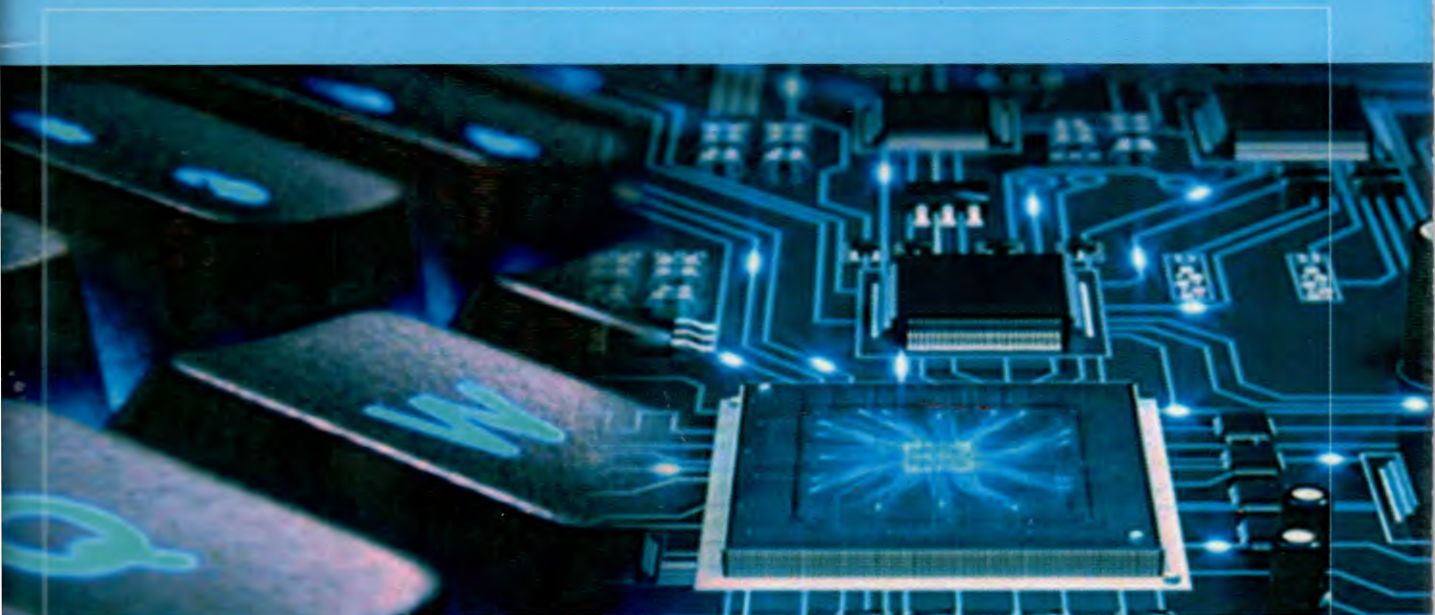
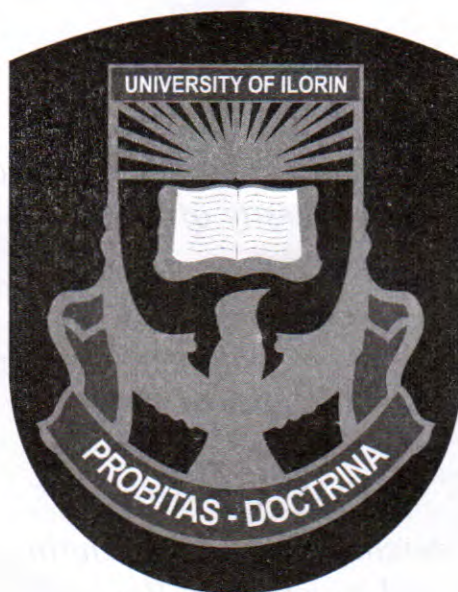




UNDERGRADUATE COMPUTER ENGINEERING LABORATORY MANUAL (VOL. 3)



**Microprocessor System Interfacing
Prototyping and Packaging
Artificial Intelligence Applications**



UNIVERSITY OF ILORIN
FACULTY OF ENGINEERING & TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING

ISBN: 978-978-8090-66-4

© January 2018

J.F. Opadiji

*Department of Computer Engineering
University of Ilorin*

A.T. Ajiboye

*Department of Computer Engineering
University of Ilorin*

S.A. Olatunji

*Department of Computer Engineering
University of Ilorin*

S.L. Ayinla

*Department of Computer Engineering
University of Ilorin*

A.R. Ajayi

*Department of Computer Engineering
University of Ilorin*

O.A. Yusuf

*Department of Computer Engineering
University of Ilorin*

H.O. Mahmud

*Department of Computer Engineering
University of Ilorin*

O.F. Adebayo

*Department of Computer Engineering
University of Ilorin*

FOREWORD



This instruction manual is the third in the series of the undergraduate Computer Engineering laboratory manuals designed to facilitate a practical-oriented approach to learning by students of Computer Engineering in Nigerian universities. In developing this manual, requirements in the Basic Minimum Academic Standard (BMAS) as defined by both the National Universities Commission (NUC) and the Council for the Regulation of Engineering in Nigeria (COREN) for courses to be audited by Computer Engineering undergraduates were considered.

This volume focuses on experiments in **Microprocessor Systems Interfacing, Prototyping and Packaging, and Artificial Intelligence and Applications**. A total of six laboratory sessions are to be performed by students in each of these areas.

At the end of each laboratory session, each student is expected to write a report on the experiments conducted in the workbook provided and submit to laboratory instructors for assessment.

It is our desire that the use of this manual in undergraduate Computer Engineering laboratories will achieve our goal of imparting students with the necessary hands-on experience that will make them productive Computer Engineers in the future.

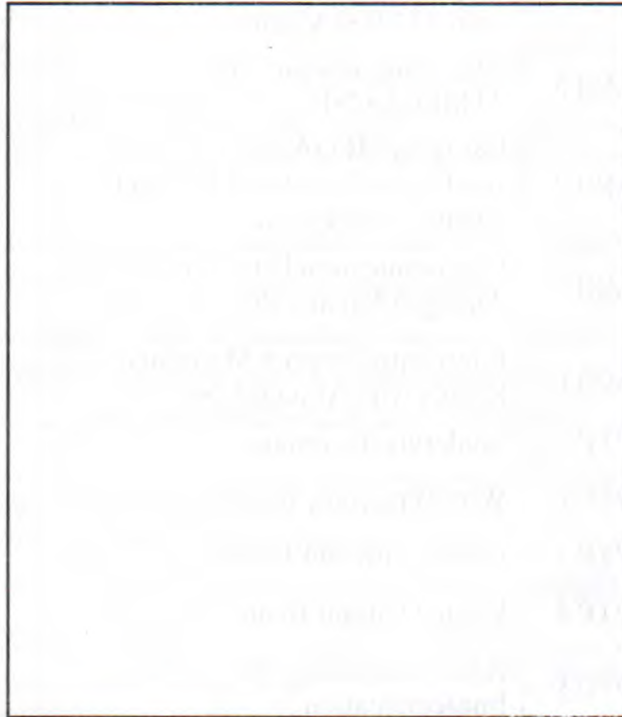
J.F. Opadiji, Dr.Eng, R.Eng.

TABLE OF CONTENTS



Student Information Page	7
Attendance Page	8
Laboratory Safety Rules and Operating Procedures	10
General Laboratory Safety Information	11
Troubleshooting Hints	13
Guidelines for Laboratory Report Notebook	14
Laboratory Grading Policy	16
UILCPE/MSI 1 – Microcontroller System	17
UILCPE/MSI 2 – Interfacing Seven Segment Display, LCD and Traffic Light with ATMEGA328P	21
UILCPE/MSI 3 – Interfacing Keypad with ATMEGA328P	27
UILCPE/MSI 4 – Using ATMEGA328P for Analog to Digital and Digital to Analog Conversion	29
UILCPE/MSI 5 – Communication between Atmega328p and PC	33
UILCPE/MSI 6 – Interfacing Stepper Motor and Relays with Atmega328p	37
UILCPE/PTP 1 – Soldering Technique	40
UILCPE/PTP 2 – Wire Wrapping Technique	43
UILCPE/PTP 3 – Breadboard and Veroboard	45
UILCPE/PTP 4 – Printed Circuit Board	49
UILCPE/PTP 5 – Radio Frequency (RF) Implementation	51
UILCPE/PTP 6 – Grounding	53
UILCPE/AIA 1 - Developing a Back Propagation Neural Network for Pattern Recognition Systems	55
UILCPE/AIA 2 – Optimizing Constrained and Unconstrained Functions using Genetic Algorithm	59
UILCPE/AIA 3 – Solving a Multiprocessor Scheduling Problem Using Simulated Annealing	64
UILCPE/AIA4 – Predicting Outcomes from Known Input and Response Data using Supervised Learning	69
UILCPE/AIA 5 – Building a Self-Organizing Map using Unsupervised Learning	73
UILCPE/AIA 6 – A reinforcement Learning Approach to Solving a Mountain Car Problem	76
APPENDIX 1 – Standard Resistor Colour Code	80

STUDENT INFORMATION PAGE



Student Passport Photograph

SURNAME: _____

FIRST NAME: _____

OTHER NAME: _____

MATRICULATION NUMBER: _____

ACADEMIC SESSION: _____

CURRENT LEVEL: _____

ATTENDANCE PAGE

DATE	EXPERIMENT NUMBER	EXPERIMENT TITLE	SIGNATURE (Technologist-In-Charge)
	UILCPE/MSI 1	Microcontroller System	
	UILCPE/MSI 2	Interfacing Seven Segment Display, LCD and Traffic Light with ATMEGA328P	
	UILCPE/MSI 3	Interfacing Keypad with ATMEGA328P	
	UILCPE/MSI 4	Using ATMEGA328P for Analog to Digital and Digital to Analog Conversion	
	UILCPE/MSI 5	Communication between Atmega328p and PC	
	UILCPE/MSI 6	Interfacing Stepper Motor and Relays with Atmega328p	
	UILCPE/PTP 1	Soldering Technique	
	UILCPE/PTP 2	Wire Wrapping Technique	
	UILCPE/PTP 3	Breadboard and Veroboard	
	UILCPE/PTP 4	Printed Circuit Board	
	UILCPE/PTP 5	Radio Frequency (RF) Implementation	
	UILCPE/PTP 6	Grounding	
	UILCPE/AIA 1	Developing a Back Propagation Neural Network for Pattern Recognition Systems	
	UILCPE/AIA 2	Optimizing Constrained and Unconstrained Functions using Genetic Algorithm	
	UILCPE/AIA 3	Solving a Multiprocessor Scheduling Problem Using Simulated Annealing	
	UILCPE/AIA4	Predicting Outcomes from Known Input and Response Data using Supervised Learning	
	UILCPE/AIA 5	Building a Self-Organizing Map using Unsupervised Learning	
	UILCPE/AIA 6	A reinforcement Learning Approach to Solving a Mountain Car Problem	

LABORATORY SAFETY RULES AND OPERATING PROCEDURES

1. Note the location of the Circuit Breaker to shut off power in an emergency. Note the location of the nearest exit to your location.
2. Students are allowed in the laboratory only when the laboratory instructor (Technologist) is present.
3. Food and drinks are strictly prohibited in the laboratory. In the case where it becomes necessary to drink water, keep open drinks away from the laboratory benches.
4. Report any broken equipment or defective parts to the laboratory instructor. Do not open, remove the cover, or attempt to repair any equipment.
5. When the laboratory exercise is over, all equipment must be turned off and returned to the designated location.
6. All equipment, tool and instruments are the property of the University and must not be taken out of the laboratory.
7. Do not move instruments from one laboratory station to another laboratory station.
8. Do not tamper with or remove security straps, locks, or other security devices.
9. When touching the FPGA development boards or any other circuit boards, please do not touch the solid-state parts on the board but handle the board from its edge.
10. Lateness to laboratory sessions will not be condoned. Students who arrive later than thirty (30) minutes into the session will not be allowed into the laboratory.

I have read and understood these rules and procedures. I agree to abide by the rules and procedures at all times while in the laboratory, failure to follow these rules and procedures will result in my immediate dismissal from the laboratory and additional disciplinary action may be taken.

SIGNATURE

DATE

GENERAL LABORATORY SAFETY INFORMATION

Introduction

The danger of injury or death from electrical shock, fire, or explosion is present while conducting experiments in this laboratory. To work safely, it is important that you understand the prudent practices necessary to minimize the risks and what to do if there is an accident.

Electrical Shock

Avoid contact with conductors in energized electrical circuits. The typical *cannot let-go* (the current in which a person cannot let go) current is about 30mA (OSHA). Muscle contractions can prevent the person from moving away from the energized circuit. Possible death can occur as low as 50mA. For a person that is wet the body resistance can be as low as 1000 ohms. A voltage of 50V can result in death.

Do not touch someone who is being shocked while still in contact with the electrical conductor or you may also be electrocuted. Instead, shut down the power source or switch off the electric circuit breaker if the power source is inaccessible.

Make sure your hands are dry. The resistance of dry, unbroken skin is relatively high and thus reduces the risk of shock. Skin that is broken, wet, or damp with sweat has a low resistance.

When working with an energized circuit, work with only your right hand, keeping your left hand away from all conductive material. This reduces the likelihood of an accident that results in current passing through your heart.

Be cautious of rings, watches, and necklaces. Skin beneath a ring or watch is damp, lowering the skin resistance. Shoes covering the feet are much safer than sandals.

Fire

Some electronic components can become extremely hot and cause severe burns if touched. In case of fire outbreak, turn off the power supply and notify the laboratory instructor. Avoid using fire extinguishers on electronic equipment.

Explosions

When using electrolytic/tantalum capacitors, be careful to observe proper polarity and do not exceed the voltage rating. Electrolytic and tantalum capacitors can explode and cause injury.

First Aid

A first aid kit is always available in the laboratory. Consult the laboratory instructor if you need to make use of the first aid kit and proceed to University Clinic if needed.

TROUBLESHOOTING HINTS

1. Be sure that the power is turned on.
2. Be sure that the supply voltages are correct and the ground connections are common. For some power supplies, the 'ON' button needs to be clicked in order to turn on the output channel.
3. Be sure the circuit you built is identical to that in the diagram. Do a node-by-node check if needed.
4. Be sure the resistors, capacitors and inductors used in your circuit all have the correct values. You can get that information using a digital multimeter, color code or the nominal value printed on the component.
5. Be sure that the experiment is set up correctly and you are measuring the correct parameter

GUIDELINES FOR LABORATORY REPORT NOTEBOOK

The laboratory workbook contains the record of all works pertaining to the experiment. This record should be sufficiently complete so that anyone of similar technical background can perform the experiment by simply following your experimental procedures. Record everything directly into the workbook during the experiment. Organization in your workbook is important. Descriptive headings should be used to separate and identify the various parts of the experiment. Record the data in chronological order. A neat, organized and complete record of an experiment is just as important as the experimental work.

1. **Heading:** The experiment number should be at the top of each page. Your name and date should be at the top of the first page of each day's experimental work.
2. **Title:** The experiment title should be clearly stated.
3. **Objective(s):** A brief but complete statement of what you intend to find out or verify in the experiment should be at the beginning of each experiment
4. **Diagram(s):** A circuit diagram should be drawn and labeled so that the actual experiment circuitry could be easily duplicated at any time in the future. Be especially careful to record all circuit changes made during the experiment.
5. **Equipment List:** List those items of equipment which have a direct effect on the accuracy of the data. It may be necessary later to locate specific items of equipment for rechecks if discrepancies develop in the results.
6. **Procedure(s):** In general, lengthy explanations of procedures are unnecessary. Short commentaries alongside the corresponding data may be used. Keep in mind the fact that the experiment must be reproducible from the information given in your notebook.
7. **Data:** Think carefully about what data is required and prepare suitable data tables. Record instrument readings directly. Do not use calculated results in place of direct data; however, calculated results may be recorded in the same table with the direct data. Data tables should be clearly identified and each data column labeled and headed by the proper units of measure.
8. **Calculations:** Not always necessary but equations and sample calculations are often given to illustrate the treatment of the experimental data in obtaining the results.
9. **Graphs:** Graphs are used to present large amounts of data in a concise visual form. Though not required for every practical session, they are often an efficient analytical tool. Data to be presented in graphical form should be plotted in the laboratory so that any questionable data points can be checked while the experiment is still set up. The grid lines in the notebook can be used for most graphs. If special graph paper is required, affix the

graph permanently into the notebook. Give all graphs a short descriptive title. Label and scale the axes. Use units of measure. Label each curve if more than one on a graph.

10. **Results:** The results should be presented in a form which makes the interpretation easy. Large amounts of numerical results are generally presented in graphical form. Tables are generally used for small amounts of results. Theoretical and experimental results should be on the same graph or arranged in the same table in a way for easy correlation of these results.
11. **Answers to Questions:** Some laboratory sessions will ask questions based on the experiments you just performed. The answer to the questions are usually gotten using the data acquired during the experiment. These questions will be graded so take time to answer them carefully. Be sure that you use only the acquired data to answer the questions
12. **Conclusion:** This is your interpretation of the results of the experiment as an engineer. Be brief and specific. Give reasons for important discrepancies.

LABORATORY GRADING POLICY

1. Students must follow the Guidelines for Laboratory Report Notebook for submission of the laboratory report.
2. Each student must submit one **REPORT** per laboratory experiment.
3. The **EXACT DURATION** of a laboratory session will be decided by the laboratory instructor prior to **the laboratory experiment**.
4. Students **should not** skip the laboratory periods. **SPECIAL PERMISSION** must be obtained from the **LABORATORY INSTRUCTOR** to do so. Otherwise, grade points will be **DEDUCTED** from the student's total grade earned for the scheduled laboratory experiment.
5. Each laboratory session will be graded out of 100 points. The average of all the scores attained for all sessions in a semester will be the scores for the student in the course for that semester.
6. Students who arrive for a laboratory session more than thirty (30) minutes after commencement of session will not be allowed in and will be given a zero (0) grade for that session
7. All laboratory report **MUST** be submitted before the deadline as decided by the laboratory instructor.

If required, Lab policy may be modified at the discretion of the lab Instructor

EXPERIMENT #: UILCPE/MSI 1**EXPERIMENT 1: Microcontroller System****OBJECTIVES:**

- To study the ATMEGA328P microcontroller.
- To use virtual instruments to make simulation measurements.

EQUIPMENT AND COMPONENTS REQUIRED

1. ATMEGA328P Development Board
2. Personal Computer
3. Simulation Software
4. Resistors
5. Breadboard
6. LED

BACKGROUND

A microcontroller can be compared to a small stand-alone computer which is a very powerful device, capable of executing a series of pre-programmed tasks and interacting with other hardware devices. AVR Atmega microcontroller is a unit-chip microcontroller developed by Atmel, founded in 1996 which was later acquired by Microchip Technology in 2016. The mega AVR is the name of a big family of microcontroller that includes ATMEGA328 we are going to use in this exercise. The pin configuration of 28-pin ATMEGA328P is illustrated in Fig 1.1 and the block diagram is shown in Fig 1.2.

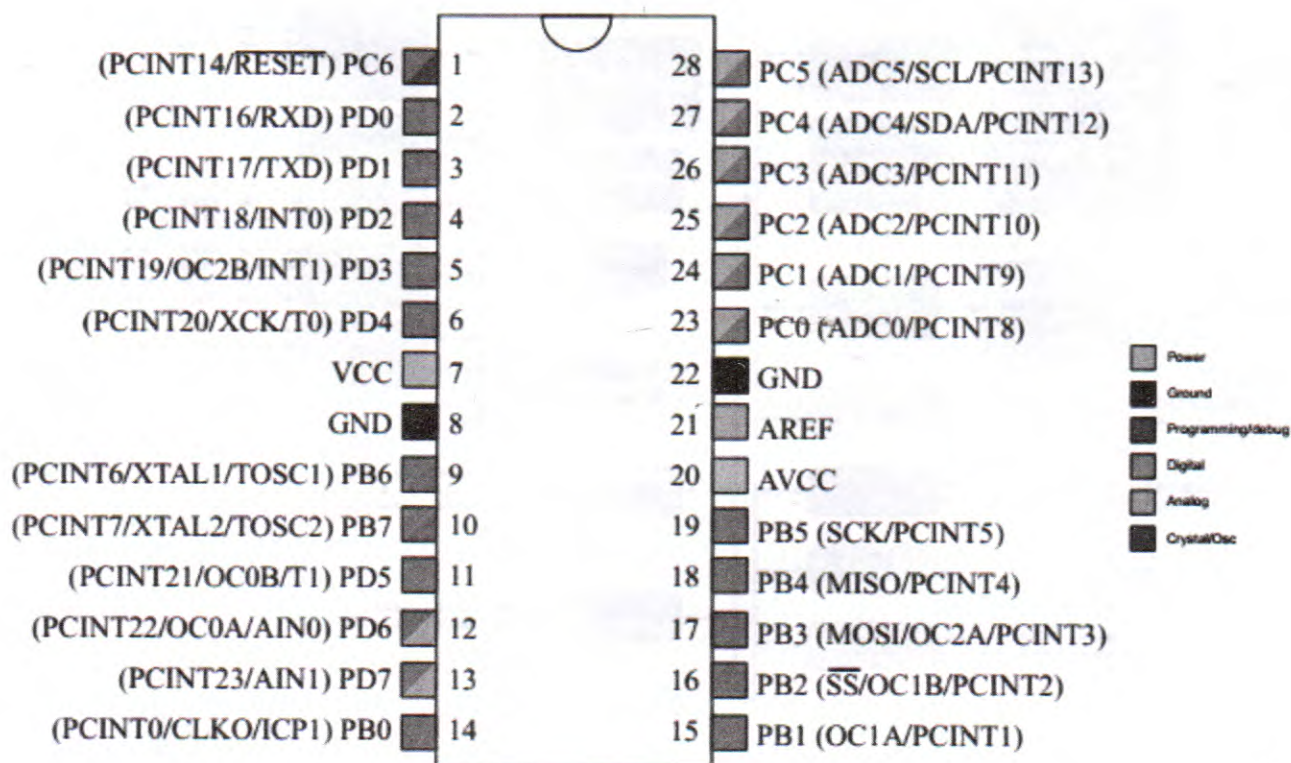


Figure 1.1: ATMEGA328P Microcontroller

The Features of ATMEGA328P are as follows:

- The ATMEGA328P is an 8-bit processor
- 32KB In-System Programmable Flash with R/W capabilities
- 1KB EEPROM
- 2KB SRAM
- It has 23 general purpose I/O programmable line
- 32 general purpose working registers
- Real Time Counter (RTC)
- Three Flexible Timer/Counters with compare modes and PWM
- 2 8-bit Timer/Counters
- 1 16-bit Timer/Counters
- 1 Serial Programmable USARTs
- 1 Byte-oriented 2-wire Serial Interface (I2C)
- 6 Channel 10-bit ADC (8 Channels in TQFP and QFN/MLF packages)
- Programmable Watchdog Timer with internal Oscillator
- SPI serial port
- 6 software selectable power saving modes

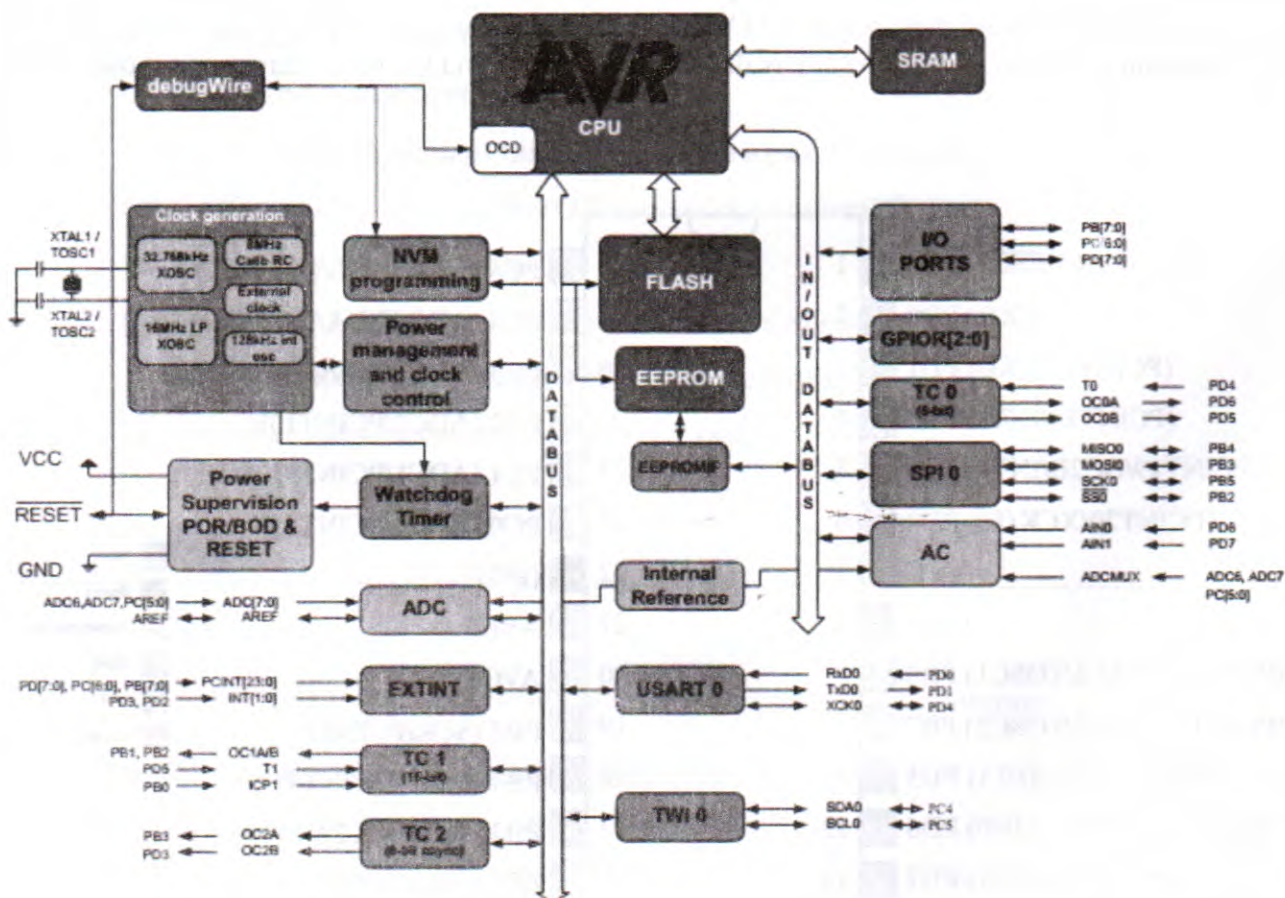


Figure 1.2: Block Diagram of ATMEGA328P Microcontroller

PREPARATION

1. Ensured you have all the required simulation software installed on your PC.
2. Write ATMEGA328P based program (embedded C++) to blink a LED.
3. Reproduce the circuit diagram of this exercise on your simulation software and run your codes in it.

EXPERIMENT

Simple Output Project: Blinking LED

To activate the hardware component of a microcontroller, Figure 1.3 shows the minimum connections required for the microcontroller to be functional. For a simple output project, LED is connected to P14 through 330Ω resistor R_1 as shown in Fig.1.3. The HEX file corresponding to the successfully written and compiled code is then transferred to the ATMEGA328P microcontroller. The HEX file is a machine language file, generated by the compiler, originally from the source code.

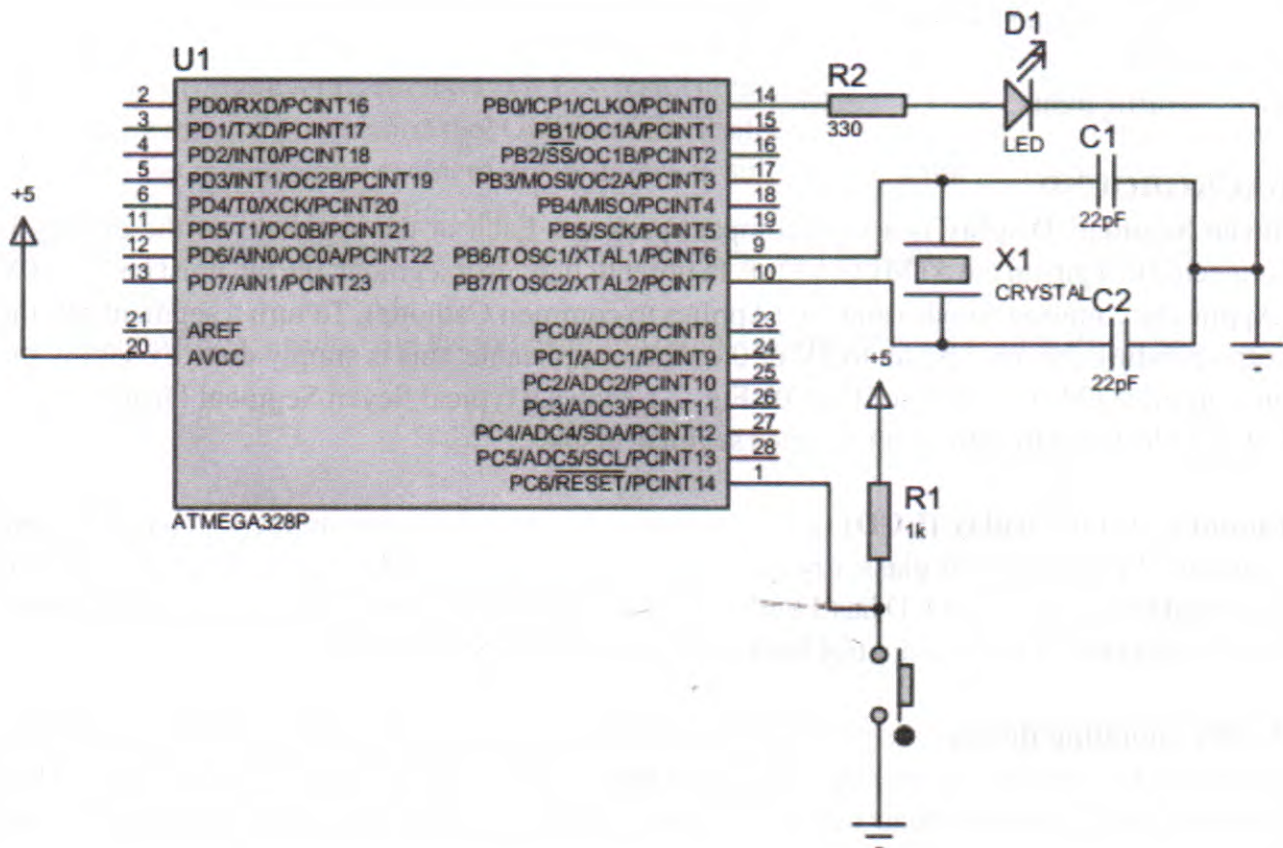


Figure 1.3: Configuration of ATMEGA328P to Blink LED

EXPERIMENT #: UILCPE/MSI 2

EXPERIMENT 2: Interfacing Seven Segment Display, LCD and Traffic Light with ATMEGA328P

OBJECTIVE:

- To understand the use of seven segment display with ATMEGA328P
- To understand LCD and how data are displayed on it.
- To control the operation of a traffic light system using ATMEGA328P.

EQUIPMENT AND COMPONENTS REQUIRED

1. ATMEGA328P Development Board
2. Personal Computer
3. Simulation Software
4. Resistors
5. Potentiometer
6. Breadboard
7. Seven Segment Display
8. LCD
9. Traffic light

BACKGROUND

Seven Segment Display is a very compact package. Each of the segments of the display is connected to a pin on the ATMEGA328P. In order to light up a segment, the pin must be set to 0V (Applies to common Anode) and 5V (Applies to common Cathode). To turn a segment off, the corresponding pin must be set to 5V or 0V where applicable, this is simply done by setting the pins on the ATMEGA328P to '1' or '0'. Fig. 2.1 shows a typical Seven Segment Display while Fig. 2.2 illustrates the pin outlet of seven segment display.

Liquid Crystal Display (LCD) is a dot matrix which displays alphanumeric, characters and symbols. It is made up of glass, organic sealant, organic fluid and polymer based polarizes as indicated in Fig. 2.3, the LCD has a built-in controller which has 3-control lines as well as 8 I/O lines for data bus. The three control lines are referred to as EN, RS and RW.

Traffic signaling devices are positioned at road intersections, pedestrian crossings, and other locations to control competing flows of traffic and manage traffic congestion. They allow/disallow access to road users by the use of light in standard colors (Red, Amber, and Green).

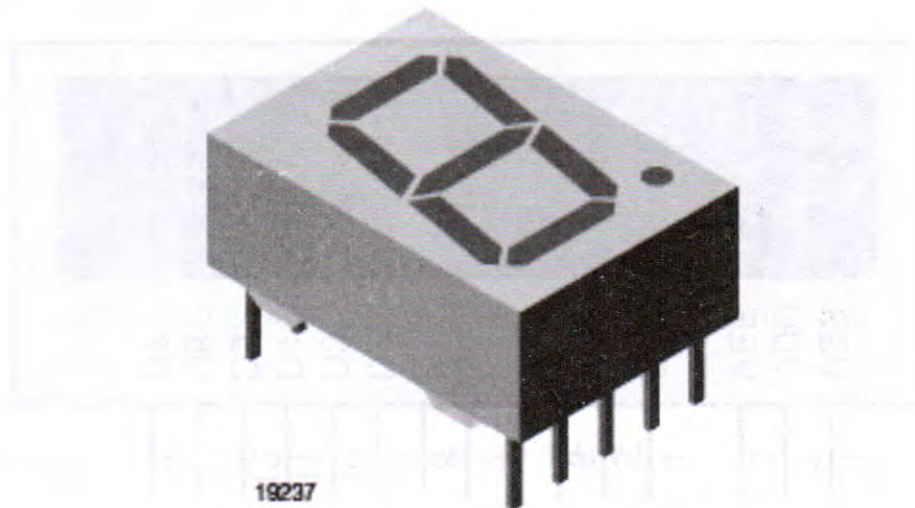


Figure 2.1: The Seven Segment Display

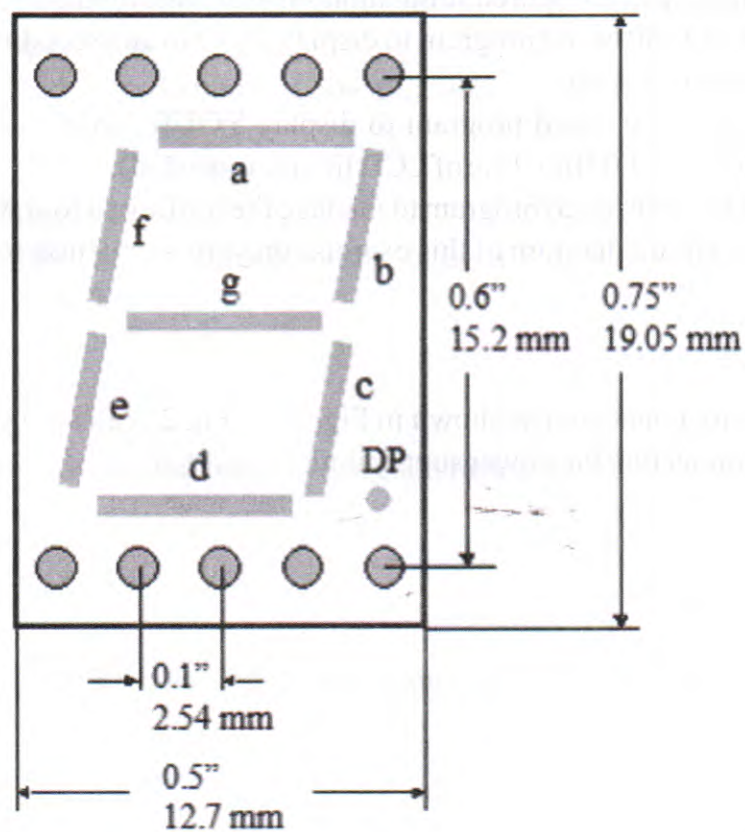


Figure 2.2: Pin Outlet of Seven Segment Display

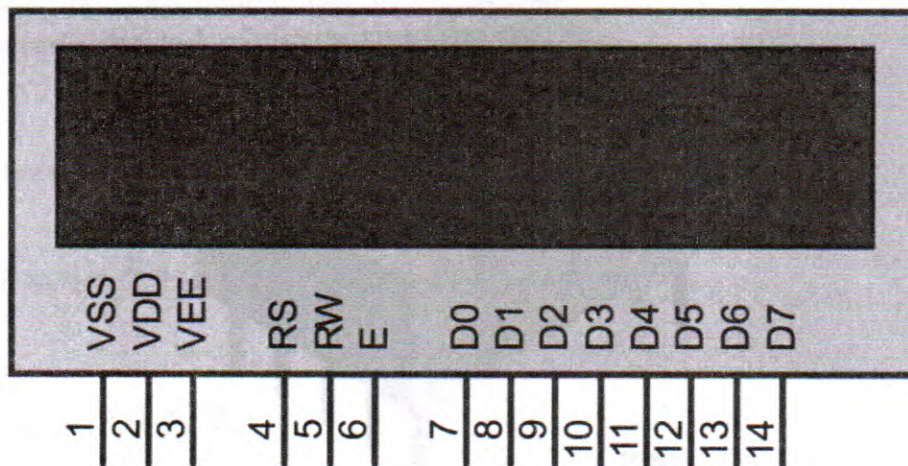


Figure 2.3: LCD pin layout

PREPARATION

1. Ensured you have all the required simulation software installed on your PC.
2. Write ATMEGA328P based program to display 0 to 9 in an ascending /descending order on a seven segment display.
3. Write ATMEGA328P based program to display YOUR NAME and YOUR MATRIC NUMBER on the LCD (Hint: Use of LCD library is needed).
4. Write ATMEGA328P based program to manage the traffic of a four way junction
5. Reproduce the circuit diagram of this exercise on your simulation software and run your codes in it.

EXPERIMENT

The circuit diagrams are connected as shown in Fig. 2.4 – Fig 2.6. Ensure your circuit is free of short circuit before connecting the power supply unit.

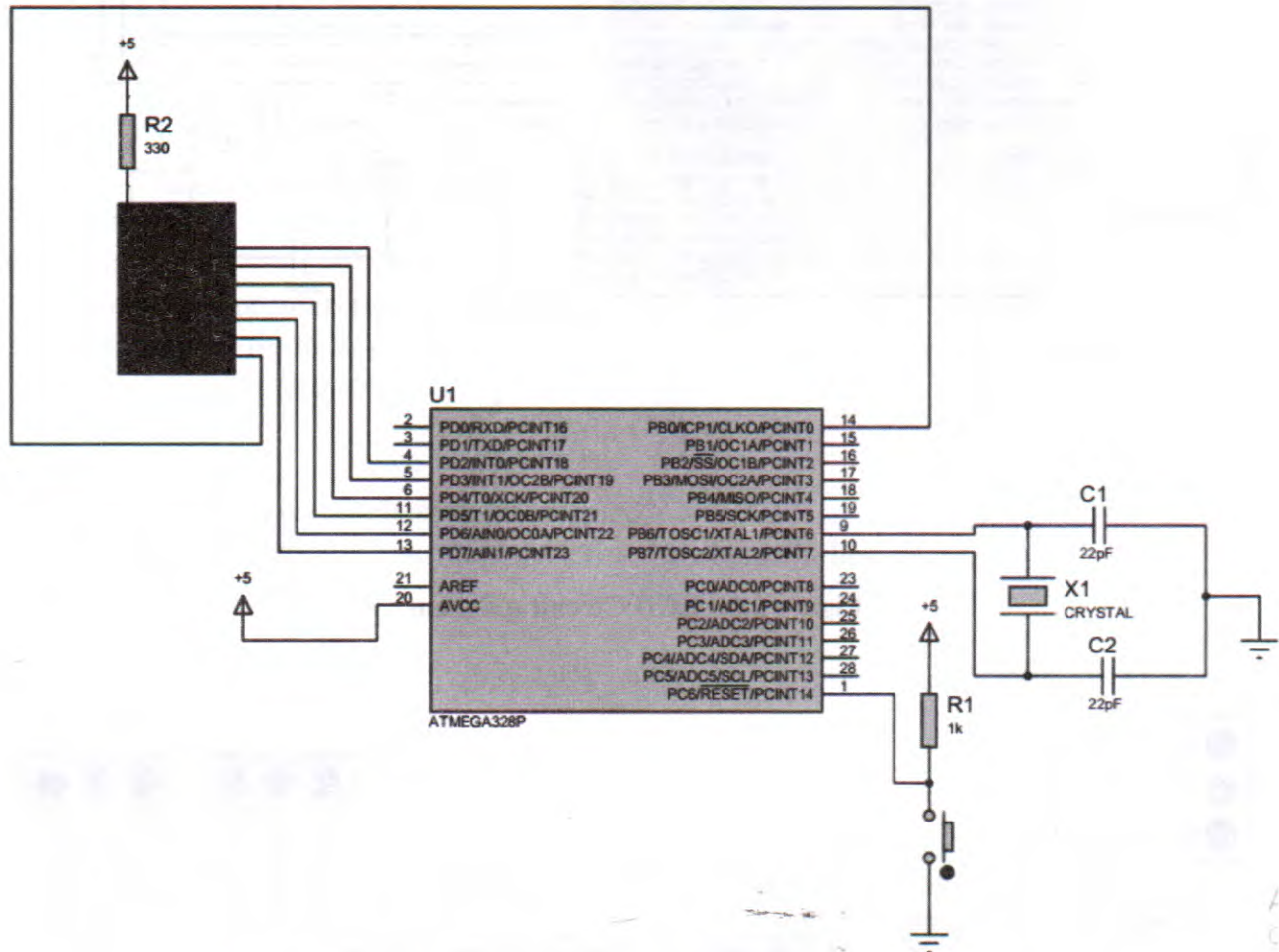


Figure 2.4: Circuit diagram for the seven segment display

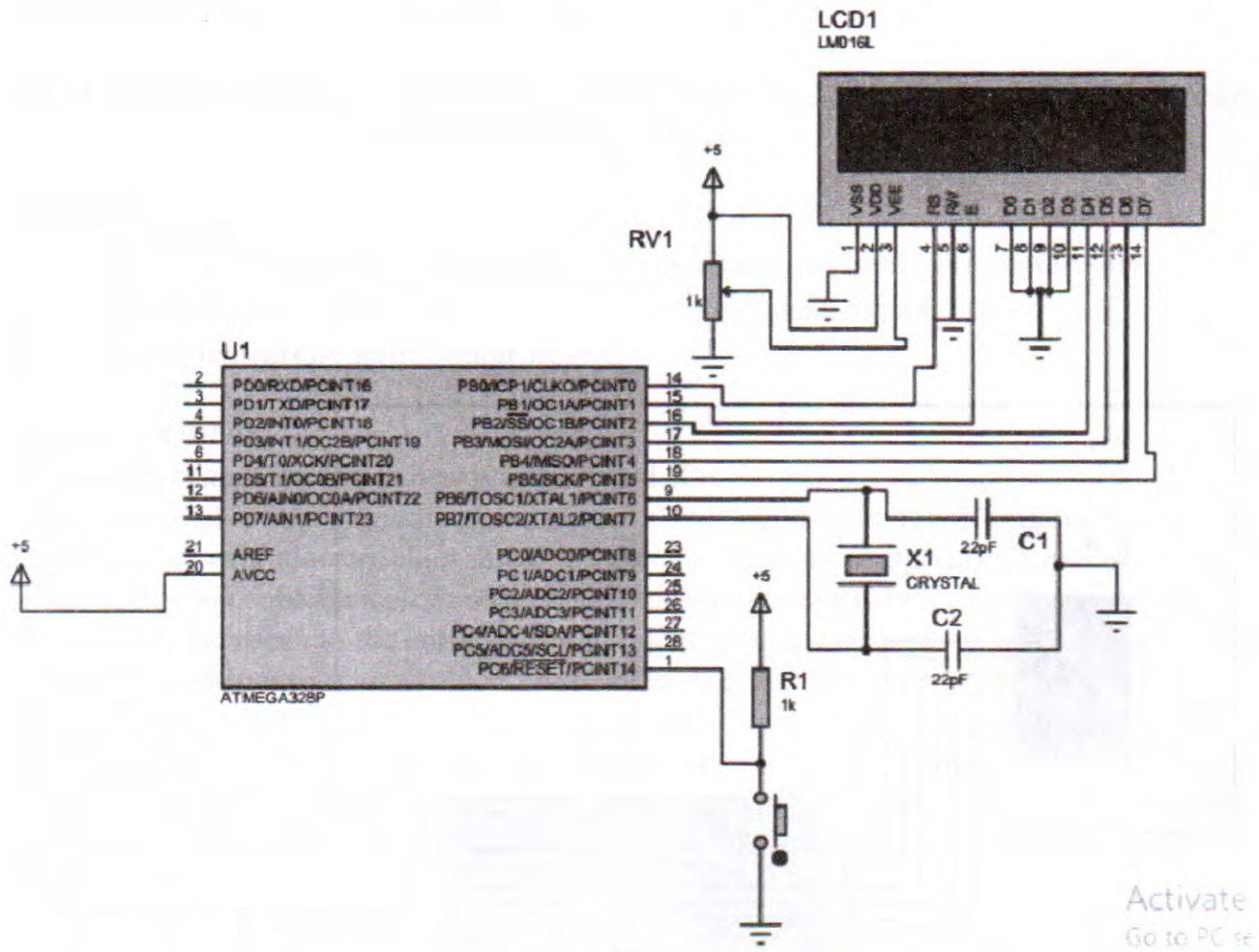


Figure 2.5: LCD Circuit diagram

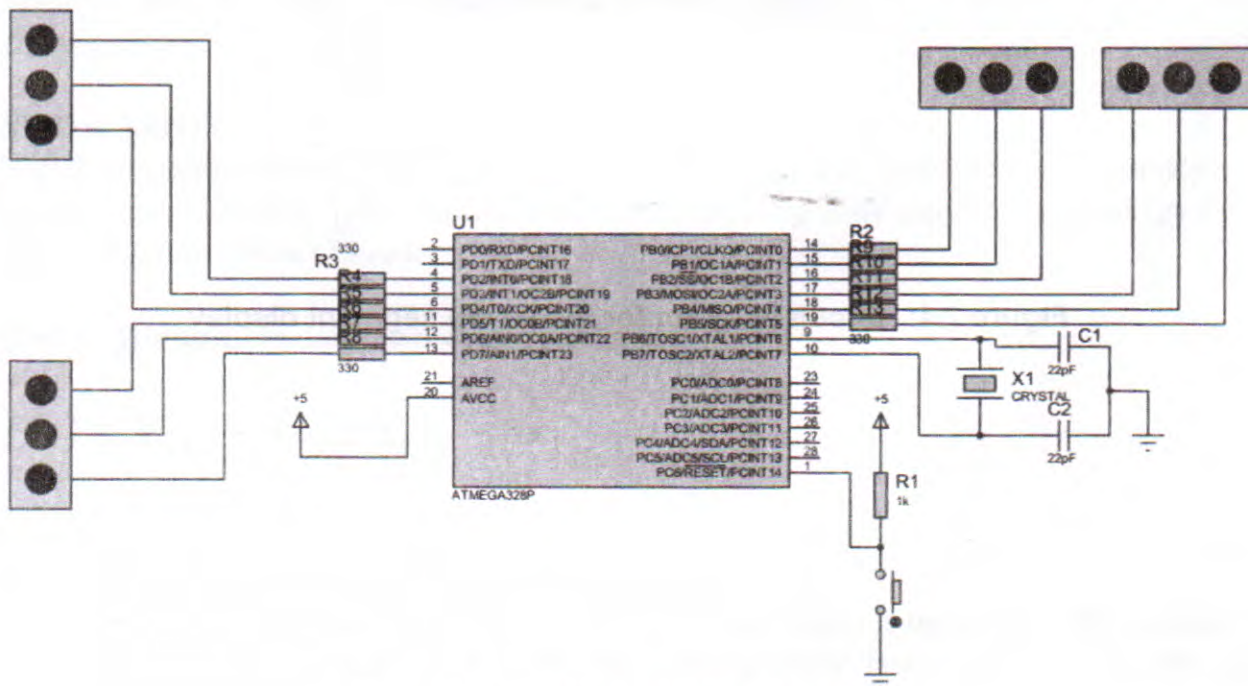


Figure 2.6: Traffic Light Circuit diagram

EXPERIMENT #: UILCPE/MSI3**EXPERIMENT 3:** Interfacing Keypad with ATMEGA328P**OBJECTIVE**

- To understand keypad as an input device
- To solve simple arithmetic operation

EQUIPMENT AND COMPONENTS REQUIRED

1. ATMEGA328P Development Board
2. Personal Computer
3. Simulation Software
4. Resistors
5. Breadboard
6. Keypad

BACKGROUND

The 4X4 keypad consists of 16 keys arranged in a 4 columns by 4 rows matrix as shown in Fig. 3.1. It can be described as a collection of multiple switches to act as a single input device.

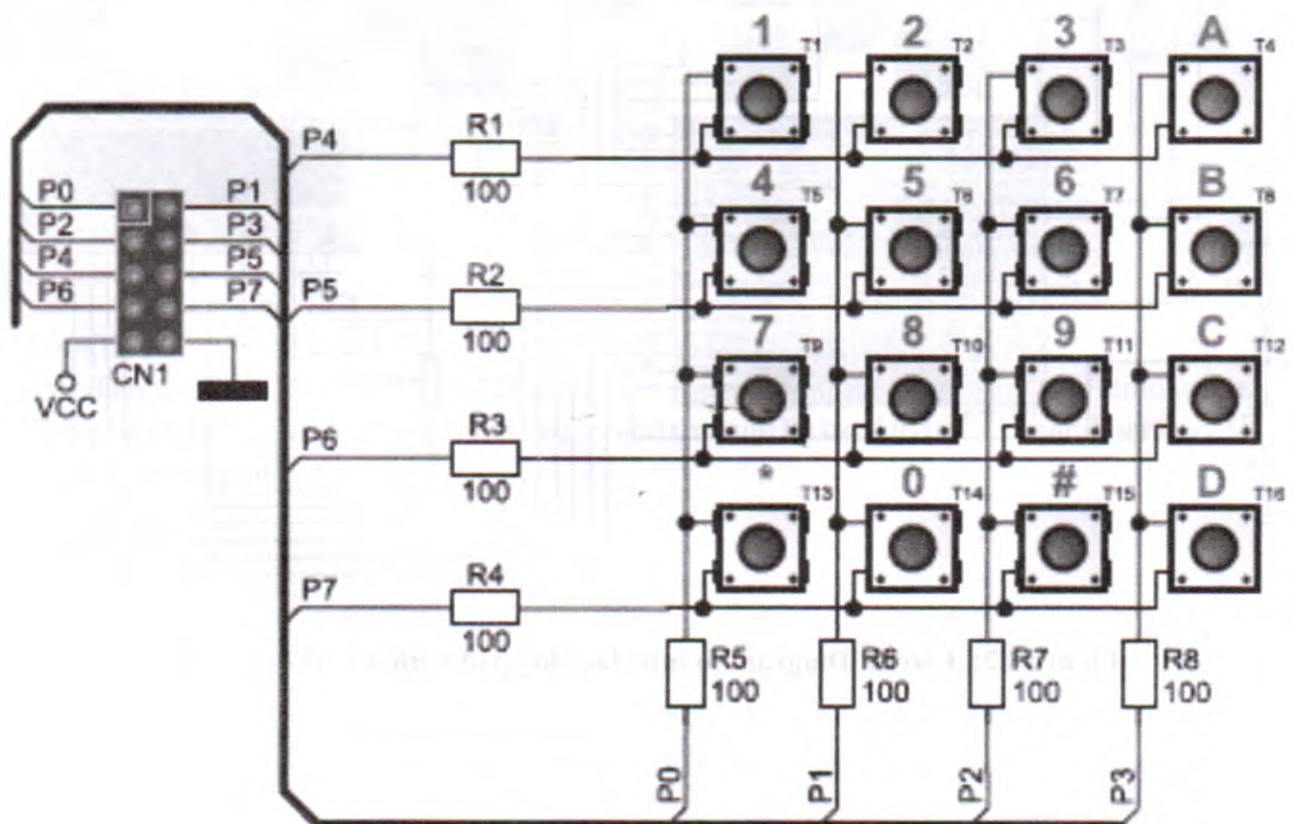


Figure 3.1: Inner structure of a 4X4 calculator keypad

PREPARATION

1. Ensured you have all the required simulation software installed on your PC.
2. Write ATMEGA328P based program to solve the Addition, Subtraction, Multiplication and Division between two 8-bit numbers and display all your inputs and result simultaneously on the LCD (Hint: Use of Keypad Library is required).
3. Reproduce the circuit diagram of this exercise on your simulation software and run your codes in it.

EXPERIMENT

The circuit diagram is connected as shown in Fig. 3.2. Ensure your circuit is free of short circuit before connecting the power supply unit.

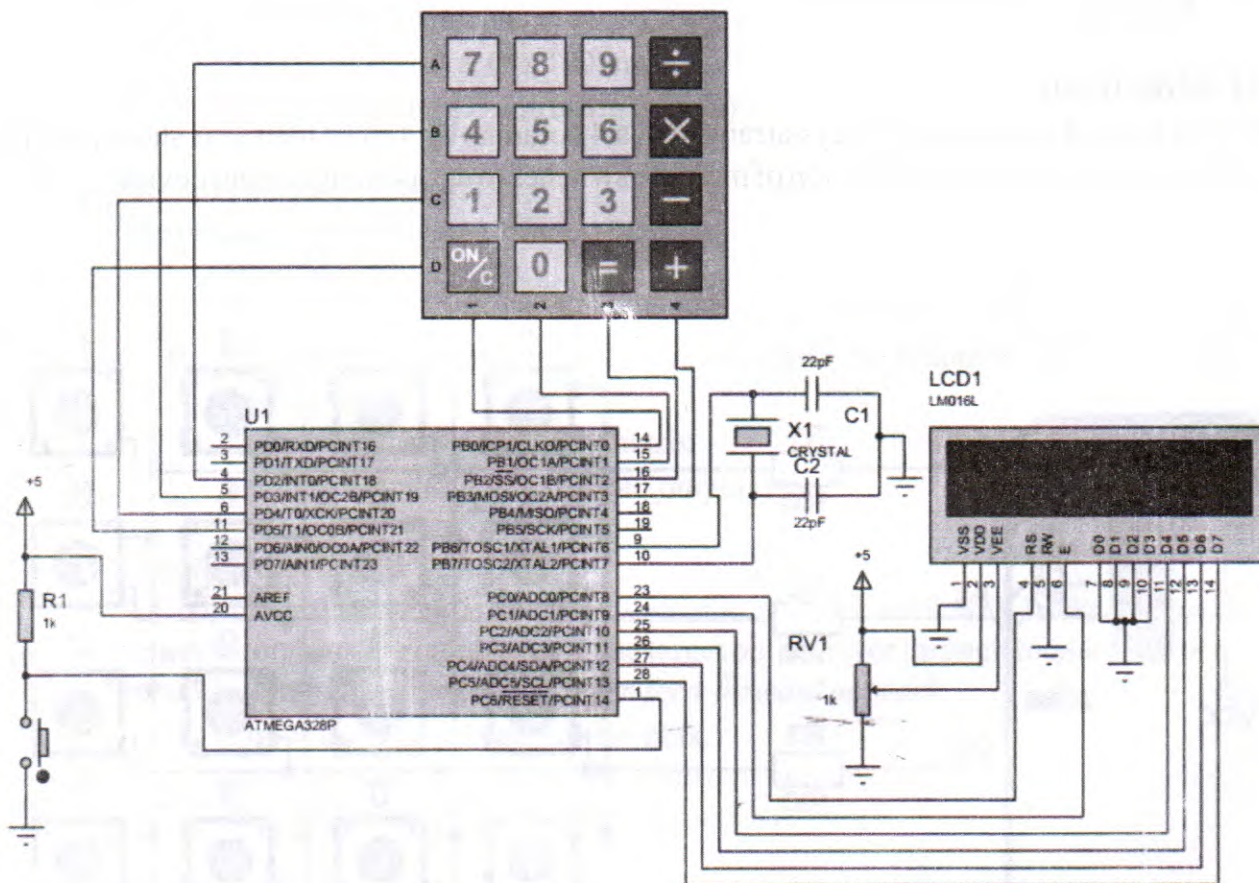


Figure 3.2: Circuit Diagram to interface keypad with ATMEGA328P

EXPERIMENT#: UILCPE/MSI4

EXPERIMENT4: Using ATMEGA328P for Analog to Digital and Digital to Analog Conversion

OBJECTIVES

- To understand Analog to Digital Conversion (ADC) using ATMEGA328P
- To understand digital to analog conversion (DAC) and generate square waveform
- To use Pulse Width Modulation (PWM) feature of ATMEGA328P

EQUIPMENT AND COMPONENTS REQUIRED

1. ATMEGA328P Development Board
2. Personal Computer
3. Simulation Software
4. Resistors
5. Breadboard
6. Temperature Sensor
7. Potentiometer
8. LCD
9. Oscilloscope
10. LED

BACKGROUND

The microcontroller can sense binary signals by interpreting 0V as a binary 0 and five volt (5V) as binary 1. However, signals fed to the microcontrollers between 0V – 5V are referred to as analog signal which can be converted to its digital equivalent value using the Analog to Digital Converter feature (ADC) embedded within the microcontroller in order to make decision or perform other processes in its program code. The ATMEGA328P has inbuilt ADC feature to convert analog voltage on its dedicated ADC pins(23 – 28) to digital number, the ADC on the microcontroller is 10-bit, which signifies that it can convert 1024 (2^{10}) discrete analog values. The ADC operation is carried out in microsecond and the result is in *ratio metric value*, which illustrate that 5V is 1023 and other value less than 5V is the ratio between 5V and 1023 as shown in Equation 4.2. The ratio metric relationship of ADC between 5V and 1023 to compute the ADC result is expressed in Equation 4.1, which is mostly applicable to operation that involves reading voltage signal such as varying voltage output from sensors to make decision or perform other processes.

$$\frac{\text{ADC RESOLUTION}}{\text{REFERENCE VOLTAGE}} = \frac{\text{ADC RESULT}}{\text{ANALOG VOLTAGE INPUT}} \quad (4.1)$$

$$\text{ADC RESULT} = \frac{\text{ANALOG VOLTAGE INPUT}}{5} \times 1023 \quad (4.2)$$

Where reference voltage = microcontroller voltage = 5V and ADC resolution = 1023

Pulse Width Modulation (PWM) is a digital output signal from the microcontroller which is used to control circuitry such as controlling the brightness of RGB led or controlling the direction of the servo motor. This is very essential in robotics which demands precise control of highly sensitive system by varying the time the signal is high (5V) or low (0V) in analog mode. The microcontroller can output 5V or 0V, while varying the time proportion the signal is high compared to when it is low over a consistent time interval gives the privilege to output analog voltage between 0V – 5V by using the concept of duty cycle. The ATMEGA328P has dedicated PWM pins which is 8-bit (2^8) that offers 256 selectable output levels of resolution allotted for the PWM operation which are pin 5, 11, 12, 15, 16 and 17, the signal output of the desired PWM pins can be deduced from equation 4.4 by calculating the PWM digital value to be set or computed in the program in order to output the desired signal level. The PWM operation is also carried out in microsecond and the result is in *ratiometric value*, which illustrate that 5V output is 255 and other value less than 5V is the ratio between 5V and 255 as shown in equation 4.3. The PWM operation is also referred to as digital to analog conversion whereby the digital value or data within the code of the microcontroller is converted to variable (analog) output level by dispersing the input source across a resistor ladder within the microcontroller, while the digital signal controls the output of the ladder which is the pin output of the PWM.

$$\frac{\text{PWMDIGITAL VALUE}}{\text{PWM RESOLUTION}} = \frac{\text{ANALOG VOLTAGE OUTPUT}}{\text{REFERENCE VOLTAGE}} \quad (4.3)$$

$$\text{PWM DIGITAL VALUE} = \frac{\text{ANALOG VOLTAGE OUTPUT}}{5} \times 255 \quad (4.4)$$

Where reference voltage = microcontroller voltage = 5V and PWM resolution = 255

PREPARATION

1. Ensured you have all the required simulation software installed on your PC.
2. Write ATMEGA328P based program to display temperature variation through the ADC port on the LCD.
3. Write ATMEGA328P based program to display a square waveform on the Oscilloscope.
4. Reproduce the circuit diagram of this exercise on your simulation software and run your codes in it.

EXPERIMENT

The circuit diagrams are connected as shown in Fig. 4.1 and Fig. 4.2. Ensure your circuit is free of short circuit before connecting the power supply unit.

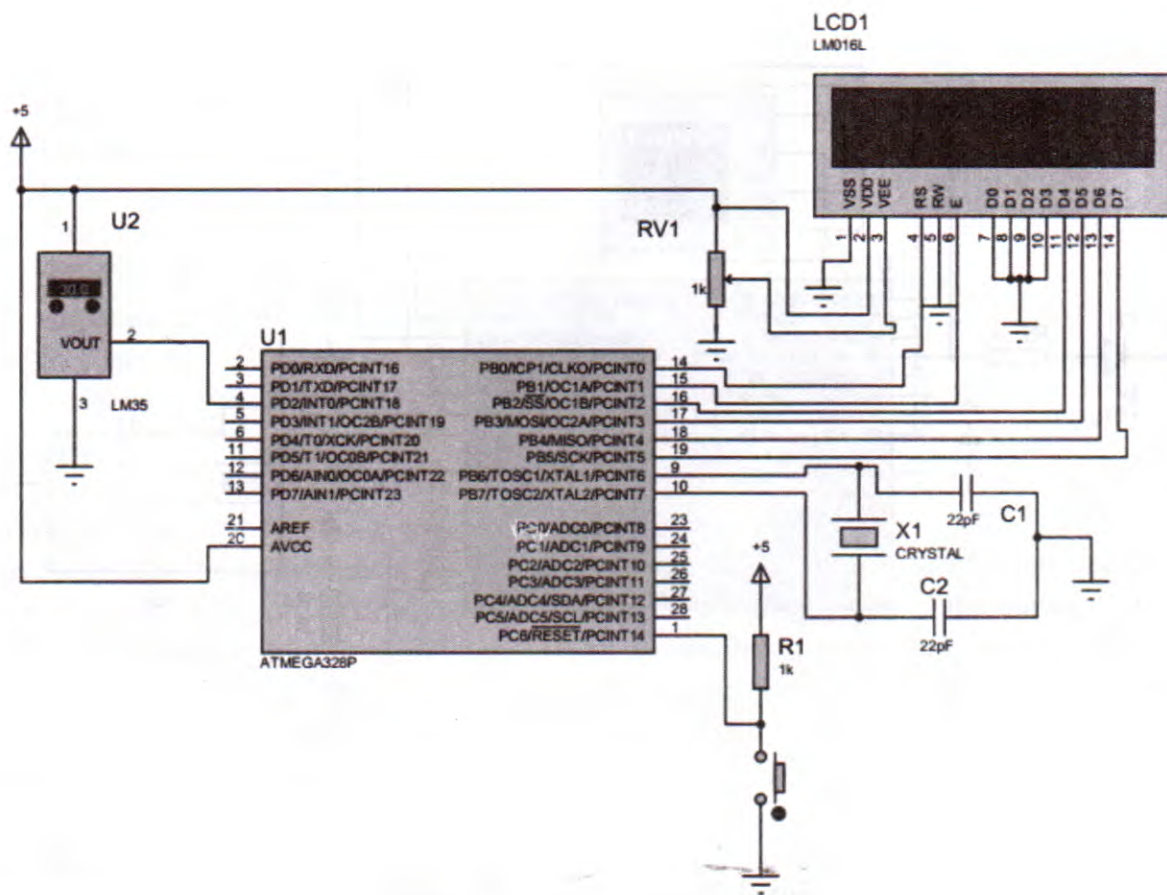


Fig. 4.1: Circuit Diagram for ADC

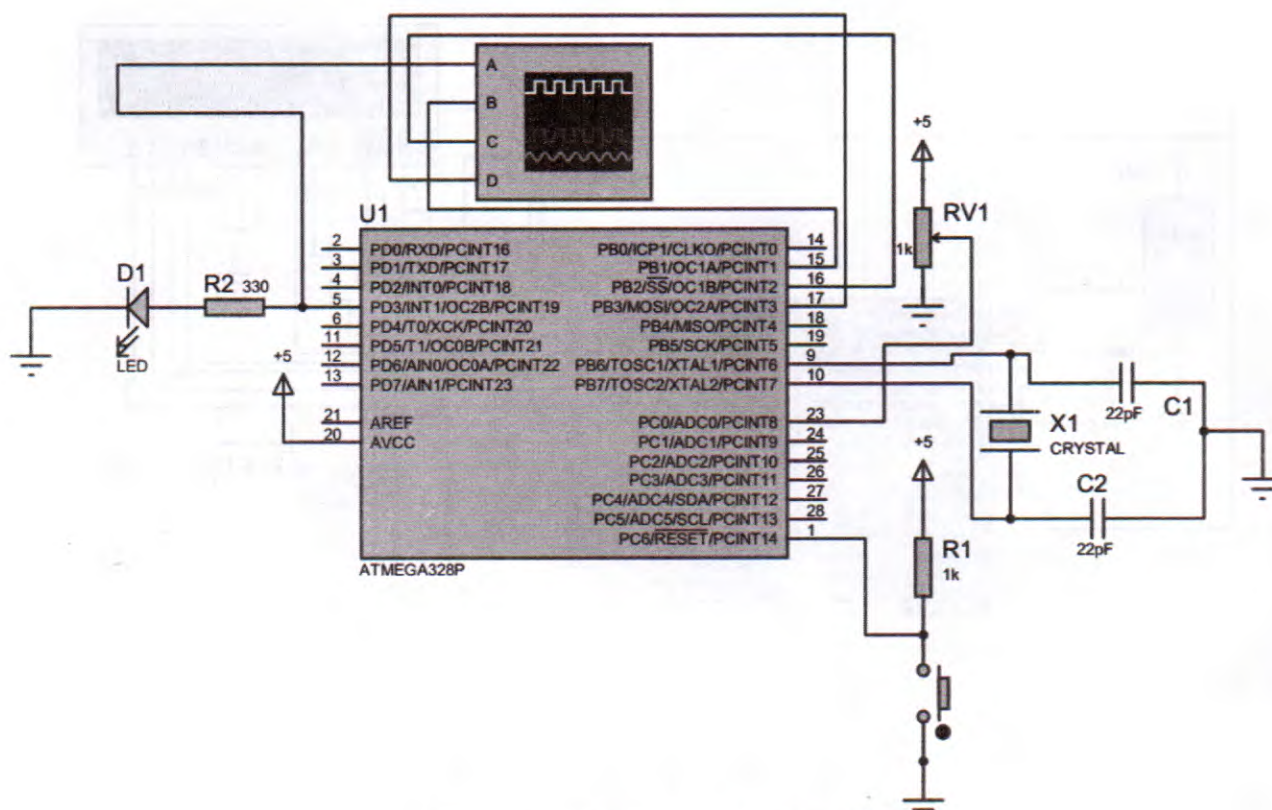


Fig. 4.2: DAC Circuit Diagram

EXPERIMENT #: UILCPE/MSI 5

EXPERIMENT 5: Communication between ATMEGA328P and PC

OBJECTIVE:

- To achieve serial communication between PC and ATMEGA328P using serial USART of the microcontroller

EQUIPMENT AND COMPONENTS REQUIRED

1. ATMEGA328P Development Board
2. Personal Computer
3. Simulation Software
4. Potentiometer
5. LCD

BACKGROUND

Most computers can communicate with other devices by serial communication. Serial communication uses a common protocol where characters are converted into numbers and then transmitted via cables. It is relatively easy to program the microcontroller to communicate to any type of computer.

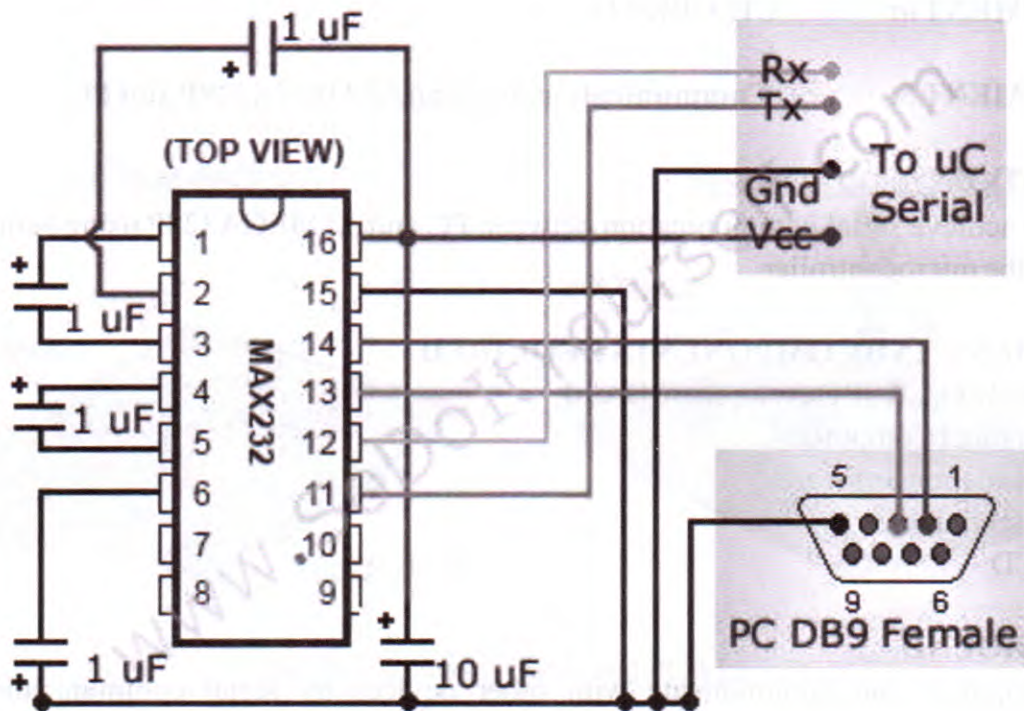


Figure 5.1: Circuit Diagram of Connection between the RS232 and MAX232

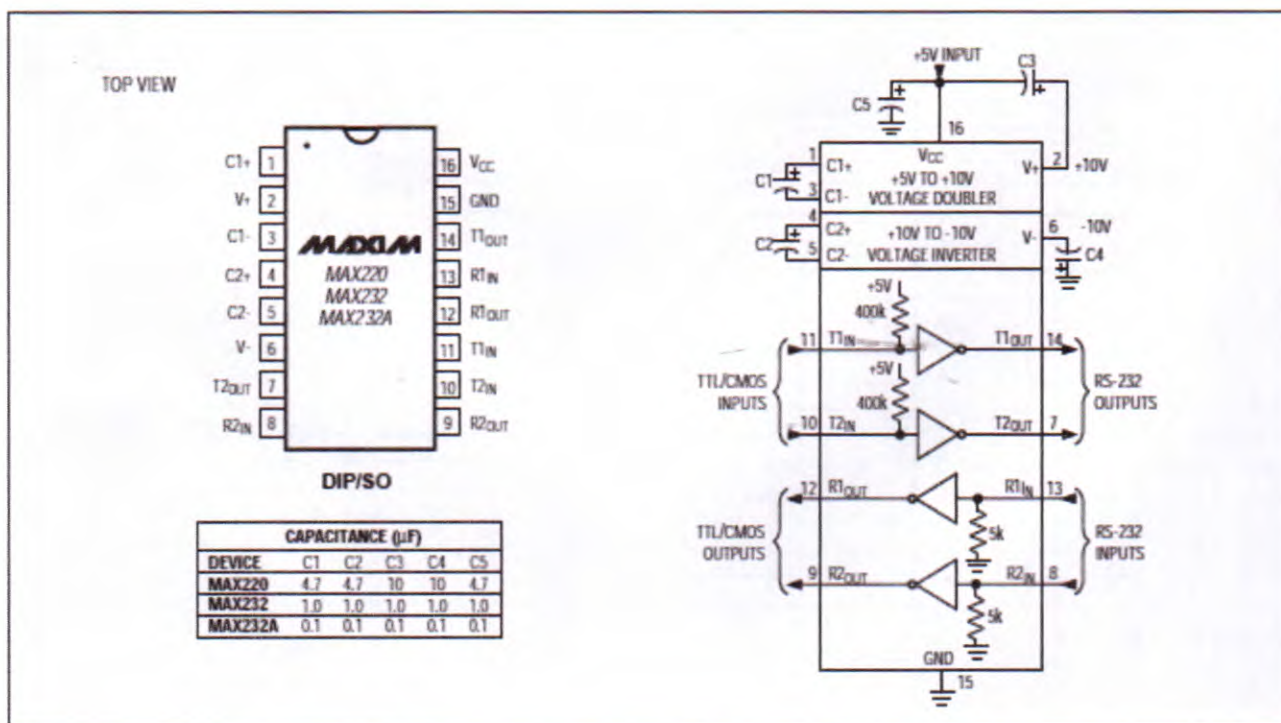


Figure 5.2: Circuit Diagram of Pin out and Inner Configuration of MAX232

MAX232 is the interface that drives the serial connection between PC and microcontroller USART. To use this system, a communication software package is required for the PC. There are various protocols that can be used for serial communication, and it is important that both the computer and the microcontroller use the same setting.

PREPARATION

1. Ensured you have all the required simulation software installed on your PC.
2. Write ATMEGA328P based program to serially connect your PC with ATMEGA328P microcontroller and confirm the connection.
3. Reproduce the circuit diagram of this exercise on your simulation software and run your codes in it.

EXPERIMENT

The circuit diagram is connected as shown in Fig. 5.3. Ensure your circuit is free of short circuit before connecting the power supply unit. The characters input should be serially received through the PC keyboard and displayed on the LCD. The received characters should be sent out serially and displayed on the terminal of the communication package on the PC monitor.

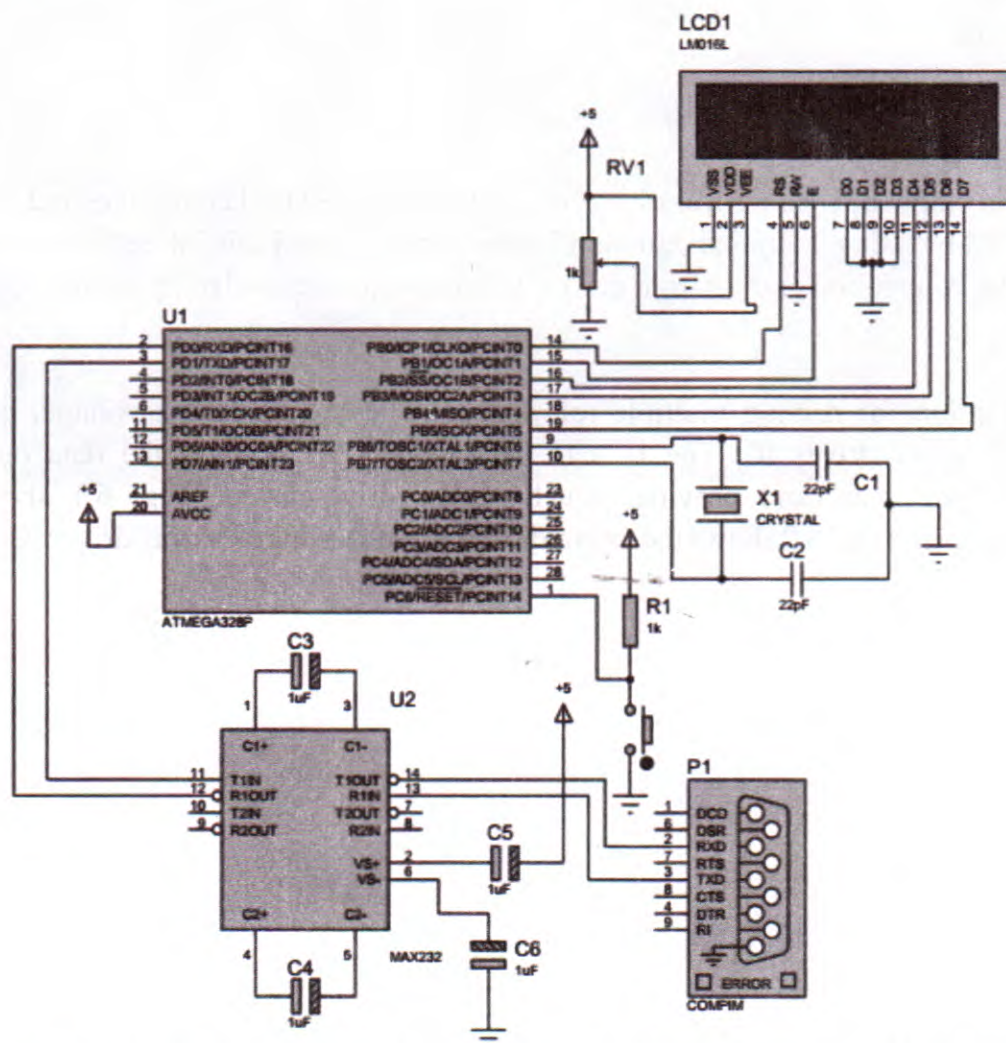


Fig. 5.3: Circuit diagram to implement the serial communication between the MCU and PC

EXPERIMENT#: UILCPE/MSI6

EXPERIMENT 6: Interfacing Stepper Motor and Relays with ATMEGA328P

OBJECTIVE

- To understand how electromagnetic devices are being interfaced with microcontroller for robotics action

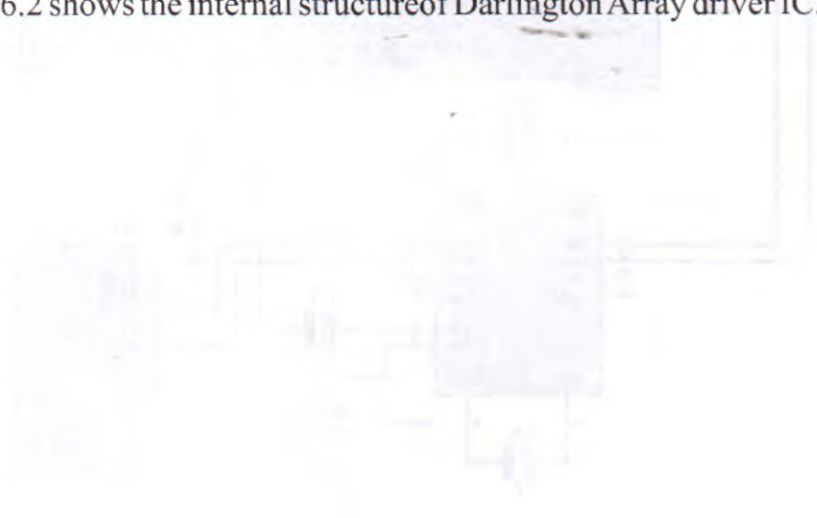
EQUIPMENT AND COMPONENTS REQUIRED

1. ATMEGA328P Development Board
2. Personal Computer
3. Simulation Software
4. Resistors
5. Breadboard
6. Relay
7. Integrated Circuit (IC)
8. Stepper Motor
9. LCD
10. Diode
11. Potentiometer

BACKGROUND

Interfacing output devices such as electromagnetic devices of higher voltages and currents have been solved by using a Bipolar Junction Transistor (BJT) or Field Effect Transistor (FET) to amplify the voltage and current from the MCU to the required level to drive the stepper motor or relay.

Another option for driving multiple relays would be to use a high-voltage, high-current, Darlington array driver IC. The IC can be directly interfaced to the data output of the ATMEGA328P pins, and provides much higher drive-current. Fig. 6.1 shows the pin configuration and Fig. 6.2 shows the internal structure of Darlington Array driver IC.



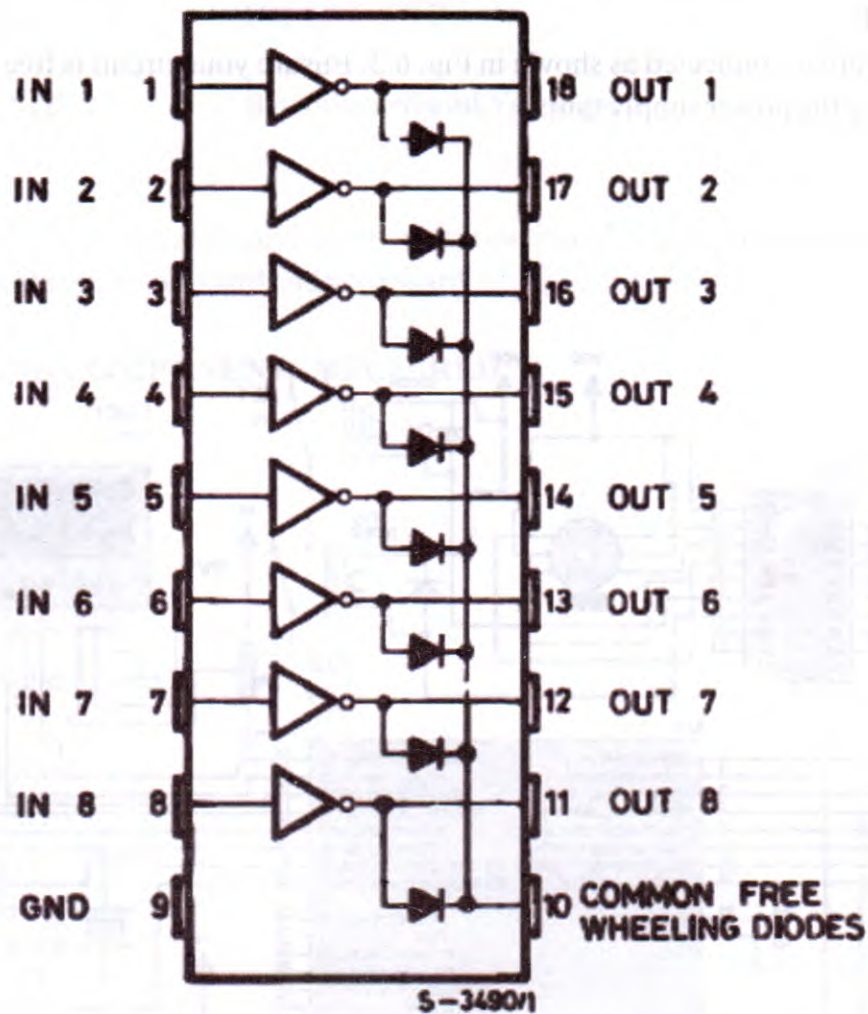


Fig. 6.1: Pin configuration of Darlington Array driver IC

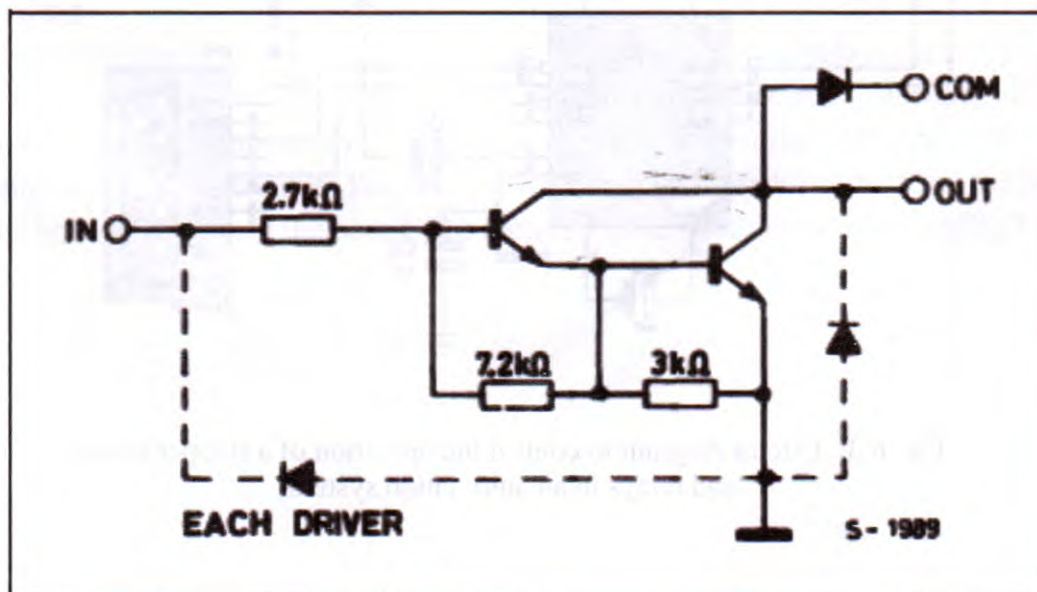


Fig. 6.2: Internal Structure of Darlington Array driver IC

EXPERIMENT

The circuit diagram is connected as shown in Fig. 6.3. Ensure your circuit is free of short circuit before connecting the power supply unit.

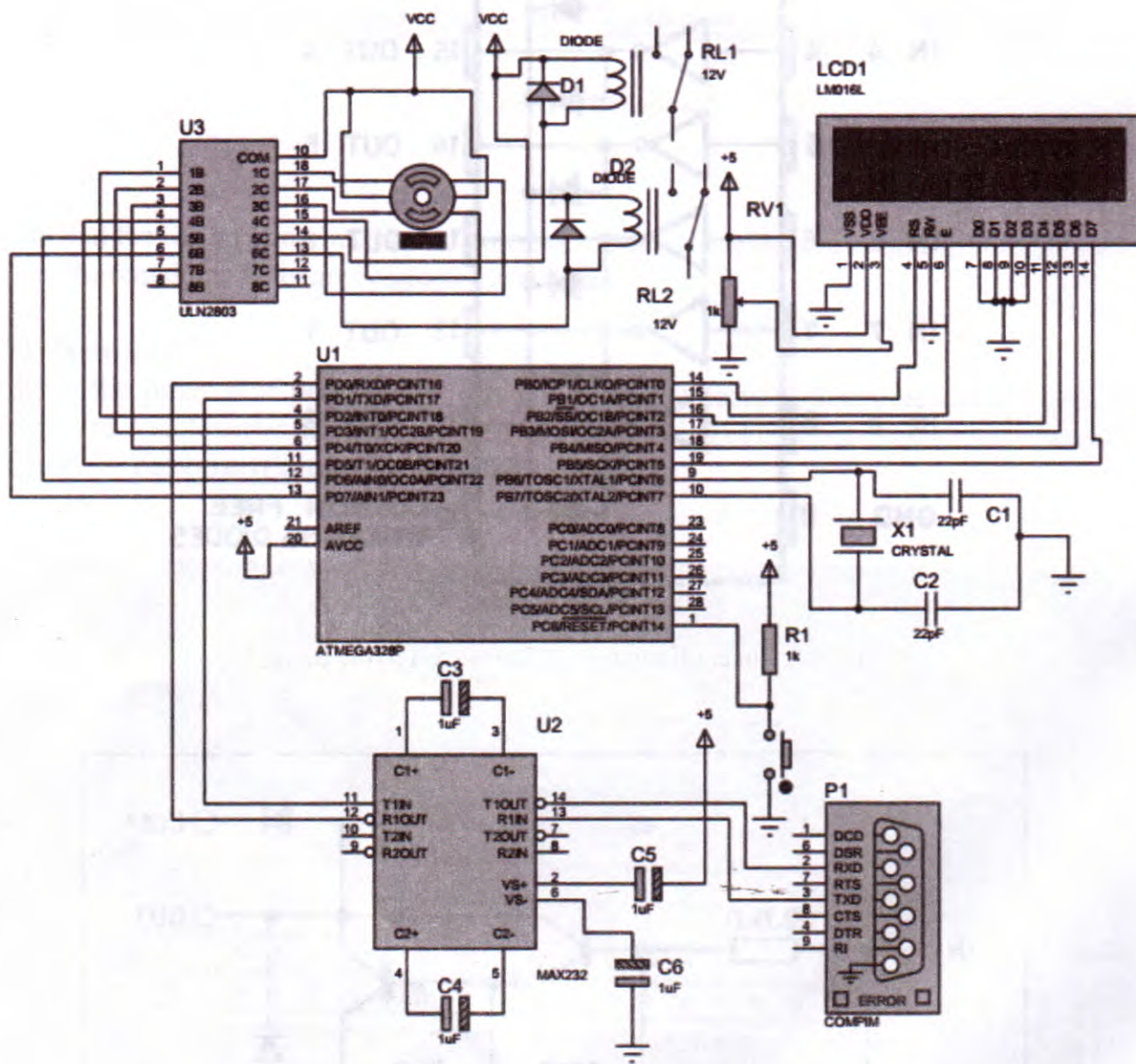


Fig. 6.3: Circuit diagram to control the operation of a stepper motor and relays in an automation system.

EXPERIMENT#: UILCPE/PTP1

EXPERIMENT TITLE: Soldering Techniques

OBJECTIVE

- To give an introduction to basic soldering, de-soldering techniques, required tools, important soldering tips, precautions for pass-through and surface mount components

EQUIPMENT AND COMPONENTS REQUIRED

1. Vero Board
2. Soldering kit
3. Connecting wires

BACKGROUND

Soldering

Soldering is a process of joining two or more metals together by melting or fusing an alloy into the joint which have a relatively lower melting point.

Required Tools and Usage

1. **Soldering iron:** used for applying heat to joints during the soldering process.
2. **Damp sponge:** for cleaning soldering iron tip.
3. **Rosin-core solder:** to electrically and mechanically bond a component to the PCB.
4. **Cutters or side cutter:** for trimming component leads and stripping insulation from wires.
5. **Nose pliers:** for holding, placing and shaping components.
6. **De-soldering pump:** for removing solder.
7. **Scotch tape:** for securing components.
8. **Safety glasses:** for eye protection. These are mandatory in the lab.
9. **Magnifier:** to provide more detail during intricate work. A magnifying glass is convenient, but an illuminated magnifier is better.
10. **Light source:** to prevent eye-strain.
11. **Ventilation:** to extract and dispel fumes generated during the soldering process.
12. **Flux:** to clean components and PCB pads.
13. **Brush:** to assist in the removal of flux residue.

Prior to soldering it is a good idea to have all components organized as it will make populating the PCB more efficient.

The following tips provide a quick guideline on how to make proper joints.

1. **Cleanliness:** All parts, including the soldering iron tip, must be clean and free from grease, oxidation and contamination.
2. **Tinning:** the soldering iron tip must also be tinned (coated with solder). Tinning the tip allows solder to flow on the components more quickly rather than the soldering iron tip itself.
3. **Temperature:** Ensure that both the component leads and the PCB's copper layer are

heated at the same time. The soldering iron tip should contact both the component and the PCB pad.

4. **Duration:** The duration that the iron is in contact with the component and PCB is dependent on the size of the joint and your soldering iron temperature.
5. **Adequate Solder Coverage:** If too little solder is applied, the joint will not make a secure connection as shown in Fig. 1.1 which may cause erratic behavior. However, if too much solder is applied, the joint may bridge with adjacent joints resulting in electrical shorts as shown in Figure 1.1.



Figure 1.1: Examples of Joints

Desoldering

This is the process of removing soldered metals with the use of desoldering pump and soldering iron. Solder Pump/Sucker is used to suck up melted solder from soldered joints as shown in Figure 1.2.



Figure 1.2: Desoldering Process

EXPERIMENT:

Make five joints on a Vero-board using the provided connecting wires.

EXPERIMENT #: UILCPE/PTP2

EXPERIMENT TITLE: Wire Wrapping Technique

OBJECTIVE

- To introduce students to electronics circuit interpretation and implementation using wire wrapping technique.

EQUIPMENT AND COMPONENTS REQUIRED

1. DMM
2. Wire Wrapping Machine
3. Pack of insulation wire
4. Cutting plier
5. Long nose plier
6. Picker
7. Power Supply Unit
8. Transistor
9. Resistors
10. Vero-board
11. Buzzer
12. LDR(Light Dependable Resistor)
13. Lamp(Bulb)
14. Button switch

BACKGROUND

Wire Wrapping

Wire wrapping is a method of constructing electronics circuit board in which electronics components mounted on an insulating board are interconnected by lengths of insulated wire run between their terminals. With the connections made by wrapping several turns around a component lead or a socket pin. Wires can be wrapped by hand or by machine, and can be hand modified afterwards as shown in Figure 2.1.

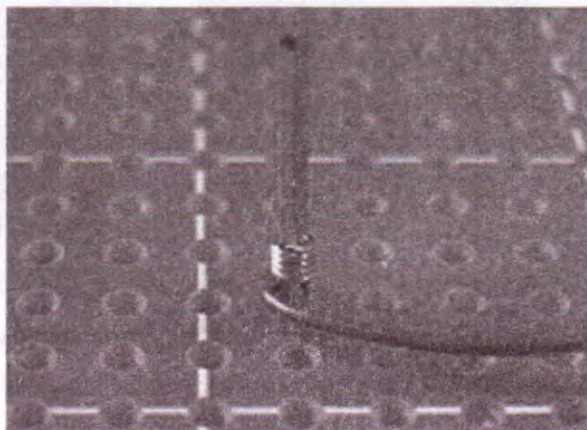


Figure 2.1: Wire-wrap Connection

EXPERIMENT

Implement the circuit diagrams shown in Figure 2.2 and Figure 2.3 using wire wrapping technique. Ensure all your connections are intact.

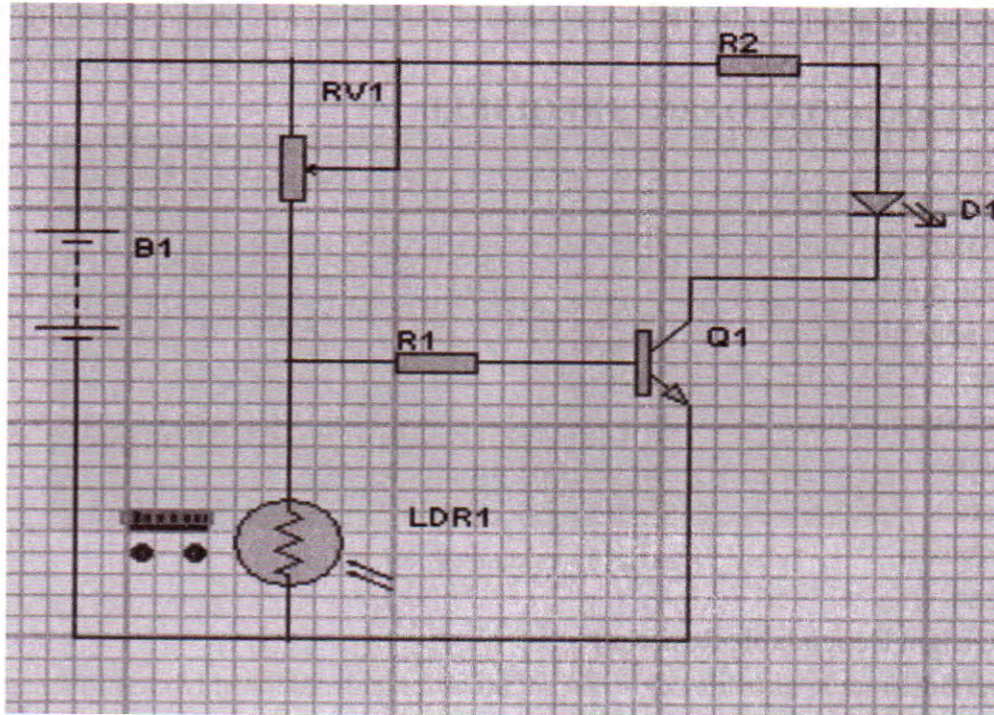


Figure 2.2: CircuitDiagram to Implement Light Switching System

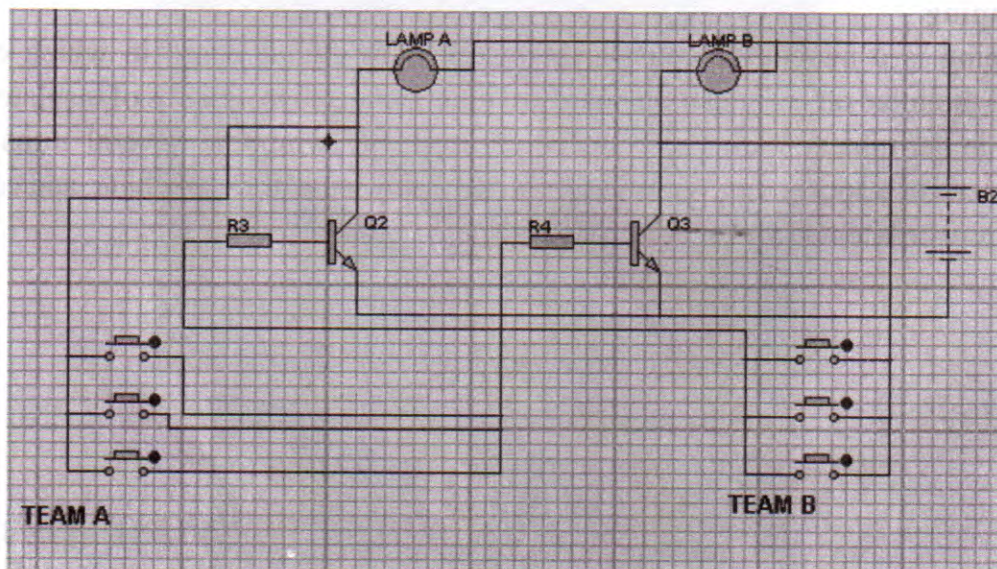


Figure 2.3: Circuit Diagram to Implement Transistor as a Switch

EXPERIMENT #: UILCPE/PTP3

EXPERIMENT TITLE: Breadboard and Veroboard

OBJECTIVE

- To improve students understanding on interpretation of electronics circuit diagrams and to implement it on breadboard and veroboard

EQUIPMENT AND COMPONENTS REQUIRED

1. DMM
2. Soldering Iron
3. Soldering Lead
4. Flux
5. Sucker
6. Cutting pliers
7. Long nose plier
8. Picker
9. Veroboard track cutter
10. Power Supply Unit
11. Loud Speaker
12. Op Amp
13. Capacitors
14. Resistors
15. Veroboard

BACKGROUND

Breadboarding

This is a process of placing and connecting electronic components on breadboard. A breadboard is a solderless device for temporary prototyping electronic circuit designs. Electronic components can be interconnected by inserting their leads or terminals into the holes and then making connections through wires. The breadboard has strips of metal underneath the board and connects the holes on the top of the board. The metal strips are laid out as shown in Fig 3.1

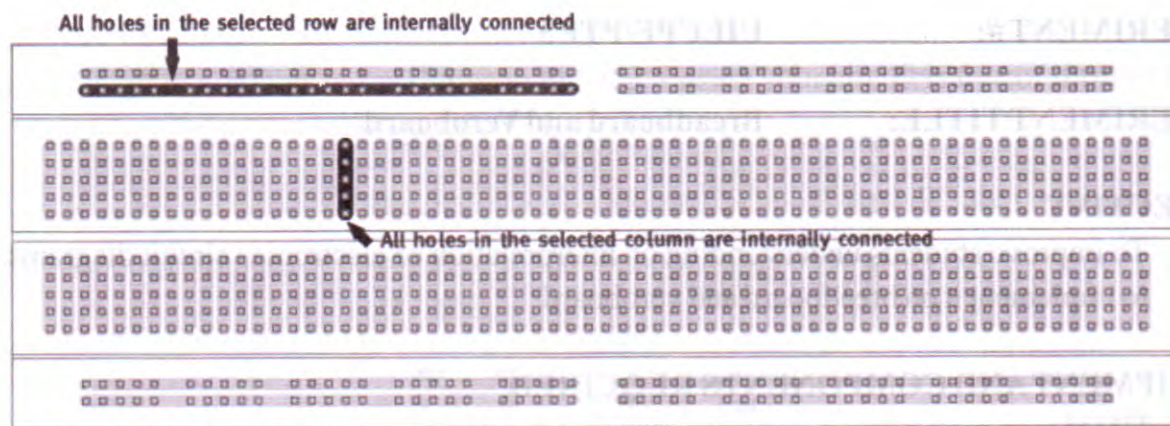


Figure 3.1: Breadboard

Note that the top and bottom rows of holes are connected horizontally and split in the middle while the remaining holes are connected vertically.

Veroboarding

This is the process of permanently arranging and connecting electronic components on a veroboard. Veroboard is designed primarily for hard wiring of discrete components as shown in Figure 3.2. Veroboard Features are as follows;

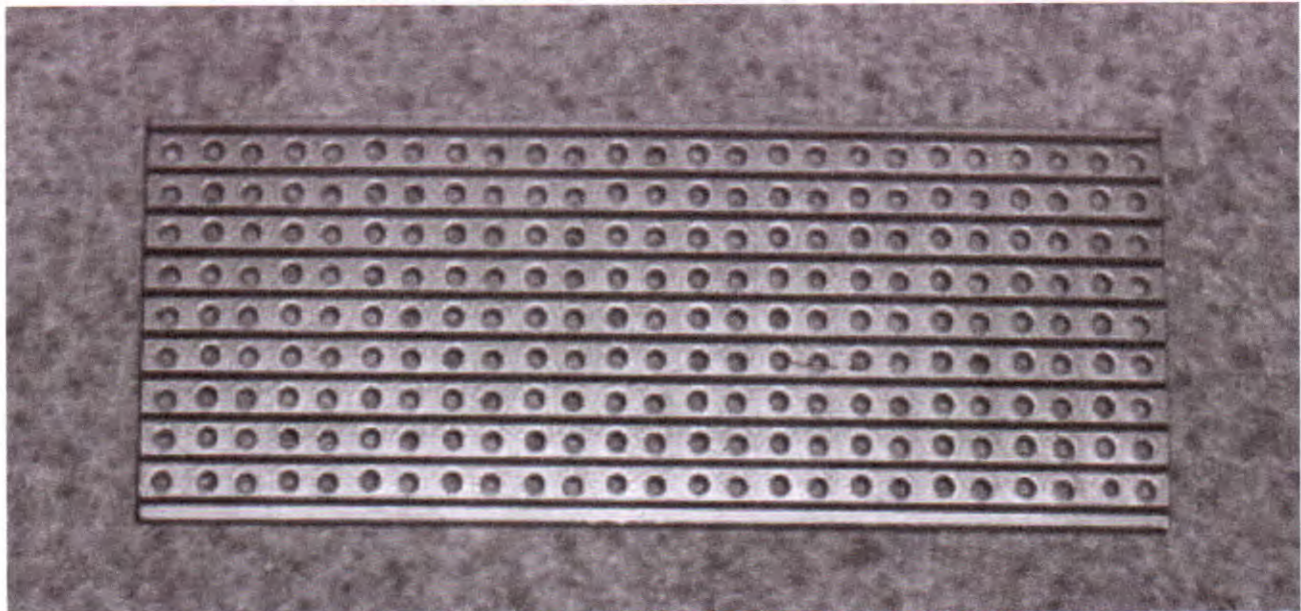


Figure 3.2: Veroboard

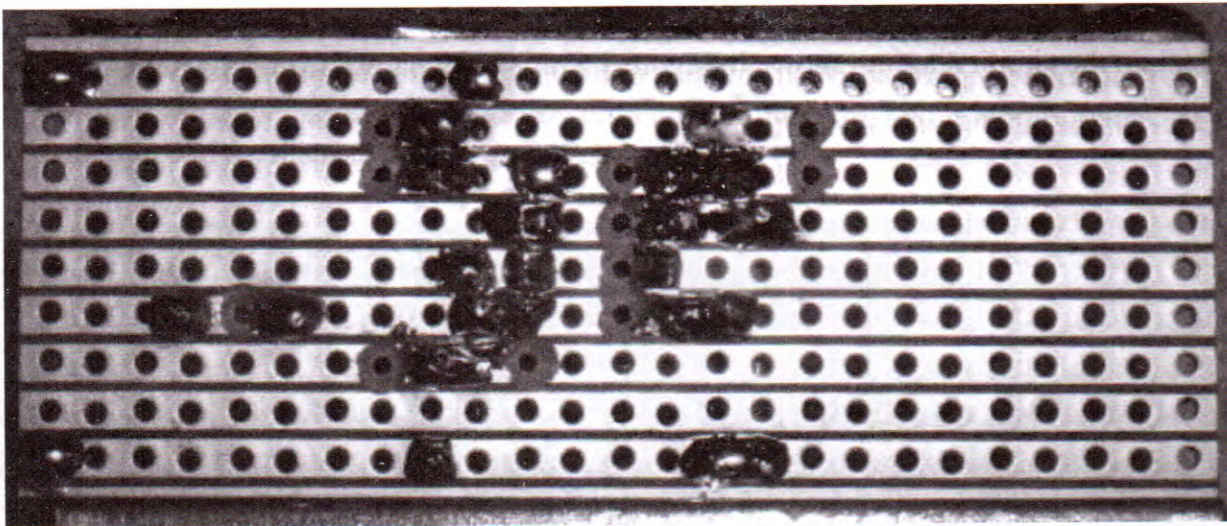


Figure 3.3: Typical Soldered Joint on a Veroboard

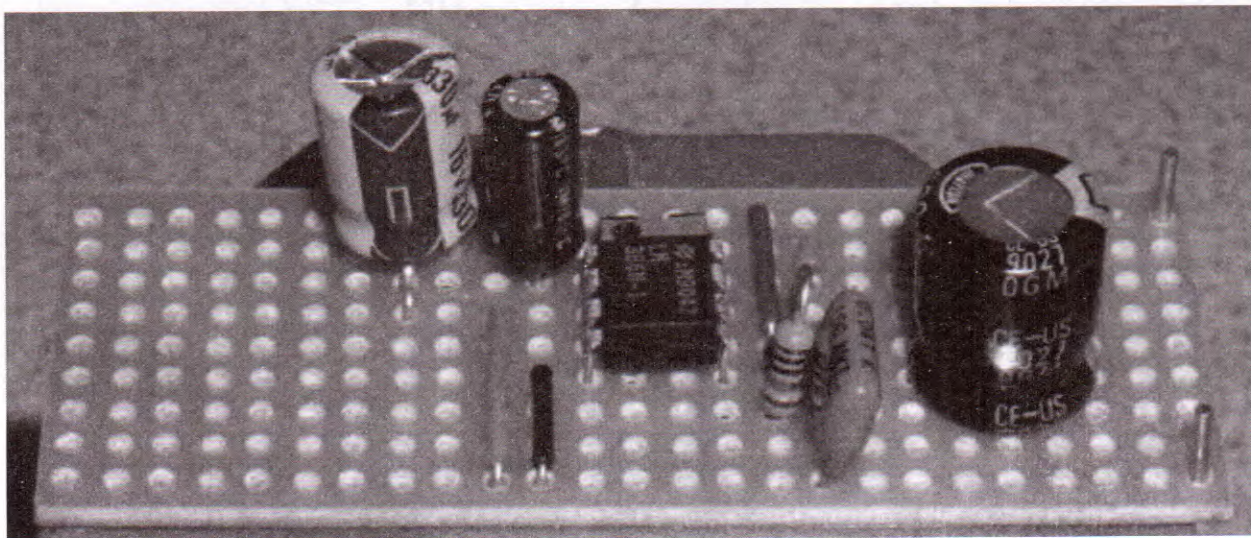


Figure 3.4: Typical Circuit Arrangement on a Veroboard

EXPERIMENT

Implement the circuit diagram shown in Figure 3.5 on a veroboard. Ensure all your connections are intact.

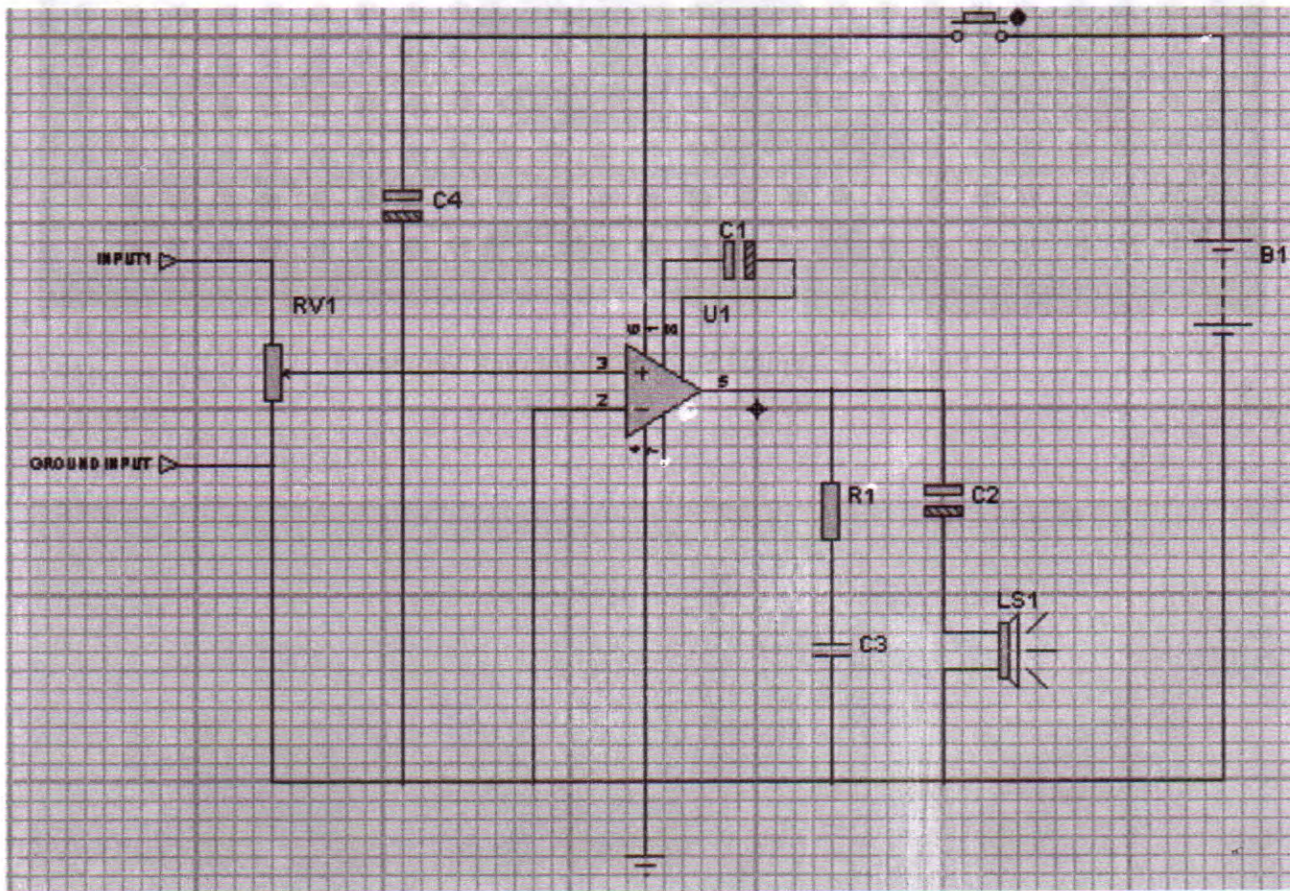


Figure 3.5: CircuitDiagram to Implement an Amplifier using Op-Amp

EXPERIMENT #: UILCPE/PTP4

EXPERIMENT TITLE: Printed Circuit Board

OBJECTIVE

- To learn to use PCB in designing for reliability, efficiency and neat circuit arrangement

EQUIPMENT AND COMPONENTS REQUIRED

1. DMM
2. Soldering Iron
3. Solder (lead)
4. Flux
5. Sucker
6. Cutting pliers
7. Long nose plier
8. Picker
9. Power Supply Unit
10. 555 Timer IC
11. Capacitors
12. Resistors
13. PCB
14. Buzzer

BACKGROUND

Electronic circuits are normally manufactured through the use of Printed Circuit Boards (PCBs). The boards are made from glass reinforced plastic with copper tracks in the place of wires. Figure 4.1 shows the copper tracks and PCB layout.

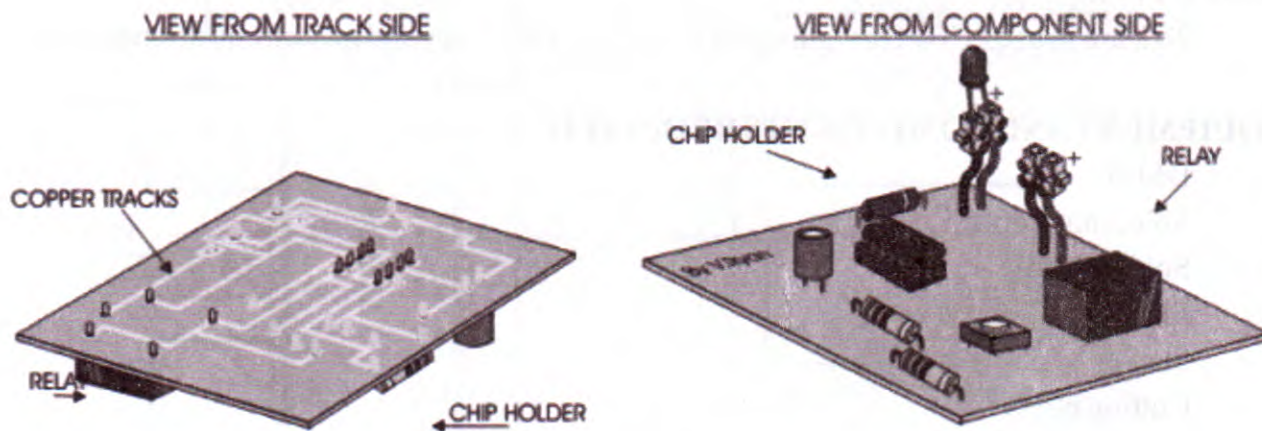


Figure 4.1: Copper tracks and PCB Layout

EXPERIMENT

Implement the circuit diagram shown in Figure 4.2 using PCB. Ensure all your connections are intact.

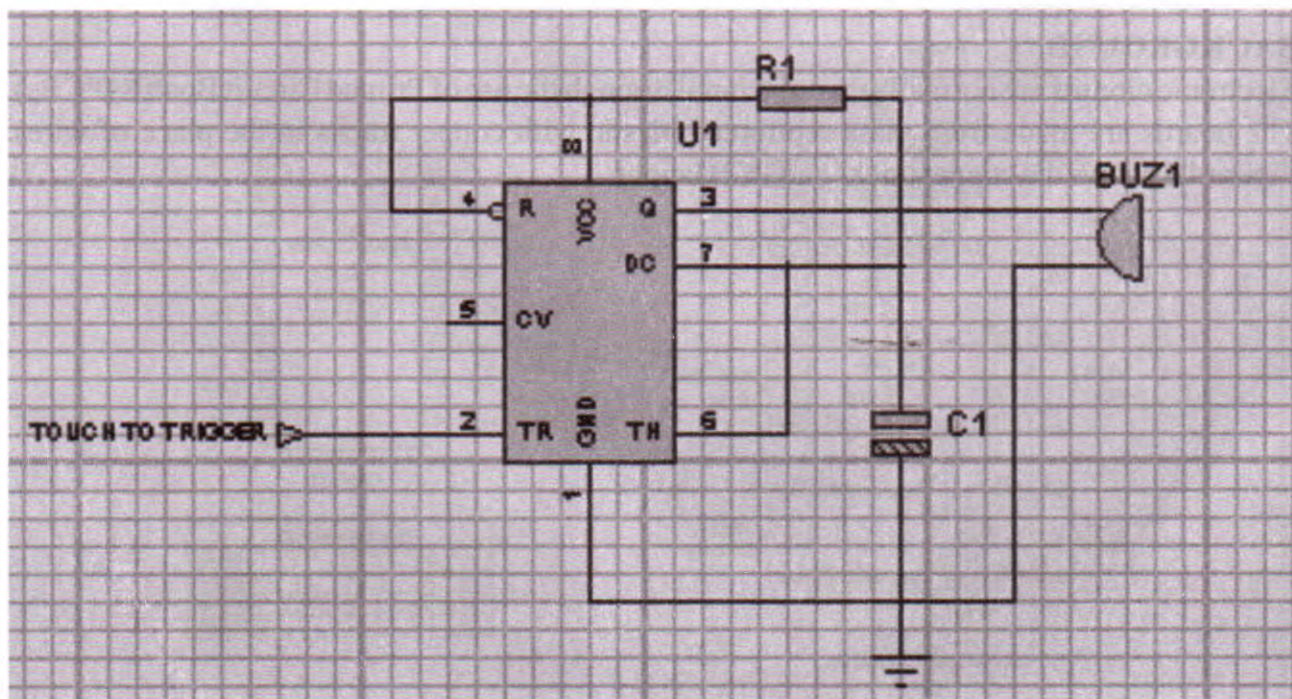


Figure 4.2: CircuitDiagram of a Burglary Alarm

EXPERIMENT #: UILCPE/PTP5

EXPERIMENT TITLE: Radio Frequency (RF) Implementation

OBJECTIVE

- To implement Radio Frequency related project

EQUIPMENT AND COMPONENTS REQUIRED

1. DMM
2. Soldering Iron
3. Solder (lead)
4. Flux
5. Sucker
6. Cutting pliers
7. Long nose plier
8. Picker
9. Power Supply Unit
10. 555 Timer IC
11. Capacitors
12. Resistors
13. Veroboard
14. Diode

BACKGROUND

Radio Frequency transmission is the process of transmitting audio signal from one point to another. It involves amplification, modulation and then transmission of the audio signal. It is carried out by amplifying the audio signal using an amplifier, generating a carrier signal and modulating the carrier signal with the amplified audio signal using a variable frequency oscillator circuit. The power of the RF signal from the oscillator is amplified using a power amplifier to produce a low impedance output, matching that with the antenna.

EXPERIMENT

Implement the circuit diagram shown in Figure 5.1 on the breadboard and then on the veroboard.



EXPERIMENT #: UILCPE/PTP6
EXPERIMENT TITLE: Grounding

OBJECTIVE

- To implement grounding technique for equipment safety and protection

EQUIPMENT AND COMPONENTS REQUIRED

1. DMM
2. Soldering Iron
3. Solder (lead)
4. Flux
5. Sucker
6. Cutting pliers
7. Long nose plier
8. Picker
9. Vero tracker cutter
10. Power Supply Unit
11. Resistors
12. Buzzer
13. Light Dependent Resistor (LDR)
14. Button switch
15. Lamp(Bulb)

BACKGROUND

Grounding is the process of electrically connecting all the ground or zero (0V) potential voltage level together; it enables a uniform reference voltage at any frequency. The components in each piece of equipment are typically tied through a common point to the case ground.

EXPERIMENT

Power Supply and Common Ground

The circuits in Experiment 2 (Figure 2.2) and Experiment 4 (Figure 4.2) should be connected to a common source of power supply and ground as shown in Fig.6.1. Implement the connection on both breadboard and veroboard.

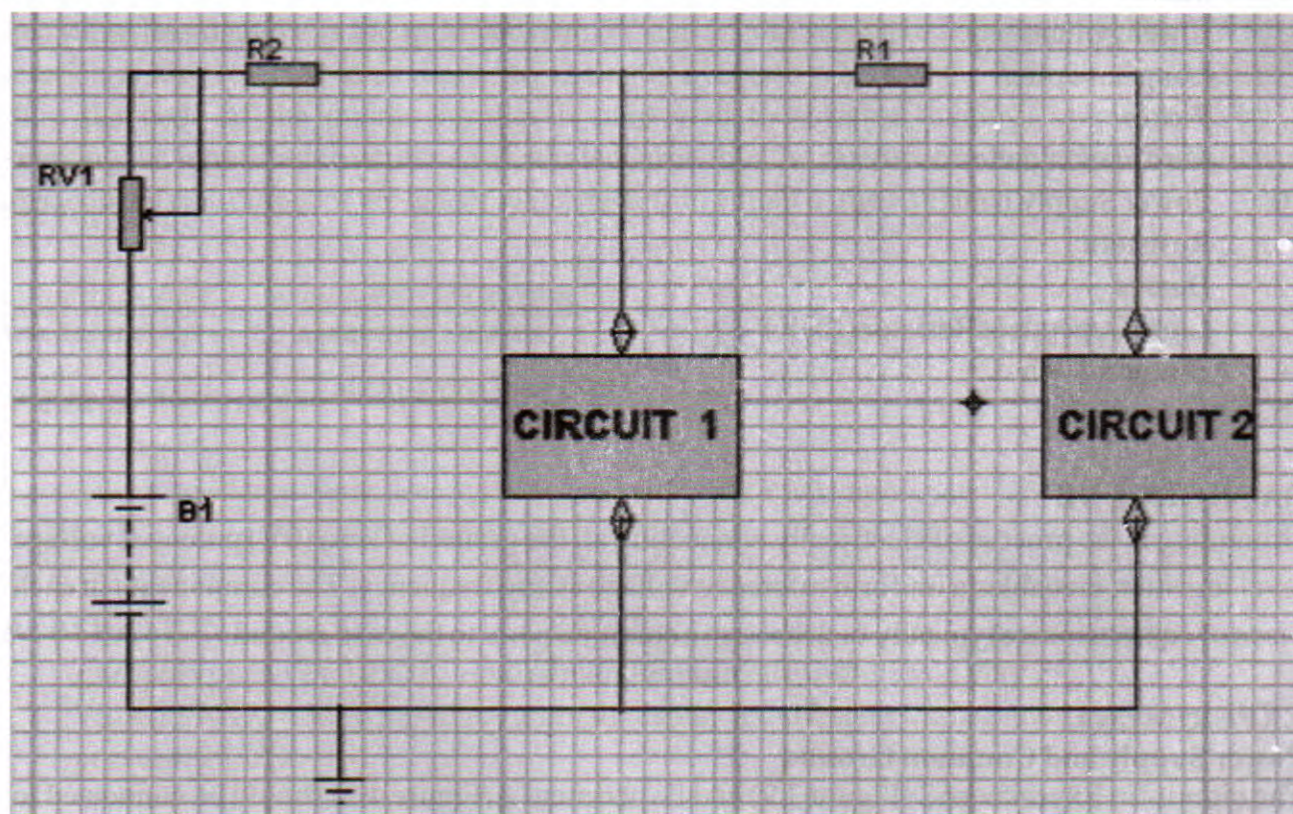


Figure 6.1: Circuit Diagram of Power Supply and Common Ground

EXPERIMENT #: UILCPE/AIA 1

EXPERIMENT TITLE: Developing a Back Propagation Neural Network for Pattern Recognition Systems

OBJECTIVE:

- To simulate a neural network with varying number of hidden nodes
- To train the network to obtain optimal results
- To compare the results of training and validation sets

BACKGROUND

A neural network is an interconnection of simple elements known as neurons which exchange information with each other. They are inspired by biological neural networks. The connected neurons have numeric weights which can be tuned or adjusted in a process termed training of the network. This training is continually done through many input / target pairs to make the neural network adaptive such that a particular input leads to a specific target output as shown in Fig. 1.1.

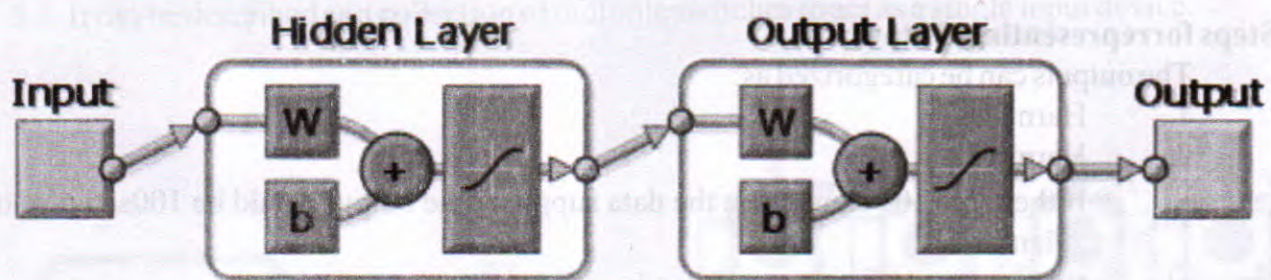


Figure 1.1: A Simple Neural Network Illustration

SIMULATION

The neural network toolbox in MATLAB would be employed for the purpose of this simulation. The Backpropagation training function in the toolbox would be used to train a neural network to solve a pattern recognition problem.

Steps for solving problems using ANN:

I. Define the Problem

This is the process of representing the data available as a set of input vectors and desired output vectors (target vectors) respectively.

Steps for representing Input:

The inputs based on the intrusion data given in the Pattern Recognition directory is in the following categories:

1. Duration: length of connection
2. Protocol Type: TCP, UDP
3. Service: Network service type
4. Flag: Status of the connection
5. Count: Number of connections to the same point in a space of 5 seconds
 - If there are 100 samples in the data supplied, the input would therefore be: 100 samples of 5 elements
 - Represent this as a: (5 x 100) matrix.
 - Save the inputs (e.g. intrusion_inputs)

Steps for representing Output:

The outputs can be categorized as:

1. Harmless
2. Harmful
 - If there are 100 samples in the data supplied, the output would be 100 samples of 2 elements
 - Represent this as a (2 x 100) matrix
 - Save the outputs (e.g. intrusion_outputs)

II. Creating the Neural Network

A feed forward network is the most common network used with backpropagation. The network connects the input layer to the target output layer through hidden nodes layer in a network commonly termed: *two-layer-feed-forward network*

- Create the network using the syntax:
`net = newff(intrusion_inputs, intrusion_outputs, 20)`
 (the '20' represents the number of hidden neurons)

III. Training the Network

- Divide the data into Training, Validation and Testing (e.g. Training (70%), Validation (15%), Testing (15%))
- Begin training using the syntax:
`net = train(net, intrusion_inputs, intrusion_outputs)`

Questions:

1. How many iterations were observed?

2. How did the training last?
3. What is the performance index?
4. At what gradient level was the best result obtained?
5. Tabulate the results for the results, validation and testing indicating the number of samples, mean-squared-error and percentage error of recognition

IV. Present Plots

Click on the training window to produce the following plots:

- Performance plot

Questions:

1. At what level of accuracy was the best performance produced?
2. At what time was the best performance reached?
3. How would you rate the performance of the training from the plot?
 - Receiver Operations Characteristic (ROC)

Questions:

1. Which axis represents the sensitivity of the network?
2. Estimate from the ROC plot, the sensitivity and specificity of the trained network
 - Training State Plot

Questions:

1. At what gradient was the best result obtained?
2. How many validation checks were performed to obtain the best result?
 - Confusion Plot

Questions:

1. What does the confusion plot suggest?
2. What do the percentages indicate?

V. Retraining the Network

If the network is not giving satisfactory results, it can be trained using the syntax:

`net = init(net) % to initialize the feed-forward network using different network parameters`
`net = train(net, intrusion_inputs, intrusion_outputs)`

- Present the four plots indicated in (iv) and compare with the first training session.
- Vary the number of hidden neurons from 20 to 100 with intervals of 20.
- Train at each session, and record the accuracy level at which the best results were obtained.
- Make a plot of the network size and the accuracy level for each of the training sessions.
- What can you infer from the plot?

Note: The GUI for Neural Network Pattern Recognition tool (nprtool) could also be used for the processes. It can be selected from the toolbox section in the MATLAB start menu.

EXPERIMENT#: UILCPE/AIA2

EXPERIMENT TITLE: Optimizing Constrained and Unconstrained Functions using Genetic Algorithm (GA)

OBJECTIVES:

- To formulating fitness functions from problems and represent them in a GA solver
- To present constraints as a function in the GA Optimization tool
- To implement GA optimization process

BACKGROUND

Genetic Algorithm is a search heuristic modelled after the process of natural selection. It can be employed to solve both constrained and unconstrained optimization problems. It belongs to the large class of evolutionary algorithms. It generates solution using various processes which include selection, inheritance, crossover, mutation, elite count etc. In the process, the population of candidate solutions to the optimization problem is continually generated through series of iterations till a better solution evolves. The population in each iteration is called a generation.

At each step, the GA randomly selects individuals from the current population as parents to produce offsprings for the generation that follows. Over successive generations, the population 'evolves' to give an optimal solution. It is often applied for standard optimization problems where the objective function is discontinuous, non-differentiable, stochastic or highly non-linear.

To optimize a constrained or an unconstrained function, an objective function would be required which is commonly referred to as the fitness function. In the selection process, the fitness of the every individual in the population is evaluated. The objective function usually contains decision variables (which represent genes) and the solution to the objective function represents the chromosomes.

SIMULATION

Simulation codes for GA may be written in any programming language to solve optimization problems. But for the purpose of this practical session, the genetic algorithm toolbox (**ga solver**) in MATLAB would be utilized.

Task 1: Minimize the unconstrained function:

$$f(x) = 4x_1^2 + 7x_2 + x_1x_2 - 3x_1$$

Steps:

- (i) **Write fitness function using MATLAB syntax**

The function should take in a vector input whose length is the same as the number of variables if the output is expected to be scalar quantity. For a vector output, the input would be a matrix.

- To write the function in task 1 in MATLAB:
- Open a new file in the editor window:
- Use the syntax:

```
function y = practice_fxn(x)
y = 4*x(1)^2 + 7*x(2) + x(1)*x(2) - 3*x(1)
```
- It could also be done in vector format such as:

```
function y = practice_fxn(x)
y = 4*x(:,1).^2 + 7*x(:,2) + x(:,1)*x(:,2) - 3*x(:,1)
```
- Save the file as a function file on the current folder

(ii) Check the fitness function

- To check if the fitness function is correctly entered,
- Type in values for the first and second variable at the command window
- For example, type:

```
practice_fxn([1 2])
```
- What was your result? _____
- Substitute the values and work it out by hand.
- Did you obtain the same result?

(iii) Perform the GA operation

Use the following command to minimize the function:

```
ObjectiveFunction = @practice_fxn;
nvars = 2; % Number of variables
x = ga(ObjectiveFunction,nvars)
```

The operation may also be combined as single statement:

```
x = ga(@practice_fxn,2)
```

(iv) Record your results.**Task 2: Maximize the unconstrained function given in task 1**

$$f(x) = 4x_1^2 + 7x_2 + x_1x_2 - 3x_1$$

Note: The **ga solver** in MATLAB solves minimization functions by default.
 Therefore, to maximize, minimize

$$-f(x)$$

This means the objective function becomes:

$$g(x) = -4x_1^2 - 7x_2 - x_1x_2 + 3x_1$$

Steps:

- (i). Follow the steps enumerated in the process of minimizing the function in task 1 to accomplish the task of maximizing the function.
- (ii). Record your results

Task 3: Solve the Optimization Problem with Constraints

$$\min f(x) = 4x_1^2 + 7x_2 + x_1x_2 - 3x_1$$

s. t.

$$x_1x_2 + x_2 + 2.4 \leq 0$$

$$7 - x_1x_2 \leq 0$$

$$0 \leq x_1 \leq 1$$

$$0 \leq x_2 \leq 4$$

Steps:

- (i). Create the function file (practice_fxn3) to serve as an fitness function,
- (ii). Write the constraints as a function file using the syntax:

```
function [c,ceq]=practice_constraint2(x)
c=[2.4+x(1)*x(2)+x(2);...
7-x(1)*x(2)];
ceq=[];
```
- (iii). **Perform the GA operation using the Syntax:**

```
ObjectiveFunction=@practice_fxn3;
nvars=2; % Number of variables
LB=[0 0]; % Lower bound
UB=[1 4]; % Upper bound
Constraint Function=@practice_constraint;
[x,fval]=ga(ObjectiveFunction,nvars,[],[],[],[],LB,UB,ConstraintFunction)
```
- (iv). **Record your results**

Task 4: Plotting Functions in the ga solver

To see the progress of the optimization with plots, use the **gaoptimset** function to create options from which two major plot functions can be selected:

- **gaplotbestf**: This plots the best and mean scores of the population at every generation
- **gaplotmaxcontr**: This plots max constraints violation at every generation.

Steps:

- (i). Create options for plots using the syntax:
`options = gaoptimset('PlotFcns',{@gaplotbestf,@gaplotmaxconstr},'Display','iter');`
- (ii). Rerun the ga solver adding 'options' to the syntax:
`[x,fval] = ga(ObjectiveFunction,nvars,[],[],[],[],...
 LB, UB, Constraint Function, options)`
- (iii). Deduce from your graphs at which generation the best value was obtained.
- (iv). What is max constraint from your plots?

Task 5: Varying other ga options

The ga solver often gives different results when options such as the crossover function, mutation and reproduction options are changed

Steps:

- (i). Run the ga function 21 times, varying the crossover function from 0 to 1 with increment of 0.05 using the syntax;
`For n:0:0.05:1
 Options = gaoptimset(options,'CrossoverFraction',n)`
- (ii). Plot final values obtained with the crossover fractions.
- (iii). At what crossover fraction was the best result obtained.?
- (iv). Open the ga Optimization tool GUI from the Toolboxes menu, select ga as the solver and run the optimization problem given in task 3 but varying some of the options such as mutation function and elite count.
- (v). Select the Best Fitness, Best Individual, Stopping and Maximum Constraint plots from the toolbox.
- (vi). Compare the results with the initial optimization results carried out using the plots.

EXPERIMENT#: UILCPE/AIA3

EXPERIMENT TITLE: Solving a Multiprocessor Scheduling Problem Using Simulated Annealing

OBJECTIVES:

- To obtain an optimal distribution of a given task on a set of processors using simulated annealing
- To set up an optimization problem that can be utilized in the simulated annealing function of MATLAB
- To implement custom data type optimization problem

BACKGROUND

Simulated annealing is an artificial intelligence algorithm modelled after the physical process of heating a material and then gently cooling it to increase the size of its crystals (larger crystals usually produce stronger metals) and to reduce defects thereby minimizing the energy consumed by the system. It is a metaheuristic algorithm which can be used to obtain globally optimized solutions in a large search space. While the same amount of cooling brings the same decrease in temperature, the rate of cooling defines the magnitude of decrease in the system energy. Therefore, slower cooling brings about greater decrease in energy. Slow cooling indicates a slow decrease in the probability of accepting solutions (even worse solutions). At high temperature, there is random movement which means the algorithm can easily jump out of local minima to search for better solutions. As cooling occurs, the randomness decreases till it reaches the global minima. This slow decrease helps the algorithm explore the solution space more, which is a potential of metaheuristic algorithms to obtain globally optimal results without being trapped in local minima.

The solution to the multiprocessor scheduling problem is to obtain an optimal distribution of tasks on a set of processors. The data given are:

- Number of processors
- Speed of each processor
- Number of tasks
- Time taken to complete one task by a processor

Scheduling is the process of assigning the jobs to the available processors. Each of these processors run independently, but can only work on one job at a time. The aim of the practical session is to obtain the shortest schedule (optimal allocation of job) for the given set of tasks.

SIMULATION

The data required for the simulation are supplied in the Simulated Annealing directory in the computer system being used. The simulated annealing tool to be used is available in MATLAB as SIMULANNEALBND function.

Steps:

- (i) Express the problem in the form of an optimization problem. Code the Time Scheme as follows:
 - Let each task be represented by an integer between 1 and total number of tasks
 - Let each processor be represented by an integer between 1 and total number of processors
 - Save the time it will take a given processor to complete a job in a matrix
(for example, the time it will take processor 'p' to complete task 't' will be stored as speed (p,t))
- (ii) Code the schedule in like manner as the time scheme:
 - Let rows (1 : number of processors) represent the processors
 - Let columns (1 : number of tasks) represent the tasks.
 - Save the schedule
(for example, [1 3 4 : 2 6 0 : 5 0 0] means
tasks 1,3,4 would be performed on processor 1,
tasks 2 and 6 would be performed on processor 2,
while task 5 would be carried out on processor 3.
- (iii) Allocate different coefficients to the rows to represent the speeds at which different processors work.
- (iv) Create a 'take-off point' as the starting point in the algorithm.

An example of the syntax for the steps given in Program Code 1:

Program Code 1: Sample Representation of the information given in the data sheet

```

numberOfProcessors = 11;
numberOfTasks = 40;
speed = [10*rand(1,numberOfTasks);
        7*rand(1,numberOfTasks);
        2*rand(1,numberOfTasks);
        5*rand(1,numberOfTasks);
        3*rand(1,numberOfTasks);
        4*rand(1,numberOfTasks);
        1*rand(1,numberOfTasks);
        6*rand(1,numberOfTasks);
        4*rand(1,numberOfTasks);
        3*rand(1,numberOfTasks);
        1*rand(1,numberOfTasks)];

% Random distribution of task on processors (starting point)
take_off_point = zeros(numberOfProcessors,numberOfTasks);
for task = 1:numberOfTasks
    processorID = 1 + floor(rand*(numberOfProcessors));
    index = find(sampleSchedule(processorID,:)==0);
    sampleSchedule(processorID,index(1)) = task;
end

```

- (v) Prepare the fitness function as indicated in the attached documentation

Sample for the syntax is given in Program Code 2:

Program Code 2: Sample representation of the fitness function syntax:

```

function timeToComplete = mulprocfitness(schedule, speeds)
% MULPROCFITNESS determines the "fitness" of the given schedule.
% In other words, it tells us how long the given schedule will take using the
% knowledge given by "speeds"
% Copyright 2006 The MathWorks, Inc.
[nrowsncols] = size(schedule);
timeToComplete = zeros(1,nrows);
for i = 1:nrows
timeToComplete(i) = 0;
    for j = 1:ncols
        if schedule(i,j)~=0
timeToComplete(i) = timeToComplete(i)+time_frame(i,schedule(i,j));
            else
                break
            end
        end
    end
end
timeToComplete = max(timeToComplete);

```

- (v) Employ the SIMULANNEALBND function to obtain the optimization

A sample syntax for the procedure is displayed in program code 3

Program Code 3: Sample Syntax to use the simulated annealing function in MATLAB

```
schedule = simulannealbnd(fitnessfcn, take_off_point, [], [], options);
```

```
% Remove zero columns (all processes are idle)
```

```
% 'simulannealbnd' is the inbuilt simulated annealing function in  
MATLAB.
```

```
% the 'fitnessfcn' represents the objective function
```

```
% the 'options' is for the plots
```

```
maxlen = 0;
```

```
fori = 1:size(schedule,1)
```

```
if max(nnz(schedule(i,:)))>maxlen
```

```
maxlen = max(nnz(schedule(i,:)));
```

```
end
```

```
end
```

```
% Display the schedule
```

```
schedule = schedule(:,1:maxlen)
```

(vi) Plot the following graphs using the simulations annealing options setup:

- Current point
- Current best point
- Fitness Value vs. iterations

(vii) From the your plots, write out the current point value and the best function value

(viii) Write out the optimized schedule for the processors

(ix) Repeat the procedure for the second set of data given in the Simulation Annealing Directory.

EXPERIMENT #: UILCPE/AIA4

EXPERIMENT TITLE: Predicting Outcomes from known input and response data using Supervised Learning

OBJECTIVES:

- To develop models that can predict the response values of a new data set
- To validate the model using test data set
- To implement classification algorithms

BACKGROUND

Supervised learning is a type of machine learning algorithm that can make predictions or inferences using labelled training data. The training data consists of a set of inputs which have labelled outputs often referred to as targets. This dataset is utilized to train the system such that a model can be built with which specific output predictions can be made for new inputs. A test dataset is usually employed to validate the model.

Usually, the larger the training datasets or examples, the more accurate the prediction power of the model generated is. The model therefore forms the basis by which new uncertain inputs can be mapped to the right outputs. The algorithm enables the system to 'learn' from observation in the training data the connection between inputs and outputs as illustrated in Figure

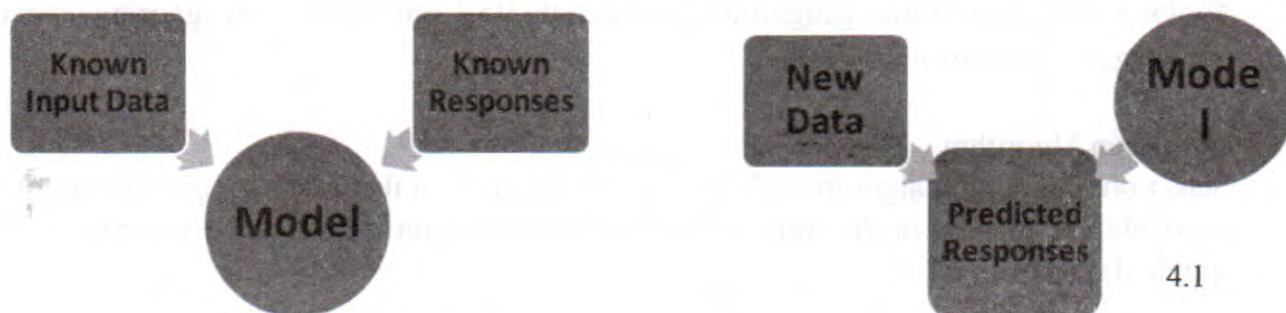


Figure 4.1: Block Diagram of Supervised Learning Algorithm

Supervised learning is of two categories:

- *Classification*: This involves mapping a class (or *label*) from a finite set of classes to an observation. The labels are categorized as nominal response values like True or False, Yes or No, Present or Absent etc. Example is the predicting whether rain would fall tomorrow or not? Output classes are True or False.
- *Regression*: This involves predicting a continuous measurement for an observation. The responses are therefore real numbers. Example is forecasting energy consumption in a location for a month.

An application of supervised learning which would be utilized in this practical session is:

- Electricity load and price forecasting
- Customer satisfaction prediction.

The latter would be treated as a classification problem while the former would be treated as a regression problem.

SIMULATION

- i. Construct the load forecasting model using the regression algorithm based on historical load data given.
- ii. Develop a similar regression algorithm model for predicting price of energy
- iii. Predict customer satisfaction in the next load and price predictions based on initial satisfaction data provided using classification algorithm

Steps:

Load Forecasting Model

- (I) Prepare the input from the historical load data given in the form

$$x_t = (t = 1, 2, \dots, N)$$

(The load forecasting historical data is provided in the LOADFORECASTING directory in the MATLAB main directory)

If the Statistics and Machine Learning Toolbox is used, enter the data as follows:

Each row of X represents one observation. Each column of X represents one variable, or predictor. Represent missing entries with NaN values in X . Statistics and Machine Learning Toolbox supervised learning algorithms can handle NaN values, either by ignoring them or by ignoring any row with a NaN value.

- (i) Choose an Algorithm
Select one regression algorithm (Sufficient information on the various regression algorithms available are present in the Mathworks documentation on the Machine Learning Toolbox supplied)
- (ii) Fit a model
The fitting function utilized to obtain the model is very closely connected with the algorithm chosen in step (2) above. The fitting functions for some common regression algorithms are provided in Table 1.

Table 1: Fitting functions for some common regression algorithms

Regression Algorithm	Fitting Function
Regression Trees	fitrtree
Support Vector Machines for Regression	fitrsvm
Regression Ensembles	fitensemble
Regression Tree Ensembles	TreeBagger

The procedure to be followed for any of the algorithms selected are all explicitly documented in the MATLAB documentation referred to in the reference section.

To use regression trees for example, the following commands may be used:

```
X = [Time Load]; % input historical load data
rtree = fitrtree(X) % create regression tree
viewrtree % view decision tree in text format
view(rtree,'mode','graph') % graphic description
```

Sketch the decision tree observed.

(i) Predict responses with the regression trees

After creating a tree, predict responses for new data. Suppose X_{new} is new data that has the same number of columns as the original data X . To predict the regression based on the tree and the new data, use the syntax:

```
Ynew = predict(tree,Xnew);
```

For each row of data in X_{new} (time), `predict` runs through the decisions in `tree` and gives the resulting prediction in the corresponding element of Y_{new} (load) .

- State the value of your X_{new} and write out the predicted Y_{new}
- Suggest 10 other new time values and write out the predicted loads
- Tabulate your results.

(ii) Choose a validation Method

- Select any of the validation techniques listed in the documentation supplied and
- Partition the data into Training (50%) and Validation (50%) sets.
- Train and validate the model

(iii) Examine, fit and update until satisfied

Prune the model by changing some parameters, especially tree depth (see documentation supplied for examples of how to prune the model).

(iv) Repeat the prediction step and note results.

(v) Compare the results using for three other algorithms using the best fit plot

(vi) Repeat the steps above for price forecasting using the supplied price data.

(vii) Predict customer satisfaction using at least one of the classifier algorithms: Decision trees, Support Vector Machines (SVM), Naïve Bayes, NearestNeighbours. Record all results and present the best fit plots.

EXPERIMENT#: UILCPE/AIA5**EXPERIMENT TITLE: Building a Self-Organizing Map (SOM) using Unsupervised Learning****OBJECTIVES:**

The objectives are to:

- prepare from the input data a corresponding mapping in two dimensions
- build a network of neurons to respond to a specific region of the input area
- refine the network for neighboring neurons to respond to adjacent regions in input area and train the SOM network to map any given set of data using unsupervised learning technique

BACKGROUND

Unsupervised learning is a type of machine learning algorithm used to draw inferences from or find a pattern in unlabeled input data given. Unlike supervised learning, labelled responses to inputs are not given and the algorithm simply builds a representation of the input and uses it for decision making. Since the data given to the learner is unlabeled, there is no error or reward sign to evaluate potential solution as obtainable in supervised and reinforcement learning respectively.

A Self Organizing Map (SOM) which is the focus of this practical session is a topographic organization in which neighbouring locations in the map represent various inputs with properties relating to the input. The SOM can be built as an application of unsupervised learning algorithm where inputs would lead to a 2-dimensional location on the map. It can also be achieved through clustering technique.

SIMULATION

A set of unknown input data is already supplied in the SOM Simulations Folder. Make out of these data a matching description (a corresponding mapping) in two dimensions. Use *unsupervised* learning with a SOM-network in 2 dimensions.

Exercise

(i) Before training the network:

- Input the data in form of elements X and Y representing the coordinates
- Data file name: input_s1.txt. How is data packed? Row number 1: X-data, Row number 2: Y-data. The data input values are all within following limits: [-2, +2]
- Save the data in your own working directory
- Build and train your SOM network to map this set of data. Use 8 x 8 neurons to map the 2-dimensions.
- Plot a figure showing how input- and output signals are connected in your SOM network (as shown in Fig. 5.1). An example to better understand building/training a SOM-network is given in the SOM Simulation folder

- (ii) After training: What are the neuron weight values ($W_{i,j}$) in your final network? Make a plot of these weights; differentiate every weight pair (w_x, w_y) with coloured markers X-axis: weight values for first input element. Y-axis: weight values for second input element. Example: If weights were named w_x and w_y then the blue diamond symbol will be plotted by this Matlab syntax: `plot( $w_x$ ,  $w_y$ , 'bd')`
- (iii) Compare your weight values to the Matlab plot named "plotsom()" - supported by the neural networks toolbox – by using this syntax:
`plotsom(net.iw{1,1},net.layers{1}.distances)`
- What are your comments to the comparison?
 - What does the blue line in the 'plotsom'- indicate?
- (A example of the GUI to help with the 'plotsom' function is shown in Fig. 2)
- (iv) Simulate with your trained network the response of the input vector [0.5; 0.7].
- Which neuron performs better in this mapping?
 - What weight values (of the 2 input elements) is the best performing neuron?
- (v) Has the SOM-network learned the topology of the input space?
- (vi) Compare all the input vectors with the trained target neurons.
- (vii) Plot the following
- Plot I: 'input vectors': X-axis: X-data of all input vectors. Y-axis: Y-data of each input vector. Plot a green dot symbol for every vector.
- Plot II: 'target neurons' – The plot that you made earlier (above, under item b with blue diamonds) – But here- Put the two plots in the same graph.
- (i) Use the graph as a legend in your answer to the given question: Has the SOM-network learned the topology of the input space?
 - (ii) Are your network weights distributed in a uniform or a non-uniform way?
 - (iii) What does this way of distribution tell you about the input space?

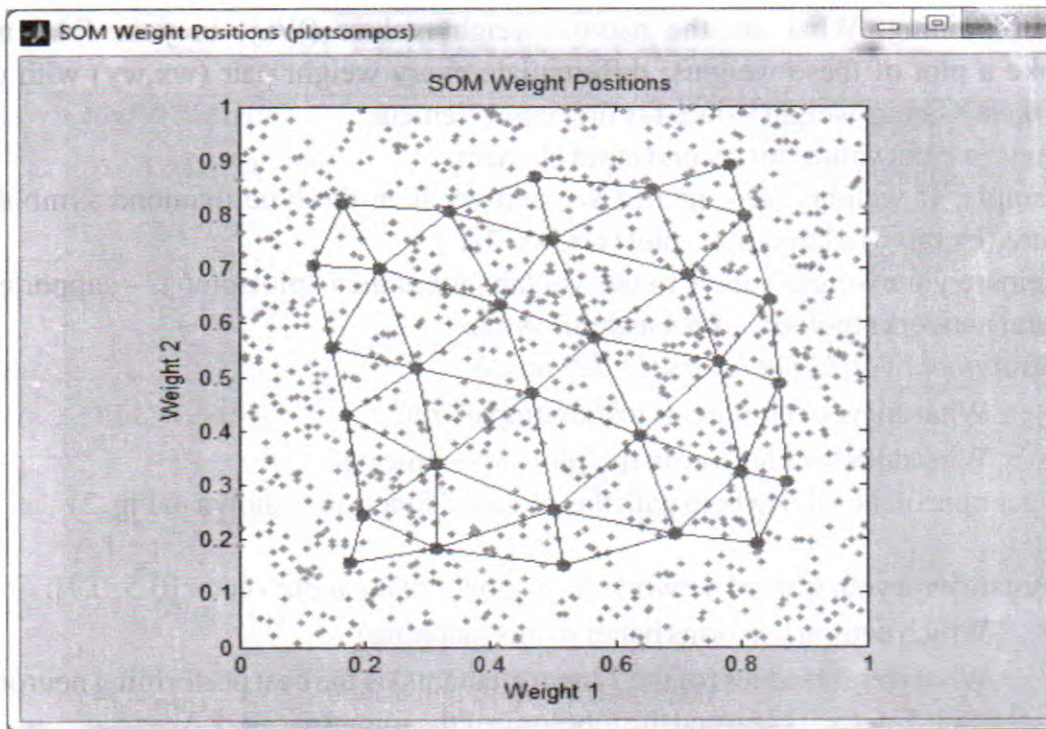


Figure 5.1: SOM Weight Positions

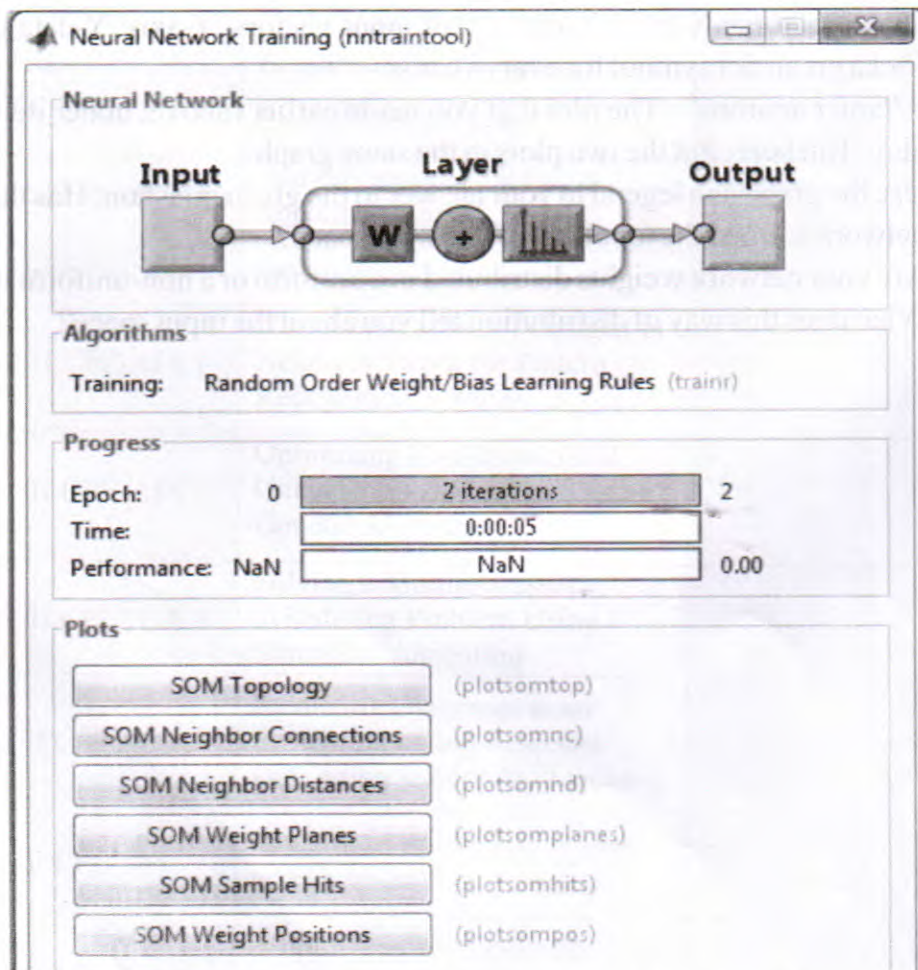


Figure 5.2: Unsupervised Learning Training tool for the SOM Network

EXPERIMENT #: UILCPE/AIA6

EXPERIMENT TITLE: A Reinforcement Learning Approach to Solving a Mountain Car Problem

OBJECTIVES:

- To implement an agent-based system using reinforcement learning algorithms such as Q-learning and SARSA
- To present the results and improvement in learning through performance plots
- To solve optimal control and navigation problems that cannot be solved using conventional dynamic programming

BACKGROUND

Reinforcement learning is a form of machine learning technique under the broad spectrum of artificial intelligence applications. It is inspired by the natural way behavior is learned, where actions taken are refined by the reward or punishment that follows such actions. The learning agent therefore takes actions so as to maximize a predefined cumulative *reward*.

Reinforcement learning differs from standard supervised learning in that correctly mapped input/output pairs are not provided. The agent is expected to learn on its own while exploring the solution space from the positive or negative rewards it gets from the actions taken.

The mountain car problem involves a two-dimensional world consisting of a valley and a mass that must be pushed back and forth to gain enough momentum to escape the valley.

SIMULATION

A GUI is utilized in this simulation to provide a more intuitive grasp of the effects of various parameter values. Controlling the running of the simulation via a graphical user interface (GUI) and in the process modifying the parameters gives a clearer understanding of the algorithm. The GUI editor *guide* is provided to quickly put together a very functional user interface.

In the **upper left** is a graph of the two-dimensional world in where the mountain car is. The car is represented by a box whose color indicates which direction, left or right, the reinforcement learning agent is currently pushing.

In the **middle region** of the figure are current parameter values in editable text fields. Changes made by the user to any of these values are immediately effective. The **lower left** graph shows the current estimate of the value function. The value function is being learned by a neural network consisting of radial basis functions.

The reinforcement learning agent is learning a prediction of the number of steps required to leave the valley from every state, where a state consists of a position and velocity of the car. The user can change the view of this three-dimensional surface by clicking and moving the mouse on this graph.

The **lower right** graph shows the actions the learning agent would take for each state of the car using the current estimate of the value function. Figure 1 shows the late stage of learning after a

good value function has been learned. The green area covers states for which the car is pushed right, and in the red area the car is pushed left.

Exercise:

- (I). On the menu bar, click on the menu **Run and pull down menu appears containing the choices Start, Reset, and Pause.**
- (ii). Click on **Start** to begin the simulation. Once started, the **Pause** menu item becomes enabled, allowing the user to pause the simulation at any time. The **Reset** menu item will re-initialize the reinforcement learning agent so it can again learn from scratch.
- (iii). Activate the display by clicking on the update button below the graph. It can also be deactivated using this same button. Deactivating it allows the simulation to run faster.
- (iv). Take 12 trials and plot the graph showing the performance of the reinforcement learning agent while it is learning. Performance is measured by the number of steps from initial random positions and velocities of the car to the step at which the car leaves the valley. Performance is plotted versus the number of trials.
- (v). Increase the temperature and learning rate parameters by at defined intervals for six episodes (taking 12 trials) and present the
 - performance plot
 - Value function plot
 - Action using Current value function
- (vi). Note the interval used for the temperature and learning rate increment.
- (vii). Compare the plots and make inferences.
- (viii). Show a plot of the trajectory of the car's state for the current trial
- (ix). A folder has been added to your MATLAB path containing a SARSA implementation of the process. Open the folder and follow the steps indicated.
- (x). Compare the results obtained through the perform plots (example is shown in fig 2.)

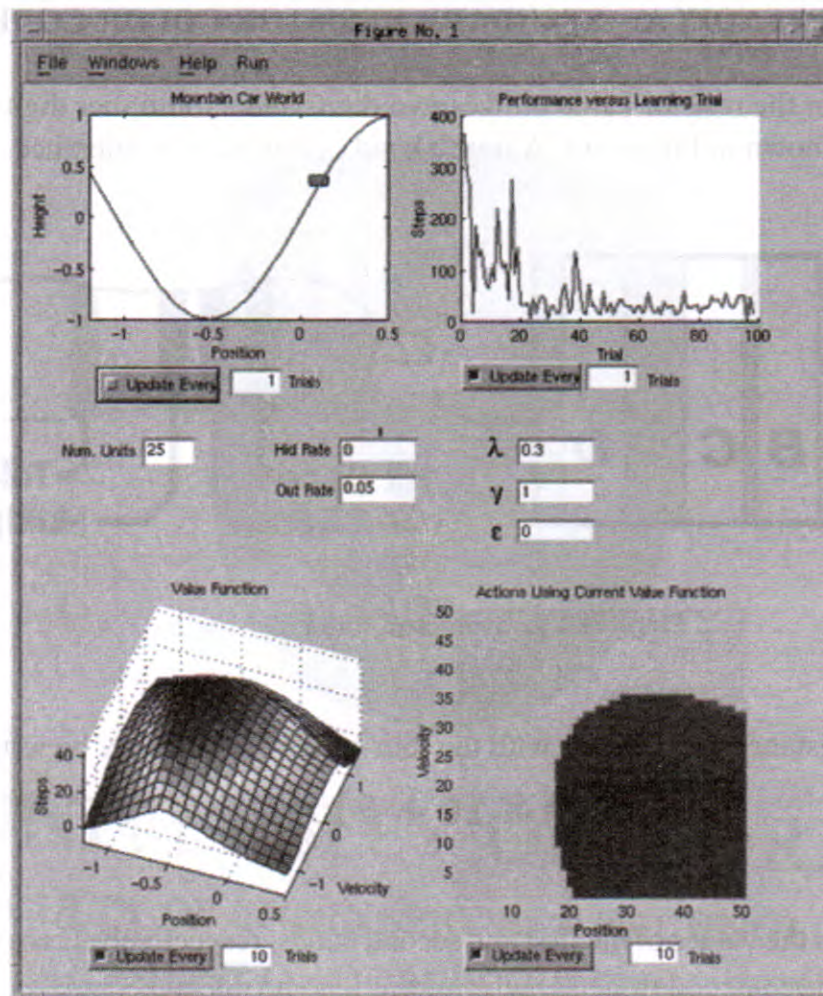


Figure 6.1: Learning GUI Implementation to solve Mountain Car Problem

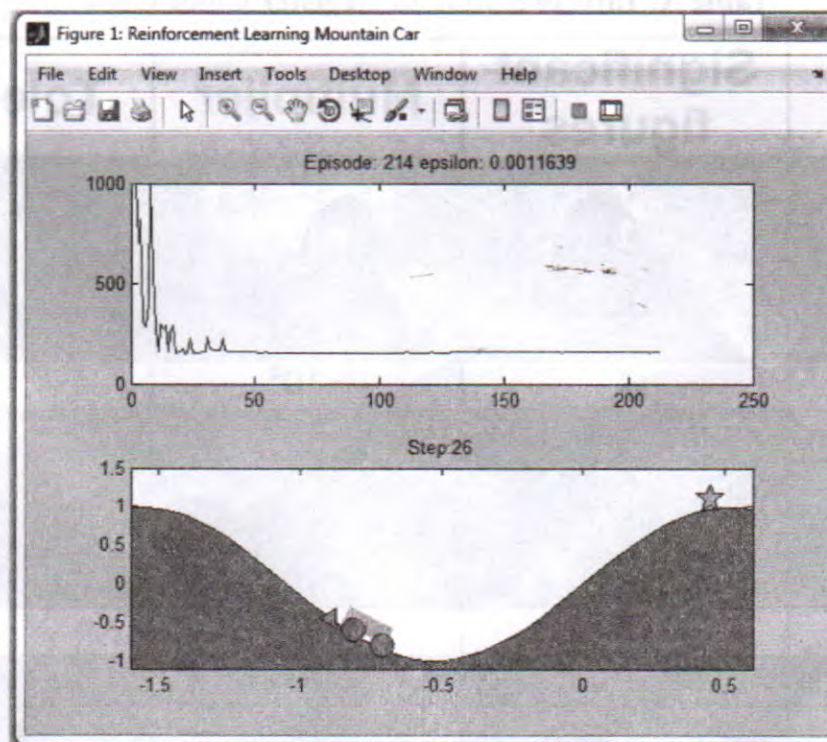


Figure 6.2: A SARSA Implementation to solve Mountain Car Problem

APPENDIX A: STANDARD RESISTOR COLOR CODE

The color code for the resistor value utilizes two digits and a multiplier digit in that order, as shown in Figure A1. A fourth band designates the tolerance.

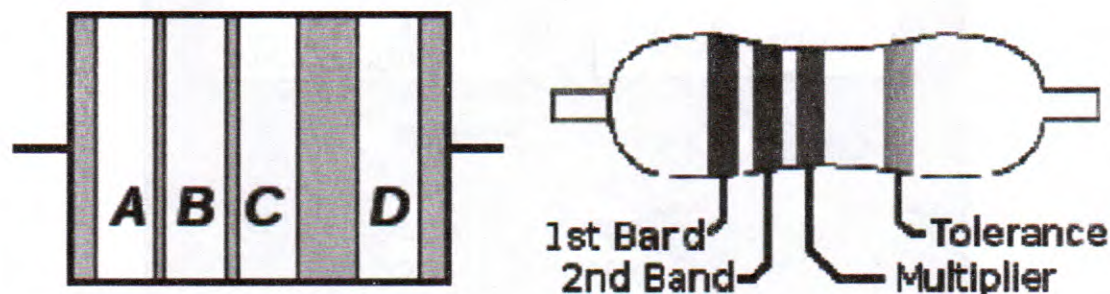


Figure A1 Resistor with four color bands.

The resistance of a resistor with the four bands of color may be written as

$$R = (A \times 10 + B) \times C \pm D$$

Where A and B are the values of the first and second bands, respectively, C is a multiplier and D is the tolerance. The color code together with multipliers and tolerances are listed in Table A1.

Table A: Interpretation of Register Color Code

Color	Significant figures	Multiplier	Tolerance
Black	0	$\times 10^0$	--
Brown	1	$\times 10^1$	--
Red	2	$\times 10^2$	$\pm 2\%$
Orange	3	$\times 10^3$	--
Yellow	4	$\times 10^4$	--
Green	5	$\times 10^5$	--
Blue	6	$\times 10^6$	--
Violet	7	$\times 10^7$	--
Gray	8	$\times 10^8$	--
White	9	--	--
Gold	--	$\times 10^{-1}$	$\pm 5\%$
Silver	--	$\times 10^{-2}$	$\pm 10\%$
None	--	--	$\pm 20\%$

