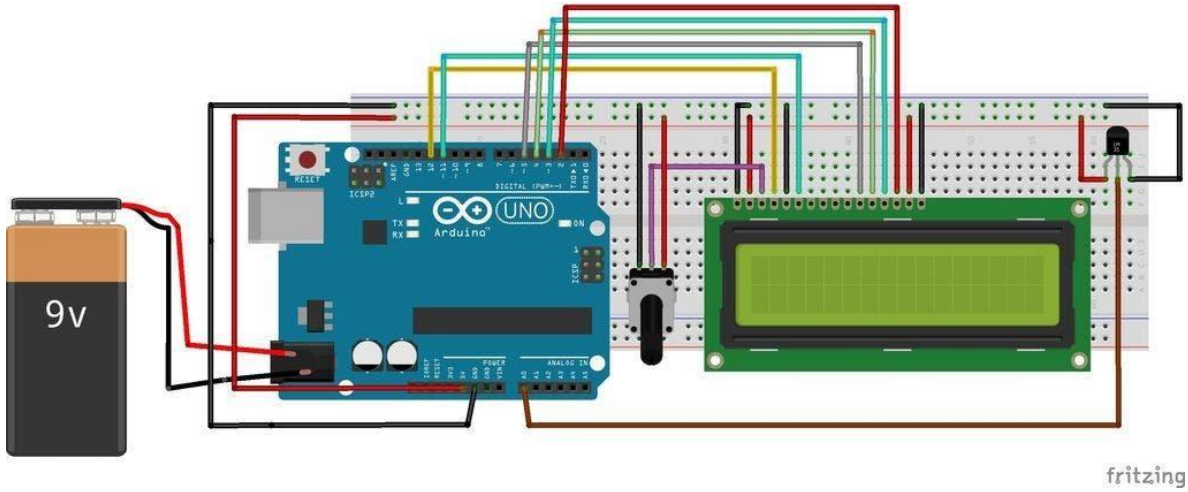


Fig 2. LM35 and Arduino – Temperature Display on 16×2 LCD





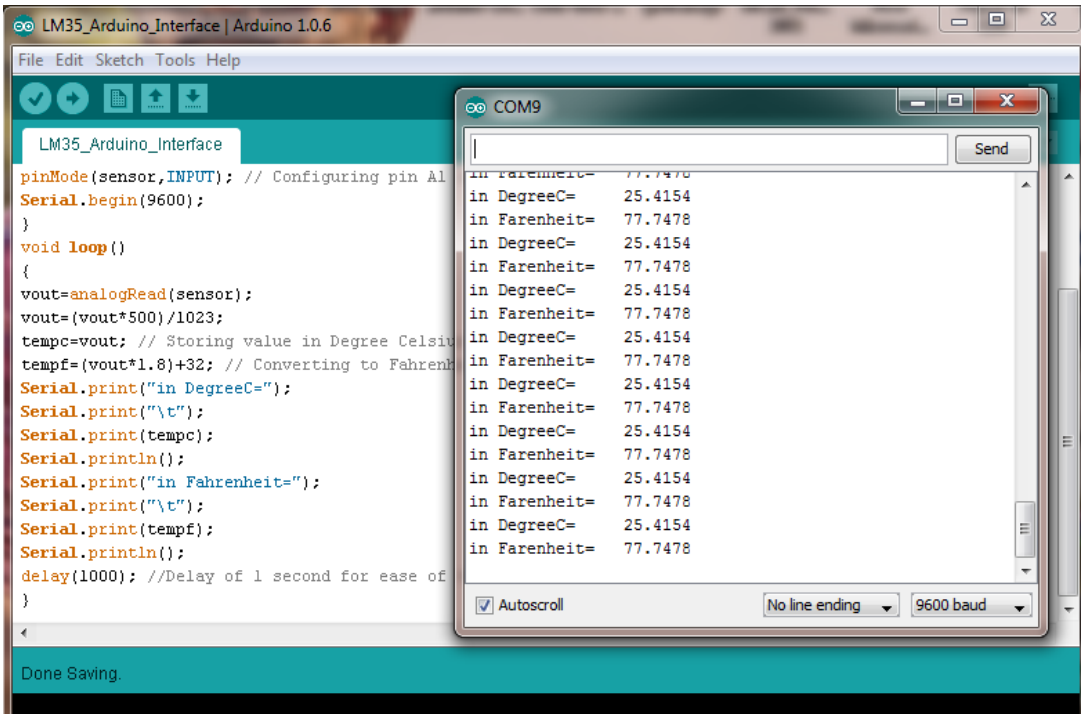
PROGRAM

The Program – LM35 and Arduino Interfacing

```
const int sensor=A1; // Assigning analog pin A1 to variable 'sensor'
float tempc;  //variable to store temperature in degree Celsius
float tempf; //variable to store temperature in Fahreinheit
float vout; //temporary variable to hold sensor reading
void setup()
{
pinMode(sensor,INPUT); // Configuring pin A1 as input
Serial.begin(9600);
}
void loop()
{
vout=analogRead(sensor);
vout=(vout*500)/1023;
tempc=vout; // Storing value in Degree Celsius
tempf=(vout*1.8)+32; // Converting to Fahrenheit
```

```
Serial.print("in DegreeC=");
Serial.print("\t");
Serial.print(tempc);
Serial.println();
Serial.print("in Fahrenheit=");
Serial.print("\t");
Serial.print(tempf);
Serial.println();
delay(1000); //Delay of 1 second for ease of viewing
}
```

So that's the arduino LM 35 code for reading temperature and displaying in degree Celsius and Fahrenheit.



To build an Arduino LM35 temperature sensor with LCD display are shown in circuit Fig 2. The arduino program for the circuit is given below.

The Program

```
#include<LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
const int sensor=A1; // Assigning analog pin A1 to variable 'sensor'
float tempc;  //variable to store temperature in degree Celsius
float tempf; //variable to store temperature in Fahreinheit
float vout; //temporary variable to hold sensor reading
void setup()
{
pinMode(sensor,INPUT); // Configuring pin A1 as input
Serial.begin(9600);
lcd.begin(16,2);
  delay(500);
}
```

```
void loop()
{
vout=analogRead(sensor);
vout=(vout*500)/1023;
tempc=vout; // Storing value in Degree Celsius
tempf=(vout*1.8)+32; // Converting to Fahrenheit
lcd.setCursor(0,0);
lcd.print("in DegreeC= ");
lcd.print(tempc);
lcd.setCursor(0,1);
lcd.print("in Fahrenheit=");
lcd.print(tempf);
delay(1000); //Delay of 1 second for ease of viewing in serial monitor
}
```

The program is very simple and self explanatory, if you have a basic idea of arduino sketches and you already learned how to interface arduino and lcd module. In another step 7 Segment display to display temperature measured using LM35 and Arduino. For displaying temperature upto 3 digit values (100 degree Celsius or higher upto 999) with corresponding unit (Celsius of Fahrenheit), its good to choose a 4 Digit 7 Segment Display unit.

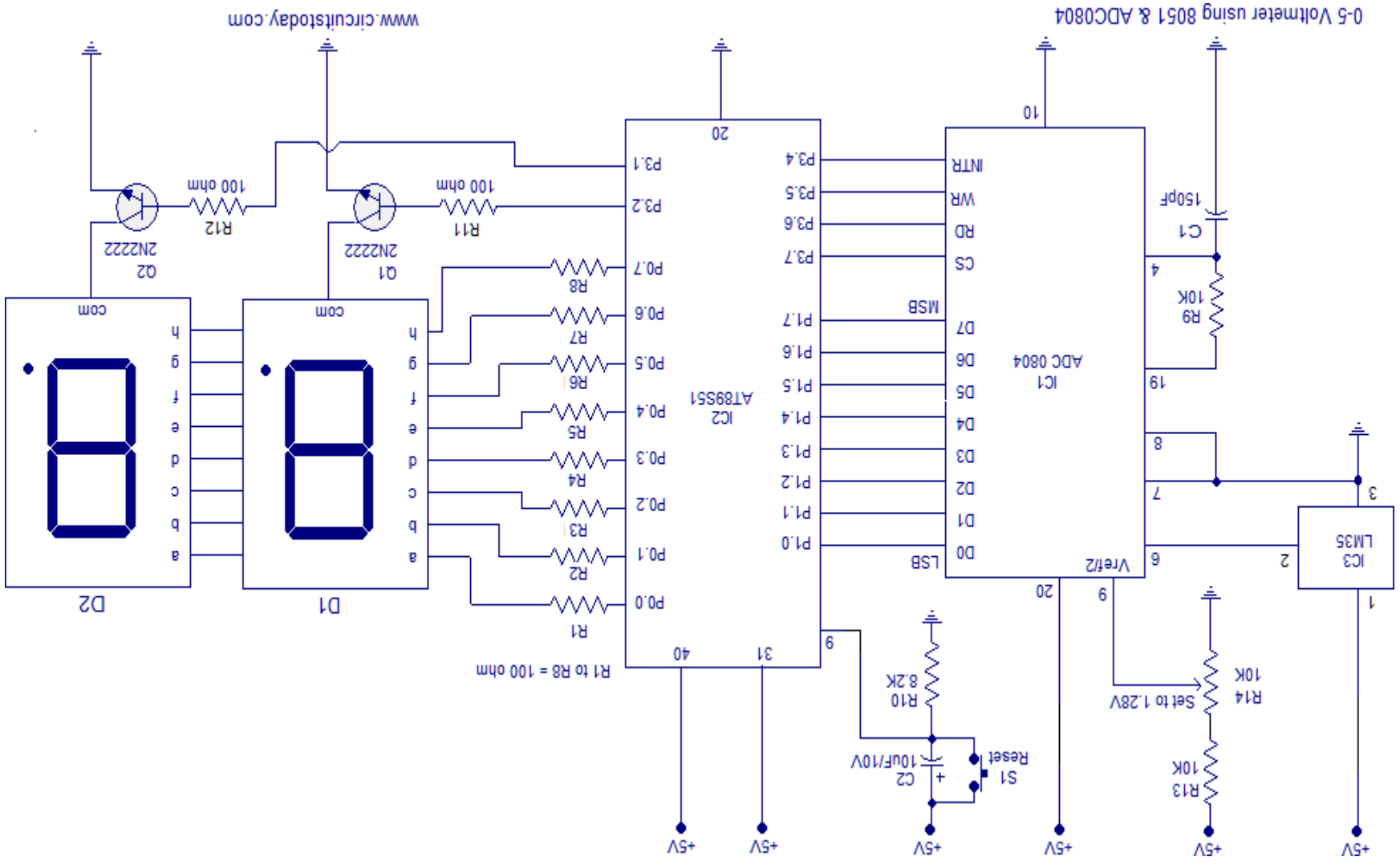
RESULT

Study and Demonstrate the LM35 Temperature senior on arduino Kit successfully

ORAL QUESTIONS

1. What is a transducer?
2. What is the form of the transducer output?
3. What is preprocessing of transducer signals to be fed into an ADC called?
4. The LM35 and LM34 produce a_____mV output for every degree of change in temperature.
5. The LM35/LM34 is a_____(linear, nonlinear) device. Discuss the advantages of linear devices and of nonlinear devices.
6. Explain signal conditioning and its role in data acquisition.
7. How does the LCD distinguish data from instruction codes when receiving information at its data pin?
8. To send the instruction code 01 to clear the display, we must make RS =____.
9. To send letter 'A' to be displayed on the LCD, we must make RS =_____.
10. What is the purpose of the E line? Is it an input or an output as far as the LCD is concerned?
11. When is the information (code or data) on the LCD pin latched into the LCD

DIAGRAM


0-5 Voltmeter using 8051 & ADC0804
www.circuitstoday.com
IC1
ADC 0804
IC2
AT89S51
IC3
LM35
Q1
2N2222
Q2
2N2222
R1 to R8 = 100 ohm
100 ohm
150pF
10K
8.2K
10uF/10V
Reset
Set to 1.28V
+5V
MSB
LSB
D1
D2
com

DEPARTMENT OF ELECTRICAL ENGINEERING

Embedded System Lab

Experiment No: 7

Write a program for interfacing GAS sensor and perform GAS leakage detection.

TITLE: Write a program for interfacing GAS sensor and perform GAS leakage detection.

AIM

Write a program for interfacing GAS sensor and perform GAS leakage detection.

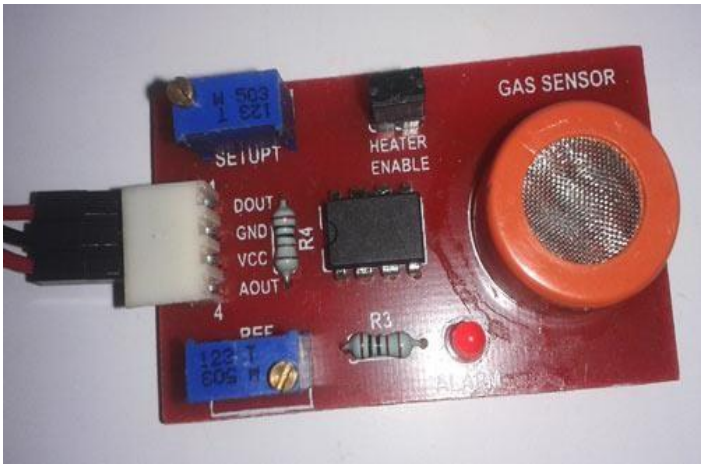
APPRATUS

1. Arduino Pro Mini
2. LPG Gas sensor Module
3. Buzzer
4. BC 547 Transistor
5. 16x2 LCD
6. 1K resistor
7. Bread board
8. 9 volt battery
9. Connecting wires

THEORY

A) LPG Gas sensor module

This module contains a MQ3 sensor which actually detects LPG gas, a comparator (LM393) for comparing MQ3 output voltage with reference voltage. It gives a HIGH output when LPG gas is sensed. A potentiometer is also used for controlling sensitivity of gas sensing. This module is very easy to interface with microcontrollers and arduino and easily available in market by name “LPG Gas Sensor Module”. We can also build it by using LM358 or LM393 and MQ3.Near infrared region — 700 nm to 1400 nm — IR sensors, fiber optic



The gas sensor MQ3 suitable for detecting of LPG, i-butane, propane, methane, alcohol, Hydrogen, smoke etc. Since It is highly sensitive and gives fast response, we can take measurements as soon as possible. This sensor can be used for gas leakage detection. The basic concept of an Infrared Sensor which is used as Obstacle detector is to transmit an infrared signal, this infrared signal bounces from the surface of an object and the signal is received at the infrared receiver.

At normal condition, sensor resistor will be high so voltage drop across the load will be low and it will be a constant. f sensor senses flammable gases, resistance of sensor will drop. That means more current will flow from load resistor. So the voltage across it increases. This output voltage increases with increase in concentration of gas in air. The sensitivity of the gas sensor can be adjusted using potentiometer. Refer MQ-3 datasheet for detailed information.

This Smoke Sensor (MQ3) Board has analog as well as digital output. For this tutorial we will use analog output. Analog output pin needs to be connecting ADC channel 0 of atmega 32 breakout as shown in hook up.

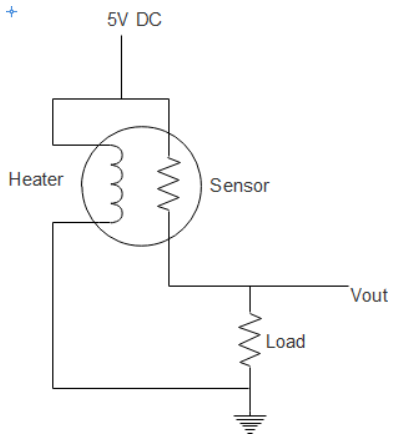
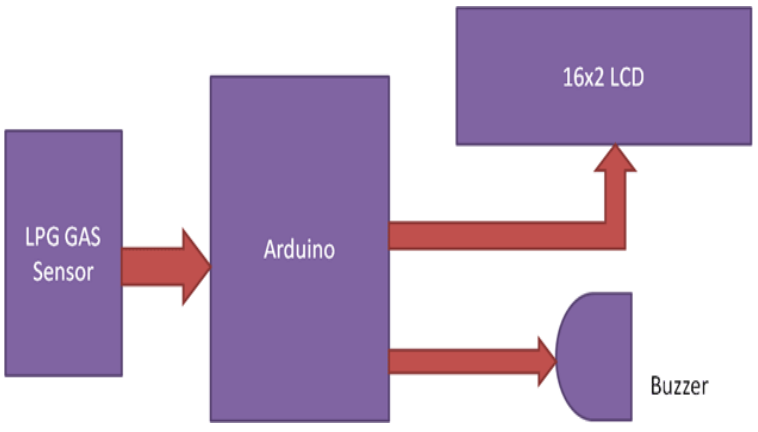


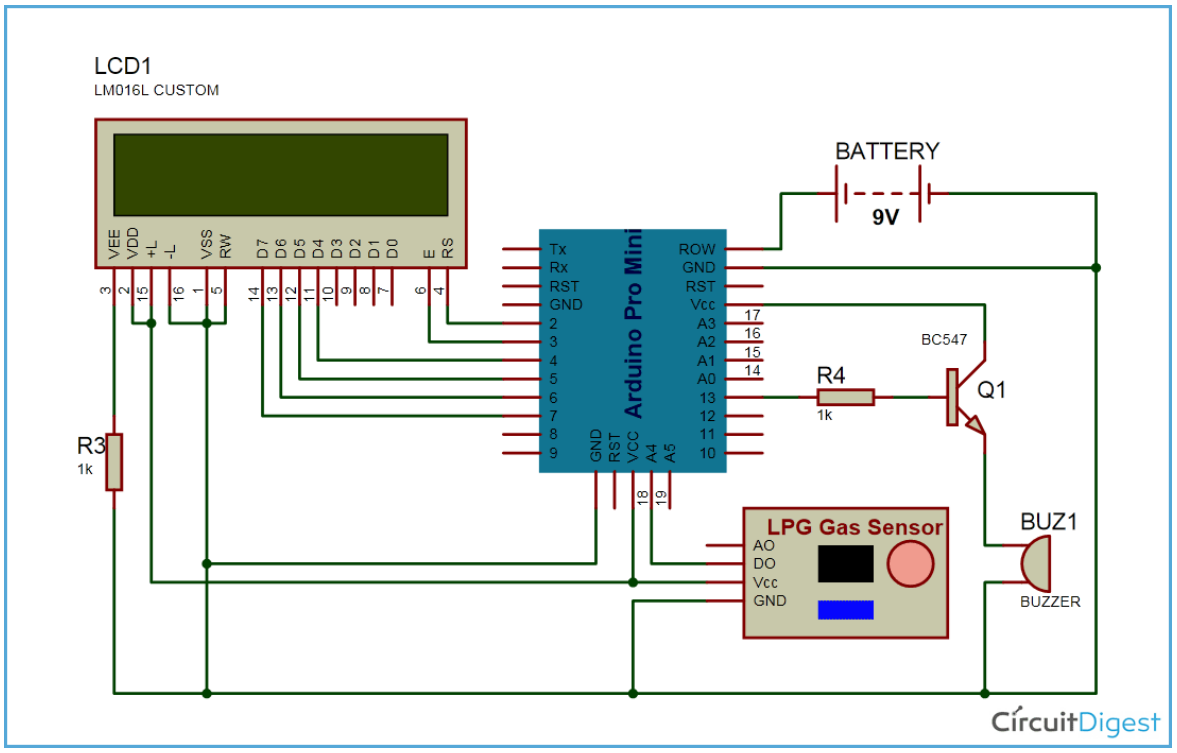
Fig : MQ-3 sensor schematic

- As MQ-3 sensor has heater inside, it is prefer to give power to the sensor from separate source.
- For stable operation sensor requires around 24 hour preheating.
- We can use Ultra AVR Development Board, Starter AVR or Atmega32 Breakout.

We have used a LPG **gas sensor module** to detect LPG Gas. When LPG gas leakage occurs, it gives a HIGH pulse on its DO pin and arduino continuously reads its DO pin. When Arduino gets a HIGH pulse from LPG Gas module it shows “LPG Gas Leakage Alert” message on 16x2 LCD and activates buzzer which beeps again and again until the gas detector module doesn't sense the gas in environment. When LPG gas detector module gives LOW pulse to arduino, then LCD shows “No LPG Gas Leakage” message.

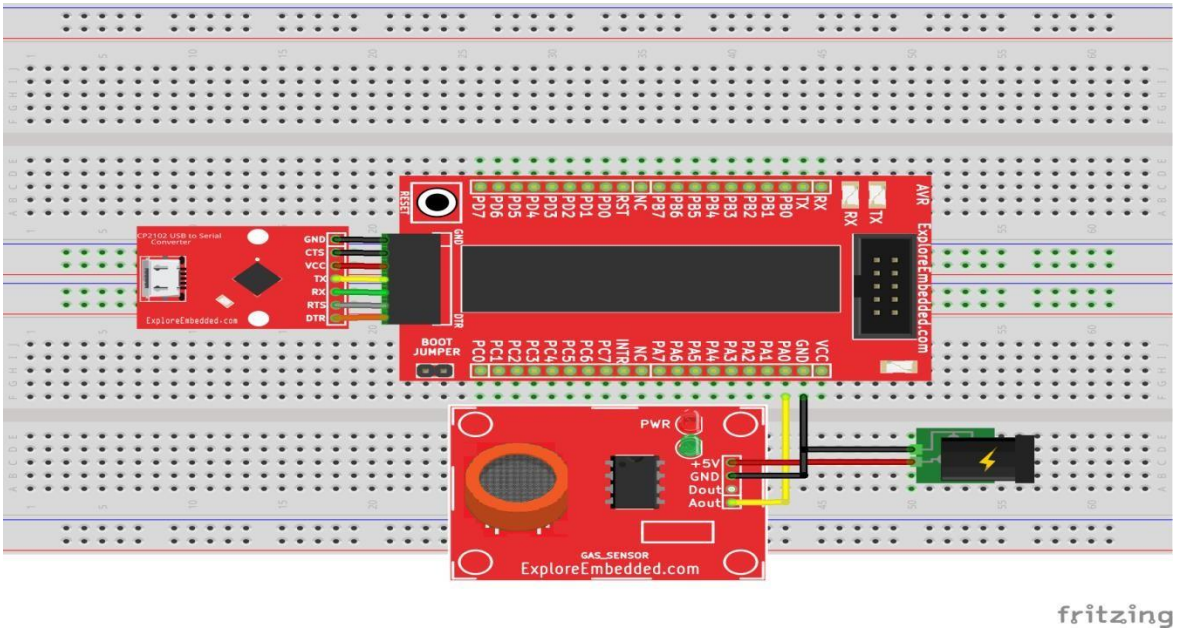


CIRCUIT DIAGRAM

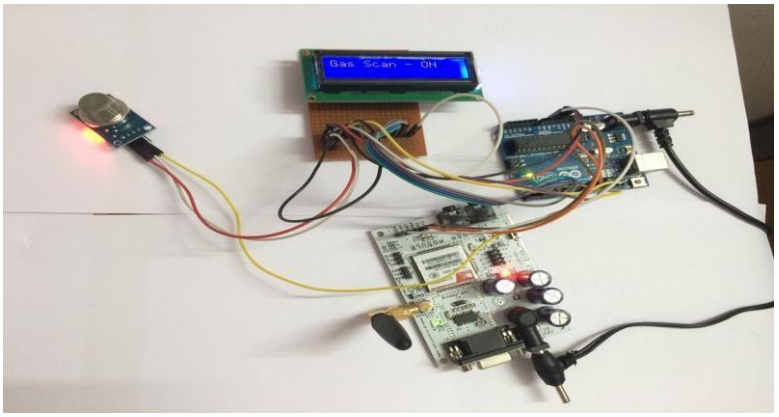


As shown in the schematic diagram above, it contains Arduino board, LPG GAS Sensor Module, buzzer and 16x2 LCD module. Arduino controls the whole process of this system like reading LPG Gas sensor module output, sending message to LCD and activating buzzer. We can set sensitivity of this sensor module by inbuilt potentiometer placed on it.

LPG gas sensor module's DO pin is directly connected to pin 18 (A4) of Arduino and Vcc and GND are connected to Vcc and GND of arduino. LPG gas sensor module consist a MQ3 sensor which detects LPG gas. This MQ3 sensor has a heater inside which needs some heater supply to heat up and it may takes up to 15 minute to get ready for detecting LPG gas.



A comparator circuit is used for converting Analog output of MQ3 in digital. A 16x2 LCD is connected with arduino in 4-bit mode. Control pin RS, RW and En are directly connected to arduino pin 2, GND and 3. And data pin D0-D7 are connected to 4, 5, 6, 7 of arduino. A buzzer is connected with arduino pin number 13 through a NPN BC547 transistor having a 1 k resistor at its base.



WORKING

MQ3 gas sensor can be used to detect the presence of LPG, Propane and Hydrogen, also could be used to detect Methane and other combustible steam, it is low cost and suitable for different application. Sensor is sensitive to flammable gas and smoke. Smoke sensor is given 5 Volt to power it. Smoke sensor indicate smoke by the voltage that it output more smoke more output. A potentiometer is provided to adjust the sensitivity. Sn02 is the sensor used which is of low conductivity when the air is clean. But when smoke exist sensor provides an Analog resistive output based on concentration of smoke. The circuit has a heater. Power is given to heater by VCC and GND from power supply. The circuit has a variable resistor. The resistance across the pin depends on the smoke in air in the sensor. The resistance will be lowered if the content is more. And voltage is increased between the sensor and load resistor.

The MQ3 has an electrochemical sensor, which changes its resistance for different concentrations of varied gasses. The sensor is connected in series with a variable resistor to form a voltage divider circuit , and the variable resistor is used to change sensitivity. When one of the above gaseous elements comes in contact with the sensor after heating, the sensors resistance change. The change in the resistance changes the voltage across the sensor, and this voltage can be read by a microcontroller. The voltage value can be used to find the resistance of the sensor by knowing the reference voltage and the other resistors resistance. The sensor has different sensitivity for different types of gasses.

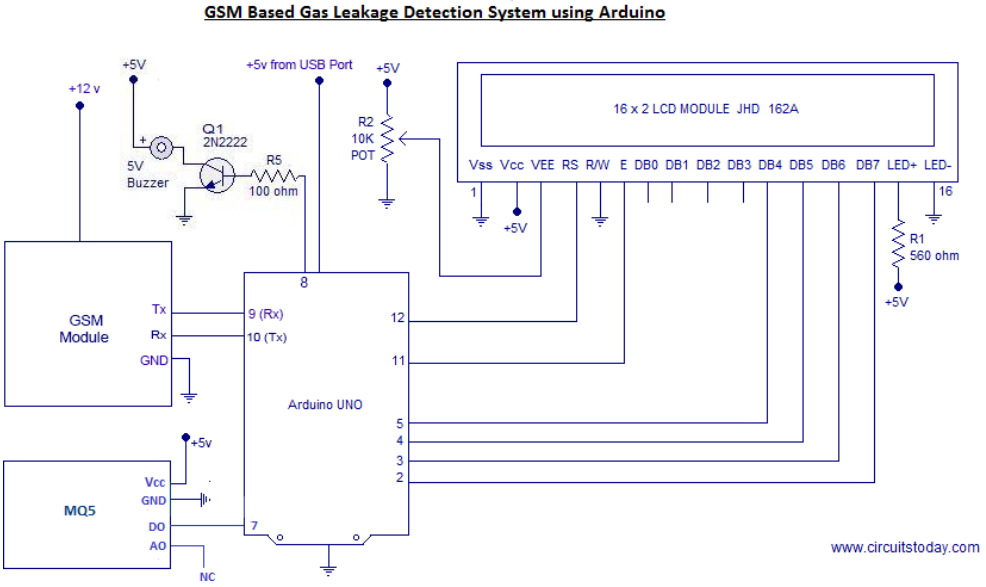
WEIGHT SENSOR (LOAD CELL)

We should know ahead of time of measure of gas in the chamber, and for this reason the dimension of gas present in the chamber must be observed persistently. We have utilized strain measure as a weight sensor. The capacity of strain gage is to give yield voltage according to the power/weight connected to it. Sensor changes over the connected power into comparing electrical flag. The yield of weight sensor is in simple structure. It is given to a Digitizer board which accompanies this weight sensor. Capacity of Digitizer board is to give advanced yield which is corresponding to simple information gotten from weight sensor. This advanced yield is given to microcontroller for further handling. We have utilized a weight sensor of 40 kg limit. So 40 kg is the greatest weight that can be connected to this weight sensor.

There are two flow charts for gas leakage detection and automatic gas booking which explain the methodology of the operation as follows:

In this model, gas spillage recognition has been given a most elevated need. MQ2 set in the region of the gas chamber. In the appearance of spillage, the obstruction of the sensor diminishes expanding its conductivity. Relating beat is sustained to microcontroller and at the same time switches on the ringer and fumes fan which we can reset by a manual reset switch. Additionally a rationale high heartbeat (+5 V) is given as a hinder to INT0 stick of Microcontroller. Microcontroller communicates something specific "EMERGENCY ALERT: LPG gas spillage found in your home" to required cell numbers by means of GSM module and a similar will be shown on LCD.

PROGRAM & PROCEDURE



The Program/Code

```
#include <SoftwareSerial.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
SoftwareSerial mySerial(9, 10);
int sensor=7;
int speaker=8;
int gas_value,Gas_alert_val, Gas_shut_val;
in0t Gas_Leak_Status;
int sms_count=0;
void setup()
{
pinMode(sensor,INPUT);
pinMode(speaker,OUTPUT);
mySerial.begin(9600);
Serial.begin(9600);
lcd.begin(16,2);
delay(500);
}
void loop()
{
CheckGas();
```

```
CheckShutDown();
}
void CheckGas()
{
lcd.setCursor(0,0);
lcd.print("Gas Scan - ON");
Gas_alert_val=ScanGasLevel();
if(Gas_alert_val==LOW)
{
 SetAlert(); // Function to send SMS Alerts
}}
int ScanGasLevel()
{
gas_value=digitalRead(sensor); // reads the sensor output (Vout of
LM35)
return gas_value; // returns temperature value in degree celsius
}
void SetAlert()
{
digitalWrite(speaker,HIGH);
while(sms_count<3) //Number of SMS Alerts to be sent
{
SendTextMessage(); // Function to send AT Commands to GSM module
}
Gas_Leak_Status=1;
lcd.setCursor(0,1);
lcd.print("Gas Alert! SMS Sent!");
}
void CheckShutDown()
{
if(Gas_Leak_Status==1)
{
Gas_shut_val=ScanGasLevel();
if(Gas_shut_val==HIGH)
{
lcd.setCursor(0,1);
lcd.print("No Gas Leaking");
digitalWrite(speaker,LOW);
sms_count=0;
Gas_Leak_Status=0;
}}}
void SendTextMessage()
{
  mySerial.println("AT+CMGF=1");    //To send SMS in Text Mode
  delay(1000);
  mySerial.println("AT+CMGS=\"+919495xxxxxx\"\r"); // change to the
phone number you using
```

```
delay(1000);
  mySerial.println("Gas Leaking!");//the content of the message
  delay(200);
  mySerial.println((char)26);//the stopping character
  delay(1000);
   mySerial.println("AT+CMGS=\"+918113xxxxxx\"\r"); // change to the
phone number you using
  delay(1000);
  mySerial.println("Gas Leaking!");//the content of the message
  delay(200);
  mySerial.println((char)26);//the message stopping character
  delay(1000);
  sms_count++;
}
```

Important Aspects about the Program

When we develop critical systems like Gas Leakage Detector or similar systems like Fire Alarm System, we need to monitor the sensor parameters continuously(24×7). So our system must monitor "gas leak" continuously.This is achieved by scanning the sensor output (digital out of MQ5) continuously inside the **ScanGasLevel()** subroutine. If you look into the program, the main function loop() has only two subroutines – CheckGas() and CheckShutDown() – which are called repeatedly. CheckGas() – is a subroutine which scans sensor output continuously and take actions if there occurs a „gas leak" at any point of time. CheckShutDown() – is a subroutine to monitor the shut down process and check if status of room is back to normal conditions (no gas leaking).

CheckGas() – is the function which monitors occurrence of a gas leak 24×7. This function fetches the gas level measured by MQ35 (by reading digital out of MQ35 using digitalRead() command) and stores it to the variable Gas_alert_val for comparison. If there is no „gas leak" – the sensor out will be HIGH. If there occurs a „gas leak" at any point of time, the sensor out will immediately change to LOW status. The statement if(Gas_alert_val==LOW) checks this and if a gas leak occurs, then an inner subroutine SetAlert() will be invoked.

SetAlert() is the function that controls number of SMS alerts sent to each mobile number loaded in the program. The number of SMS alerts sent can be altered by changing the stopping condition of while loop. The stopping condition sms_count<3 – means 3 SMS alerts will be sent to each mobile number. If you want to send 5 alerts, just change the stopping condition to sms_count<5 – you got it ? The function to send SMS (using AT Commands) – SendTextMessage() will be called 3 times if SMS alert count is 3. This function SendTextMessage() will be invoked as many times as the number SMS alerts set in the program. In addition to sending SMS alerts, this subroutine also controls the sound alarm. The alarm is invoked using command digitalWrite(speaker,HIGH) – which will activate the speaker connected at pin 8 of arduino.

CheckShutDown() − is the function which monitors if gas leak was shut down. We need to entertain this function only if a „gas leak" has occurred. To limit the entry to the statements inside this routine, we have introduced a variable **Gas_Leak_Status**. This variable value will be set to value **1** when a gas leak occurs (check the statement inside SetAlert()). The statements inside CheckShutDown() will be executed only if the value of **Gas_Leak_Status==1**. (If there was no gas leak occurred, we don"t need to waste time executing ShutDown checking statements). We consider the „gas leak" has been eliminated once room temperature is back to normal.

So if our variable **Gas_shut_val** falls back to HIGH status, we consider gas leak has been eliminated and surroundings are safe. The subroutine has statement to stop the gas leakage alarm (refer statement – digitalWrite(speaker,LOW) – which cuts the supply to pin 8 of arduino and stops the sound alarm) which will be executed when gas leak is eliminated completely (as the status of Gas_shut_val == HIGH). We start our Gas Leakage monitoring again with SMS Alerts active! (We reset the **Gas_Leak_Status** variable and **sms_count** variable back to zero – which are essential variable conditions for monitoring gas leak again and to send alert sms if gas leak repeats.

CONCLUSION

Successfully design the circuit of interfacing GAS sensor and perform GAS leakage detection and implement on hardware at Adriano microcontroller kit.

VIVA QUESTIONS:

1. Can you explain the principle behind gas sensor operation?
2. What is the significance of interfacing a gas sensor with a microcontroller?
3. How does the microcontroller communicate with the gas sensor?
4. What are the key components required to interface a gas sensor with a microcontroller?
5. How do you ensure proper calibration of the gas sensor in the setup?
6. What factors can affect the accuracy of gas leakage detection using the sensor?
7. Can you explain the steps involved in programming the microcontroller for gas leakage detection?
8. What methods can be employed to visualize the gas leakage detection results?
9. How do you handle potential false positives or false negatives in gas leakage detection?
10. What safety precautions should be taken while handling gas sensors and conducting gas leakage detection experiments?
11. How do you select the appropriate gas sensor for a specific application?
12. What measures can be implemented to enhance the sensitivity of the gas sensor?
13. Can you discuss any potential limitations or challenges encountered while interfacing gas sensors with microcontrollers?
14. How do you troubleshoot common issues encountered during the setup or operation of the gas sensor system?
15. In what ways can the gas leakage detection program be optimized for better efficiency and performance?

DEPARTMENT OF ELECTRICAL ENGINEERING

Embedded System Lab

Experiment No: 8

Write a program to design the Traffic Light System and implement the same using suitable hardware.

TITLE: Write a program to design the Traffic Light System and implement the same using suitable hardware.

AIM

Write a program to design the Traffic Light System

APPRATUS

1) ATMEGA16 microcontroller.
2) LED
3) Crystal Oscilator
4) Resistor
5) Capacitors
6) Kit power supply or Dual power supply with +/- 12volt/ 200mA.
7) Proteus Software

DEFINATION

How to interface an LED matrix with a micro-controller and use the same to display patterns on the matrix with a delay.

THEORY

ABOUT MICROCONTROLLERS FROM THE AVR FAMILY:

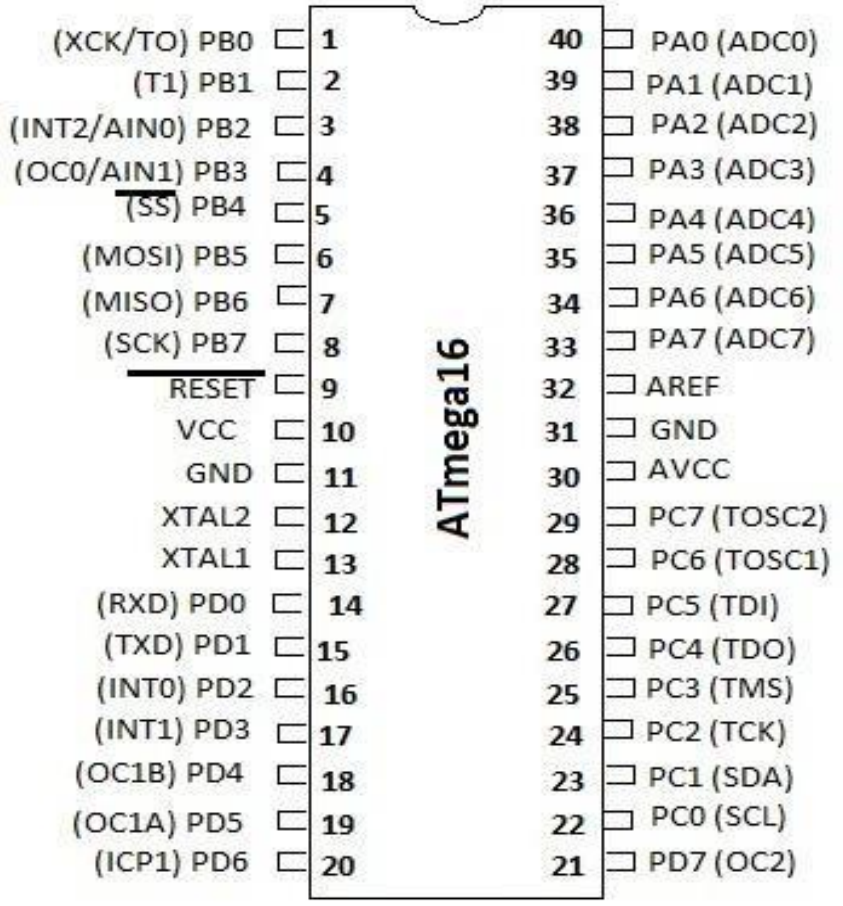
The AVR architecture is based upon modified Harvard architecture where program and data are stored in separate physical memory systems. The AVR family can be briefly classified as

- Tiny AVR
- Mega AVR
- XMEGA AVR
- Application specific AVR
- FPSLIC
- 32-bit AVR

Atmel's AVRs have a two stage, single level pipeline design. This means the next machine instruction is fetched as the current one is executing. Most instructions take just one or two clock cycles, making AVRs relatively fast among the eight-bit microcontrollers. The AVR processors were designed for efficient execution of compiled C code in mind and have several built-in pointers for the task.

KNOWLEDGE ON ATMEGA16

ATmega16 is an 8-bit high performance microcontroller from Atmel's Mega AVR family with low power consumption. The architecture of ATmega16 is based on enhanced RISC (Reduced Instruction Set Computing) architecture with 131 powerful instructions. Most of the instructions are executed in one machine cycle. It can work on a maximum frequency of 16MHz.



Pin description of ATmega16

FEATURES OF ATMEGA 16

a) 16 kilo bytes of Flash memory.
b) 512 bytes of EEPROM.
c) 1 kilo bytes of SRAM.
d) 131 instructions set.
e) 32 x 8 bit general purpose working registers.
f) On chip 2 cycle multiplier.
g) Programming locks for software security.
h) Two 8-bit timers.
i) One 16 bit timer.
j) 8 channel 10-bit ADC.
k) On chip Analog comparator.
l) Internal calibrated RC Oscillator.

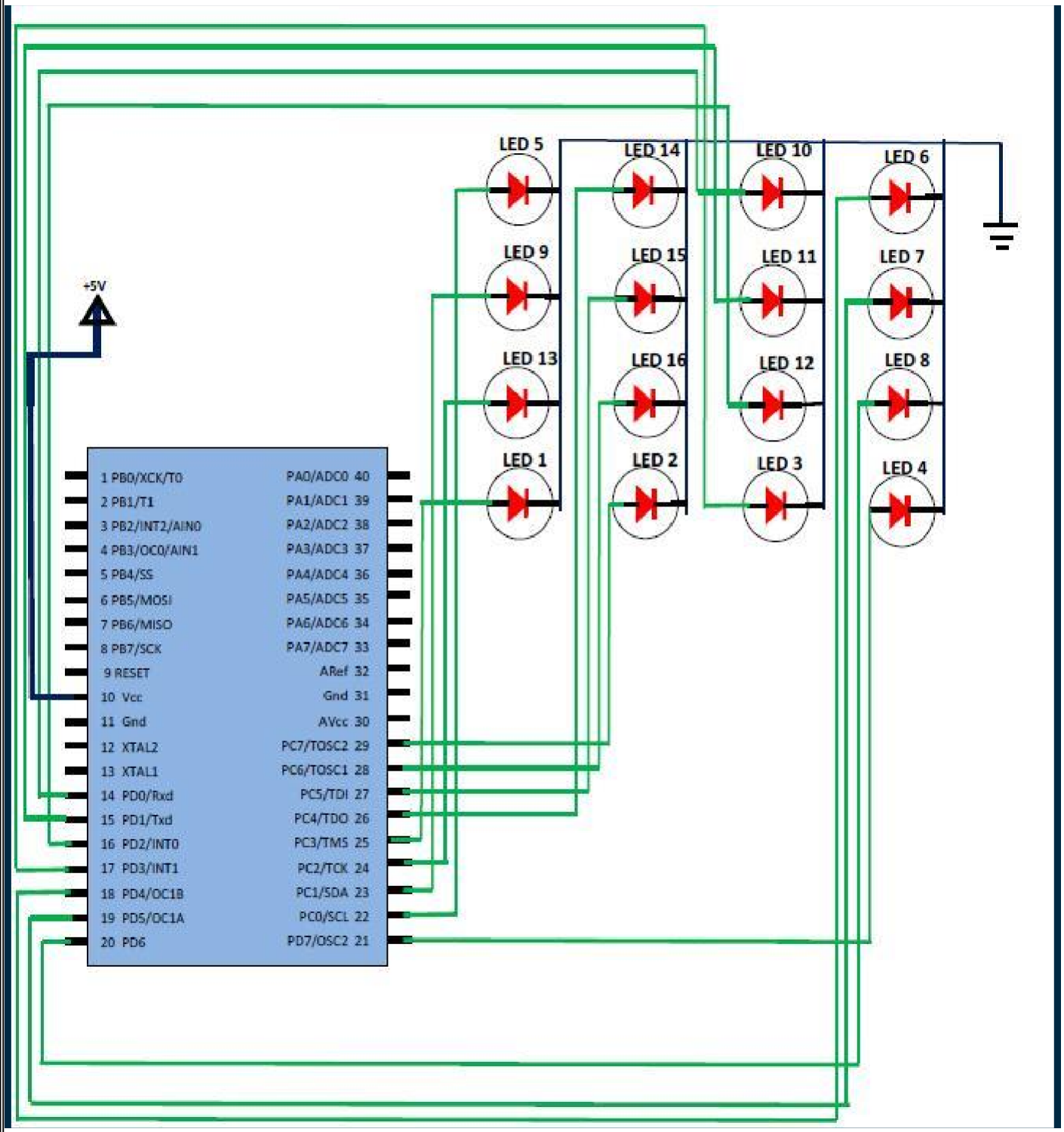
m) 32 programmable I/O lines.
n) Programmable serial USART.
o) Max Speed : 16 MHz(ATmega16).
p) Watchdog timer.
q) USB controller support.
r) Ethernet controller support.
s) LCD controller support.
t) DMA controller.

BASIC KNOWLEDGE OF LEDs

Light Emitting Diode (LED) is a diode that will give off visible light when it is energized. In any forward-biased p-n junction there is, within the structure and primarily close to the junction, a recombination of holes and electrons. This recombination requires that the energy possessed by the unbound free electron be transferred to another state. In all semiconductor p-n junctions, some of this energy will be given off as heat and some in the form of photons. In materials, such as gallium phosphide (GaP) or gallium arsenide phosphide (GaAsP), the number of photons of light energy emitted is sufficient to create a visible light source. There are also two-lead LED lamps that contain two LEDs, so that a reversal in biasing will change the colour from green to red, or vice versa. LEDs are presently available in red, green, yellow, orange, and white. In general, LEDs operate at voltage levels from 1.7 to 3.3 V, which makes them completely compatible with solid-state circuits. They have a fast response time (nanoseconds) and offer good contrast ratios for visibility. The power requirement is typically from 10 to 150 mW with a lifetime of 100,000 hours. Their semiconductor construction adds a significant ruggedness factor.
Typical characteristics of a semiconductor diode is shown below:

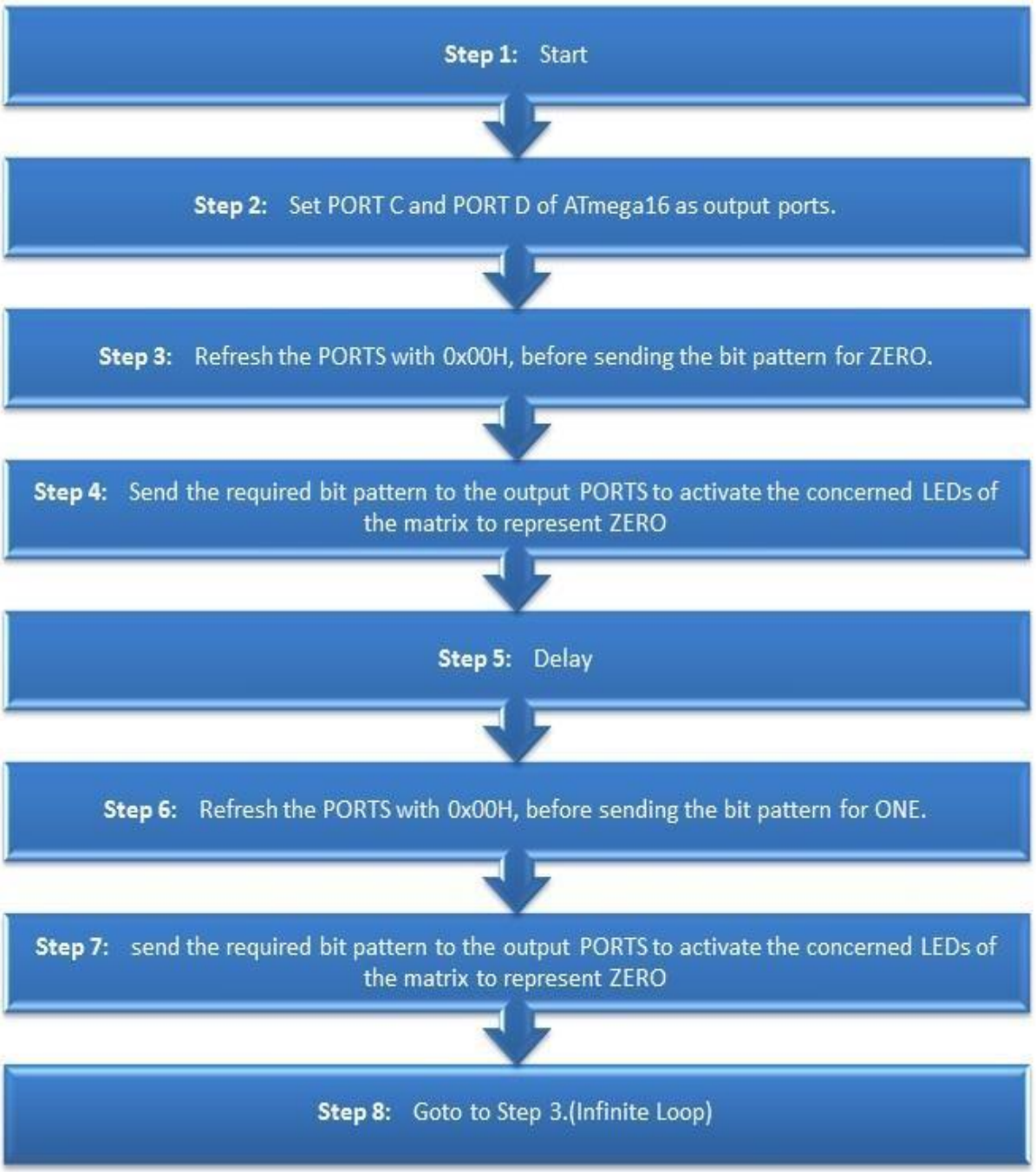
PARAMETER	CHARACTERISTICS	UNIT
Power dissipation	110	MW
Forward Current	40	MA
Peak Forward current(f=1 KHz, DF=10%)	200	MA
Lead Soldering Time at 260^{0}C	5	SEC
Operating Temperature	-40 TO +100	^{0}C
Storage Temperature	-40 TO +100	^{0}C

BLOCK DIAGRAM



PROCEDURE

Algorithm for Programming ATmega16:



CONCLUSION

Successfully design the circuit of Traffic Light System in Proteus software and implement on hardware at microcontroller kit.

VIVA QUESTIONS:

1. Can you explain the basic concept behind a traffic light system?
2. What are the essential components required to build a traffic light system?
3. How does the program determine the timing for each signal in the traffic light system?

4. Can you explain the algorithm used to control the sequence of signals in the traffic light system?
5. How does the hardware interface with the program to control the lights?
6. What safety measures are implemented in the program to ensure proper functioning of the traffic light system?
7. How does the program handle emergency situations, such as power outages or hardware failures?
8. Can you discuss any optimizations made in the program to improve the efficiency of the traffic light system?
9. How do you ensure synchronization between multiple traffic light systems in a network?
10. What methods are employed to detect and resolve conflicts in the traffic flow using this system?
11. Can you explain the role of sensors, if any, in the traffic light system?
12. How does the program adapt to varying traffic conditions, such as rush hour or late-night traffic?
13. What considerations are taken into account regarding visibility and readability of the signals?
14. Can you discuss any environmental considerations in the design and implementation of the traffic light system?
15. How do you test the reliability and accuracy of the program in simulating real-world traffic scenarios?

DEPARTMENT OF ELECTRICAL ENGINEERING

Embedded System
Lab

Experiment No: 9

Write a program for interfacing finger print sensor

TITLE: Write a program for interfacing finger print sensor

AIM

Write a program for interfacing finger print sensor.

APPRATUS

1) Ardunio kit
2) DSO
3) GT511C3 Finger Print Sensor
4) 16x2 LCD screen
5) Pot – 10k and 1k,10k,22k resistors
6) Push button
7) Kit power supply or Dual power supply with +/- 12volt/ 200mA.
8) Proteus Software (Test in software form)
9) Connecting code and DSO Probe

THEORY

A) Introduction

Biometrics has been used as a reliable authentication system for a long time now. Today there exist complex biometric systems which can identify a person by his heart beat rhythm or even by his DNA. Other feasible methods include voice recognition, Face recognition, Iris scanning and Finger print Scanning. Out of which the finger print recognition is the most widely used method, we can find it from a simple attendance system to smart phones to Security checks and much more.

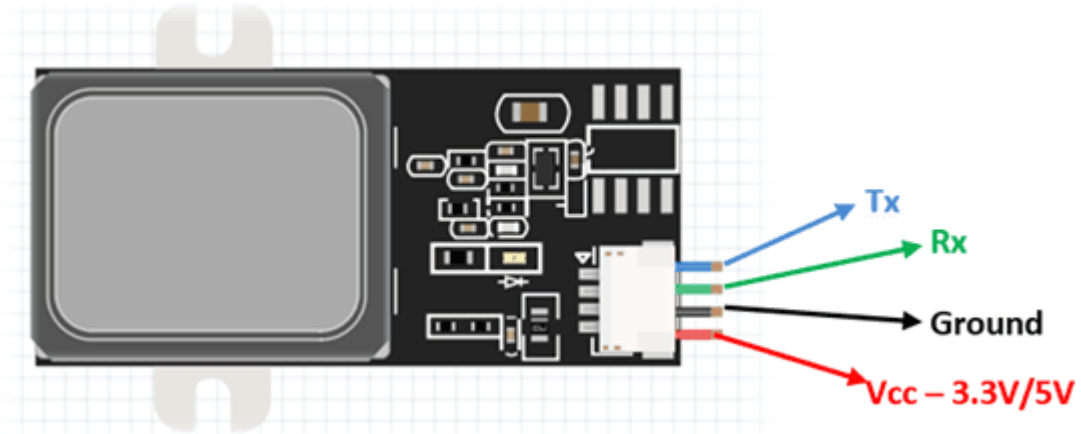
In this experiment we will learn how to use the popular **GT511C3 Finger Print Sensor (FPS) with Arduino**. There are many FPS available and we have already learnt how to use them to build designs like Attendance system, Voting Machine, Security system etc. But the GT511C3 is more advanced with high accuracy and faster response time, so we will learn **how to use it with Arduino to enroll finger prints on it and then detect the fingerprints whenever required**. So let's get started.

B) GT511C3 Fingerprint Sensor (FPS) Module

Before diving into the project let us understand about the **GT511C3 fingerprint sensor Module** and how it works. This sensor is very different form the Capacitive and Ultrasonic Fingerprint sensor that are commonly used in our smart phones. The **GT511C3 is an optical Fingerprint sensor**, meaning it relies on images of your fingerprint to recognize its pattern. Yes you read that right, the

sensor actually has a camera inside it which takes pictures of your fingerprint and then processes these images using powerful in-built ARM Cortex M3 IC. The below image shows the front and back side of the sensor with pinouts.

As you can see the sensor has a camera (black spot) surrounded by blue LEDs, these LEDs have to be lit up to take a clear image of the fingerprint. These images are then processed and converted into binary value by using the **ARM Microcontroller** coupled with EEPROM. The module also has a green color SMD LED to indicate power. Each fingerprint image is of 202x258 pixels with a resolution of 450dpi. The **sensor can enroll upto 200 fingerprints and for each finger print template it assigns an ID form 0 to 199**. Then during detection it can automatically compare the scanned fingerprint with all 200 templates and if a match is found it gives the ID number of that particular fingerprint using the **Smack Finger 3.0** Algorithm on the ARM Microcontroller. The sensor can operate from 3.3V to 6V and communicates through Serial communication at 9600. The communication pins (Rx and Tx) is said to be only 3.3V tolerant, however the datasheet does not specify much about it. The pin-out of a GT511C3 FPS is shown below.



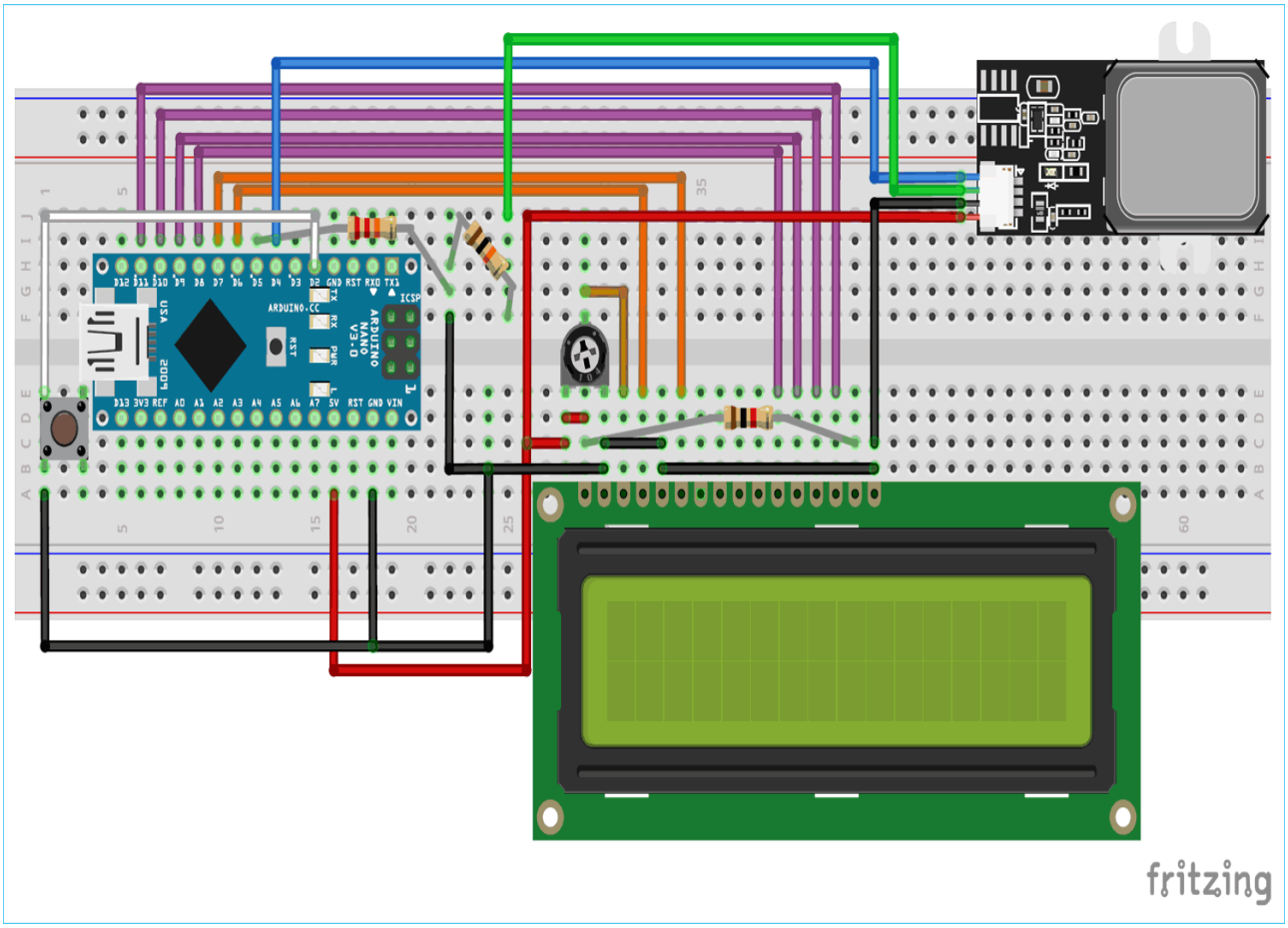
Apart from serial communication the **module can also be directly interfaced to computer though USB connection** using the pins shown in previous image. Once connected to computer the module can be controlled using the SDK_DEMO.exe application which can be downloaded from the link. This application allows the user to enroll/verify/delete fingerprints and also to recognize fingerprints. The software can also help you to read the image captured by the sensor which is worth giving it a try. Alternatively you can also use this Software even if the sensor is connected with Arduino, we will discuss on this later in this article.

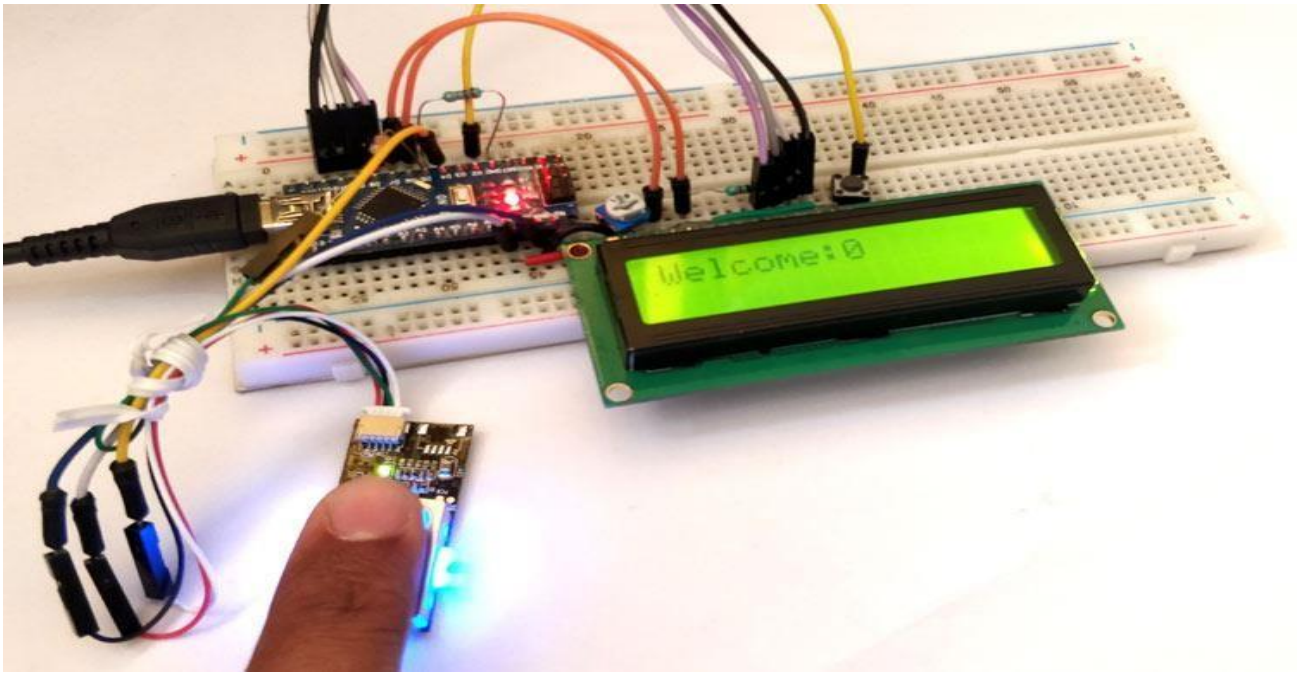
Another interesting feature about the sensor is the metal casing around sensing region. As I told earlier the **blue LED has to be turned on for the sensor to work**. But in applications where the sensor should actively wait for a fingerprint it is not possible to keep the LED turned on always since it will heat up the sensor and thus damage it. Hence in those cases the **metal casing can be wired to a capacitive touch input pin of a MCU to detect if it is being touched**. If yes the LED can be turned on and the sensing process can be started. This method is not demonstrated here as it is outside the scope of this article.

When operating at 3.3V the sensor consumes about 130mA. It requires nearly 3 seconds for enrolling a finger and 1 second to identify it. However if the enrolled template count is less the recognition speed will be high. For more details about the sensor you can refer to this datasheet from ADH-Tech who is the official manufacturer of the module. There are different types of infrared transmitters depending on their wavelengths, output power and response time. A simple infrared transmitter can be constructed using an infrared LED, a current limiting resistor and a power supply. The schematic of a typical IR transmitter is shown below.

CIRCUIT DIAGRAM

The GT511C3 FPS has two power pins which can be powered by +5V pin of Arduino and two communication pins Rx and Tx which can be connected to any digital pin of Arduino for serial communication. Additionally we have also added a push button and a LCD to display the sensor status. The complete circuit diagram for **interfacing GT511C3 FPS with Arduino** can be found below.





Since the Rx and Tx pins are 3.3V tolerant we have used a potential divider on the Rx side to convert 5V to 3.3V. The 10k resistor and 22k resistor converts the 5V signal from the Arduino Tx pin to 3.3V before it reaches the Rx pin of the FPS. The Sensor can also be powered by 3.3V but make sure your Arduino can source enough current for the sensor. We have connected the LCD in 4-bit mode powered by 5V pin of Arduino. A push button is connected to pin D2 which when pressed **will put the program in enroll mode** where the user can enroll new finger. After enrolling the program will remain in scanning mode to scan for any finger touching the sensor.

PROGRAM & PROCEDURE

Our aim here is to write a program that will enroll a finger when a button is pressed and display the ID number of the finger that is already enrolled. We should also be able to display all information on the LCD to enable the project to be a stand-alone one. The complete code to do the same is give at the bottom of this page. Here I am breaking the same into small snippets to help you understand better. As always we begin the program by including the required libraries, here we will need the FPS_GT511C3 library for our FPS module, Software serial to use D4 and D5 on serial communication and Liquid crystal for LCD interfacing. Then we need to mention to which pins the FPS and LCD is connected to. If you had followed the circuit diagram as such then it is 4 and 5 for FPS and D6 to D11 for LCD. The code for the same is shown below

```
#include "FPS_GT511C3.h" //Get library from https://github.com/sparkfun/Fingerprint_Scanner-TTL
#include "SoftwareSerial.h" //Software serial library
#include <LiquidCrystal.h> //Library for LCD
FPS_GT511C3 fps(4, 5); //FPS connected to D4 and D5
const int rs = 6, en = 7, d4 = 8, d5 = 9, d6 = 10, d7 = 11; //Mention the pin number for LCD connection
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);//Initialize LCD method
```

Inside the *setup* function, we display some introductory message on the LCD and then initialize the FPS module. The command fps.SetLED(true) will turn on the blue LED on the sensor, you can turn it off by fps.SetLED(false) when not required as it would heat up the sensor if left on continuously. We have also made the pin D2 as input pin and connected it to internal pull-up resistor so as to connect a push button to the pin.

```
void setup()
{
  Serial.begin(9600);
  lcd.begin(16, 2); //Initialise 16*2 LCD
  lcd.print("GT511C3 FPS"); //Intro Message line 1
  lcd.setCursor(0, 1);
  lcd.print("with Arduino"); //Intro Message line 2
  delay(2000);
  lcd.clear();
  fps.Open();         //send serial command to initialize fp
  fps.SetLED(true);   //turn on LED so fps can see fingerprint
  pinMode(2,INPUT_PULLUP); //Connect to internal pull up resistor as input pin
}
```

Inside the *void loop* function we have to check if the button is pressed, if pressed we will enroll a new finger and save its template with an ID number by using the enroll function. If not we will keep waiting for a finger to be pressed in the sensor. If pressed we will indentify the fingerprint by comparing it to all enrolled fingerprints template using the 1:N method. Once the ID number is discovered we will display welcome followed by the ID number. If the finger print did not match with any of the enrolled fingers the id count will be 200, in that case we will display welcome unknown.

.

```
if (digitalRead(2))//If button pressed
  {
  Enroll(); //Enroll a fingerprint
  }
// Identify fingerprint test
  if (fps.IsPressFinger())
  {
    fps.CaptureFinger(false);
    int id = fps.Identify1_N();
      lcd.clear();
      lcd.print("Welcome:");
      if (id==200) lcd.print("Unkown  "); //If not recognisedlcd.print(id);
      delay(1000);
  }
```

The *enroll* function would have to take three sample inputs to enroll one finger successfully. Once enrolled a template for that particular finger will be created which will not be deleted unless the user forced it though HEX commands. The code to enroll a finger is shown below. The method *Is Press Finger* is used to check if a finger is detected, if yes then the image is captured using *Capture Finger* and then finally Enroll1, Enroll2 and Enroll3 is used for three different samples to successfully enroll one finger.

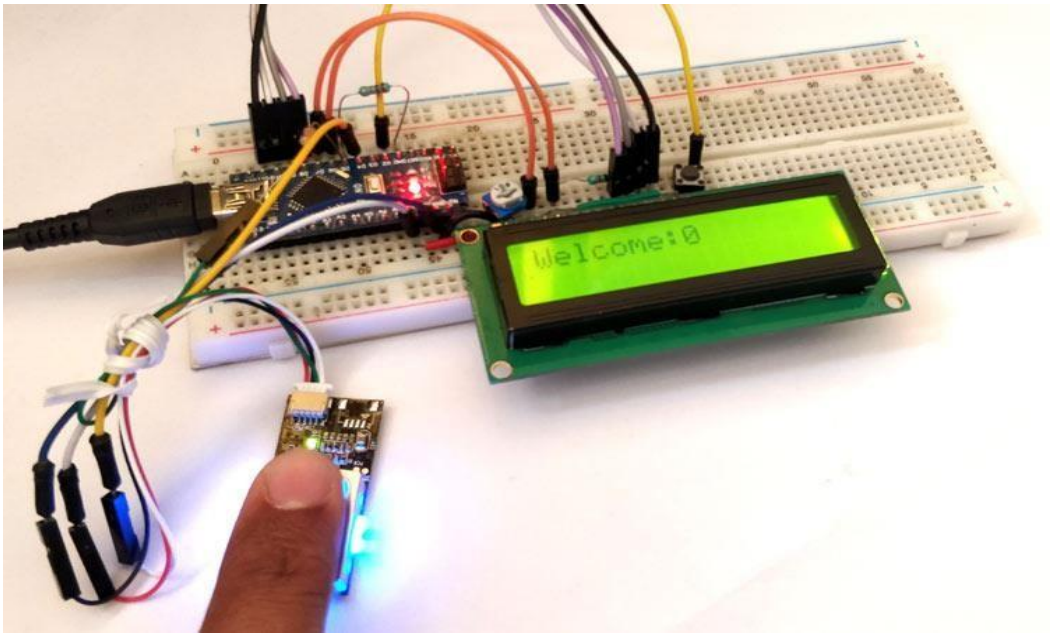
The LCD displays the ID number of the finger if enrolled successfully else it would display a failure message with code.Code 1 means the finger print was not captured clearly and hence you have to try again. Code 2 is a memory fail indication and code 3 is to indicate that the finger has already been enrolled.

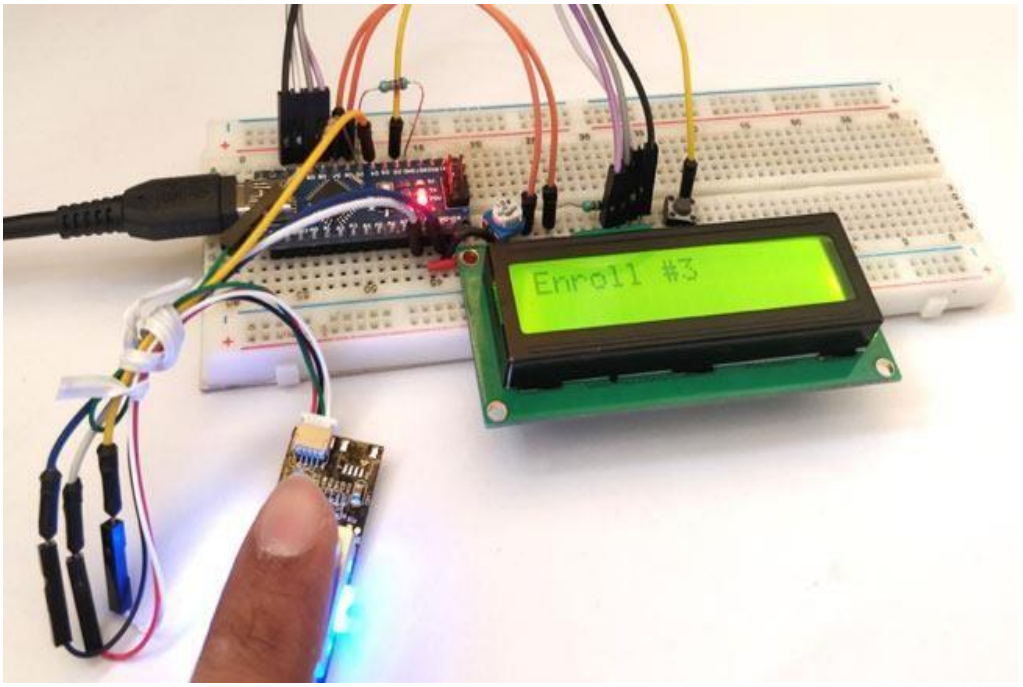
```
void Enroll() //Enrol function from library exmaple program
{
  int enrollid = 0;
  bool usedid = true;
  while (usedid == true)
  {
    usedid = fps.CheckEnrolled(enrollid);
    if (usedid==true) enrollid++;
  }
  fps.EnrollStart(enrollid);
  // enroll
  lcd.print("Enroll #");
  lcd.print(enrollid);
  while(fps.IsPressFinger() == false) delay(100);
  bool bret = fps.CaptureFinger(true);
  int iret = 0;
  if (bret != false)
  {
    lcd.clear();
    lcd.print("Remove finger");
    fps.Enroll1();
    while(fps.IsPressFinger() == true) delay(100);
    lcd.clear(); lcd.print("Press again");
    while(fps.IsPressFinger() == false) delay(100);
    bret = fps.CaptureFinger(true);
    if (bret != false)
    {
      lcd.clear(); lcd.print("Remove finger");
      fps.Enroll2();
      while(fps.IsPressFinger() == true) delay(100);
       lcd.clear(); lcd.print("Press yet again");
       while(fps.IsPressFinger() == false) delay(100);
       bret = fps.CaptureFinger(true);
       if (bret != false)
       {
         lcd.clear(); lcd.print("Remove finger");
         iret = fps.Enroll3();
         if (iret == 0)
         {
           lcd.clear(); lcd.print("Enrolling Success");
         }
         else
         {
           lcd.clear();
           lcd.print("Enroll Failed:");
           lcd.print(iret);
         }
       }
       else lcd.print("Failed 1");
     }
     else lcd.print("Failed 2");
   }
   else lcd.print("Failed 3");
}
```

WORKING

Working of GT511C3 Finger Print Sensor with Arduino

Upload the code to Arduino and power it up, I am just using the micro-usb port to power the project. On booting we should see the intro message on the LCD and then it should display "Hi!". This means that FPS is ready to scan for finger, if any enrolled finger is pressed it would say "Welcome" followed by the ID number of that finger as shown below. If a new finger has to be enrolled then we can **use the push button to get into enroll mode** and follow the LCD instruction to enroll a finger. After the enrolling process is complete the LCD will display "Hi!.." again to indicate that it is read to indentify fingers again. The **complete working can be found at the video linked below**.





CODE

```
* Arduino with GT511C2 FingerPrint Sensor (FPS)
 * Code to enroll and Detect Fingers
 * For: www.circuitdigest.com
 * Dated: 6-5-19
 * Code By: Aswinth
 *
 * Connect Tx of FPS to Arduino Pin D4 and Rx of FPS to D5
 */
#include "FPS_GT511C3.h" //Get library from https://github.com/sparkfun/Fingerprint_Scanner-TTL
#include "SoftwareSerial.h" //Software serial library
#include <LiquidCrystal.h> //Library for LCD
FPS_GT511C3 fps(4, 5); //FPS connected to D4 and D5
const int rs = 6, en = 7, d4 = 8, d5 = 9, d6 = 10, d7 = 11; //Mention the pin number for LCD connection
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);//Initialize LCD method
void setup()
{
  Serial.begin(9600);
  lcd.begin(16, 2); //Initialise 16*2 LCD
  lcd.print("GT511C3 FPS"); //Intro Message line 1
  lcd.setCursor(0, 1);
  lcd.print("with Arduino"); //Intro Message line 2
  delay(2000);
  lcd.clear();
  fps.Open();         //send serial command to initialize fps
  fps.SetLED(true); //turn on LED so fps can see fingerprint
  pinMode(2,INPUT_PULLUP); //Connect to internal pull up resistor as input pin
}
void loop()
{
  if (digitalRead(2)==0)//If button pressed
  {
  Enroll(); //Enroll a fingerprint
  }

// Identify fingerprint test
  if (fps.IsPressFinger())
  {
    fps.CaptureFinger(false);
    int id = fps.Identify1_N();
    lcd.clear();
    lcd.print("Welcome:");
      if (id==200) lcd.print("Unkown "); //If not recognised
      lcd.print(id);
      delay(1000);
  }
  else
  {
    lcd.clear(); lcd.print("Hi! ...."); //Display hi when ready to scan
  }
}
void Enroll() //Enrol function from library exmaple program
{
  int enrollid = 0;
  bool usedid = true;
  while (usedid == true)
  {
    usedid = fps.CheckEnrolled(enrollid);
    if (usedid==true) enrollid++;
  }
  fps.EnrollStart(enrollid);
```

```
 // enroll
 lcd.clear();
 lcd.print("Enroll #");
 lcd.print(enrollid);
 while(fps.IsPressFinger() == false) delay(100);
 bool bret = fps.CaptureFinger(true);
 int iret = 0;
 if (bret != false)
 {
  lcd.clear();
  lcd.print("Remove finger");
  fps.Enroll1();
  while(fps.IsPressFinger() == true) delay(100);
  lcd.clear(); lcd.print("Press again");
  while(fps.IsPressFinger() == false) delay(100);
  bret = fps.CaptureFinger(true);
  if (bret != false)
  {
   lcd.clear(); lcd.print("Remove finger");
   fps.Enroll2();
   while(fps.IsPressFinger() == true) delay(100);
   lcd.clear(); lcd.print("Press yet again");
   while(fps.IsPressFinger() == false) delay(100);
   bret = fps.CaptureFinger(true);
   if (bret != false)
   {
    lcd.clear(); lcd.print("Remove finger");
    iret = fps.Enroll3();
    if (iret == 0)
    {
     lcd.clear(); lcd.print("Enrolling Success");
    }
    else
    {
     lcd.clear();
     lcd.print("Enroll Failed:");
     lcd.print(iret);
    }
   }
   else lcd.print("Failed 1");
  }
  else lcd.print("Failed 2");
 }
 else lcd.print("Failed 3");
}
```

CONCLUSION

Successfully design the circuit of interfacing finger print sensor in Proteus software and implement on hardware at Adriano microcontroller kit.

VIVA QUESTIONS:

1. Can you explain the basic working principle of a fingerprint sensor?
2. What are the main components required to interface a fingerprint sensor with a microcontroller?
3. How does the fingerprint recognition process occur in the sensor?
4. What considerations should be taken into account when choosing a fingerprint sensor for interfacing with a microcontroller?
5. Can you outline the steps involved in setting up the hardware for interfacing a fingerprint sensor with a microcontroller?
6. How does the microcontroller communicate with the fingerprint sensor? Describe the

communication protocol.

7. What are the key challenges in interfacing a fingerprint sensor with a microcontroller, and how can they be overcome?
8. What programming language and IDE would you recommend for writing the program to interface with the fingerprint sensor? Why?
9. What are the different methods for storing and managing fingerprint data in the microcontroller's memory?
10. How can you ensure the security and integrity of the fingerprint data stored in the microcontroller?
11. Can you explain the steps involved in capturing and processing fingerprint data in the program?
12. How can you optimize the performance of the fingerprint recognition algorithm in terms of speed and accuracy?
13. What measures can be taken to handle errors and exceptions that may occur during fingerprint recognition?
14. How would you test the functionality and reliability of the program for interfacing with the fingerprint sensor?
15. Can you suggest any potential enhancements or additional features that could be implemented in the program to improve its usability or security?

DEPARTMENT OF ELECTRICAL ENGINEERING

Embedded System
Lab

Experiment No: 10

Write a program for Master Slave Communication between using suitable hardware and using SPI

TITLE: Write a program for Master Slave Communication between using suitable hardware and using SPI

AIM

Write a program for Master Slave Communication between using suitable hardware and using SPI

APPRATUS

1) Ardunio kit
2) DSO
3) Breadboard
4) Resistor
5) Kit power supply or Dual power supply with +/- 12volt/ 200mA.
6) Proteus Software
7) Connecting code and DSO Probe

THEORY

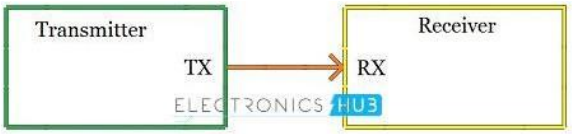
A) Introduction

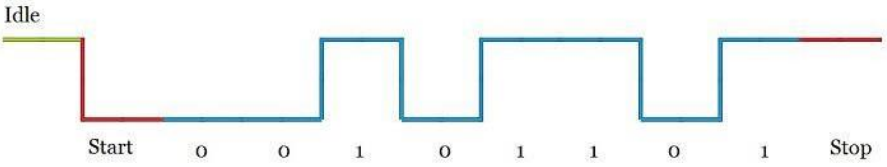
Serial Peripheral Interface or SPI is a synchronous serial communication protocol that provides full – duplex communication at very high speeds. Serial Peripheral Interface (SPI) is a master – slave type protocol that provides a simple and low cost interface between a microcontroller and its peripherals. SPI Interface bus is commonly used for interfacing microprocessor or microcontroller with memory like EEPROM, RTC (Real Time Clock), ADC (Analog – to – Digital Converters), DAC (Digital – to – Analog Converters), displays like LCDs, Audio ICs, sensors like temperature and pressure, memory cards like MMC or SD Cards or even other microcontrollers.

We have seen about UART in the previous article. In UART (or any common serial port), where the communication happens over RX and TX line, there is no clock signal i.e. it is an asynchronous communication. In this type of communication, there is no control over the data sent or whether the transmitter and receiver have same data rates.

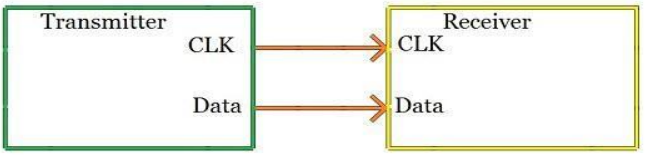
In order to overcome this, UART uses synchronisation bits i.e. Start bit and Stop bits and also a pre agreed data transfer speeds (typically 9600 bps). If the baud rates of transmitter and receiver are not matched, the data sent from the transmitter will not reach the receiver properly and often garbage or junk values are received.

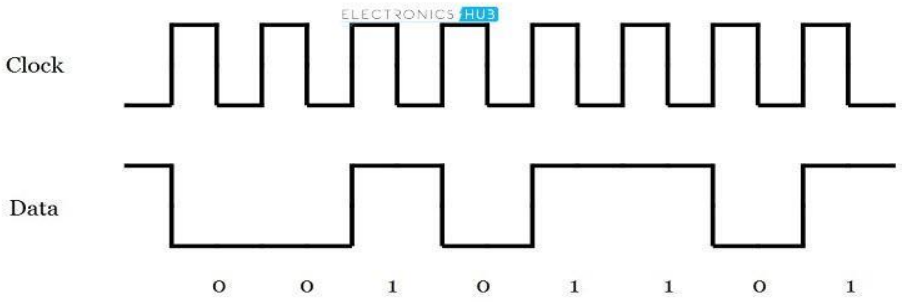
For short distance communication, Synchronous Serial Communication would be a better choice and in that Serial Peripheral Interface or SPI in particular is the best choice. When we say short distance communication, it often means communication with in a device or between the devices on the same board (PCB).





The other type of Synchronous Serial Communication Protocol is I2C (Inter – Integrated Communication, often called as I Squared C or I Two C). For this article, we will focus on SPI. SPI is a Synchronous type serial communication i.e. it uses a dedicated clock signal to synchronise the transmitter and receiver or Master and Slave, speaking in SPI terms. The transmitter and receiver are connected with separate data and clock lines and the clock signal will help the receiver when to look for data on the bus.





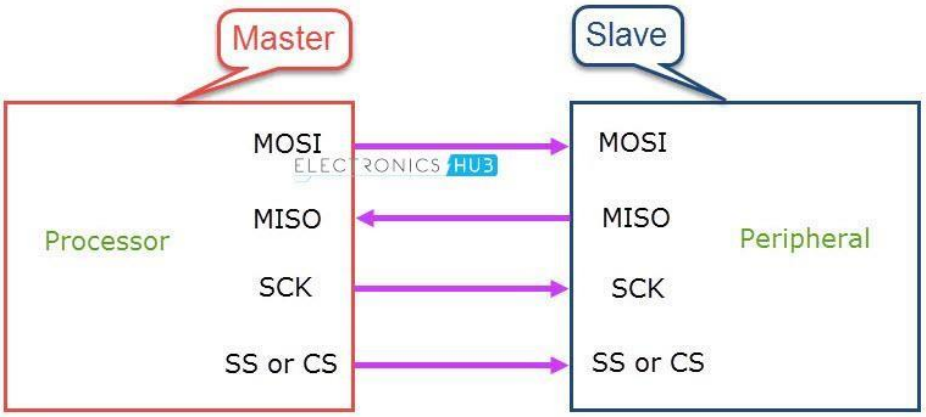
The clock signal must be supplied by the Master to the slave (or all the slaves in case of multiple slave setup). There are two types of triggering mechanisms on the clock signal that are used to intimate the receiver about the data: Edge Triggering and Level Triggering. The most commonly used triggering is edge triggering and there are two types: rising edge (low to high transition on the clock) and falling edge (high to low transition). Depending on how the receiver is configured, up on detecting the edge, the receiver will look for data on the data bus from the next bit. Since both the clock and data are sent by the Master (or transmitter), we need not worry about the speed of data transfer. What makes SPI so popular among other Synchronous Serial Communication protocols (or any serial communication for that matter) is that it provides a high speed secured data transfer with reasonably simple hardware like shift registers at relatively less cost. SPI or Serial Peripheral Interface was developed by Motorola in the 1980's as a standard, low – cost and reliable interface between the Microcontroller (microcontrollers by Motorola in the beginning) and its peripheral ICs. Because of its simple interface, flexibility and ease of use, SPI has become a standard and soon other semiconductor manufacturers started implementing it in their chips. In SPI protocol, the devices are connected in a Master – Slave relationship in a multi – point interface. In this type of interface, one device is considered the Master of the bus (usually a Microcontroller) and all the other devices (peripheral ICs or even other Microcontrollers) are considered as slaves.

SPI protocol, there can be only one master but many slave devices.The SPI bus consists of 4 signals or pins. They are

Master – Out / Slave – In (MOSI)
Master – In / Slave – Out (MISO)
Serial Clock (SCLK) and
Chip Select (CS) or Slave Select (SS)

Since, the SPI bus is implemented using 4 signals or wires, it is sometimes called as Four Wire Interface. Let us first see a simple interface between a single master and single slave that are connected using SPI protocol and then we will explain about the 4 wires.

The following image depicts a Master (Processor) connected to a Slave (Peripheral) using SPI bus.

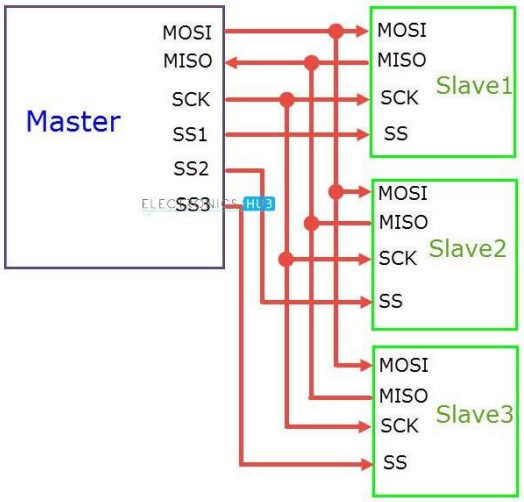


Master – Out / Slave – In or MOSI, as the name suggests, is the data generated by the Master and received by the Slave. Hence, MOSI pins on both the master and slave are connected together. Master – In / Slave – Out or MISO is the data generated by Slave and must be transmitted to Master.

MISO pins on both the master and slave are ties together. Even though the Signal in MISO is produced by the Slave, the line is controlled by the Master. The Master generates a clock signal at SCLK and is supplied to the clock input of the slave. Chip Select (CS) or Slave Select (SS) is used to select a particular slave by the master. Since the clock is generated by the Master, the flow of data is controlled by the master. For every clock cycle, one bit of data is transmitted from master to slave and one bit of data is transmitted from slave to master.

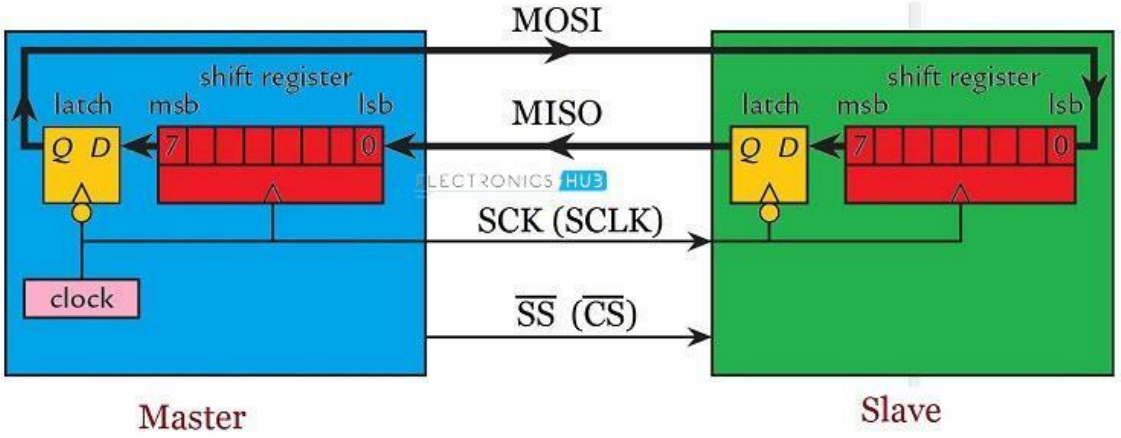
This process happen simultaneously and after 8 clock cycles, a byte of data is transmitted in both directions and hence, SPI is a full – duplex communication. If the data has to be transmitted by only one device, then the other device has to send something (even garbage or junk data) and it is up to the device whether the transmitted data is actual data or not. This means that for every bit transmitted by one device, the other device has to send one bit data i.e. the Master simultaneously transmits data on MOSI line and receive data from slave on MISO line. If the slave wants to transmit the data, the master has to generate the clock signal accordingly by knowing when the slave wants to send the data in advance. If more than one slave has to be connected to the master, then the setup will be something similar to the following image.

Even though multiple slaves are connected to the master in the SPI bus, only one slave will be active at any time. In order to select the slave, the master will pull down the SS (Slave Select) or CS (Chip Select) line of the corresponding slave. Hence, there must by a separate CS pin on the Master corresponding to each of the slave device. We need to pull down the SS or CS line to select the slave because this line is active low.



SPI Hardware

The hardware requirement for implementing SPI is very simple when compared to UART and I2C. Consider a Master and a single Slave are connected using SPI bus. The following image shows the minimal system requirements for both the devices.



From the image, the Master device consists of a Shift Register, a data latch and a clock generator. The slave consists of similar hardware: a shift register and a data latch. Both the shift registers are connected to form a loop. Usually, the size of the register is 8 – bits but higher size registers of 16 – bits are also common. During the positive edge of the clock signal, both the devices (master and slave) read input bit into LSB of the register. During the negative cycle of the clock signal, both the master and slave places a bit on its corresponding output from the MSB of the shift register. Hence, for each clock cycle, a bit of data is transferred in each direction i.e. from master to slave and slave to master. So, for a byte of data to be transmitted from each device, it will take 8 clock cycles.

SPI Modes of Operation

We have already seen that it is the job of the Master device to generate the clock signal and distribute it to the slave in order to synchronise the data between master and slave. The work of master doesn't end at generating clock signal at a particular frequency. In fact, the master and slave have to agree on certain synchronization protocols. For this, two features of the clock i.e. the Clock Polarity (CPOL or CKP) and Clock Phase (CPHA) come in to picture. Clock Polarity determines the state of the clock. When CPOL is LOW, the clock generated by the Master i.e. SCK is LOW when idle and toggles to HIGH during active state (during a transfer).

Similarly, when CPOL is HIGH, SCK is HIGH during idle and LOW during active state. Clock Phase determines the clock transition i.e. rising (LOW to HIGH) or falling (HIGH to LOW), at which the data is transmitted. When CPHA is 0, the data is transmitted on the rising edge of the clock. Data is transmitted on the falling edge when CPHA is 1. Depending on the values of Clock Polarity (CPOL) and Clock Phase (CPHA), there are 4 modes of operation of SPI: Modes 0 through 3.

Mode 0:

Mode 0 occurs when Clock Polarity is LOW and Clock Phase is 0 (CPOL = 0 and CPHA = 0). During Mode 0, data transmission occurs during rising edge of the clock.

Mode 1:

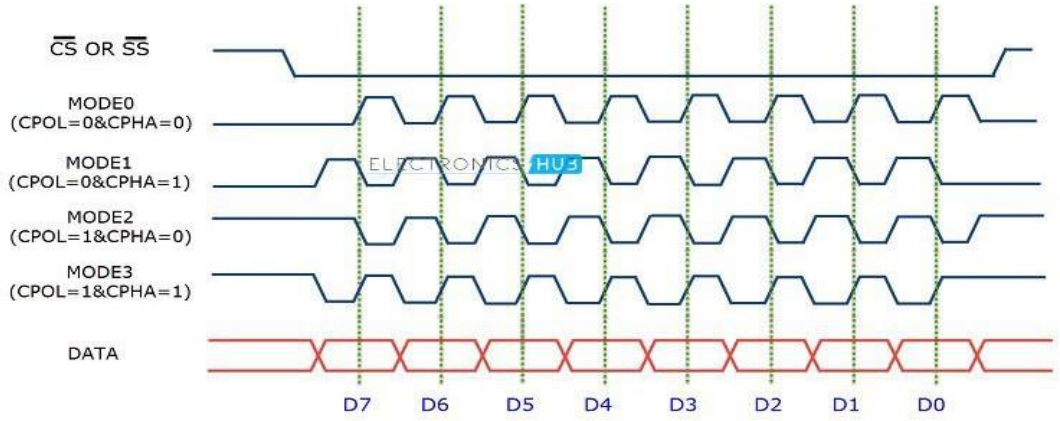
Mode 1 occurs when Clock Polarity is LOW and Clock Phase is 1 (CPOL = 0 and CPHA = 1). During Mode 1, data transmission occurs during falling edge of the clock.

Mode 2:

Mode 2 occurs when Clock Polarity is HIGH and Clock Phase is 0 (CPOL = 1 and CPHA = 0). During Mode 2, data transmission occurs during rising edge of the clock.

Mode 3:

Mode 3 occurs when Clock Polarity is HIGH and Clock Phase is 1 (CPOL = 1 and CPHA = 1). During Mode 3, data transmission occurs during rising edge of the clock.



SPI WORKING

THE CLOCK

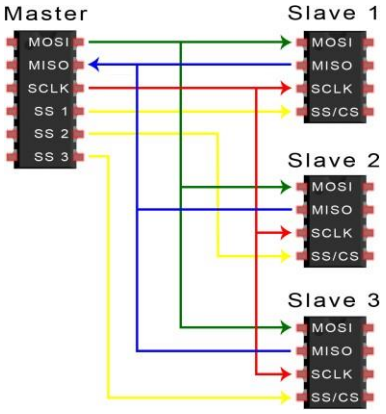
The clock signal synchronizes the output of data bits from the master to the sampling of bits by the slave. One bit of data is transferred in each clock cycle, so the speed of data transfer is determined by the frequency of the clock signal. SPI communication is always initiated by the master since the master configures and generates the clock signal. Any communication protocol where devices share a clock signal is known as *synchronous.* SPI is a synchronous communication protocol. There are also *asynchronous* methods that don't use a clock signal. For example, in UART communication, both sides are set to a pre-configured baud rate that dictates the speed and timing of data transmission. The clock signal in SPI can be modified using the properties of *clock polarity* and *clock phase*. These two properties work together to define when the bits are output and when they are sampled. Clock polarity can be set by the master to allow for bits to be output and sampled on either the rising or falling edge of the clock cycle. Clock phase can be set for output and sampling to occur on either the first edge or second edge of the clock cycle, regardless of whether it is rising or falling.

SLAVE SELECT

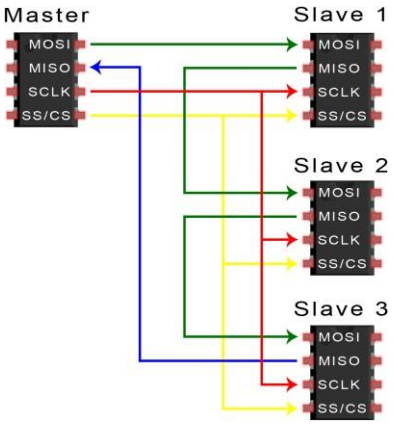
The master can choose which slave it wants to talk to by setting the slave's CS/SS line to a low voltage level. In the idle, non-transmitting state, the slave select line is kept at a high voltage level. Multiple CS/SS pins may be available on the master, which allows for multiple slaves to be wired in parallel. If only one CS/SS pin is present, multiple slaves can be wired to the master by daisy-chaining.

MULTIPLE SLAVES

SPI can be set up to operate with a single master and a single slave, and it can be set up with multiple slaves controlled by a single master. There are two ways to connect multiple slaves to the master. If the master has multiple slave select pins, the slaves can be wired in parallel like this:



If only one slave select pin is available, the slaves can be daisy-chained like this:



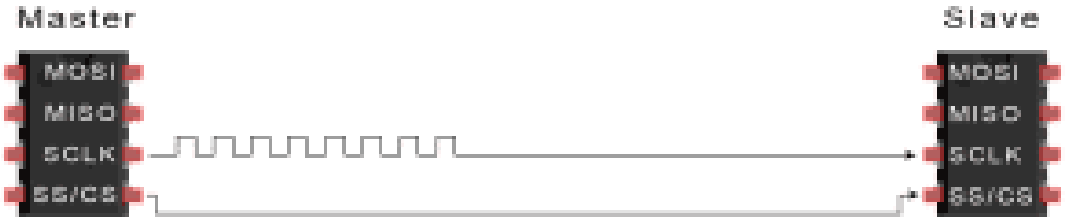
MOSI AND MISO

The master sends data to the slave bit by bit, in serial through the MOSI line. The slave receives the data sent from the master at the MOSI pin. Data sent from the master to the slave is usually sent with the most significant bit first. The slave can also send data back to the master through the MISO line in serial. The data sent from the slave back to the master is usually sent with the least significant bit first.

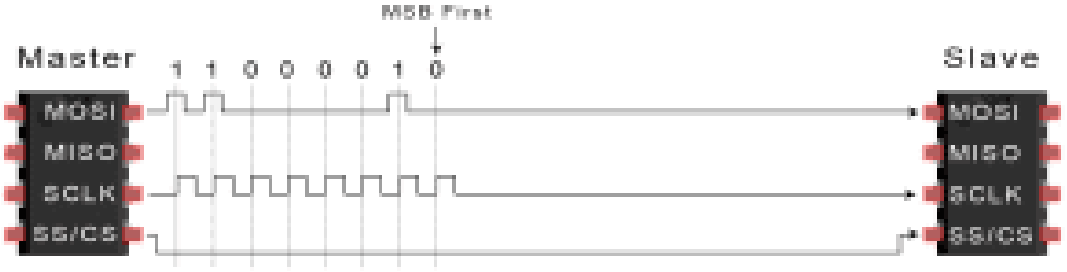
STEPS OF SPI DATA TRANSMISSION

1. The master outputs the clock signal:

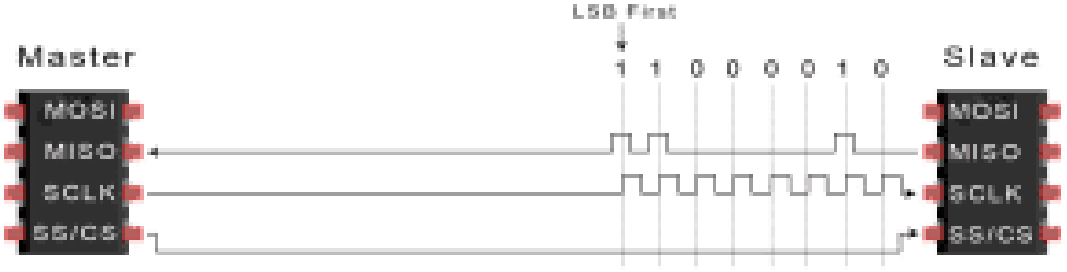
2. The master switches the SS/CS pin to a low voltage state, which activates the slave:



3. The master sends the data one bit at a time to the slave along the MOSI line. The slave reads the bits as they are received:

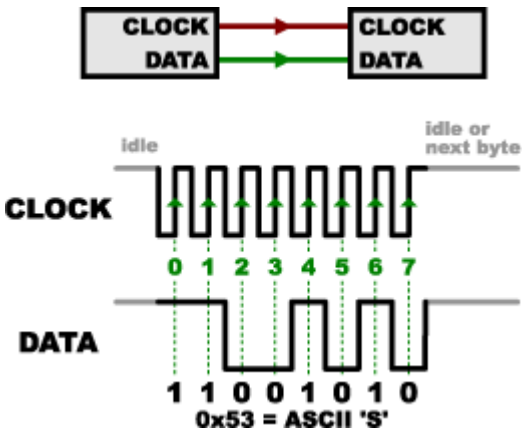


4. If a response is needed, the slave returns data one bit at a time to the master along the MISO line. The master reads the bits as they are received:



DATA WORKING

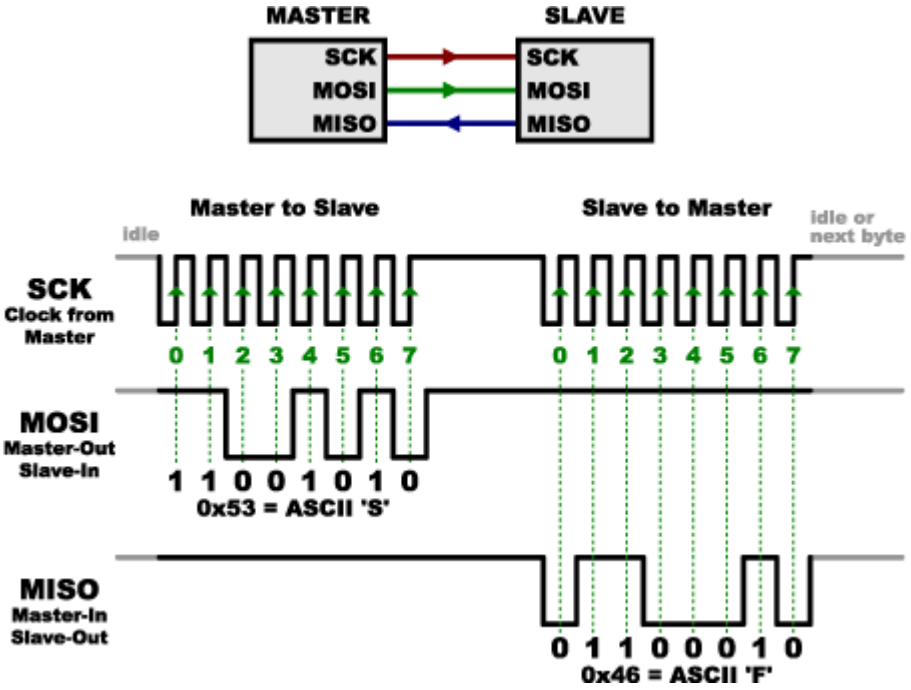
SPI works in a slightly different manner. It's a "synchronous" data bus, which means that it uses separate lines for data and a "clock" that keeps both sides in perfect sync. The clock is an oscillating signal that tells the receiver exactly when to sample the bits on the data line. This could be the rising (low to high) or falling (high to low) edge of the clock signal; the datasheet will specify which one to use. When the receiver detects that edge, it will immediately look at the data line to read the next bit (see the arrows in the below diagram). Because the clock is sent along with the data, specifying the speed isn't important, although devices will have a top speed at which they can operate (We'll discuss choosing the proper clock edge and speed in a bit).



One reason that SPI is so popular is that the receiving hardware can be a simple shift register. This is a much simpler (and cheaper!) piece of hardware than the full-up UART (Universal Asynchronous Receiver / Transmitter) that asynchronous serial requires.

Receiving Data

You might be thinking to yourself, self, that sounds great for one-way communications, but how do you send data back in the opposite direction? Here's where things get slightly more complicated. In SPI, only one side generates the clock signal (usually called CLK or SCK for Serial ClocK). The side that generates the clock is called the "master", and the other side is called the "slave". There is always only one master (which is almost always your microcontroller), but there can be multiple slaves (more on this in a bit). When data is sent from the master to a slave, it's sent on a data line called MOSI, for "Master Out / Slave In". If the slave needs to send a response back to the master, the master will continue to generate a prearranged number of clock cycles, and the slave will put the data onto a third data line called MISO, for "Master In / Slave Out".

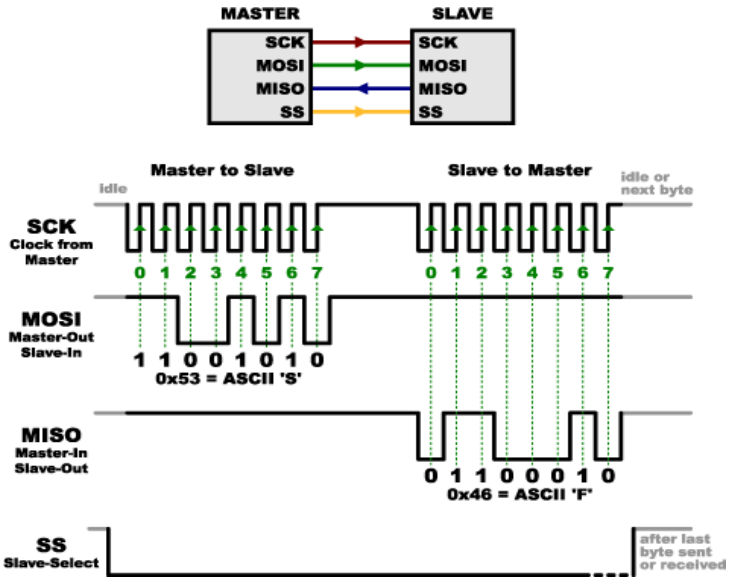


Notice we said "prearranged" in the above description. Because the master always generates the clock signal, it must know in advance when a slave needs to return data and how much data will be returned. This is very different than asynchronous serial, where random amounts of data can be sent in either direction at any time. In practice this isn't a problem, as SPI is generally used to talk to sensors that have a very specific command structure. For example, if you send the command for "read data" to a device, you know that the device will always send you, for example, two bytes in return. (In cases where you might want to return a variable amount of data, you could always return one or two bytes specifying the length of the data and then have the master retrieve the full amount.)

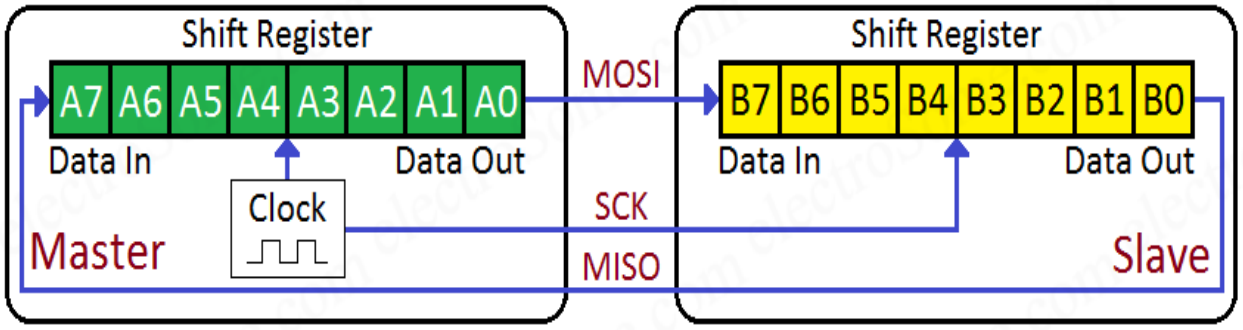
Note that SPI is "full duplex" (has separate send and receive lines), and, thus, in certain situations, you can transmit and receive data at the same time (for example, requesting a new sensor reading while retrieving the data from the previous one). Your device's datasheet will tell you if this is possible.

Slave Select (SS)

There's one last line you should be aware of, called SS for Slave Select. This tells the slave that it should wake up and receive / send data and is also used when multiple slaves are present to select the one you'd like to talk to.

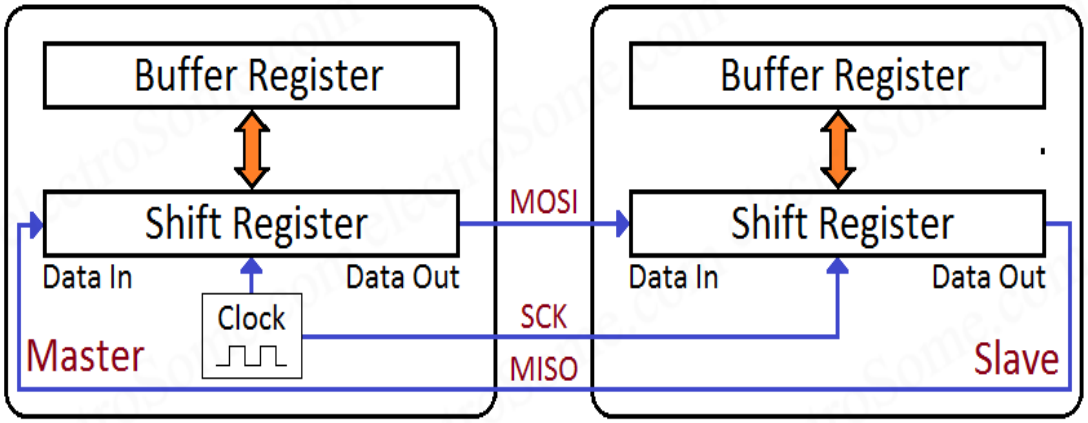


The SS line is normally held high, which disconnects the slave from the SPI bus. (This type of logic is known as “active low,” and you’ll often see used it for enable and reset lines.) Just before data is sent to the slave, the line is brought low, which activates the slave. When you're done using the slave, the line is made high again. In a shift register, this corresponds to the "latch" input, which transfers the received data to the output lines.



We can easily understand the working from the above animation. **Master will generate clock whenever it wants to write data to a Slave device.** After 8 clock pulses data in the master device (A7 ~ A0) is transferred to slave device and data in the slave device (B7 ~ B0) is transferred to the master device.

Buffer



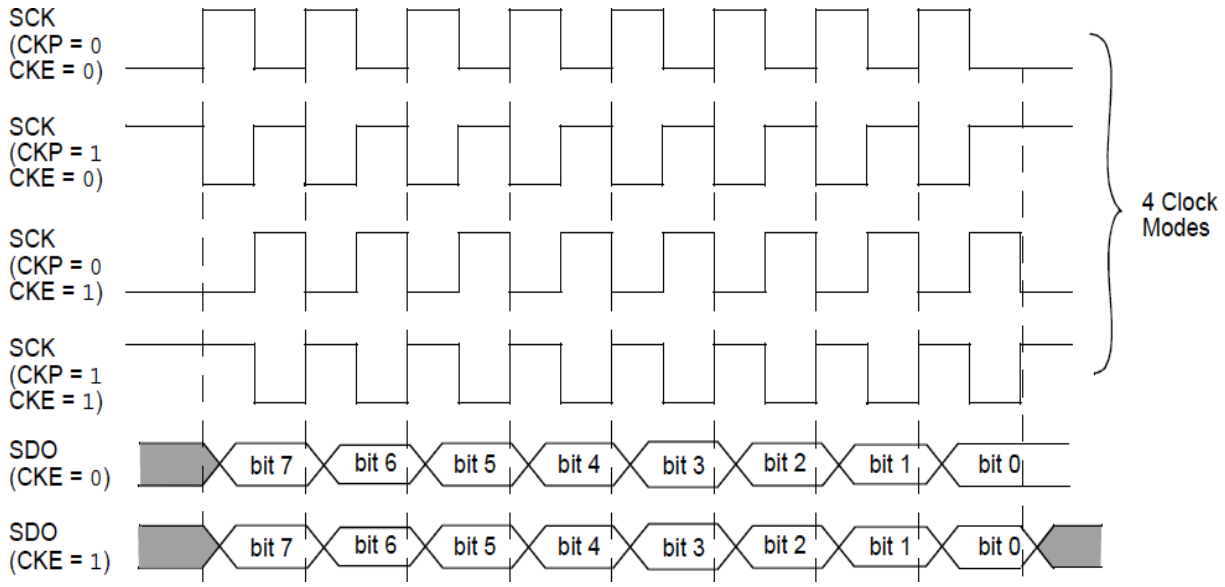
For the sake of explanation, I omitted Buffer Register. It acts as an interface between user (processor, programmer) and SPI. Usually shift register won't be directly accessible. So if we need to transmit data, we will write it to the buffer register. So it will automatically written to shift register when it is free and transmission will start. Similarly data is received in the shift register is automatically transferred to buffer register once the reception is complete. We can easily read from it. Thus buffer register will avoid all glitches that can happen if we try to read or write to shift register directly while transmission is taking place.

SPI Modes – Clock Polarity & Phase

We already seen that clock for data transfer is generated by the SPI master. So the master should set the clock frequency for SPI transfer. In addition to this **clock polarity** and **clock phase** are there, which has to match with SPI slaves for proper data transfer.

Clock Polarity : **CPOL** or **CKP**
Clock polarity is the idle / active state of the clock. If idle state is 0, active state will be 1 and vice versa.

Clock Phase : **CPHA**, Inverted Clock Phase (**Clock Edge**) : **NCPHA** or **CKE**
Clock phase or clock edge defines when to transfer data. Data can be transferred during LOW (0) to HIGH (1) or HIGH to LOW transitions.

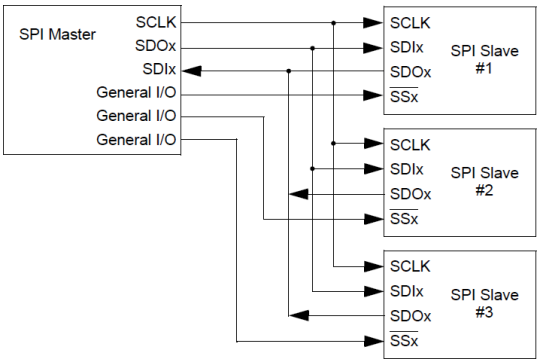


- **Clock Polarity (CKP) = 0**
  This means that the base value of clock is zero. Which implies idle state is 0 and active state is 1.
  - **Clock Edge (CKE) = 0**
    Data transmission occurs during idle to active clock state, ie LOW to HIGH transition.
  - **Clock Edge (CKE) = 1**
    Data transmission occurs during active to idle clock state, ie HIGH to LOW transition
- **Clock Polarity (CKP) = 1**
  This means that the base value of clock is one. Which implies idle state is 1 and active state is 0.
  - **Clock Edge (CKE) = 0**
    Data transmission occurs during idle to active clock state, ie HIGH to LOW transition.
  - **Clock Edge (CKE) = 1**
    Data transmission occurs during active to idle clock state, ie LOW to HIGH transition.

Configurations

Independent Slave Configuration

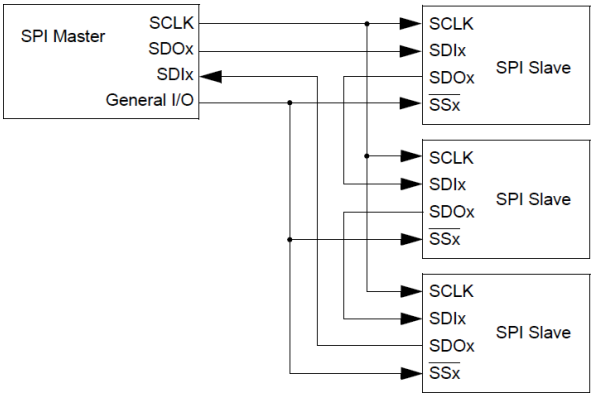
In the independent slave configuration, there is an independent slave select line from master to each slave. SPI is commonly used in this way. SPI slave output (SDOx or MISO) will be a tri-state pin, so it will be in high impedance state when the slave is not selected. In this configuration master can select any slave device and start communication as per requirements.



Daisy Chain Configuration

In this configuration SPI can be connected one after another in a serial form. In this configuration a single slave select line is used to select all daisy chain slaves. Whole chain acts like a communication through shift registers connected in series. Each daisy chain slave is supposed to send out exact copy of data received in the first group of clock cycles during the second group of clock cycles.

This configuration is commonly used in JTAG.



SPI PROGRAMMING

Many microcontrollers have built-in SPI peripherals that handle all the details of sending and receiving data, and can do so at very high speeds. The SPI protocol is also simple enough that you (yes, you!) can write your own routines to manipulate the I/O lines in the proper sequence to transfer data. If you're using an Arduino, there are two ways you can communicate with SPI devices:

1. You can use the shiftIn() and shiftOut() commands. These are software-based commands that will work on any group of pins, but will be somewhat slow.
2. Or you can use the SPI Library, which takes advantage of the SPI hardware built into the microcontroller. This is vastly faster than the above commands, but it will only work on certain pins.
3. You will need to select some options when setting up your interface. These options must match those of the device you're talking to; check the device's datasheet to see what it requires.

4. The interface can send data with the most-significant bit (MSB) first, or least-significant bit (LSB) first. In the Arduino SPI library, this is controlled by the setBitOrder() function.
5. The slave will read the data on either the rising edge or the falling edge of the clock pulse. Additionally, the clock can be considered "idle" when it is high or low. In the Arduino SPI library, both of these options are controlled by the setDataMode() function.
6. SPI can operate at extremely high speeds (millions of bytes per second), which may be too fast for some devices. To accommodate such devices, you can adjust the data rate. In the Arduino SPI library, the speed is set by the setClockDivider() function, which divides the master clock (16MHz on most Arduinos) down to a frequency between 8MHz (/2) and 125kHz (/128).
7. If you're using the SPI Library, you must use the provided SCK, MOSI and MISO pins, as the hardware is hardwired to those pins. There is also a dedicated SS pin that you can use (which must, at least, be set to an output in order for the SPI hardware to function), but note that you can use any other available output pin(s) for SS to your slave device(s) as well.
8. On older Arduinos, you'll need to control the SS pin(s) yourself, making one of them low before your data transfer and high afterward. Newer Arduinos such as the Due can control each SS pin automatically as part of the data transfer; see the Due SPI documentation page for more information.

ADVANTAGES AND DISADVANTAGES OF SPI

There are some advantages and disadvantages to using SPI, and if given the choice between different communication protocols, you should know when to use SPI according to the requirements of your project:

ADVANTAGES

- No start and stop bits, so the data can be streamed continuously without interruption
- No complicated slave addressing system like I2C
- Higher data transfer rate than I2C (almost twice as fast)
- Separate MISO and MOSI lines, so data can be sent and received at the same time
- Simple hardware
- Full duplex communication
- Simple software implementation
- High Speed
- No speed limit (practically it will be limited by the clock frequency, rise time, fall time etc.)
- Not Limited to 8 bit data
- Signals are unidirectional through all lines, makes easy isolation
- No need of unique address in slaves like in RS485 or I2C.
- No need of precision oscillators in slave devices as it uses master’s clock
- No complex transceivers are required

DISADVANTAGES

- Uses four wires (I2C and UARTs use two)
- No acknowledgement that the data has been successfully received (I2C has this)

- No form of error checking like the parity bit in UART
- Only allows for a single master
- More pins/wires are required. Minimum 3 wires (in single slave) are required.
- Can be used only from short distances
- No error detection protocol is defined
- Usually supports only one master

Applications

- SD Cards
- LCD Displays
- RTC
- Ethernet Controllers

CONCLUSION

Successfully Study the program for Master Slave Communication between using suitable hardware and using SPI

VIVA QUESTIONS

1. Can you explain the concept of Master-Slave communication in the context of SPI (Serial Peripheral Interface)?
2. What are the key components required for implementing Master-Slave communication using SPI?
3. How does SPI differ from other communication protocols like I2C or UART?
4. What are the advantages of using SPI for Master-Slave communication?
5. Describe the hardware setup required for establishing Master-Slave

communication using SPI.

6. What role does the Master device play in SPI communication, and how does it differ from the Slave device?
7. How is data transferred between the Master and Slave devices in SPI communication?
8. What are the different modes of SPI communication, and how do they affect data transfer?
9. Can you explain the significance of clock polarity and phase in SPI communication?
10. How is data synchronization ensured between the Master and Slave devices in SPI communication?
11. What considerations should be taken into account while selecting suitable hardware for implementing SPI communication?
12. Discuss the potential challenges or limitations associated with SPI communication in a Master-Slave setup.
13. How does the choice of microcontroller or microprocessor impact the implementation of SPI communication?

14. Can you explain any alternative methods or protocols that can be used for Master-Slave communication besides SPI?
15. What are some practical applications or use cases where Master-Slave communication using SPI is commonly employed?

DEPARTMENT OF ELECTRICAL ENGINEERING

Embedded System Lab

Experiment No: 11

Write a program for variable frequency square wave generation using with suitable hardware.

TITLE: Write a program for variable frequency square wave generation using with suitable hardware

AIM

Write a program for variable frequency square wave generation using with suitable hardware.

APPRATUS

1) Ardunio kit
2) DSO
3) Variable Resistor (10K)
4) Resistor
5) Switch
6) Kit power supply or Dual power supply with +/- 12volt/ 200mA.
7) Proteus Software
8) Connecting code and DSO Probe

THEORY

A) Generating a square wave:

A square wave with frequency f = 1 kHz (kilohertz) is shown in the Figure below. This is a periodic signal with period T = 1/f = 1/1000 = 1 millisecond. The amplitude of the signal is 5 V (volts). This signal can be generated by making a bit "1" when "high" voltage is required and "0" when "low" voltage is required. If one is added repeatedly to a binary number, the least significant bit (LSB) of the result will alternate between 0 and 1. We use this technique of continually incrementing a number to generate a square wave.

The problem then becomes: write a program that will the LSB of an accumulator and send the result to one of the output lines of the EVB. As can be seen in the Table, the EVB has output ports A-E. They are accessed through connector P1. Connector P1 includes the lines of five ports, namely Port A, B, C, D, and E, and other signals like E-clock, interrupt request (IRQ), VDD, etc. Except for Port D, each port has eight pins. Note that Port B is the only port that generates only output lines. Hence, we will use Port B in this lab. We will generate a square wave with a program and send the signal to the Port B.

The address for Port B is $1004. This location does not belong to the user programmable space in the RAM (you can see this on the EVB memory map). Any data that is stored at this location will appear at the eight pins of Port B. The instruction STAA $1004 will store the value in Accumulator A into $1004 and thus the eight bits in Accumulator A appear at Port B.

The frequency of the square wave can be measured by connecting the appropriate pins to the oscilloscope. The pin assignment of the 60-pin connector P1 is shown in Table 6-1 of the EVB User's Manual.

We start by writing a program that can do these manipulations. We will use Accumulator A to increment the value we send to output Port B. We first have to clear Accumulator A, then continuously increment it and write the result to address $1004. Program 1 (below) performs this task. Note that the BRA instruction always branches back, so this program has an infinite (never ending) loop. We can connect an oscilloscope to the appropriate pin on connector Pl to see the generated square wave, and can calculate the frequency of this signal. All of the square waves will be observed on bit 0 of Port B (abbreviated "PB0"). You may want to look at the signals generated on the other bits of Port B, too.

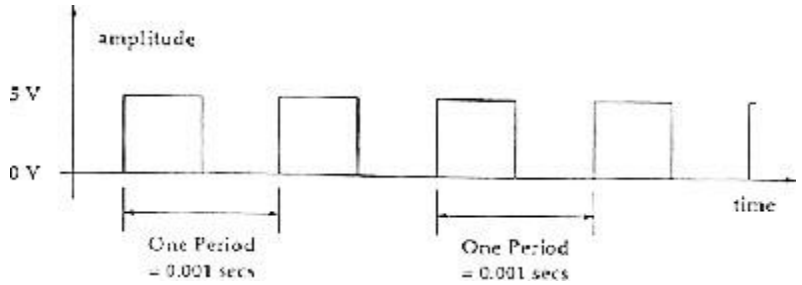


Figure 1: Square wave of l kHz frequency

Pins on P1	Purpose
9-16	Port C, General-purpose I/O lines
20-25	Port D, General-purpose I/O lines. Used with the Serial Communication Interface (SCI) and Serial Peripheral Interface (SPI).
27-34	Port A, General-purpose I/O lines.
35-42	Port B, General-purpose output lines.
43-50	Port E, General-purpose input or A/D channel input lines

Table 1: Pin numbers of MCU ports available on connector P1

Program 1

```
Address   Label    Mnemonic   Operand   Comment
C010               CLRA                 ; Clear Accumulator A
C011      LOOP     INCA                 ; Increment Accumulator A
C012               STAA       $1004     ; Send ACCA to Port B
C015               BRA        LOOP      ; Branch back always
C017      FINISH   SWI                  ; We never get here!
```

How can we calculate the frequency of the square wave? The M68HC11 operates with a clock of frequency 2 MHz (megahertz), i.e. 2 million cycles per second. Hence each cycle has a period of 1 second/(2 × 106 cycles) = 0.5 µsec/cycle. Therefore, if an instruction takes 4 cycles, it takes 4 × 0.5 = 2 µsec to execute. The number of cycles each instruction takes can be found in the Instruction Set Summary (Appendix A in the textbook). From these tables, we see that CLRA takes 2 cycles, INCA takes 2 cycles, STAA with extended addressing takes 4 cycles, and BRA takes 3 cycles. Hence

the total number of cycles involved in each loop are 2+4+3 = 9 cycles. Note that the loop consists of only INCA, STAA, and BRA instructions. CLRA and SWI are outside the loop. The total time involved in executing 9 cycles is 9 × 0.5 µsec/cycle = 4.5 µsec. Our output square wave goes from "up" to "down", or from "down" to "up", every 4.5 µsec. This makes the period T = 2 × 4.5 µsec = 9.0 µsec and the corresponding frequency f = 1/(9.0 µsec) = 0.1111 MHz.

B) Generating a square wave of desired frequency:

In the previous section we generated a square wave. This section deals with generating a square wave of a desired frequency. We showed the calculations required for computing the total number of clock cycles in a given loop. To generate a square wave of lower frequency (< 0.1111 MHz), we need to introduce a "wait loop", i.e., a sequence of instructions that takes some time to execute but does nothing. The following assembly program shows one such wait loop.

WAIT LOOP

```
Address   Label    Mnemonic   Operand    Comment
C010               LDX        #$0010     ; load loop counter
C011      LOOP     DEX                   ; Decrement Index Register
C012               BNE        LOOP       ; Branch till the end of loop
C014      FINISH   SWI                   ; Stop
```

This wait loop does nothing, but it spends some time executing. Inserting such a loop in Program l, we can increase the execution time and hence can increase the period of the generated square wave. Program 2 shows Program 1 modified with a wait loop.

Program 2

```
Address   Label    Mnemonic   Operand    Comment
C010               CLRA                  ; Clear Accumulator A
C011      LOOP     INCA                  ; Increment Accumulator A
C012               LDX        #$N        ; Load Index Register X with a number N
C015      WAIT     DEX                   ; Decrement Index Register
C016               BNE        WAIT       ; Branch back to wait loop
C018               STAA       $1004      ; Send ACCA to Port B
C01B               BRA        LOOP       ; Branch back always
C01C      FINISH   SWI                   ; We never get here!
```

Note that an unknown number N is used in Program 2. For a given frequency we wish to generate, we must compute the value of N. We compute the execution time involved in executing LOOP by analyzing the loop WAIT, and the rest of the instructions in loop LOOP with cycle-by-cycle

calculations:

WAIT loop timing:

- *N* = Total number of times the loop WAIT is executed
- DEX takes 3 cycles.
- BNE takes 3 cycles.
- Total cycles in WAIT loop = 6*N*.

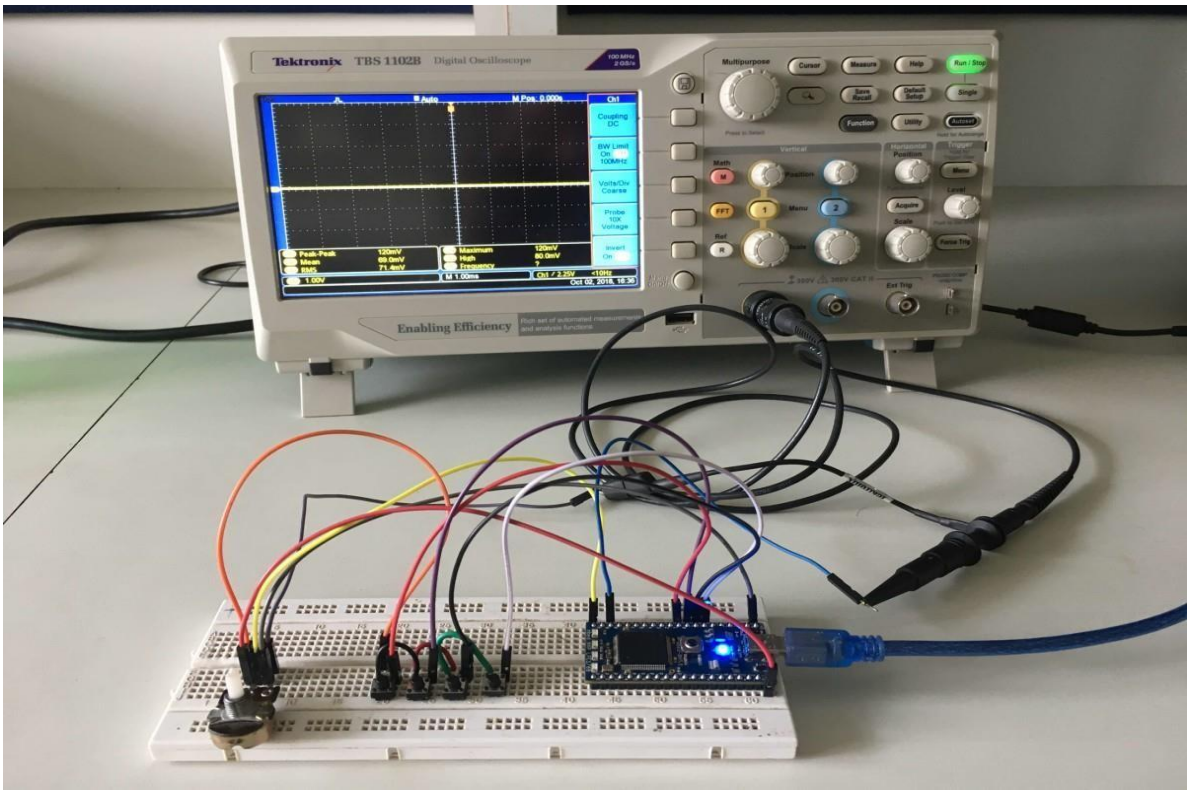
LOOP loop timing:
- INCA takes 2 cycles.
- LDX takes 3 cycles in immediate addressing mode.
- STAA takes 4 cycles in extended addressing mode.
- BRA takes 3 cycles.
- Total cycles in LOOP loop = 2 + 3 + 6*N* + 4 + 3 = 6*N* + 12 cycles

Square wave characteristics:
Period of square wave: $T = 2 \times (6N + 12)$ cycles × 0.5 μsec/cycle = $6N + 12$ μsec
A desired frequency of f MHz implies: $N = (1/f - 12) / 6$

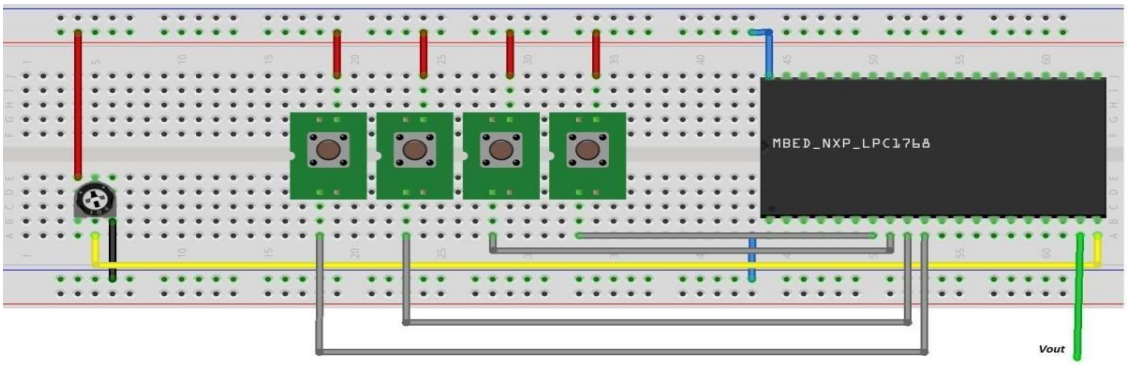
The number N calculated from this formula for a desired frequency, fdesired , should be substituted for N in Program 2. Care should be taken since the resulting N is a real number, not an integer. The integer value closest to the calculated real number should be chosen. The resulting frequency will not always equal the desired frequency, fdesired . For example, fdesired = 25 kHz results in N = 4.67; we choose the closest integer, N = 5. The calculated frequency is fcalc = 23.8 kHz.

A solution to this problem is to introduce more instructions to change the loop timing. A good choice is the NOP instruction (no operation). This instruction does nothing but use two CPU cycles (it doesn't change any registers, affect the CCR, etc.). Since it doesn't affect the CPU, it is a good choice to use in wait loops. If additional instructions are introduced, all the cycle-by-cycle calculations must be redone since the above formula for N is no longer valid.

BLOCK DIAGRAM



.



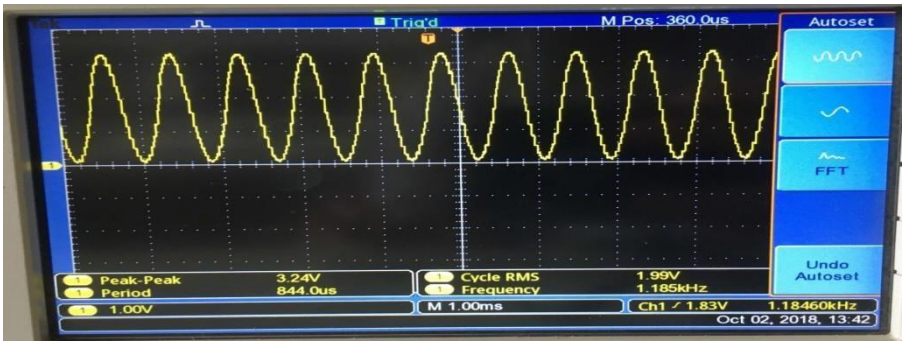
a)

PROGRAM & OUTPUT

Sine wave:

- When a1=1 sine wave is selected. We get value of potentiometer in range of 0 to 1, so convert it from 0 to 100 multiply it by 100.
- Put this value of 'A' in sine function to get different frequencies.

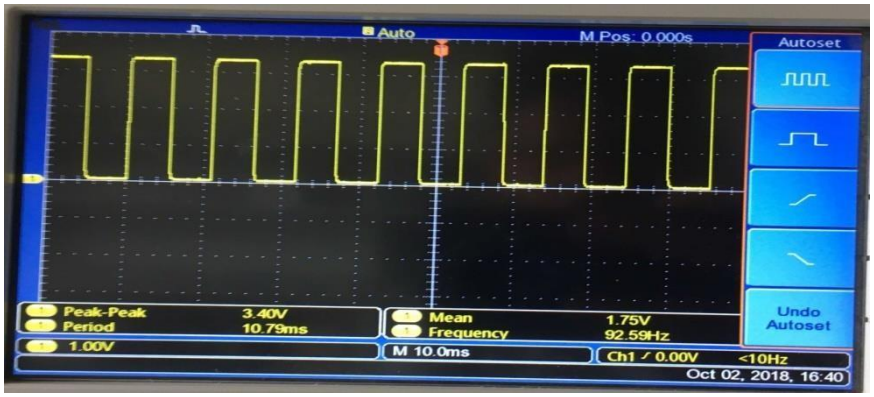
```
while(a1==1)      //For generating sine wave
  {
    A=Ain*100;
    for (n=0;n<A;n++)
    {
        Aout= 0.5 + 0.5*sin(n*2*pi/A);    //note the 0.5 V of offset since DAC
outputs voltage between 0 and 3.3V
    }

  }
```



Square wave:

- When a2 is '1' square wave is selected. To get different delay Ain is divided by 10 so we get delay in range of 0 to 0.1 Sec.

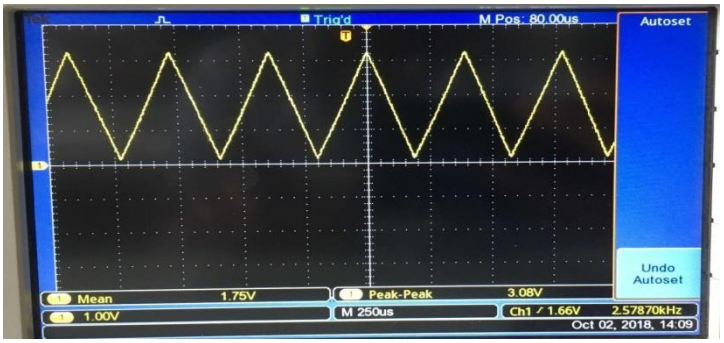
```
    while(a2==1)          //for generating square wave
    {
     Aout=0X00;
    wait(Ain/10);

    Aout=0XFF;
    wait(Ain/10);
  }
```



Triangular wave:

- We get value of potentiometer in range of 0 to 1, so convert it from 0 to 100 multiply Ain by 100 and get A.
- Divide i by A so that we get Aout in range of 0 to 1.

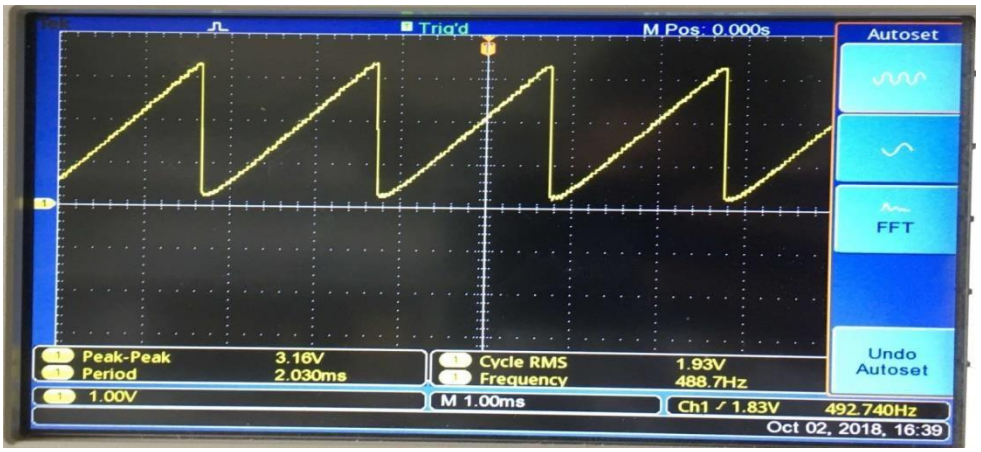
```
while(a3==1)              //for generating trianguler wave
{
A=Ain*100.00;
for (i = 0.00; i < A; i++)
{
Aout = (float)i /A;
            wait(0.0001);
      }
   for (i = A-1.00; i > 0.00; i--)
        {
            Aout = (float)i / A;
            wait(0.0001);
        }
 }
```



Saw-tooth wave:

- Divide i by (Ain*100) to get Aout in range of 0 to 1.

```
while(a4==1)                //for generating saw-tooth wave
 {

  for (i = 0.00; i < (Ain*100); i++)
 {
            Aout = (float)i / (Ain*100);
 }
 }
```



CONCLUSION

Successfully design the circuit of variable frequency square wave generation in Proteus software and implement on hardware at microcontroller kit.

QUESTION

a. Can you describe the objective of the experiment on variable frequency square wave generation?
b. What hardware components are required for this experiment?
c. How does the square wave generator circuit work?
d. What is the significance of generating square waves with variable frequency?
e. How do you control the frequency of the square wave in your program?
f. What programming language did you use for writing the program, and why?
g. Can you explain the role of timers or counters in the program?
h. How did you ensure the accuracy and stability of the generated square wave frequencies?
i. What methods did you employ for testing the functionality of your program?
j. How does the hardware interface with the program for frequency

control?

k. What challenges did you encounter during the implementation of the program and how did you overcome them?
l. Can you discuss any limitations or constraints of your hardware setup for this experiment?
m. How did you verify that the generated square wave frequencies meet the desired specifications?
n. Did you encounter any issues with signal integrity or noise during the generation process? If so, how did you address them?
o. How would you further enhance or optimize the program or hardware setup for better performance or versatility in future experiments?

DEPARTMENT OF ELECTRICAL ENGINEERING

EMBEDDED SYSTEM LAB MANUAL

Experiment No : 12

Write a program to implement a PWM based speed controller for 12 V/24V DC Motor incorporating a suitable potentiometer to provide the set point.

TITLE : DC Motor control

AIM

To study interfacing of DC motor control with microcontroller.

APPARATUS

1) DC Motor control card.
2) 8051 board.
3) FRC Connector.
4) Dual power supply.

THEORY

DC motor control card gives the complete control of motor i.e. direction and speed control. DC motor module consists of two controls speed and direction. A DAC 0808 is used to control the speed of motor. This converts the digital data equivalent to speed into analog value as a reference of speed. Speed is controlled by PWM IC LM 3524 direction control uses port C (PC1) of 8255.

DC motor control using PWM:-

Pulse width modulation is a different way of controlling the speed of DC motor. The duty cycle controls the armature voltage & hence the motor speed. The total period of a PWM waveform is kept such that mechanical inertia will smooth out the power brush. PWM waveforms can easily be generated using microcontroller. The driver amplifier circuit can be a power transistor, MOSFET or power operational amplifier. IC LM 35Q4 accepts logical signals (OV & 5V) from a microcontroller. Motor supply voltage can be minimum 50V & maximum load current of 2A output pins of LM3524 are converted to DC motor armature directly. IC switches supply voltage & ground to the pins.

Changing the phase signal reverses the direction of motor. For logic 1 phase signal, the motor direction is clockwise & for a low phase signal, it will be counter clock wise. Enable input is connected to PWM output of the microcontroller. The microcontroller pin may directly drive the MOSFET IRF 130. The pin (1) shows a PWM drive for DC motor using microcontroller. Power MOSFET is used as a switching device. 12 V supply is used to drive the MOSFET gate. If 12V supply is not available then it may be derived using simple zener diode circuit from DC supply used for load.

PROCEDURE

1) The DC motor control card supplied to you comes in the from of well finished wooden box.
2) Keep development board to the RHS & DC motor control card to LHS.
3) Connect +5v supply to kit.

4) Connect the ports of the microcontroller (Available on 26 pin FRC connector) to the DC motor control card with the help of 26 pin FRC.
5) Switch ON the power supply.
6) Feed or down load the sample program given. As per following the instruction & execute the program.
7) Press 'I' key on your LGS 51 kit key board for increase the speed of motor.
8) Press 'D' key on your LGS 51 kit key board for decrease the speed ofmotor.
9) Press 'C' key on your LGS 51 kit key board for rotate the motor in clockwise/ forward direction.
10) Press 'A' key on your LGS 51 kit key board for rotate the motor in anticlockwise/ reverse direction.

PROGRAM

Memory address	Opcode	Label	Mnemonic	Comment
8000	75 81 60		MOV SP,# 60H	
8003	79 00		MOV R1,# 00H	
8005	90 80 C4		MOV DPTR, # LINE 2	
8008	12 00 5A		LCALL MSGOUT	
800B	79 04		MOV R1, # 04H	
800D	90 80 CD		MOV DPTR, #F2	
8010	12 00 5A		LCALL MSGOUT	
8013	79 08		MOV R1, # 08H	
8015	90 80 D1		MOV DPTR, #F3	
8018	12 00 5A		LCALL MSGOUT	
801B	79 0C		MOV R1, # OCH	
801D	90 80 E0		MOV DPTR, #F6	
8020	12 00 5A		LCALL MSGOUT	
8023	74 80		MOV A,# 80H	
8025	90 00 03		MOV DPTR, # CW55	
8028	F0		MOVX @ DPTR, A	
8029	FA		MOV R2, A	

802A	74 1F		MOV A,# 1FH	
802C	90 00 00		MOV DPTR, # PORTA	
802F	F0		MOVX @ DPTR, A	
8030	FA		MOV R2, A	
8031	74 00		MOV A, # 00H	
8033	90 00 01		MOV DPTR, # PORTB	
8036	F0		MOVX @DPTR, A	
8037	12 00 65	LOOP :	LCALL RDKEY	
803A	79 04		MOV R1, # 04H	
803C	B4 41 1B		CJNE A, # 'A', CHKC	
803F	90 80 C9		MOV DPTR, # F1	
8042	12 00 5A		LCALL MSGOUT	
8045	74 80		MOV A, # 80H	
8047	90 00 03		MOV DPTR, # CW55	
804A	F0		MOVX @DPTR, A	
804B	74 00		MOV A, # 00H	
804D	90 00 01		MOV DPTR, # PORTC	
8050	F0		MOVX @DPTR, A	
8051	EA		MOV A, R2	
8052	90 00 00		MOV DPTR, # PORTA	
8055	F0		MOVX @DPTR, A	
8056	FA		MOV R2, A	
8057	02 80 37		LJMP LOOP	
805A	B4 43 1B	CHKC :	CJNE A, # 'C', CHKD	
805D	90 80 CD		MOV DPTR, # F2	
8060	12 00 5A		LCALL MSGOUT	
8063	74 80		MOV A, # 80H	

8065	90 00 03		MOV DPTR, # CW55	
8068	F0		MOVX @DPTR, A	
8069	74 01		MOV A, # 02H	
806B	90 00 01		MOV DPTR, # PORTC	
806E	F0		MOVX @DPTR, A	
806F	EA		MOV A, R2	
8070	90 00 00		MOV DPTR, # PORTA	
8073	F0		MOVX @DPTR, A	
8074	FA		MOV R2, A	
8075	02 80 37		LJMP LOOP	
8078	79 0C	CHKD :	MOV R1, # 0CH	
807A	B4 44 15		CJNE A, # 'D', CHKI	
807D	90 80 DB		MOV DPTR, # F5	
8080	12 00 5A		LCALL MSGOUT	
8083	90 00 00		MOV DPTR, # PORTA	
8086	BA 18 03		CJNE R2, # 18H, NODEC	
8089	02 80 8D		LJMP NODEC1	
808C	1A	NODEC :	DEC R2	
808D	EA	NODEC 1 :	MOV A, R2	
808E	F0		MOVX @DPTR, A	
808F	02 80 37		LJMP LOOP	
8092	B4 49 15	CHKI :	CJNE A, # 'I', CHESC	
8095	90 80 D6		MOV DPTR, # F4	
8098	12 00 5A		LCALL MSGOUT	
809B	90 00 00		MOV DPTR, # PORTA	
809E	BA FF 03		CJNE R2, # 0OFFH, NOINC	
80A1	02 80 A5		LJMP NOINC1	

80A4	0A	NOINC:	INC R2	
80A5	EA	NOINC 1 :	MOV A, R2	
80A6	F0		MOVX @DPTR, A	
80A7	02 80 37		LJMP LOOP	
80AA	B4 20 8A	CHESC :	CJNE A, # 20H, LOOP	
80AD	74 80		MOV A, # 80H	
80AF	90 00 03		MOV DPTR, # CW55	
80B2	F0		MOVX @DPTR, A	
80B3	74 00		MOV A, # 00H	
80B5	90 00 00		MOV DPTR, # PORTA	
80B8	F0		MOVX @DPTR, A	
80B9	01 37		AJMP LOOP	
80BB	44 43 20 4D	LINE 1 :	DB DC MOTOT, 03H	
80BF	4F 54 4F 52			
80C3	03			
80C4	44 49 52 3A	LINE 2 :	DB DIR : , 03H	
80C8	03			
80C9	52 45 56 03	F1 :	DB REV, 03H	
80CD	46 57 44 03	F2 :	DB FWD, 03H	
80D1	53 50 44 3A	F3 :	DB SPD, 03H	
80D5	03			
80D6	49 4E 43 20	F4 :	DB INC, 03H	
80DA	03			
80DB	44 45 43 20	F5 :	DB DEC, 03H	
80DF	03			
80E0	4E 4F 52 4D	F6 :	DB NORM, 03H	
80E4	03			

Simple program for understanding:

```
ORG 00H // initial starting address
MAIN: MOV P1,#00000001B // motor runs clockwise
ACALL DELAY // calls the 1S DELAY
MOV P1,#00000010B // motor runs anti clockwise
ACALL DELAY // calls the 1S DELAY
SJMP MAIN // jumps to label MAIN for repeating the cycle
DELAY: MOV R4,#0FH
WAIT1: MOV R3,#00H
WAIT2: MOV R2,#00H
WAIT3: DJNZ R2,WAIT3
DJNZ R3,WAIT2
DJNZ R4,WAIT1
RET
```

CONCLUSION

Thus we have studied interfacing of DC motor where it moves in clockwise and anticlockwise direction.

VIVA QUESTIONS:

1. Can you explain what PWM stands for and how it is utilized in motor speed control?
2. How does a PWM-based speed controller differ from other methods of motor speed control?
3. What are the advantages of using a PWM-based controller for motor speed regulation?
4. Can you outline the basic components required for implementing a PWM-based speed controller for a DC motor?
5. How does a potentiometer integrate into the PWM-based speed control system, and what is its role?
6. What factors should be considered when selecting a suitable potentiometer for this application?
7. Can you describe the process of setting the desired speed using the potentiometer in this setup?
8. What are the safety precautions to be taken when working with high voltage DC motors?
9. How does the voltage level (12V/24V) affect the design and implementation of the PWM-based speed controller?
10. What is duty cycle in the context of PWM, and how does it influence motor speed?
11. How does the frequency of the PWM signal affect the performance of the motor controller?
12. Can you explain how feedback mechanisms can be incorporated into this PWM-based speed control system for improved accuracy?
13. What are the potential challenges or limitations one might encounter when implementing this PWM-based speed controller?
14. How would you calibrate the system to ensure accurate speed control?
15. Can you discuss any potential applications or industries where PWM-based motor speed control is commonly used, and why?