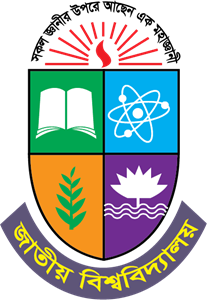
**PROJECT REPORT**

**Design and Implementation of a Mobile Application Controlled Scrolling Display**

A project report submitted in partial fulfillment of the requirement for the Degree of Bachelor of Science in Electronics and Communication Engineering.



**Prepared By:**

Roll No: 1500038

Registration No: 14508000045

Session: 2014-2015

Department of Electronics and communication Engineering.

**DECLARATION**

This is to declare that the project entitled “**Design and Implementation of a Mobile Application Controlled Scrolling Display**” submitted by the student (**Exam Roll**: 1500038**, Reg No:**14508000045) to the Department of Electronics and communication Engineering, National University for the degree of B. Sc. in Electronics & Communication Engineering is an original project work carried out by the student under my supervision. The content of this paper has not been submitted to any other Institute or University for the award of any degree.

…………………………………………………..

Abdur Rahman

Registration No: 14508000045

Roll No : 1500038

Session : 2014-15

Supervisor

……………………………………………………

Dr. Galib Hashmi

Assistant Professor

Department of ECE

**ACKNOWLEDGEMENT**

We would like to thank our Almighty. Then we would like to express our gratitude to the outstanding group of people who have helped us in completing our project successfully. We would like to convey our sincere thanks to them for their unabated support rendered to us at all time. . Firstly, we would like to thank **Dr. Galib Hashmi,** Assistant Professor of Department of Electronics and Communication Engineering for his guidance, ideas, suggestion and support to come up with this project. We would like to thank all our friends for the help and constructive criticism during our project period. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve higher goals.

**ABSTRACT**

This paper presents the overall design of “Smart Scrolling LED Display using Arduino, Bluetooth and mobile application” with low cost and user can access multiple applications. If anyone wants to display the message, they can send message through using android Bluetooth by using this project. This project deals with advanced wireless Arduino development board. The main objective of this project is to design a wireless board that displays messages sent from android phone user using Bluetooth. The main controlling device of the whole system is Arduino. Matrix LED Display module, Bluetooth module are interfaced to Arduino and android development tool with APK application.

Table of Contents

[Chapter 1: Introduction 1](#_Toc34262311)

[1.1 Introduction 1](#_Toc34262312)

[1.2 Motivation 2](#_Toc34262313)

[1.3: Objectives of the project 2](#_Toc34262314)

[1.4: Chapter Outline 2](#_Toc34262315)

[Chapter 2: Background Study 3](#_Toc34262316)

[2.1: Microprocessor 3](#_Toc34262317)

[2.2: Shift Register 7](#_Toc34262318)

[2.3: Transistor 13](#_Toc34262319)

[2.4: Dot Matrix 20](#_Toc34262320)

[2.5: Resistor 26](#_Toc34262321)

[2.6: Power Supply 35](#_Toc34262322)

[Chapter 3: Component Description 37](#_Toc34262323)

[3.1: Arduino Nano 37](#_Toc34262324)

[3.2: Bluetooth Module 42](#_Toc34262325)

[Chapter 4 System Methodology 47](#_Toc34262326)

[4.1 Bluetooth overview 47](#_Toc34262327)

[4.1.1 The basics 47](#_Toc34262328)

[4.1.2 Bluetooth permissions 47](#_Toc34262329)

[4.1.3 Set up Bluetooth 48](#_Toc34262330)

[4.1.4 Find devices 50](#_Toc34262331)

[4.1.5 Query paired devices 50](#_Toc34262332)

[4.1.6 Discover devices 51](#_Toc34262333)

[4.1.7 Enable discoverability 53](#_Toc34262334)

[4.1.8 Connect devices 54](#_Toc34262335)

[4.1.9 Connection techniques 54](#_Toc34262336)

[4.2 Hardware Part 55](#_Toc34262337)

[4.2.1 Block Diagram 55](#_Toc34262338)

[4.2.2 Circuit Diagram 56](#_Toc34262339)

[4.2.3 PCB Design 57](#_Toc34262340)

[4.3 Software Part 58](#_Toc34262341)

[4.3.1 Flowchart of the system 58](#_Toc34262342)

[4.3.2 Arduino code 58](#_Toc34262343)

[4.4 Structure and Measurements 61](#_Toc34262344)

[4.5 Implementation (Hardware) 62](#_Toc34262345)

[4.6 Implementation (Software) 63](#_Toc34262346)

[4.7 Code generation for character display 64](#_Toc34262347)

[Chapter 5 Discussion and Conclusion 65](#_Toc34262348)

[5.1 Cost Analysis 65](#_Toc34262349)

[5.2 Advantages 66](#_Toc34262350)

[5.3 Disadvantages 66](#_Toc34262351)

[5.4 Applications 66](#_Toc34262352)

[REFERENCES 67](#_Toc34262353)

**List of figures**

[Figure 1: Block Diagram of a Computer 12](#_Toc34261126)

[Figure 2Block of Serial In − Serial Out SISOSISO Shift Register 17](#_Toc34261127)

[Figure 3 Block Diagram of Serial In - Parallel Out SIPOSIPO Shift Register 18](#_Toc34261128)

[Figure 4Block of Parallel In − Serial Out PISOPISO Shift Register 20](#_Toc34261129)

[Figure 5 Block diagram of Parallel In - Parallel Out PIPOPIPO Shift Register 21](#_Toc34261130)

[Figure 6 Bipolar Junction Transistor (BJT) 23](#_Toc34261131)

[Figure 7 Field Effect Transistors(FET) 23](#_Toc34261132)

[Figure 8 Circuit Diagram of PNP Transistor 24](#_Toc34261133)

[Figure 9Circuit diagram of NPN Transistor 25](#_Toc34261134)

[Figure 10 Bipolar Junction Transistor (BJT) Biasing common emitter 26](#_Toc34261135)

[Figure 11 Bipolar Junction Transistor (BJT) Biasing common collector 28](#_Toc34261136)

[Figure 12 Structure of a 8x8 LED dot matrix 30](#_Toc34261137)

[Figure 13 Output Structure of a 8x8 LED dot matrix 31](#_Toc34261138)

[Figure 14 Scanning across the columns and feeding with appropriate row values 32](#_Toc34261139)

[Figure 15 Pin diagram and schematic of ULN2003A (Darlington transistor arrays) 33](#_Toc34261140)

[Figure 16 Circuit diagram for interfacing a 5x7 LED dot matrix with PIC18F2550 34](#_Toc34261141)

[Figure 17 Circuit setup showing a Start USB board with a 6x7 LED dot matrix 34](#_Toc34261142)

[Figure 18 Two common resistor schematic symbols. R1 is an American-style 1kΩ resistor, and R2 is an international-style 47kΩ resistor. 35](#_Toc34261143)

[Figure 19 In this circuit, resistors play a key role in setting the frequency of the 555 timer's output. Another resistor (R3) limits the current through an LED. 36](#_Toc34261144)

[Figure 20 A half-watt (½W) resistor (above) sized up to a quarter-watt (¼W). 37](#_Toc34261145)

[Figure 21 A tiny 0603 330Ω resistor hovering over shiny George Washington's nose on top of a [U.S. quarter] (http://en.wikipedia.org/wiki/Quarter\_(United\_States\_coin). 37](#_Toc34261146)

[Figure 22 Peek inside the guts of a few carbon-film resistors. Resistance values from top to bottom: 27Ω, 330Ω and a 3.3MΩ. Inside the resistor, a carbon film is wrapped around an insulator. More wraps mean a higher resistance. Pretty neat! 38](#_Toc34261147)

[Figure 23 An array of five 330Ω resistors, all tied together at one end. 38](#_Toc34261148)

[Figure 24 A smattering of potentiometers. From top-left, clockwise a standard 10k trimpot,2-axis joystick, soft pot, slide pot, classic right-angle ,and a breadboard friendly 10k trim pot 39](#_Toc34261149)

[Figure 25 Decoding color code 39](#_Toc34261150)

[Figure 26 4.7kΩ resistor with four color bands 40](#_Toc34261151)

[Figure 27 Color code table 41](#_Toc34261152)

[Figure 28 These power resistors can handle a lot more power before they blow. From top-right to bottom-left there are examples of 25W, 5W and 3W resistors, with values of 2Ω, 3Ω 0.1Ω and 22kΩ. Smaller power-resistors are often used to sense current. 43](#_Toc34261153)

[Figure 29 Pull-up resistor 44](#_Toc34261154)

[Figure 30 Arduino Nano 46](#_Toc34261155)

[Figure 31Pin Configuration of Arduino Nano 47](#_Toc34261156)

[Figure 32HC-06 Bluetooth Module 52](#_Toc34261157)

[Figure 33Bluetooth Module 54](#_Toc34261158)

[Figure 34 2D Model of Bluetooth Module 55](#_Toc34261159)

[Figure 35 The enabling Bluetooth dialog. 58](#_Toc34261160)

[Figure 36 The enabling discoverability dialog. 62](#_Toc34261161)

[Figure 37 The Bluetooth pairing dialog. 63](#_Toc34261162)

[Figure 38 Block Diagram of Android controlled message display 64](#_Toc34261163)

[Figure 39 Schematic Circuit Diagram of Android controlled message display 65](#_Toc34261164)

[Figure 40 PCB design of Android controlled message display 66](#_Toc34261165)

[Figure 41 Flowchart of Android controlled message display 67](#_Toc34261166)

[Figure 42 Structure of message display board 70](#_Toc34261167)

[Figure 43 Hardware implementation of message display board 71](#_Toc34261168)

[Figure 44 Software implementation of message display board 72](#_Toc34261169)

[Figure 45 Code generation for character display 73](#_Toc34261170)

**List of Table**

**Table Page No**

**Table2.1** Truth table of Serial In − Serial Out SISOSISO Shift Register…………………….15

**Table2.2** Truth table of SerialIn - Parallel Out SIPOSIPO Shift Register…………………...17

**Table2.3** Truth table of Parallel In − Serial Out PISOPISO Shift Register…………………..18

**Table 2.4**: The table below gives the logic levels to be applied to R1 through R7 for each of the columns in order to display the alphabet ‘A’………………………………………………....29

**Table-3.1: Pin Configuration table of Bluetooth Nano…………………………………...….51**

**Table 5.1** cost analysis of a message display…………………………………………………74

# **Chapter 1: Introduction**

## **1.1 Introduction**

The project aims at designing a LED based scrolling message display controlled from an android mobile phone. The proposed system makes use of Bluetooth technology to communicate from android phone to LED display board. This project is to develop an embedded system, which is used for instant information display using LED’s by using android Bluetooth module. Now a day’s every advertisement or information is displayed digitally. The big shops and shopping centers are using the digital moving/scrolling displays now. In railway station, bus stands everything that is ticket information, platform number etc. is displaying in digital moving display. But in these displays if they want to change the message, they can send message through using android Bluetooth by using this project. If they want to display messages about something crucial within minimum time, it displays whatever wants. LED displays are used in variety of applications, like store signs, billboards and many more. In recent years it is commonly used in destination signs on public transport vehicles. LED panels are also used for the purpose of general illumination, task lighting and for stage lighting. Display systems are classified into single line displays, and multiline displays. A standard LED display board consists of led lights arranged in 3 sets of 16 rows x 32 columns, with each led placed at a pixel of 5mm. Displays boards of any length and breadth can be made by combining more than one of these standard units. These display units are capable of displaying messages of any kind, including alphanumeric, numbers etc., in static or scrolling formats. This system is comprised of a red color matrix display panel. It also includes an executive program that runs on the Arduino for the display control of data information on the display board. Led provides several advantages over traditional light bulbs, such as small size and longer life. A red color led can be used to advertise even day-light conditions. The led display board displays images and messages entered by using a microcontroller

## **1.2 Motivation**

As an engineering student a project is required as part of the final year work. The project chosen is the Arduino based mobile Application controlled message display using dot matrix LED display. This was selected because of a monochrome (single color) LED dot matrix display is used for displaying the Characters and Symbols which is interface with a microcontroller. This project will deliberate on displaying a scrolling text message on a 16x128 LED dot matrix display. The microcontroller used is Arduino Nano which is open source prototype Electronic platform. The 128 columns of the LED matrix are driven individually by six shifts.

## **1.3: Objectives of the project**

* To build scrolling message display.
* It will control Message text with mobile phone via Bluetooth.
* It will be able to control brightness and text scroll speed.
* It will be Long lasting device.
* It could survive at any critical environment.
* Cost Minimize.
* It is suitable for any weather
* It will be able to operate low power.

## **1.4: Chapter Outline**

The project is divided into 6 chapters:

**Chapter 1:** In this chapter, an introduction of the works & motivation is described. Some recent works related to the project are attached.

**Chapter 2:** Here, the scrolling display theories are described which are needed behind the works.

**Chapter 3:** In this chapter, the overviews on the proposed model are described briefly which are needed for implementation.

**Chapter 4:** This chapter presents the design and implementation and project result of a Mobile Application controlled Message Display.

**Chapter 5:** The conclusions are discussed here.

# **Chapter 2: Background Study**

## **2.1: Microprocessor**

A microprocessor is an integrated circuit (IC) which incorporates core functions of a computer’s central processing unit (CPU). It is a programmable multipurpose silicon chip, clock driven, register based, accepts binary data as input and provides output after processing it as per the instructions stored in the memory.

**Block Diagram of a Computer**

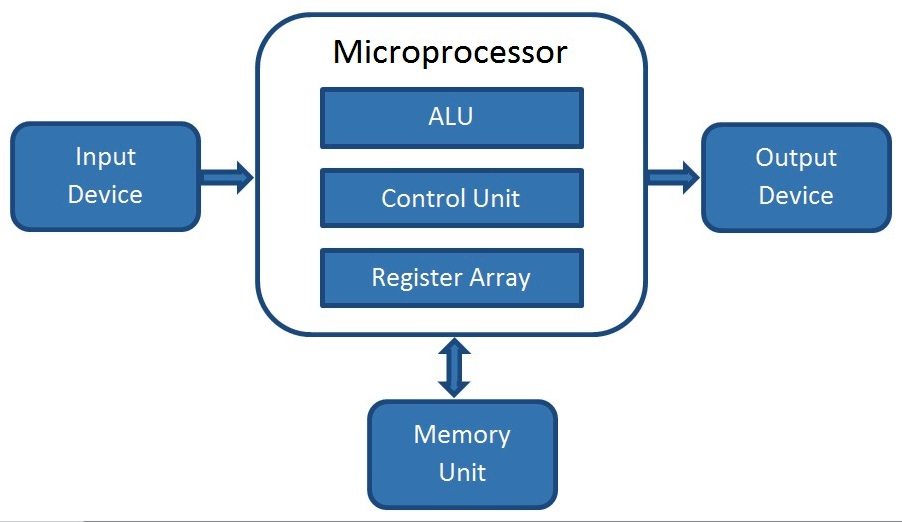


Figure 1: Block Diagram of a Computer

**How does a Microprocessor work?**

A processor is the brain of a computer which basically consists of Arithmetical and Logical Unit (ALU), Control Unit and Register Array. As the name indicates ALU performs all arithmetic and logical operations on the data received from input devices or memory. Register array consists of a series of registers like accumulator (A), B, C, D etc. which acts as temporary fast access memory locations for processing data. As the name indicates, control unit controls the flow of instructions and data throughout the system.

So basically, a microprocessor takes input from input devices, process it as per instructions given in the memory and produces output.

**Advantages of a Microprocessor**

* **Low Cost**  
  Microprocessors are available at low cost due to integrated circuit technology. Which will reduce the cost of a computer system.
* **High Speed**  
  Microprocessor chips can work at very high speed due to the technology involved in it. It is capable of executing millions of instructions per second.
* **Small Size**  
  Due to very large scale and ultra-large-scale integration technology, a microprocessor is fabricated in a very less footprint. This will reduce the size of the entire computer system.
* **Versatile**  
  Microprocessors are very versatile; the same chip can be used for a number of applications by simply changing the program (instructions stored in the memory).
* **Low Power Consumption**  
  Microprocessors are usually manufactured using metal oxide semiconductor technology, in which MOSFETs (Metal Oxide Semiconductor Field Effect Transistors) are working in saturation and cut off modes. So, the power consumption is very low compared to others.
* **Less Heat Generation**  
  Compared to vacuum tube devices, semiconductor devices won’t emit that much heat.
* **Reliable**  
  Microprocessors are very reliable; failure rate is very less as semiconductor technology is used.
* **Portable**  
  Devices or computer system made with microprocessors can be made portable due to the small size and low power consumption.

**Common Terms used in a Microprocessor**

Here are some common terms that we will use in microprocessor field.

**Bus**

A bus is a set of conductors intended to transmit data, address or control information to different elements in a microprocessor. Usually a microprocessor will have 3 types of buses: Data Bus, Control Bus and Address Bus. An 8-bit processor will be using 8-bit wide bus.

**Instruction Set**

Instruction set is the group of commands that a microprocessor can understand. So, instruction set is an interface between hardware and software (program). An instruction commands the processor to switch relevant transistors for doing some processing in data. For e.g. ADD A, B; is used to add two numbers stored in the register A and B.

**Word Length**

Word Length is the number of bits in the internal data bus of a processor or it is the number of bits a processor can process at a time. For e.g. An 8-bit processor will have an 8-bit data bus, 8-bit registers and will do 8-bit processing at a time. For doing higher bits (32-bit, 16-bit) operations, it will split that into a series of 8-bit operations.

**Cache Memory**

Cache memory is a Random-Access memory that is integrated into the processor. So, the processor can access data in the cache memory more quickly than from a regular RAM. It is also known as CPU Memory. Cache memory is used to store data or instructions that are frequently referenced by the software or program during the operation. So, it will increase the overall speed of the operation.

**Clock Speed**

Microprocessors uses a clock signal to control the rate at which instructions are executed, synchronize other internal components and to control the data transfer between them. So clock speed refers to the speed at which a microprocessor executes instructions. It is usually measured in Hertz and are expressed in megahertz (MHz), gigahertz (GHz) etc.

**Classification of Microprocessors**

**Based on Word Length**

Hope you read about word length above. So, based on the word length of a processor we can have 8-bit, 16-bit, 32-bit and 64-bit processors.

**RISC – Reduced Instruction Set Computer**

RISC is a type of microprocessor architecture which uses small, general purpose and highly optimized instruction set rather than more specialized set of instructions found in others. RISC offers high performance over its opposing architecture CISC (see below). In a processor, execution of each instruction requires a special circuit to load and process the data. So, by reducing instructions, the processor will be using simple circuits and faster in operation.

* Simple instruction set
* Larger program
* Consists of large number of registers
* Simple processor circuitry (small number of transistors)
* More RAM usage
* Fixed length instructions
* Simple addressing modes
* Usually fixed number of clock cycles for executing one instruction

**CISC – Complex Instruction Set Computer**

CISC is the opposing microprocessor architecture for RISC. It is made to reduce the number of instructions per program, ignoring the number of cycles per instruction. So complex instructions are directly made into hardware making the processor complex and slower in operation.

This architecture is actually designed to reduce the cost of memory by reducing the program length.

* Complex instruction set
* Smaller program
* Less number of registers
* Complex processor circuitry (a greater number of transistors)
* Little RAM usage
* Variable length instructions
* Variety of addressing modes
* Variable number of clock cycles for each instruction

**Special Purpose Processors**

There are some processors which are designed to handle some specific functions.

* DSP – Digital Signal Processors
* Coprocessors – processors used along with a main processor (8087 math-coprocessor used with 8086)
* Input/output processors
* Transputer – Transistor Computer: Microprocessor with its own local memory

Examples

* Intel 4004 – The First Microprocessor
* Intel 8085
* Intel 8086
* Intel Pentium 4
* Intel Core i7
* AMD Athlon

## **2.2: Shift Register**

We know that one flip-flop can store one-bit of information. In order to store multiple bits of information, we require multiple flip-flops. The group of flip-flops, which are used to hold store the binary data is known as **register**.

If the register is capable of shifting bits either towards right hand side or towards left hand side is known as **shift register**. An ‘N’ bit shift register contains ‘N’ flip-flops. Following are the four types of shift registers based on applying inputs and accessing of outputs.

* Serial In − Serial Out shift register
* Serial In − Parallel Out shift register
* Parallel In − Serial Out shift register
* Parallel In − Parallel Out shift register

**Serial In − Serial Out SISOSISO Shift Register**

The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out SISOSISO shift register. The **block diagram** of 3-bit SISO shift register is shown in the following figure.

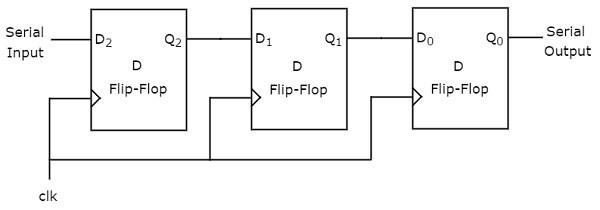


Figure 2Block of Serial In − Serial Out SISOSISO Shift Register

This block diagram consists of three D flip-flops, which are **cascaded**. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we can receive the bits serially from the output of right most D flip-flop. Hence, this output is also called as **serial output**.

**Example**

Let us see the working of 3-bit SISO shift register by sending the binary information **“011”** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000Q2Q1Q0=000. We can understand the **working of 3-bit SISO shift register** from the following table.

**Table2.1** Truth table of Serial In − Serial Out SISOSISO Shift Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No of positive edge of Clock** | **Serial Input** | **Q2** | **Q1** | **Q0** |
| 0 | - | 0 | 0 | 0 |
| 1 | 1LSBLSB | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 0MSBMSB | 0 | 1 | 1LSBLSB |
| 4 | - | - | 0 | 1 |
| 5 | - | - | - | 0MSB |

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000Q2Q1Q0=000. Here, the serial output is coming from Q0Q0. So, the LSB 11 is received at 3rd positive edge of clock and the MSB 00 is received at 5th positive edge of clock.

Therefore, the 3-bit SISO shift register requires five clock pulses in order to produce the valid output. Similarly, the **N-bit SISO shift register** requires **2N-1** clock pulses in order to shift ‘N’ bit information.

**Serial In - Parallel Out SIPOSIPO Shift Register**

The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out SIPOSIPO shift register. The **block diagram** of 3-bit SIPO shift register is shown in the following figure.

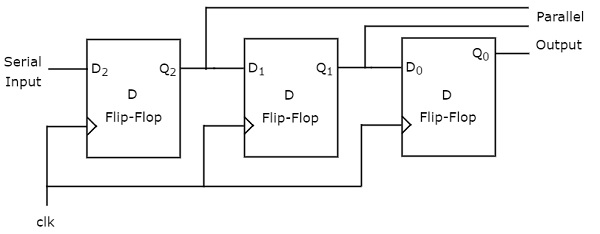


Figure 3 Block Diagram of Serial In - Parallel Out SIPOSIPO Shift Register

This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can send the bits serially from the input of left most D flip-flop. Hence, this input is also called as **serial input**. For every positive edge triggering of clock signal, the data shifts from one stage to the next. In this case, we can access the outputs of each D flip-flop in parallel. So, we will get **parallel outputs** from this shift register.

**Example**

Let us see the working of 3-bit SIPO shift register by sending the binary information **“011”** from LSB to MSB serially at the input.

Assume, initial status of the D flip-flops from leftmost to rightmost is Q2Q1Q0=000Q2Q1Q0=000. Here, Q2Q2 & Q0Q0 are MSB & LSB respectively. We can understand the **working of 3-bit SIPO shift register** from the following table.

**Table2.2** Truth table of SerialIn - Parallel Out SIPOSIPO Shift Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No of positive edge of Clock** | **Serial Input** | **Q2**MSBMSB | **Q1** | **Q0**LSBLSB |
| 0 | - | 0 | 0 | 0 |
| 1 | 1LSBLSB | 1 | 0 | 0 |
| 2 | 1 | 1 | 1 | 0 |
| 3 | 0MSBMSB | 0 | 1 | 1 |

The initial status of the D flip-flops in the absence of clock signal is Q2Q1Q0=000Q2Q1Q0=000. The binary information **“011”** is obtained in parallel at the outputs of D flip-flops for third positive edge of clock.

So, the 3-bit SIPO shift register requires three clock pulses in order to produce the valid output. Similarly, the **N-bit SIPO shift register** requires **N** clock pulses in order to shift ‘N’ bit information.

**Parallel In − Serial Out PISOPISO Shift Register**

The shift register, which allows parallel input and produces serial output is known as Parallel In − Serial Out PISOPISO shift register. The **block diagram** of 3-bit PISO shift register is shown in the following figure.

This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. For every positive edge triggering of clock signal, the data shifts from one stage to the next. So, we will get the **serial output** from the right most D flip-flop.

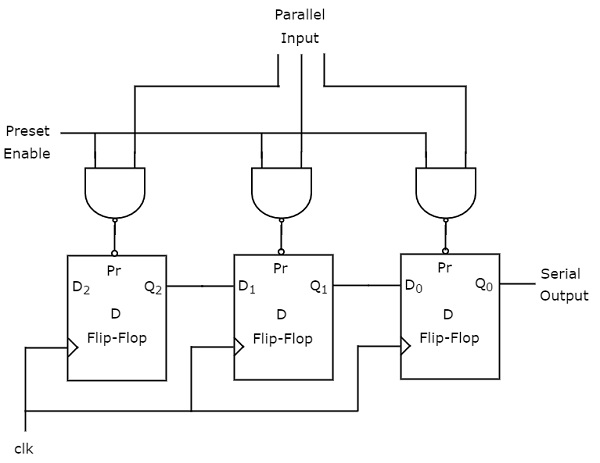
****

Figure 4Block of Parallel In − Serial Out PISOPISO Shift Register

**Example**

Let us see the working of 3-bit PISO shift register by applying the binary information **“011”** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011Q2Q1Q0=011. We can understand the **working of 3-bit PISO shift register** from the following table

**Table2.3** Truth table of Parallel In − Serial Out PISOPISO Shift Register

|  |  |  |  |
| --- | --- | --- | --- |
| **No of positive edge of Clock** | **Q2** | **Q1** | **Q0** |
| 0 | 0 | 1 | 1LSBLSB |
| 1 | - | 0 | 1 |
| 2 | - | - | 0LSBLSB |

Here, the serial output is coming from Q0Q0. So, the LSB 11 is received before applying positive edge of clock and the MSB 00 is received at 2nd positive edge of clock.

Therefore, the 3-bit PISO shift register requires two clock pulses in order to produce the valid output. Similarly, the **N-bit PISO shift register** requires **N-1** clock pulses in order to shift ‘N’ bit information.

**Parallel In - Parallel Out PIPOPIPO Shift Register**

The shift register, which allows parallel input and produces parallel output is known as Parallel In − Parallel Out PIPOPIPO shift register. The **block diagram** of 3-bit PIPO shift register is shown in the following figure.

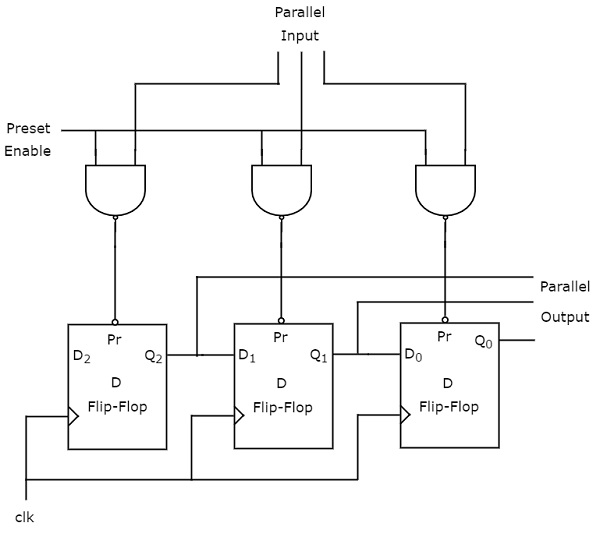


Figure 5 Block diagram of Parallel **In - Parallel Out PIPOPIPO Shift Register**

This circuit consists of three D flip-flops, which are cascaded. That means, output of one D flip-flop is connected as the input of next D flip-flop. All these flip-flops are synchronous with each other since, the same clock signal is applied to each one.

In this shift register, we can apply the **parallel inputs** to each D flip-flop by making Preset Enable to 1. We can apply the parallel inputs through preset or clear. These two are asynchronous inputs. That means, the flip-flops produce the corresponding outputs, based on the values of asynchronous inputs. In this case, the effect of outputs is independent of clock transition. So, we will get the **parallel outputs** from each D flip-flop.

**Example**

Let us see the working of 3-bit PIPO shift register by applying the binary information **“011”** in parallel through preset inputs.

Since the preset inputs are applied before positive edge of Clock, the initial status of the D flip-flops from leftmost to rightmost will be Q2Q1Q0=011Q2Q1Q0=011. So, the binary information **“011”** is obtained in parallel at the outputs of D flip-flops before applying positive edge of clock.

Therefore, the 3-bit PIPO shift register requires zero clock pulses in order to produce the valid output. Similarly, the **N-bit PIPO shift register** doesn’t require any clock pulse in order to shift ‘N’ bit information.

## **2.3: Transistor**

Earlier, the critical and important component of an electronic device was a vacuum tube; it is an electron tube used to control electric current. The vacuum tubes worked but they are bulky, require higher operating voltages, high power consumption, yield lower efficiency and cathode electron-emitting materials are used up in operation. So, that ended up as heat which shortened the life of the tube itself. To overcome these problems, John Bardeen, Walter Brattain and William Shockley were invented a transistor at Bell Labs in the year of 1947. This new device was a much more elegant solution to overcome many of the fundamental limitations of vacuum tubes.

Transistor is a semiconductor device that can both conduct and insulate. A transistor can act as a switch and an amplifier. It converts audio waves into electronic waves and resistor, controlling electronic current. Transistors have very long life, smaller in size, can operate on lower voltage supplies for greater safety and required no filament current. The first transistor was fabricated with germanium. A transistor performs the same function as a vacuum tube triode, but using semiconductor junctions instead of heated electrodes in a vacuum chamber. It is the fundamental building block of modern electronic devices and found everywhere in modern electronic systems.

**Transistor Basics:**

A transistor is a three-terminal device. Namely,

* Base: This is responsible for activating the transistor.
* Collector: This is the positive lead.
* Emitter: This is the negative lead.

The basic idea behind a transistor is that it lets you control the flow of current through one channel by varying the intensity of a much smaller current that’s flowing through a second channel.

**Types of Transistors:**

There are two types of transistors in present; they are bipolar junction transistor (BJT), field effect transistors (FET). A small current is flowing between the base and the emitter; base terminal can control a larger current flow between the collector and the emitter terminals. For a field-effect transistor, it also has the three terminals, they are gate, source, and drain, and a voltage at the gate can control a current between source and drain. The simple diagrams of BJT and FET are shown in figure below:

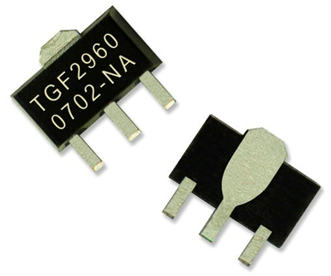
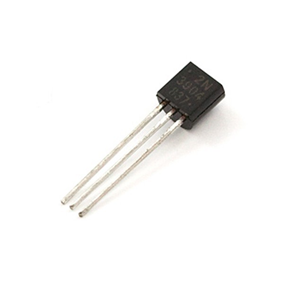


Figure 6 Bipolar Junction Transistor (BJT) Figure 7 Field Effect Transistors (FET)

* **Bipolar Junction Transistor:**

A Bipolar Junction Transistor (BJT) has three terminals connected to three doped semiconductor regions. It comes with two types, P-N-P and N-P-N.

P-N-P transistor, consisting of a layer of N-doped semiconductor between two layers of P-doped material. The base current entering in the collector is amplified at its output.

That is when PNP transistor is ON when its base is pulled low relative to the emitter. The arrows of PNP transistor symbol the direction of current flow when the device is in forward active mode.

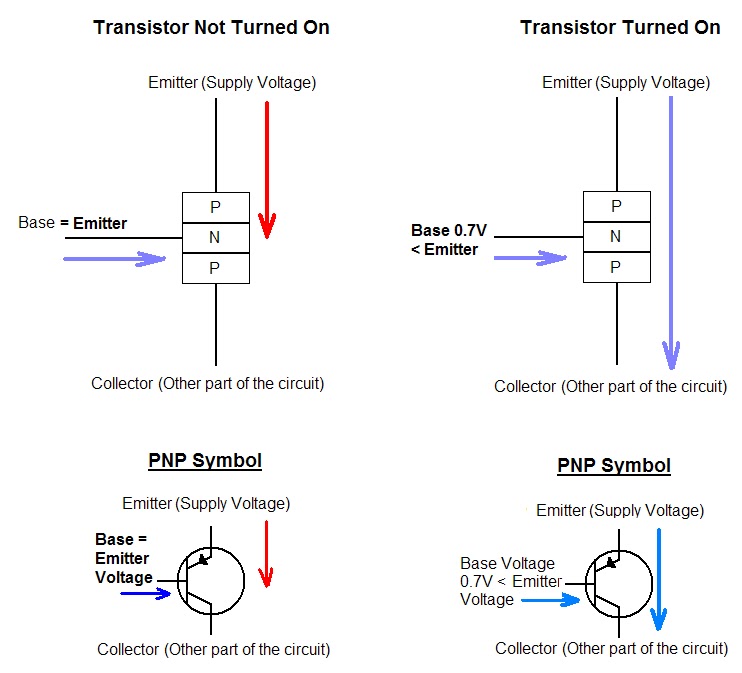
[](https://www.elprocus.com/wp-content/uploads/2013/04/PNP-working-diagram1.jpg)

Figure 8 Circuit Diagram of PNP Transistor

N-P-N transistor consisting a layer of P-doped semiconductor between two layers of N-doped material. By amplifying current the base, we get the high collector and emitter current.

That is when NPN transistor is ON when its base is pulled low relative to the emitter. When the transistor is in ON state, current flow is in between the collector and emitter of the transistor. Based on minority carriers in P-type region the electrons moving from emitter to collector. It allows the greater current and faster operation; because of this reason most bipolar transistors used today are NPN.

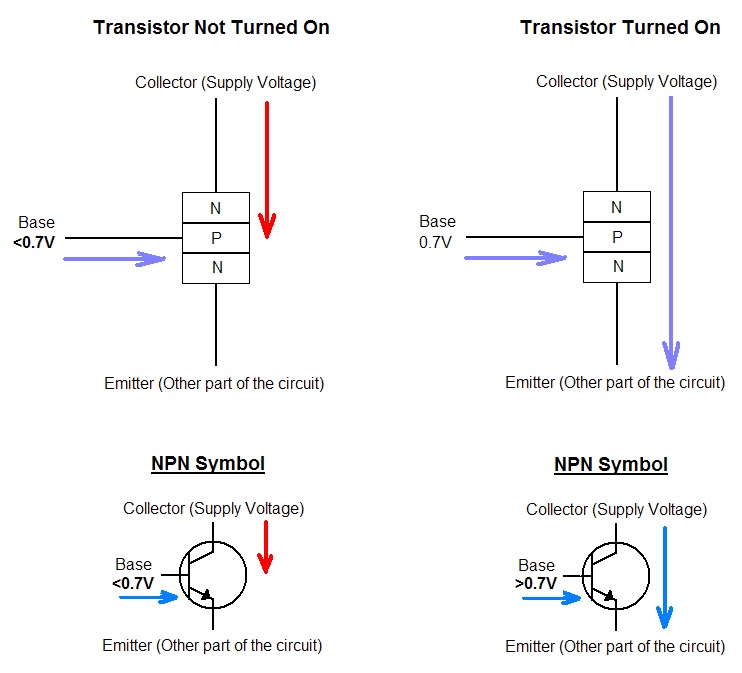
[](https://www.elprocus.com/wp-content/uploads/2013/04/NPN-working-diagram1.jpg)

Figure 9Circuit diagram of NPN Transistor

* **Field Effect Transistor (FET):**

The field-effect transistor is a unipolar transistor, N-channel FET or P-channel FET are used for conduction. The three terminals of FET are source, gate and drain. The basic n-channel and p-channel FET’s are shown above. For an n-channel FET, the device is constructed from n-type material. Between the source and drain then-type material acts as a resistor.

This transistor controls the positive and negative carriers with respect to holes or electrons. FET channel is formed by moving of positive and negative charge carriers. The channel of FET which is made by silicon.

There are many types of FET’s, MOSFET, JFET and etc. The applications of FET’s are in low noise amplifier, buffer amplifier and analog switch.

**Bipolar Junction Transistor Biasing**

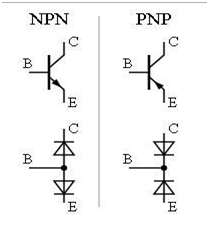
[](https://www.elprocus.com/wp-content/uploads/2013/10/Diode-NPN-PNP.png)

Figure 10 Bipolar JunctionTransistor **(**BJT) Biasing common emitter

Transistors are the most important semiconductor active devices essential for almost all circuits. They are used as electronic switches, amplifiers etc. in circuits. Transistors may be NPN, PNP, FET, JFET etc. which have different functions in electronic circuits. For the proper working of the circuit, it is necessary to bias the transistor using resistor networks. Operating point is the point on the output characteristics that shows the Collector-Emitter voltage and the Collector current with no input signal. The Operating point is also known as the Bias point or Q-Point (Quiescent point).

Biasing is referred to provide resistors, capacitors or supply voltage etc. to provide proper operating characteristics of the transistors. DC biasing is used to obtain DC collector current at a particular collector voltage. The value of this voltage and current are expressed in terms of the Q-Point. In a transistor amplifier configuration, the IC (max) is the maximum current that can flow through the transistor and VCE (max) is the maximum voltage applied across the device. To work the transistor as an amplifier, a load resistor RC must be connected to the collector. Biasing set the DC operating voltage and current to the correct level so that the AC input signal can be properly amplified by the transistor. The correct biasing point is somewhere between the fully ON or fully OFF states of the transistor. This central point is the Q-Point and if the transistor is properly biased, the Q-point will be the central operating point of the transistor. This helps the output current to increase and decrease as the input signal swings through the complete cycle.

For setting the correct Q-Point of the transistor, a collector resistor is used to set the collector current to a constant and steady value without any signal in its base. This steady DC operating point is set by the value of the supply voltage and the value of the base biasing resistor. Base bias resistors are used in all the three transistor configurations like common base, common collector and Common emitter configurations.

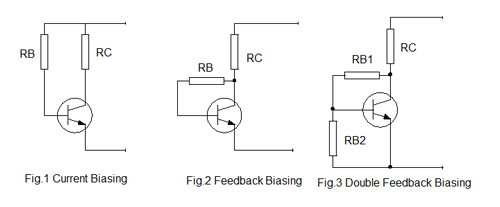
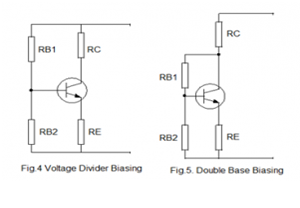
[](https://www.elprocus.com/wp-content/uploads/2013/03/TRANSISTOR-BIASING-1.png)[](https://www.elprocus.com/wp-content/uploads/2013/10/Transistor.png)

Figure 11 Bipolar JunctionTransistor **(**BJT) Biasing common collector

**Modes of biasing:**

Following are the different modes of transistor base biasing:

**1. Current biasing:**

As shown in the Fig.1, two resistors RC and RB are used to set the base bias. These resistors establish the initial operating region of the transistor with a fixed current bias.

The transistor forward biases with a positive base bias voltage through RB.  The forward base-Emitter voltage drop is 0.7 volts. Therefore the current through RB is IB = (Vcc– VBE ) / IB

**2. Feedback biasing:**

Fig.2 shows the transistor biasing by the use of a feedback resistor. The base bias is obtained from the collector voltage. The collector feedback ensures that the transistor is always biased in the active region. When the collector current increases, the voltage at the collector drops. This reduces the base drive which in turn reduces the collector current. This feedback configuration is ideal for transistor amplifier designs.

**3. Double Feedback Biasing:**

Fig.3 shows how the biasing is achieved using double feedback resistors.

By using two resistors RB1 and RB2 increases the stability with respect to the variations in Beta by increasing the current flow through the base bias resistors. In this configuration, the current in RB1 is equal to 10 % of the collector current.

**4. Voltage Dividing Biasing:**

Fig.4 shows the Voltage divider biasing in which two resistors RB1 and RB2 are connected to the base of the transistor forming a voltage divider network. The transistor gets biases by the voltage drop across RB2. This kind of biasing configuration is used widely in amplifier circuits.

**5. Double Base Biasing:**

Fig.5 shows a double feedback for stabilization. It uses both Emitter and Collector base feedback to improve the stabilization through controlling the collector current. Resistor values should be selected so as to set the voltage drop across the Emitter resistor 10% of the supply voltage and the current through RB1, 10% of the collector current.

**Advantages of Transistor:**

1. Smaller mechanical sensitivity.
2. Lower cost and smaller in size, especially in small-signal circuits.
3. Low operating voltages for greater safety, lower costs and tighter clearances.
4. Extremely long life.
5. No power consumption by a cathode heater.
6. Fast switching.

## **2.4: Dot Matrix**

LED dot matrices are very popular means of displaying information as it allows both static and animated text and images. Perhaps, you have encountered them at gas stations displaying the gas prices, or in the public places and alongside highways, displaying advertisements on large dot matrix panels.

In a dot matrix display, multiple LEDs are wired together in rows and columns. This is done to minimize the number of pins required to drive them. For example, a 8×8 matrix of LEDs (shown below) would need 64 I/O pins, one for each LED pixel. By wiring all the anodes together in rows (R1 through R8), and cathodes in columns (C1 through C8), the required number of I/O pins is reduced to 16. Each LED is addressed by its row and column number. In the figure below, if R4 is pulled high and C3 is pulled low, the LED in fourth row and third column will be turned on. Characters can be displayed by fast scanning of either rows or columns. This tutorial will discuss the method of column scanning.

The LED matrix used in this experiment is of size 5×7. We will learn how to display still characters in a standard 5×7-pixel format. The figure below shows which LEDs are to be turned on to display the English alphabet ‘A’. The 7 rows and 5 columns are controlled through the microcontroller pins. Now, let’s see in detail how it works.

Suppose, we want to display the alphabet A. We will first select the column C1 (which means C1 is pulled low in this case), and deselect other columns by blocking their ground paths (one way of doing that is by pulling C2 through C5 pins to logic high). Now, the first column is active, and you need to turn on the LEDs in the rows R2 through R7 of this column, which can be done by applying forward bias voltages to these rows. Next, select the column C2 (and deselect all other columns), and apply forward bias to R1 and R5, and so on. Therefore, by scanning across the column quickly (> 100 times per second), and turning on the respective LEDs in each row of that column, the persistence of vision comes in to play, and we perceive the display image as still.

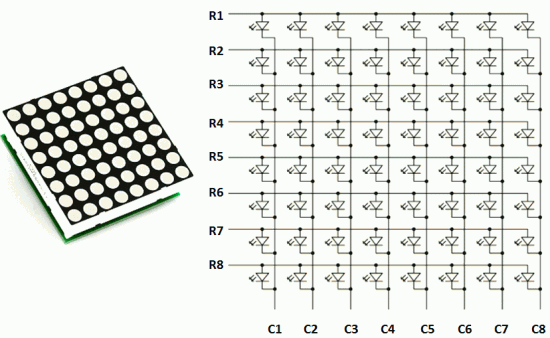


Figure 12 Structure of a 8x8 LED dot matrix

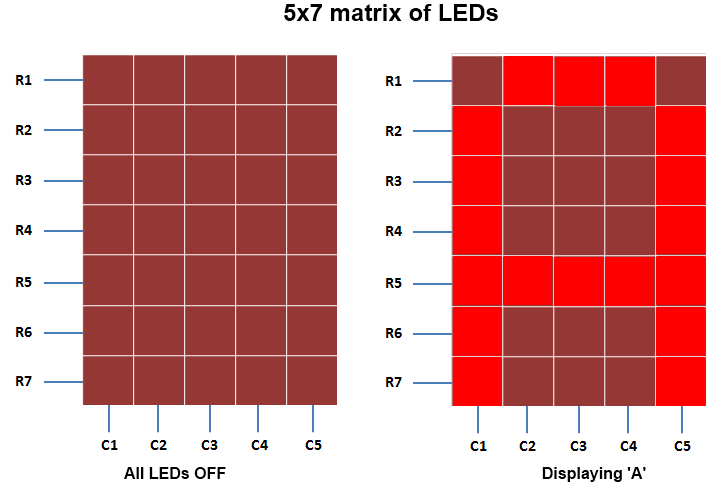
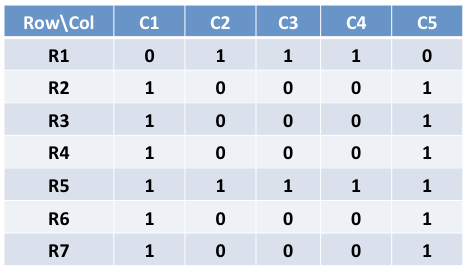


Figure 13 Output Structure of a 8x8 LED dot matrix

**Table 2.4:** The table below gives the logic levels to be applied to R1 through R7 for each of the columns in order to display the alphabet ‘A’.

[](http://embedded-lab.com/blog/wp-content/uploads/2011/04/RowColValuesforA.png)

Row values of each column for displaying the alphabet A

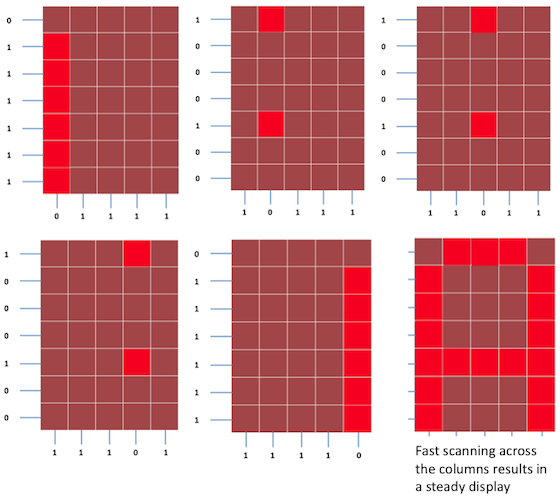


Figure 14 Scanning across the columns and feeding with appropriate row values

You should have noted that across each row, one pin is sourcing the current for only one LED at a time, but a column pin may have to sink the currents from more than one LED. For example, the column C1 should be able to sink the currents from 6 LEDs while displaying the alphabet ‘A’. A microcontroller’s I/O pin cannot sink this much of current, so external transistor arrays are required. I am using ULN2003A IC which has seven built-in Darlington transistor arrays (see below). The inputs of ULN2003A are active high. This means the input pins must be supplied with logic high in order to bring the corresponding output pins to ground. The schematic of the Darlington transistor array inside the ULN2003A chip is shown below.

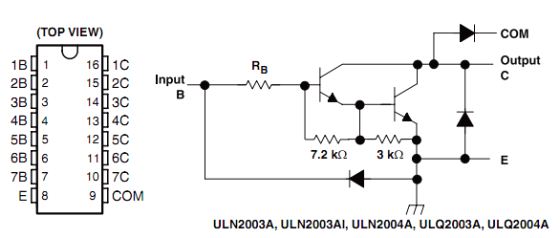


Figure 15 Pin diagram and schematic of ULN2003A (Darlington transistor arrays)

**Circuit Setup**

The circuit setup for this experiment is quite simple. You need seven 330? resistors in series with rows R1 through R7 to limit the current through the LEDs. Then the rows are driven by RB0 through RB6 pins of PIC18F2550. The columns are connected to the five outputs of ULN2003A. The corresponding five input pins of ULN2003A IC are controlled by RA0 through RA4 pins of PIC18F2550. The microcontroller will, therefore, scan across the column by sending appropriate bits to PORTA. For example, setting RA0 to 1 and clearing RA1 through RA4 bits, will select the first column. The microcontroller will wait for about 1 MS before switching to the next column. At each column, the microcontroller will output the corresponding row value at PORTB to turn on the appropriate LEDs in the column that are required to display the specific character. The switching between columns is fast enough to deceive the human eyes and a steady character is displayed.

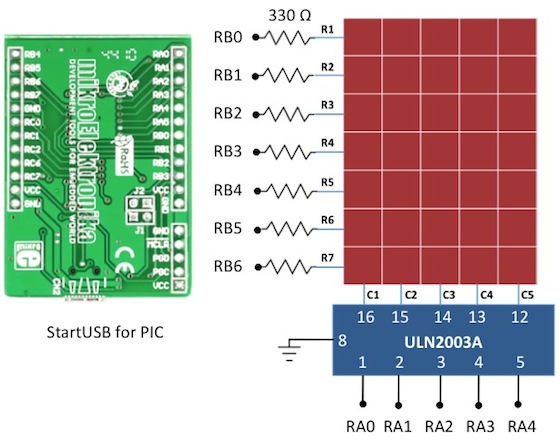


Figure 16 Circuit diagram for interfacing a 5x7 LED dot matrix with PIC18F2550

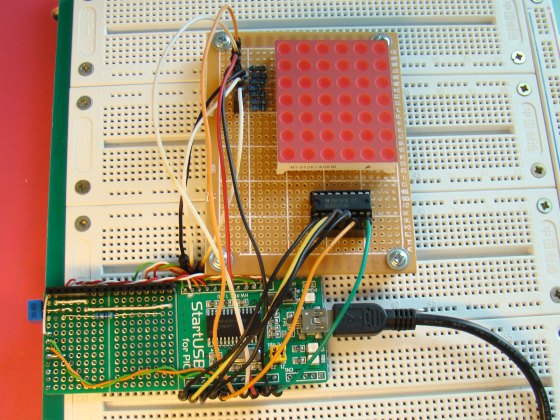


Figure 17 Circuit setup showing a Start USB board with a 6x7 LED dot matrix

**Software**

The major part of this experiment is the software routine to scan the columns and feed the rows with appropriate values. The column-specific row values for display characters can be either defined in RAM or stored in the program memory in case the on-board RAM is not sufficient enough. In micro C, the variables are saved in RAM and constants are stored in program memory. So, if your PIC does not have enough RAM, you can define a constant array to store the row values so that a part of the program memory is occupied by it to free up the on-board RAM. PIC18F2550 has quite a bit of RAM (2 KB), so I have used RAM to store the row values for alphabets A through Z.

## **2.5: Resistor**

Resistors are electronic components which have a specific, never-changing electrical resistance. The resistor's resistance **limits the flow of electrons** through a circuit.

They are **passive** components, meaning they only consume power (and can't generate it). Resistors are usually added to circuits where they complement **active** components like op-amps, microcontrollers, and other integrated circuits. Commonly resistors are used to limit current, divide voltage, and pull-up I/O lines.

**Resistor units**

The electrical resistance of a resistor is measured in **ohms**. The symbol for an ohm is the Greek capital-omega: Ω. The (somewhat roundabout) definition of 1Ω is the resistance between two points where 1 volt (1V) of applied potential energy will push 1 ampere (1A) of current.

As SI units go, larger or smaller values of ohms can be matched with a prefix like kilo-, mega-, or giga-, to make large values easier to read. It's very common to see resistors in the kilohm (kΩ) and megaohm (MΩ) range (much less common to see milliohm (mΩ) resistors). For example, a 4,700Ω resistor is equivalent to a 4.7kΩ resistor, and a 5,600,000Ω resistor can be written as 5,600kΩ or (more commonly as) 5.6MΩ.

**Schematic symbol**

All resistors have **two terminals**, one connection on each end of the resistor. When modeled on a schematic, a resistor will show up as one of these two symbols:

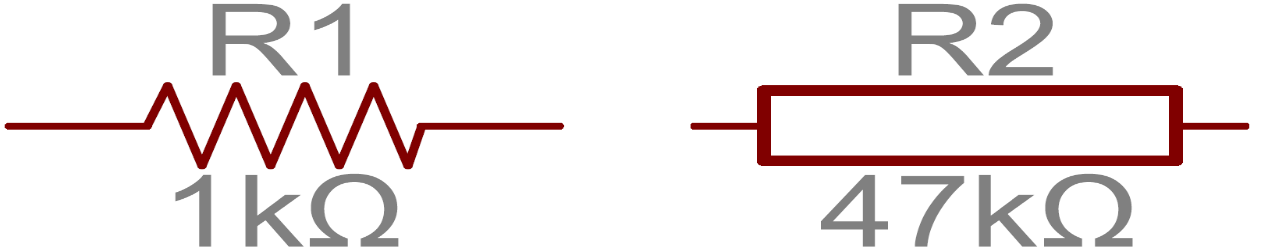


Figure 18 *Two common resistor schematic symbols. R1 is an American-style 1kΩ resistor, and R2 is an international-style 47kΩ resistor.*

The terminals of the resistor are each of the lines extending from the squiggle (or rectangle). Those are what connect to the rest of the circuit.

The resistor circuit symbols are usually enhanced with both a resistance value and a name. The value, displayed in ohms, is obviously critical for both evaluating and actually constructing the circuit. The name of the resistor is usually an R preceding a number. Each resistor in a circuit should have a unique name/number. For example, here's a few resistors in action on a 555-timer circuit:

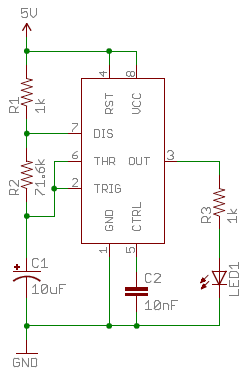
[](https://cdn.sparkfun.com/assets/7/f/4/f/7/515dc512ce395f4e58000001.PNG)

Figure 19 *In this circuit, resistors play a key role in setting the frequency of the 555 timer's output. Another resistor (R3) limits the current through an LED.*

**Types of Resistors**

Resistors come in a variety of shapes and sizes. They might be through-hole or surface-mount. They might be a standard, static resistor, a pack of resistors, or a special variable resistor.

**Termination and Mounting**

Resistors will come in one of two termination-types: through-hole or surface-mount. These types of resistors are usually abbreviated as either PTH (plated through-hole) or SMD/SMT (surface-mount technology or device).

**Through-hole** resistors come with long, pliable leads which can be stuck into a breadboard or hand-soldered into a prototyping board or printed circuit board (PCB). These resistors are usually more useful in breadboarding, prototyping, or in any case where you'd rather not solder tiny, little 0.6mm-long SMD resistors. The long leads usually require trimming, and these resistors are bound to take up much more space than their surface-mount counterparts.

The most common through-hole resistors come in an axial package. The size of an axial resistor is relative to its power rating. A common ½W resistor measure about 9.2mm across, while a smaller ¼W resistor is about 6.3mm long.

[](https://cdn.sparkfun.com/assets/6/9/c/4/3/515dcac7ce395f7259000000.png)

Figure 20 *A half-watt (½W) resistor (above) sized up to a quarter-watt (¼W).*

**Decoding Resistor Markings**

Though they may not display their value outright, most resistors are marked to show what their resistance is. PTH resistors use a color-coding system (which really adds some flair to circuits), and SMD resistors have their own value-marking system.

**Decoding Resistor Color Bands**

When decoding the resistor color bands, consult a resistor color code table like the one below. For the first two bands, find that color's corresponding digit value. The 4.7kΩ resistor shown here has color bands of yellow and violet to begin - which have digit values of 4 and 7 (47). The third band of the 4.7kΩ is red, which indicates that the 47 should be multiplied by 102 (or 100). 47 times 100 is 4,700!

[](https://cdn.sparkfun.com/assets/1/0/4/0/9/5165df6ace395f343f000003.jpg)

Figure 23 *4.7kΩ resistor with four color bands*

If you're trying to commit the color band code to memory, a mnemonic device might help. There are a handfuls of (sometimes unsavory) mnemonics out there to help remember the resistor color code. A good one, which spells out the difference between black and brown is:

"**B**ig **b**rown **r**abbits **o**ften **y**ield **g**reat **b**ig **v**ocal **g**roans **w**hen **g**ingerly **s**napped."

Or, if you remember "ROY G. BIV", subtract the indigo (poor indigo, no one remembers indigo), and add black and brown to the front and gray and white to the back of the classic rainbow color-order.

**Resistor Color Code Table**

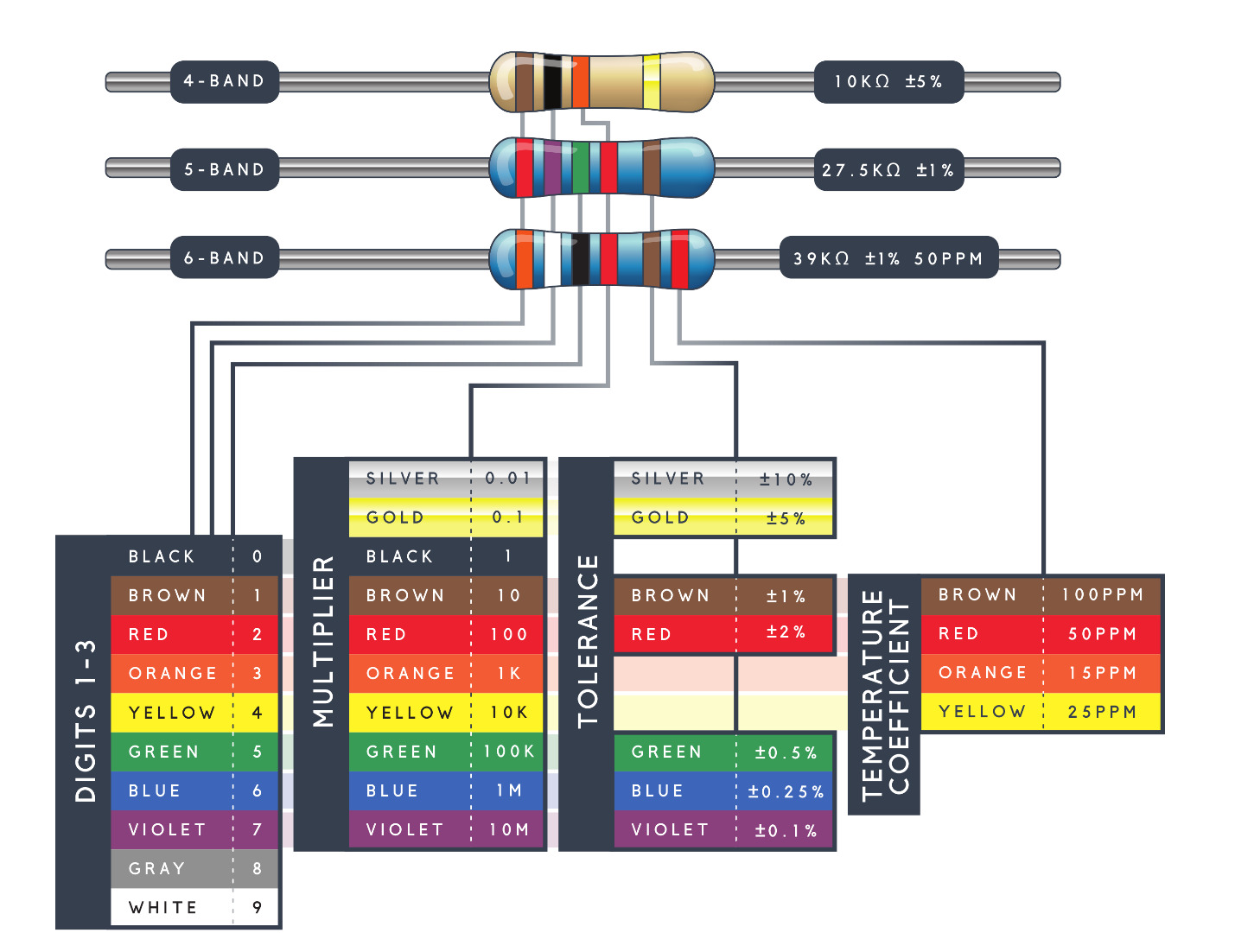


Figure 24 Color code table

**Pull-up Resistors**

A pull-up resistor is used when you need to bias a microcontroller's input pin to a known state. One end of the resistor is connected to the MCU's pin, and the other end is connected to a high voltage (usually 5V or 3.3V).

Without a pull-up resistor, inputs on the MCU could be left floating. There's no guarantee that a floating pin is either high (5V) or low (0V).

Pull-up resistors are often used when interfacing with a button or switch input. The pull-up resistor can bias the input-pin when the switch is open. And it will protect the circuit from a short when the switch is closed.

In the circuit above, when the switch is open the MCU's input pin is connected through the resistor to 5V. When the switch closes, the input pin is connected directly to GND.

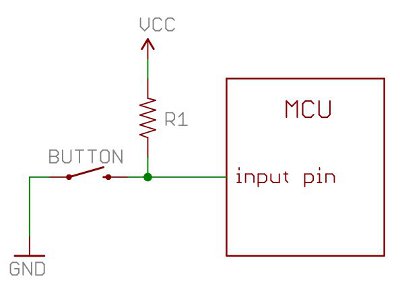
[](https://cdn.sparkfun.com/assets/7/a/5/1/3/515c86cfce395fb61d000000.jpg)

Figure 25 Pull-up resistor

The value of a pull-up resistor doesn't usually need to be anything specific. But it should be high enough that not too much power is lost if 5V or so is applied across it. Usually values around 10kΩ work well.

## **2.6: Power Supply**

A power supply is a component that supplies power to at least one electric load. Typically, it converts one type of electrical power to another, but it may also convert a a different form of energy – such as solar, mechanical, or chemical - into electrical energy.

A power supply provides components with electric power. The term usually pertains to devices integrated within the component being powered. For example, computer power supplies convert AC current to DC current and are generally located at the rear of the computer case, along with at least one fan.

A power supply is also known as a power supply unit, power brick or power adapter.

Most computer power supplies also have an input voltage switch, which can be set to 110v/115v or 220v/240v, depending on the geographic location. This switch position is crucial because of the different power voltages supplied by power outlets in different countries.

Most computers now use a switched-mode power supply, which changes AC current to DC voltage. This voltage can be switched on and off electronically. A switched- mode power supply can also shut itself down before damage is done when a short is detected.

Most computer power supplies include a number of switched-mode supplies, which operate independently by producing a single voltage. These are linked together, so that they shut down as a group in case of a computer fault.

**Parts found on the back of a power supply**

Below is a list of parts you may find on the back of the power supply.

* A connection for the power cord to the computer.
* A fan opening to heat out of the power supply.
* A red switch to change the power supply voltage.
* A rocker switch to turn the power supply on and off.

**Parts found inside a power supply**

Below is a list of parts you will find inside a power supply.

* A rectifier that converts AC (alternating current) into DC.
* A filter that smooths out the DC (direct current) coming from a rectifier.
* A transformer that controls the incoming voltage by stepping it up or down.
* A voltage regulator that controls the DC output, allowing the correct amount of power, volts or watts, to be supplied to the computer hardware.

The order in which these internal power supply components function is as follows.

1. Transformer
2. Rectifier
3. Filter
4. Voltage Regulator

**What items are powered by the computer PSU?**

Everything contained in the computer chassis is powered by the power supply. For example, the motherboard, RAM, CPU, hard drive, disc drives, and most video card (if the computer has one) are all drawing power from the power supply. Any other external devices and peripherals, such as the computer monitor and printer, have their own power source.

# **Chapter 3: Component Description**

## **3.1: Arduino Nano**

The Arduino Nano is a small, complete, and breadboard-friendly board based on the ATmega328 (Arduino Nano 3.0) or ATmega168 (Arduino Nano 2.x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. The Nano was designed and is being produced by Gravitech.

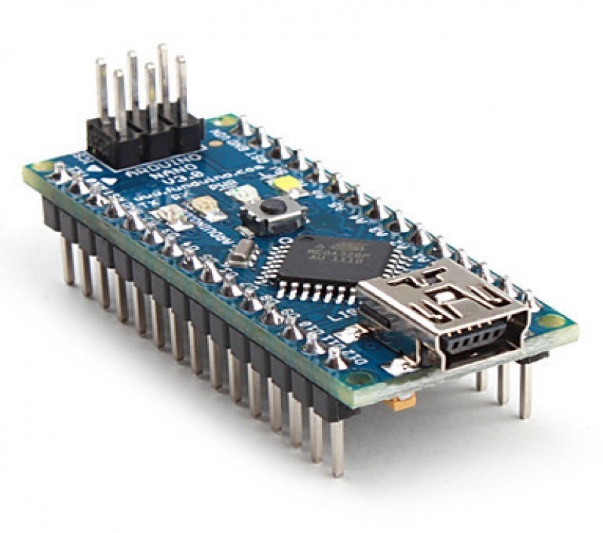


Figure 26 Arduino Nano

**Arduino Nano Pin Configuration**

**Vin.** It is input power supply voltage to the board when using an external power source of 7 to 12 V.

**5V.** It is a regulated power supply voltage of the board that is used to power the controller and other components placed on the board.

**3.3V.** This is a minimum voltage generated by the voltage regulator on the board.

**GND.** These are the ground pins on the board. There are multiple ground pins on the board that can be interfaced accordingly when more than one ground pin is required.

**Reset.** Reset pin is added on the board that resets the board. It is very helpful when running program goes too complex and hangs up the board. LOW value to the reset pin will reset the controller.

**Analog Pins.** There are 8 analog pins on the board marked as A0 – A7. These pins are used to measure the analog voltage ranging between 0 to 5V.

**Rx, Tx.** These pins are used for serial communication where Tx represents the transmission of data while Rx represents the data receiver.

**13.** This pin is used to turn on the built-in LED.

**AREF.** This pin is used as a reference voltage for the input voltage.

**PWM.** Six pins 3,5,6,9,10, 11 can be used for providing 8-pit PWM (Pulse Width Modulation) output. It is a method used for getting analog results with digital sources.

**SPI.** Four pins 10(SS),11(MOSI),12(MISO),13(SCK) are used for SPI (Serial Peripheral Interface). SPI is an interface bus and mainly used to transfer data between microcontrollers and other peripherals like sensors, registers, and SD card.

**External Interrupts.** Pin 2 and 3 are used as external interrupts which are used in case of emergency when we need to stop the main program and call important instructions at that point. The main program resumes once interrupt instruction is called and executed.

**I2C.** I2C communication is developed using A4 and A5 pins where A4 represents the serial data line (SDA) which carries the data and A5 represents the serial clock line (SCL) which is a clock signal, generated by the master device, used for data synchronization between the devices on an I2C bus.

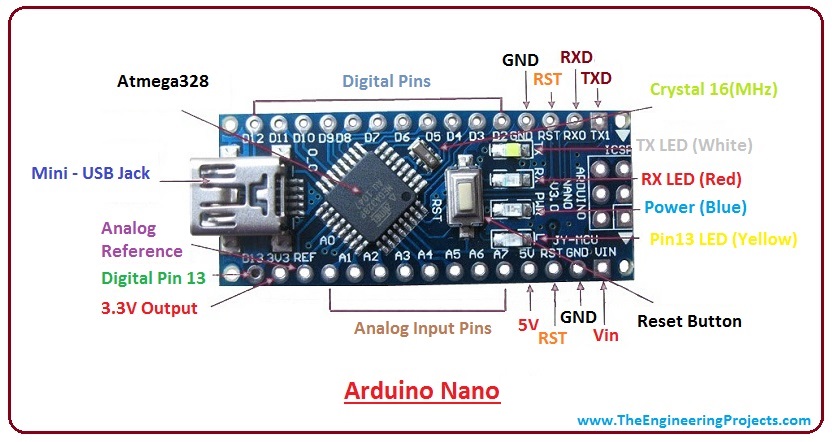


Figure 27Pin Configuration of Arduino Nano

**Communication and Programming**

* The Nano device comes with an ability to set up a communication with other controllers and computers. The serial communication is carried out by the digital pins like pin 0 (Rx) and pin 1 (Tx) where Rx is used for receiving data and Tx is used for the transmission of data. The serial monitor is added on the Arduino Software which is used to transmit textual data to or from the board. FTDI drivers are also included in the software which behave as a virtual com port to the software.
* The Tx and Rx pins come with an LED which blinks as the data is transmitted between FTDI and USB connection to the computer.
* Arduino Software Serial Library is used for carrying out a serial communication between the board and the computer.
* Apart from serial communication the Nano board also support I2C and SPI communication. The Wire Library inside the Arduino Software is accessed to use the I2C bus.
* The Arduino Nano is programmed by Arduino Software called IDE which is a common software used for almost all types of board available. Simply download the software and select the board you are using. There are two options to program the controller i.e either by the bootloader that is added in the software which sets you free from the use of external burner to compile and burn the program into the controller and another option is by using ICSP (In-circuit serial programming header).
* Arduino board software is equally compatible with Windows, Linux or MAC, however, Windows are preferred to use.

**How to reset Arduino Nano Board?**

There are two ways to reset the board i.e. electronically or programmatically.

In order to reset the board electronically, you need to connect the reset pin of the board with any of digital pins on the controller. Don’t forget to add 1K or 2K Ohm resistor while setting up this connection. Now, use the digital pin as an output and keep it HIGH before the reset. Once the reset is required, set this digital pin to LOW. This method is very useful because using it sends a hardware reset signal to the controller once the digital pin is set to LOW. You can use the following program to reset the controller electronically.

int Reset = 4;

void setup() {

digitalWrite(Reset, HIGH);

delay(200);

pinMode(Reset, OUTPUT);

Serial.begin(9600);

Serial.println("How to Reset Arduino Programmatically");

Serial.println("www.TheEngineeringProjects.com");

delay(200);

}

void loop() {

Serial.println("A");

delay(1000);

Serial.println("B");

delay(1000);

Serial.println("Now we are Resetting Arduino Programmatically");

Serial.println();

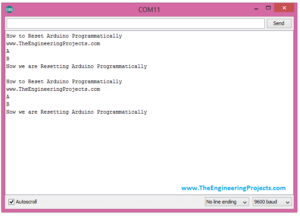
delay(1000);

digitalWrite(Reset, LOW);

Serial.println("Arduino will never reach there.");

}

Once you upload the program, open your Arduino serial monitor that shows output as follows.

[](https://www.theengineeringprojects.com/wp-content/uploads/2018/06/introduction-to-arduino-nano-3.png)

Another method we can use to reset the board is by software only without using any hardware pin. Nano board comes with a built-in function known as resetFunc(). The board will reset automatically as we define this function and then call it. Without using any hardware pin you can upload the following program to reset the board programmatically.

void(\* resetFunc) (void) = 0;

void setup() {

Serial.begin(9600);

Serial.println("How to Reset Arduino Programmatically");

Serial.println("www.TheEngineeringProjects.com");

delay(200);

}

void loop() {

Serial.println("A");

delay(1000);

Serial.println("B");

delay(1000);

Serial.println("Now we are Resetting Arduino Programmatically");

Serial.println();

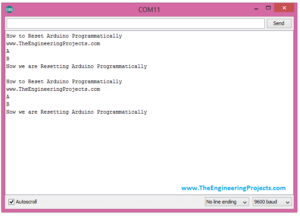
delay(1000);

resetFunc();

Serial.println("Arrduino will never reach there.");

}

As you open the Arduino Serial Terminal you will get the output below.

[](https://www.theengineeringprojects.com/wp-content/uploads/2018/06/introduction-to-arduino-nano-3.png)

However, this method comes with some limitations. Once the board is connected to the computer, the board will be reset each time the connection is laid out between the board and the computer. So, it is preferred to reset the controller electronically using a digital pin.

**Application:**

Arduino Nano is a very useful device that comes with a wide range of applications and covers less space as compared to other Arduino board. Breadboard friendly nature makes it stand out from other board. Following are the main applications of the board.

* Arduino Metal Detector
* Real-Time Face Detection
* Medical Instruments
* Industrial Automation
* Android Applications
* GSM Based Projects
* Embedded System
* Automation and Robotics
* Home Automation and Defense Systems
* Virtual Reality Applications

## **3.2: Bluetooth Module**

HC-06 Bluetooth Module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Its communication is via serial communication which makes an easy way to interface with controller or PC. HC-06 Bluetooth module provides switching mode between master and slave mode which means it able to use neither receiving nor transmitting data.



Figure 28HC-06 Bluetooth Module

**Table-3.1: Pin Configuration table of Bluetooth Nano**

|  |  |  |
| --- | --- | --- |
| **Pin Number** | **Pin Name** | **Description** |
| 1 | Enable / Key | This pin is used to toggle between Data Mode (set low) and AT command mode (set high). By default, it is in Data mode |
| 2 | Vcc | Powers the module. Connect to +5V Supply voltage |
| 3 | Ground | Ground pin of module, connect to system ground. |
| 4 | TX – Transmitter | Transmits Serial Data. Everything received via Bluetooth will be given out by this pin as serial data. |
| 5 | RX – Receiver | Receive Serial Data. Every serial data given to this pin will be broadcasted via Bluetooth |
| 6 | State | The state pin is connected to on board LED, it can be used as a feedback to check if Bluetooth is working properly. |
| 7 | LED | Indicates the status of Module   * Blink once in 2 sec: Module has entered Command Mode * Repeated Blinking: Waiting for connection in Data Mode * Blink twice in 1 sec: Connection successful in Data Mode |
| 8 | Button | Used to control the Key/Enable pin to toggle between Data and command Mode |

**HC-05 Default Settings**

Default Bluetooth Name: “HC-06”

Default Password: 1234 or 0000

Default Communication: Slave

Default Mode: Data Mode

Data Mode Baud Rate: 9600, 8, N, 1

Command Mode Baud Rate: 38400, 8, N, 1

Default firmware: LINVOR

**HC-06 Technical Specifications**

* Serial Bluetooth module for Arduino and other microcontrollers
* Operating Voltage: 4V to 6V (Typically +5V)
* Operating Current: 30mA
* Range: <100m
* Works with Serial communication (USART) and TTL compatible
* Follows IEEE 802.15.1 standardized protocol
* Uses Frequency-Hopping Spread spectrum (FHSS)
* Can operate in Master, Slave or Master/Slave mode
* Can be easily interfaced with Laptop or Mobile phones with Bluetooth
* Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.

**Where to use HC-06 Bluetooth module**

The **HC-06** is a very cool module which can add two-way (full-duplex) wireless functionality to your projects. You can use this module to communicate between two microcontrollers like Arduino or communicate with any device with Bluetooth functionality like a Phone or Laptop. There are many android applications that are already available which makes this process a lot easier. The module communicates with the help of USART at 9600 baud rates hence it is easy to interface with any microcontroller that supports USART. We can also configure the default values of the module by using the command mode. So, if you looking for a Wireless module that could transfer data from your computer or mobile phone to microcontroller or vice versa then this module might be the right choice for you. However, do not expect this module to transfer multimedia like photos or songs; you might have to look into the CSR8645 module for that.

**How to Use the HC-06 Bluetooth module**

The **HC-06** has two operating modes, one is the Data mode in which it can send and receive data from other Bluetooth devices and the other is the AT Command mode where the default device settings can be changed. We can operate the device in either of these two modes by using the key pin as explained in the pin description.

It is very easy to pair the HC-05 module with microcontrollers because it operates using the Serial Port Protocol (SPP). Simply power the module with +5V and connect the Rx pin of the module to the Tx of MCU and Tx pin of module to Rx of MCU as shown in the figure below

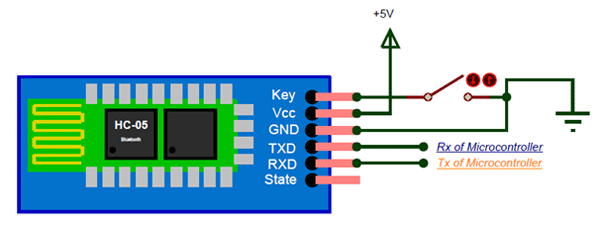


Figure 29Bluetooth Module

During power up the key pin can be grounded to enter into Command mode, if left free it will by default enter into the data mode. As soon as the module is powered you should be able to discover the Bluetooth device as “HC-05” then connect with it using the default password 1234 and start communicating with it.

**Applications**

1. Wireless communication between two microcontrollers

2. Communicate with Laptop, Desktops and mobile phones

3. Data Logging application

4. Consumer applications

5. Wireless Robots

6. Home Automation

**2D Model**

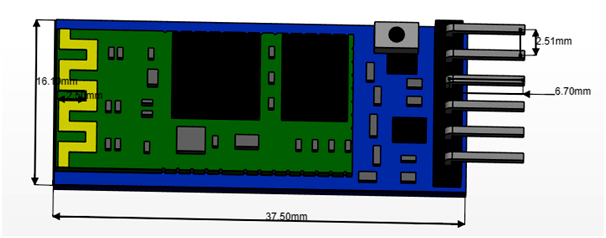
****

Figure 30 2D Model of Bluetooth Module

# **Chapter 4 System Methodology**

## **4.1 Bluetooth overview**

The Android platform includes support for the Bluetooth network stack, which allows a device to wirelessly exchange data with other Bluetooth devices. The application framework provides access to the Bluetooth functionality through the Android Bluetooth APIs. These APIs let applications wirelessly connect to other Bluetooth devices, enabling point-to-point and multipoint wireless features.

Using the Bluetooth APIs, an Android application can perform the following:

* Scan for other Bluetooth devices
* Query the local Bluetooth adapter for paired Bluetooth devices
* Establish RFCOMM channels
* Connect to other devices through service discovery
* Transfer data to and from other devices
* Manage multiple connections

### **4.1.1 The basics**

In order for Bluetooth-enabled devices to transmit data between each other, we must first form a channel of communication using a *pairing* process. One device, a *discoverable device*, makes itself available for incoming connection requests. Another device finds the discoverable device using a *service discovery* process. After the discoverable device accepts the pairing request, the two devices complete a *bonding* process where we exchange security keys. The devices cache these keys for later use. After the pairing and bonding processes are complete, the two devices exchange information. When the session is complete, the device that initiated the pairing request releases the channel that had linked it to the discoverable device. The two devices remain bonded, however, so we can reconnect automatically during a future session as long as we're in range of each other and neither device has removed the bond.

### **4.1.2 Bluetooth permissions**

In order to use Bluetooth features in our application, we must declare two permissions. The first of these is [BLUETOOTH](https://developer.android.com/reference/android/Manifest.permission#BLUETOOTH). We need this permission to perform any Bluetooth communication, such as requesting a connection, accepting a connection, and transferring data.

The other permission that we must declare is [ACCESS\_FINE\_LOCATION](https://developer.android.com/reference/android/Manifest.permission#ACCESS_FINE_LOCATION). Our app needs this permission because a Bluetooth scan can be used to gather information about the location of the user. This information may come from the user's own devices, as well as Bluetooth beacons in use at locations such as shops and transit facilities.

Alternatively, on devices running Android 8.0 (API level 26) and higher, we can use the [CompanionDeviceManager](https://developer.android.com/reference/android/companion/CompanionDeviceManager) to perform a scan of nearby companion devices on behalf of our app without requiring the location permission.

If our app targets Android 9 (API level 28) or lower, we can declare the [ACCESS\_COARSE\_LOCATION](https://developer.android.com/reference/android/Manifest.permission#ACCESS_COARSE_LOCATION) permission instead of the ACCESS\_FINE\_LOCATION permission.

If we want our app to initiate device discovery or manipulate Bluetooth settings, we must declare the [BLUETOOTH\_ADMIN](https://developer.android.com/reference/android/Manifest.permission#BLUETOOTH_ADMIN) permission in addition to the [BLUETOOTH](https://developer.android.com/reference/android/Manifest.permission#BLUETOOTH) permission. Most applications need this permission solely for the ability to discover local Bluetooth devices. The other abilities granted by this permission should not be used, unless the application is a "power manager" that modifies Bluetooth settings upon user request.

Declare the Bluetooth permissions in application manifest file. For example:

<manifest ... >  
 <uses-permission android:name="android.permission.BLUETOOTH" />

  <uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />

  <!-- If app target SDK Android 9 or lower declare

       ACCESS\_COARSE\_LOCATION instead. -->

  <uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

</manifest>

### **4.1.3 Set up Bluetooth**

Before our application can communicate over Bluetooth, we need to verify that Bluetooth is supported on the device, and if so, ensure that it is enabled.

If Bluetooth isn't supported, then we should gracefully disable any Bluetooth features. If Bluetooth is supported, but disabled, then we can request that the user enable Bluetooth without leaving your application. This setup is accomplished in two steps, using the [BluetoothAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter):

1. **Get the** [**BluetoothAdapter**](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter)**.**

The [BluetoothAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter) is required for any and all Bluetooth activity. To get the [BluetoothAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter), call the static [getDefaultAdapter()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#getDefaultAdapter()) method. This returns a [BluetoothAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter) that represents the device's own Bluetooth adapter (the Bluetooth radio). There's one Bluetooth adapter for the entire system, and our application can interact with it using this object. If [getDefaultAdapter()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#getDefaultAdapter()) returns null, then the device doesn't support Bluetooth. For example:

**In** [**Java**](https://developer.android.com/guide/topics/connectivity/bluetooth#java) **Code**

BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

if (bluetoothAdapter == null) {

    // Device doesn't support Bluetooth

}

**Enable Bluetooth.**

Next, we need to ensure that Bluetooth is enabled. Call [isEnabled()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#isEnabled()) to check whether Bluetooth is currently enabled. If this method returns false, then Bluetooth is disabled. To request that Bluetooth be enabled, call [startActivityForResult()](https://developer.android.com/reference/android/app/Activity#startActivityForResult(android.content.Intent, int)), passing in an [ACTION\_REQUEST\_ENABLE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#ACTION_REQUEST_ENABLE) intent action. This call issues a request to enable Bluetooth through the system settings (without stopping our application). For example:

**In** [**Java**](https://developer.android.com/guide/topics/connectivity/bluetooth#java) **Code**

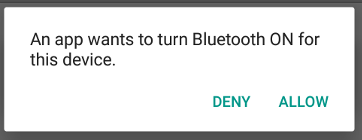
if (!bluetoothAdapter.isEnabled()) {

    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

    startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);

}

1. A dialog appears requesting user permission to enable Bluetooth, as shown in Figure 1. If the user responds "Yes", the system begins to enable Bluetooth, and focus returns to our application once the process completes (or fails).



**Figure 31** **The enabling Bluetooth dialog**.

The REQUEST\_ENABLE\_BT constant passed to [startActivityForResult()](https://developer.android.com/reference/android/app/Activity#startActivityForResult(android.content.Intent, int)) is a locally defined integer that must be greater than 0. The system passes this constant back to you in our [onActivityResult()](https://developer.android.com/reference/android/app/Activity#onActivityResult(int, int, android.content.Intent)) implementation as the requestCode parameter.

If enabling Bluetooth succeeds, our activity receives the [RESULT\_OK](https://developer.android.com/reference/android/app/Activity#RESULT_OK) result code in the [onActivityResult()](https://developer.android.com/reference/android/app/Activity#onActivityResult(int, int, android.content.Intent)) callback. If Bluetooth was not enabled due to an error (or the user responded "No") then the result code is [RESULT\_CANCELED](https://developer.android.com/reference/android/app/Activity#RESULT_CANCELED).

Optionally, our application can also listen for the [ACTION\_STATE\_CHANGED](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#ACTION_STATE_CHANGED) broadcast intent, which the system broadcasts whenever the Bluetooth state changes. This broadcast contains the extra fields [EXTRA\_STATE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#EXTRA_STATE) and [EXTRA\_PREVIOUS\_STATE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#EXTRA_PREVIOUS_STATE), containing the new and old Bluetooth states, respectively. Possible values for these extra fields are [STATE\_TURNING\_ON](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#STATE_TURNING_ON), [STATE\_ON](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#STATE_ON), [STATE\_TURNING\_OFF](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#STATE_TURNING_OFF), and [STATE\_OFF](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#STATE_OFF). Listening for this broadcast can be useful if our app needs to detect runtime changes made to the Bluetooth state.

### **4.1.4 Find devices**

Using the [BluetoothAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter), we can find remote Bluetooth devices either through device discovery or by querying the list of paired devices.

Device discovery is a scanning procedure that searches the local area for Bluetooth-enabled devices and requests some information about each one. This process is sometimes referred to as *discovering*, *inquiring*, or *scanning*. However, a nearby Bluetooth device responds to a discovery request only if it is currently accepting information requests by being *discoverable*. If a device is discoverable, it responds to the discovery request by sharing some information, such as the device's name, its class, and its unique MAC address. Using this information, the device that is performing the discovery process can then choose to initiate a connection to the discovered device.

Once a connection is made with a remote device for the first time, a pairing request is automatically presented to the user. When a device is paired, the basic information about that device, such as the device's name, class, and MAC address—is saved and can be read using the Bluetooth APIs. Using the known MAC address for a remote device, a connection can be initiated with it at any time without performing discovery, assuming the device is still within range.

There is a difference between being paired and being connected:

* To be *paired* means that two devices are aware of each other's existence, have a shared link-key that can be used for authentication, and are capable of establishing an encrypted connection with each other.
* To be *connected* means that the devices currently share an RFCOMM channel and are able to transmit data with each other. The current Android Bluetooth API's require devices to be paired before an RFCOMM connection can be established. Pairing is automatically performed when we initiate an encrypted connection with the Bluetooth APIs.

The following sections describe how to find devices that have been paired, or discover new devices using device discovery.

Android-powered devices are not discoverable by default. A user can make the device discoverable for a limited time through the system settings, or an application can request that the user enable discoverability without leaving the application.

### **4.1.5 Query paired devices**

Before performing device discovery, it's worth querying the set of paired devices to see if the desired device is already known. To do so, call [getBondedDevices()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#getBondedDevices()). This returns a set of [BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice) objects representing paired devices. For example, we can query all paired devices and get the name and MAC address of each device, as the following code snippet demonstrates:

**In** [**Java**](https://developer.android.com/guide/topics/connectivity/bluetooth#java) **Code**

Set<BluetoothDevice> pairedDevices = bluetoothAdapter.getBondedDevices();

if (pairedDevices.size() > 0) {

    // There are paired devices. Get the name and address of each paired device.

    for (BluetoothDevice device : pairedDevices) {

        String deviceName = device.getName();

        String deviceHardwareAddress = device.getAddress(); // MAC address

    }

}

To initiate a connection with a Bluetooth device, all that's needed from the associated [BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice) object is the MAC address, which we retrieve by calling [getAddress()](https://developer.android.com/reference/android/bluetooth/BluetoothDevice#getAddress()). You can learn more about creating a connection in the section about [Connecting Devices](https://developer.android.com/guide/topics/connectivity/bluetooth#ConnectDevices).

Performing device discovery consumes a lot of the Bluetooth adapter's resources. After we have found a device to connect to, be certain that we stop discovery with [cancelDiscovery()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#cancelDiscovery()) before attempting a connection. Also, we shouldn't perform discovery while connected to a device because the discovery process significantly reduces the bandwidth available for any existing connections.

### **4.1.6 Discover devices**

To start discovering devices, simply call [startDiscovery()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#startDiscovery()). The process is asynchronous and returns a boolean value indicating whether discovery has successfully started. The discovery process usually involves an inquiry scan of about 12 seconds, followed by a page scan of each device found to retrieve its Bluetooth name.

In order to receive information about each device discovered, our application must register a BroadcastReceiver for the [ACTION\_FOUND](https://developer.android.com/reference/android/bluetooth/BluetoothDevice#ACTION_FOUND) intent. The system broadcasts this intent for each device. The intent contains the extra fields [EXTRA\_DEVICE](https://developer.android.com/reference/android/bluetooth/BluetoothDevice#EXTRA_DEVICE) and [EXTRA\_CLASS](https://developer.android.com/reference/android/bluetooth/BluetoothDevice#EXTRA_CLASS), which in turn contain a [BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice) and a [BluetoothClass](https://developer.android.com/reference/android/bluetooth/BluetoothClass), respectively. The following code snippet shows how we can register to handle the broadcast when devices are discovered:

**In** [**Java**](https://developer.android.com/guide/topics/connectivity/bluetooth#java) **Code**

@Override

protected void onCreate(Bundle savedInstanceState) {

    ...

    // Register for broadcasts when a device is discovered.

    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION\_FOUND);

    registerReceiver(receiver, filter);

}

// Create a BroadcastReceiver for ACTION\_FOUND.

private final BroadcastReceiver receiver = new BroadcastReceiver() {

    public void onReceive(Context context, Intent intent) {

        String action = intent.getAction();

        if (BluetoothDevice.ACTION\_FOUND.equals(action)) {

            // Discovery has found a device. Get the BluetoothDevice

            // object and its info from the Intent.

            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA\_DEVICE);

            String deviceName = device.getName();

            String deviceHardwareAddress = device.getAddress(); // MAC address

        }

    }

};

@Override

protected void onDestroy() {

    super.onDestroy();

    ...

    // Don't forget to unregister the ACTION\_FOUND receiver.

    unregisterReceiver(receiver);

}

To initiate a connection with a Bluetooth device, all that's needed from the associated [BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice) object is the MAC address, which you retrieve by calling [getAddress()](https://developer.android.com/reference/android/bluetooth/BluetoothDevice#getAddress()).

Performing device discovery consumes a lot of the Bluetooth adapter's resources. After we have found a device to connect to, be certain that you stop discovery with [cancelDiscovery()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#cancelDiscovery()) before attempting a connection. Also, we shouldn't perform discovery while connected to a device because the discovery process significantly reduces the bandwidth available for any existing connections.

### **4.1.7 Enable discoverability**

If we would like to make the local device discoverable to other devices, call [startActivityForResult(Intent, int)](https://developer.android.com/reference/android/app/Activity#startActivityForResult(android.content.Intent, int)) with the [ACTION\_REQUEST\_DISCOVERABLE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#ACTION_REQUEST_DISCOVERABLE) intent. This issues a request to enable the system's discoverable mode without having to navigate to the Settings app, which would stop our own app. By default, the device becomes discoverable for 120 seconds, or 2 minutes. We can define a different duration, up to 3600 seconds (1 hour), by adding the [EXTRA\_DISCOVERABLE\_DURATION](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#EXTRA_DISCOVERABLE_DURATION) extra.

If we set the [EXTRA\_DISCOVERABLE\_DURATION](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#EXTRA_DISCOVERABLE_DURATION) extra's value to 0, the device is always discoverable. This configuration is insecure and therefore highly discouraged.

The following code snippet sets the device to be discoverable for 5 minutes (300 seconds):

**In** [**Java**](https://developer.android.com/guide/topics/connectivity/bluetooth#java) **Code**

Intent discoverableIntent =

        new Intent(BluetoothAdapter.ACTION\_REQUEST\_DISCOVERABLE);

discoverableIntent.putExtra(BluetoothAdapter.EXTRA\_DISCOVERABLE\_DURATION, 300);  
startActivity(discoverableIntent);

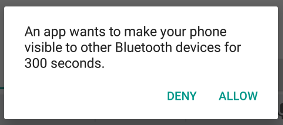


Figure 32 **The enabling discoverability dialog.**

A dialog is displayed, requesting the user's permission to make the device discoverable, as shown in Figure 2. If the user responds "Yes," then the device becomes discoverable for the specified amount of time. Our activity then receives a call to the [onActivityResult()](https://developer.android.com/reference/android/app/Activity#onActivityResult(int, int, android.content.Intent)) callback, with the result code equal to the duration that the device is discoverable. If the user responded "No", or if an error occurred, the result code is [RESULT\_CANCELED](https://developer.android.com/reference/android/app/Activity#RESULT_CANCELED).

If Bluetooth has not been enabled on the device, then making the device discoverable automatically enables Bluetooth.

The device silently remains in discoverable mode for the allotted time. If we would like to be notified when the discoverable mode has changed, we can register a BroadcastReceiver for the [ACTION\_SCAN\_MODE\_CHANGED](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#ACTION_SCAN_MODE_CHANGED) intent. This intent contains the extra fields [EXTRA\_SCAN\_MODE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#EXTRA_SCAN_MODE) and [EXTRA\_PREVIOUS\_SCAN\_MODE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#EXTRA_PREVIOUS_SCAN_MODE), which provide the new and old scan mode, respectively. Possible values for each extra are as follows:

[SCAN\_MODE\_CONNECTABLE\_DISCOVERABLE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#SCAN_MODE_CONNECTABLE_DISCOVERABLE)

The device is in discoverable mode.

[SCAN\_MODE\_CONNECTABLE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#SCAN_MODE_CONNECTABLE)

The device isn't in discoverable mode but can still receive connections.

[SCAN\_MODE\_NONE](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#SCAN_MODE_NONE)

The device isn't in discoverable mode and cannot receive connections.

If we are initiating the connection to a remote device, we don't need to enable device discoverability. Enabling discoverability is only necessary when we want our application to host a server socket that accepts incoming connections, as remote devices must be able to discover other devices before initiating connections to those other devices.

### **4.1.8 Connect devices**

In order to create a connection between two devices, we must implement both the server-side and client-side mechanisms because one device must open a server socket, and the other one must initiate the connection using the server device's MAC address. The server device and the client device each obtain the required [Bluetooth Socket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket) in different ways. The server receives socket information when an incoming connection is accepted. The client provides socket information when it opens an RFCOMM channel to the server.

The server and client are considered connected to each other when they each have a connected [Bluetooth Socket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket) on the same RFCOMM channel. At this point, each device can obtain input and output streams, and data transfer can begin, which is discussed in the section about [manage a connection](https://developer.android.com/guide/topics/connectivity/bluetooth#ManageAConnection). This section describes how to initiate the connection between two devices.

### **4.1.9 Connection techniques**

One implementation technique is to automatically prepare each device as a server so that each device has a server socket open and listening for connections. In this case, either device can initiate a connection with the other and become the client. Alternatively, one device can explicitly host the connection and open a server socket on demand, and the other device initiates the connection.

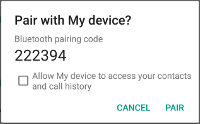


Figure 33 ***The Bluetooth pairing dialog*.**

If the two devices have not been previously paired, then the Android framework automatically shows a pairing request notification or dialog to the user during the connection procedure, as shown in Figure 3. Therefore, when our application attempts to connect devices, it doesn't need to be concerned about whether or not the devices are paired. Our RFCOMM connection attempt gets blocked until the user has successfully paired the two devices, and the attempt fails if the user rejects pairing, or if the pairing process fails or times out.

## **4.2 Hardware Part**

A monochrome (single color) LED dot matrix display is used for displaying the Characters and Symbols which is interface with a microcontroller. This project will deliberate on displaying a scrolling text message on a 48×8 LED dot matrix display.

## **4.2.1 Block Diagram**

The block diagram of mobile application controlled scrolling display as shown as figure

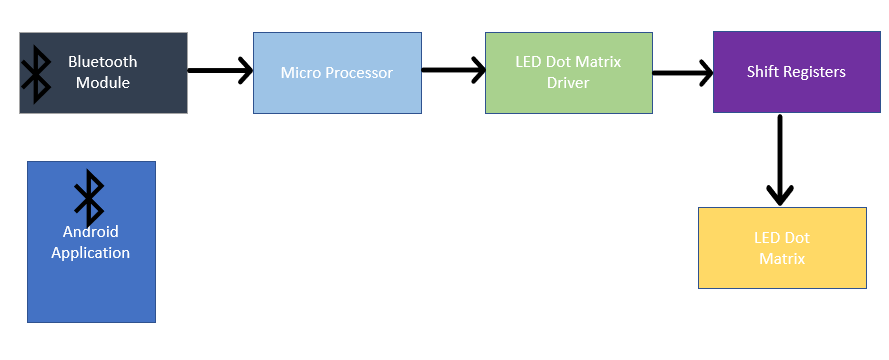


Figure 34 **Block Diagram of Android controlled message display**

## **4.2.2 Circuit Diagram**

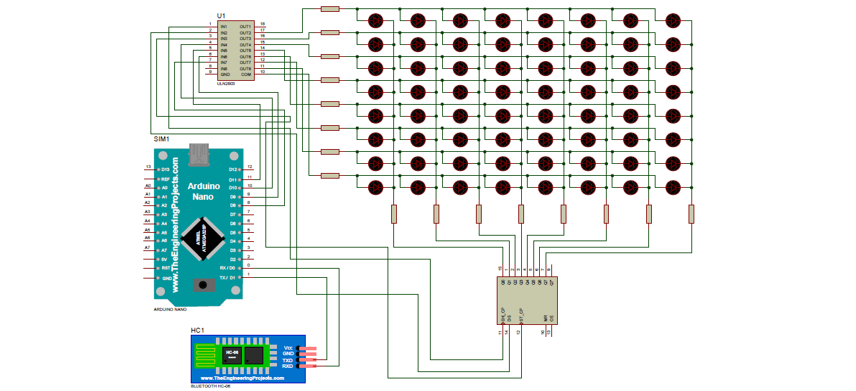
The circuit diagram of mobile application controlled scrolling display as shown as figure

Figure 35 **Schematic Circuit Diagram of Android controlled message display**

## **4.2.3 PCB Design**

The PCB design of mobile application controlled scrolling display as shown as figure

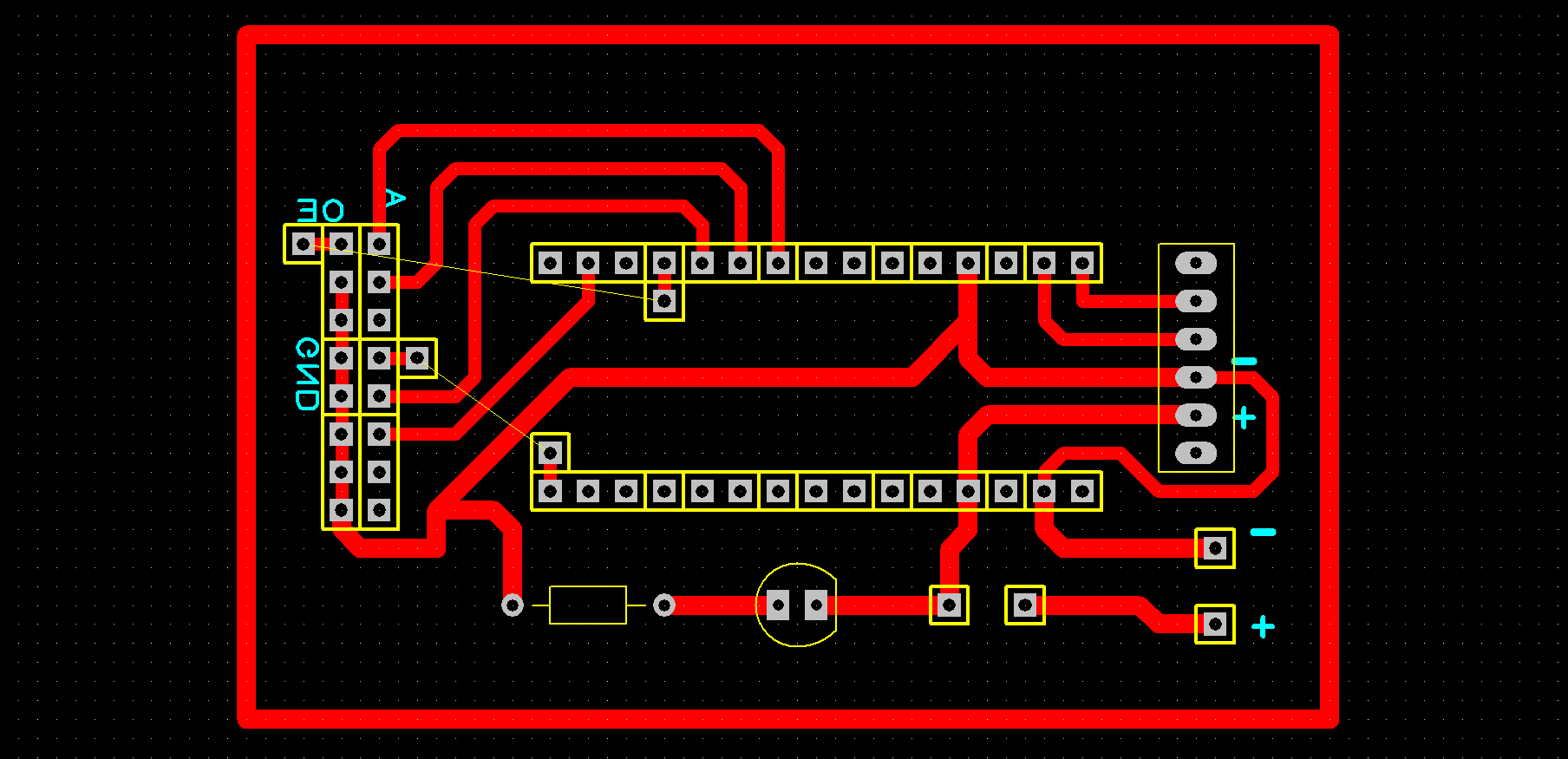


Figure 36  **PCB design of Android controlled message display**

## **4.3 Software Part**

## **4.3.1 Flowchart of the system**

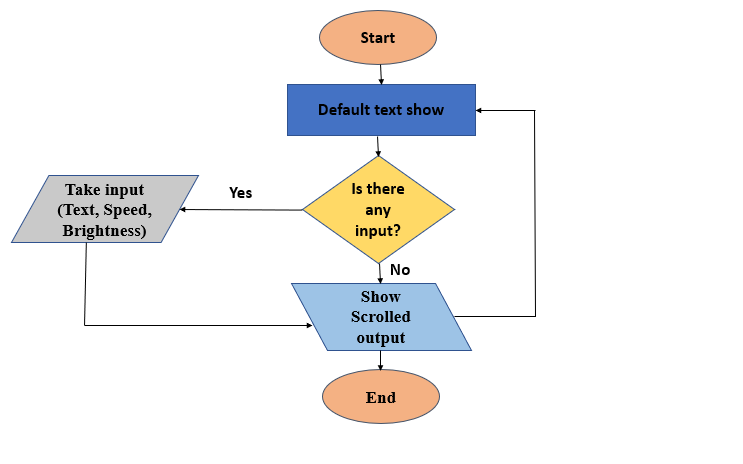


Figure 37 **Flowchart of Android controlled message display**

## **4.3.2 Arduino code**

#include <SPI.h> //Include all libraries

#include <DMD.h>

#include <TimerOne.h> //Timer

#include "SystemFont5x7.h"

#include "Arial\_black\_16.h"//front\_liblary

#define max\_char 1000 //Define how many characters we are going to show

#define DISPLAYS\_ACROSS 4 //Numbers of DMD-board(16x32) are using

#define DISPLAYS\_DOWN 1 //DMD up or down

DMD dmd(DISPLAYS\_ACROSS, DISPLAYS\_DOWN);//Display starting position

char message[max\_char];

char r\_char;

byte index = 0;

int i;

char greeting[] = "WELCOME TO ECRC - IST"; //Print message

void ScanDMD(){

dmd.scanDisplayBySPI(); // Scan From Bluetooth

}

void setup(void){

Timer1.initialize( 1000 );

Timer1.attachInterrupt( ScanDMD ); // Bluetooth scanner function called

dmd.clearScreen( true );//to clear RAM

Serial.begin(9600);//begin serial communication

strcpy(message,greeting);// copying greeting to message

//Serial.print(greeting);/////////

}

void loop(void){

Serial.print(message);// Console print

if(Serial.available()){

for(i=0; i<999; i++){

message[i] = '\0';

}

index=0;

}

while(Serial.available() > 0){

dmd.clearScreen( true );

if(index < (max\_char-1)){

r\_char = Serial.read();

message[index] = r\_char;

index++;

}

}

dmd.selectFont(Arial\_Black\_16);//Font

dmd.drawMarquee(message ,max\_char,(32\*DISPLAYS\_ACROSS)-1 ,0);// Matrix Size

long start=millis();

long timer=start;

boolean ret=false;

while(!ret)

{

if ((timer+30) < millis()) {

ret=dmd.stepMarquee(-1,0);

timer=millis();

}

}

}

## **4.4 Structure and Measurements**

We are using 16x128 LED dot matrix display which is used 25W electric power. Our display is 48” length,6” width and 3’’ thick.

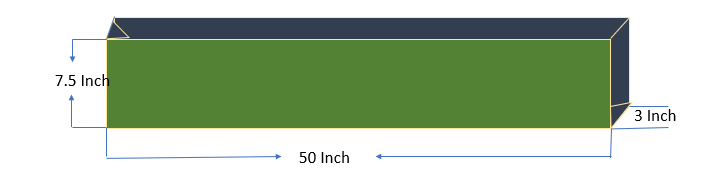


Figure 38 **Structure of message display board**

## **4.5 Implementation (Hardware)**



**Figure 39 Hardware implementation of message display board**

## **4.6 Implementation (Software)**



Figure 40 **Software implementation of message display board**

## **4.7 Code generation for character display**

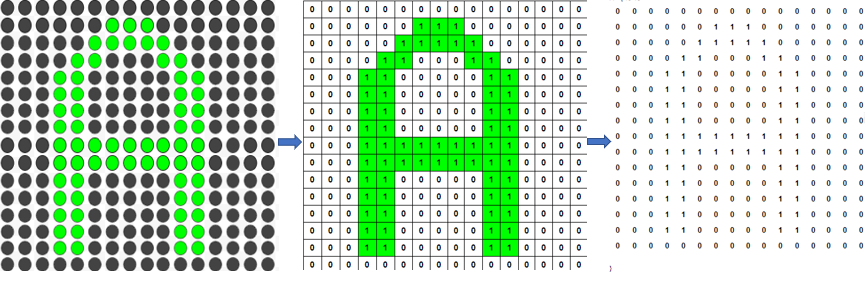


Figure41 **Code generation for character display**

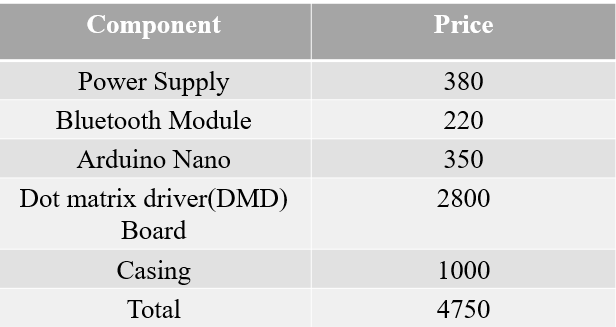
# **Chapter 5 Discussion and Conclusion**

Our scrolling display is doing well. In future, it will be developed for commercial and industrial uses.

We have successfully completed our project and it’s running quite well.

## **5.1 Cost Analysis**

**Table 5.1 cost analysis of a message display**



## **5.2 Advantages**

* Cost efficiency.
* Low power consumption.
* Miniaturization.
* High performance and integration.
* Functionality can be integration on the same silicon or in the same package, which reduces the component count.
* This contributes to overall cost saving.

## **5.3 Disadvantages**

* Farm establishment requires huge investment.
* Micro components are costly compared to macro component.
* Design includes very much complex procedures.

## **5.4 Applications**

* **Educational Institution and Organization:** Currently we rely on putting up papers on notice boards to inform people of events.
* **Railway Station:** Instead of announcing the delay in arrival of trains we can display the information.
* **Hotels:** To display the availability of the rooms and the room rents the type of rooms.
* **Nursing homes**: To display the staff attendance, the availability of the doctors, the list of the specialized doctors, no of in patients etc.

# REFERENCES

[1] A.Göransson, D.C.Ruiz ; “Professional Android™ Open Accessory Programming with Arduino™”, John Wiley & Sons, Inc., Indianapolis, Indiana, 2013, ISBN: 978-1-118-45476-3

[2] L.D.Jasio, T.Wilmshurst, D.Ibrahim, J.Morton, M.P.Bates, J.Smith, D.W.Smith, C.Hellebuyck; “PIC Microcontrollers”, Elsevier Inc., 2008, ISBN: 978-0-7506-8615-0

[3] M.R.McRoberts; “A Complete Beginners Guide to the Arduino”, Earthshine Design, 2009 [4] <http://tronixstuff.com/2013/10/11/tutorial-arduino-max7219-led-displaydriver-ic/>  -

[5] G. B. Stringfellow and M. G. Craford, “High brightness light emitting diodes,” in Semiconductors and Semimetals. New York: Academic, 1997, vol. 48.

[6] M. G. Craford, “LEDs challenge the incandescents,” IEEE Circuits Devices Mag., vol. 8, pp. 24–29, Sept. 1992.

[7] K. Werner, “Higher visibility for LEDs,” IEEE Spectr., vol. vol. 31, pp. 30–39, July 1994. [4] J. O’Connell, “The Philadelphia story,” Traffic Technol. Int., pp. 106–110, Aug./Sept. 1997.

[8] P. P. Smyth et al., “Optical wireless—A prognosis,” Proc. Wireless Data Transmission, vol. 2601, pp. 212–225, Oct. 23–25, 1995.

[9] G. Pang et al., “LED traffic lights as a communication device,” in Proc. Int. Conf. Intelligent Transportation Systems, Tokyo, Japan, Oct. 5–8, 1999, pp. 788–793.

[10] G. Pang et al., “Dual use of LEDs: Signaling and communications in ITS,” presented at the 5th World Congress on Intelligent Transport Systems, Seoul, Korea, Oct. 12–16, 1998,

[11] G.Pang et al., “Opticalwireless based on highbrightness visibleLEDs,” inConf.Rec.IEEE-IASAnnu.Meeting,Phoenix,AZ,Oct.3–7,1999,pp. 1693–1699.

[12] G. Pang et al., “Dual use of LEDs: Signaling and communications in ITS,” presented at the 5th World Congress on Intelligent Transport Systems, Seoul, Korea, Oct. 12–16, 1998, .

[13] G.Pang et al., “Opticalwireless based on highbrightness visibleLEDs,” inConf.Rec.IEEE-IASAnnu.Meeting,Phoenix,AZ,Oct.3–7,1999,pp. 1693–1699.  
[14] Neeraj Khera ,  Divya Shukla , Shambhavi Awasthi, “Development of simple and low cost Android based wireless notice board”, 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 7-9 Sept. 2016, Noida, India.

[15] Rajesh G.P et. al.,” Near field application: NFC smart notice board”, Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN), 21-23 July 2016, Hyderabad, India.

[16] Aniket Pramanik et. al.,“GSM based smart home and digital notice board”, International Conference on Computational Techniques in Information and Communication Technologies (ICCTICT), 11-13 March 2016, New Delhi, India.

[17] Sayidul Morsalin et. al., “Password protected multiuser wireless electronic noticing system by GSM with robust algorithm”, 2nd International Conference on Electrical Information and Communication Technologies (EICT), 10-12 Dec. 2015, Khulna, Bangladesh.

[18] Jianbo Zhang , Yin Qun “Design of the electronic bulletin board based on wireless sensor”, 3rd Information Technology and Mechatronics Engineering Conference (ITOEC), 3-5 Oct. 2017, Chongqing, China.

[19] Dejan Nedelkovski , “Arduino Wireless Communication – NRF24L01 Tutorial”, [Online] Available: <https://howtomechatronics.com/> tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/ (Accessed: March 2018).

[20] Justin Rajewski, “Serial Peripheral Interface (SPI)”, [Online]. Available at:<https://embeddedmicro.com/blogs/tutorials/serial-peripheral-interfacespi> [Accssed: February 11, 2018].

[21] Nordic Semiconductor, “Single chip 2.4 GHz transceiver”, NRF24L01 datasheet, September 2008 (Version 1.0).

[22] M.A. Kader, Md. Mahbubur Rahman , S.M Imdad Bin Haider , Mojaharul Islam, “LED matrix based digital learning display for children with wireless control”, 17th International Conference on Computer and Information Technology (ICCIT), 22-23 Dec. 2014. DOI:10.1109/ICCITechn.2014.7073134

# **Appendix**

**Mobile Application Source code:**

AndroidManifest:

<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"

package="subrota.shuvro.sendstringandroidaplication">

<uses-permission android:name="android.permission.BLUETOOTH" />

<uses-permission android:name="android.permission.BLUETOOTH\_ADMIN" />

<application

android:allowBackup="true"

android:icon="@mipmap/ic\_launcher"

android:label="@string/app\_name"

android:roundIcon="@mipmap/ic\_launcher\_round"

android:supportsRtl="true"

android:theme="@style/Theme.SendStringAndroidAplication">

<activity android:name=".ControlBoard"></activity>

<activity android:name=".SplashScreen">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

<activity android:name=".MainActivity"></activity>

</application>

</manifest>

**Splash screen class:**

package subrota.shuvro.sendstringandroidaplication;

import androidx.appcompat.app.AppCompatActivity;

import android.app.ProgressDialog;

import android.content.Intent;

import android.os.Bundle;

import android.os.Handler;

public class SplashScreen extends AppCompatActivity {

private ProgressDialog pd;

private final int SPLASH\_DISPLAY\_LENGTH = 1000;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_splash\_screen);

pd = new ProgressDialog(SplashScreen.this);

Intent intent = getIntent();

Bundle bundle = intent.getExtras();

if (bundle != null){

String message =(String) bundle.get("msg");

progressShow(message);

}

new Handler().postDelayed(new Runnable(){

@Override

public void run() {

/\* Create an Intent that will start the Menu-Activity. \*/

Intent mainIntent = new Intent(SplashScreen.this, MainActivity.class);

SplashScreen.this.startActivity(mainIntent);

SplashScreen.this.finish();

}

}, SPLASH\_DISPLAY\_LENGTH);

}

private void progressShow(String message){

pd.setMessage(message);

pd.show();

}

}

**Main Activity Class:**

package subrota.shuvro.sendstringandroidaplication;

import androidx.appcompat.app.AppCompatActivity;

import androidx.recyclerview.widget.LinearLayoutManager;

import androidx.recyclerview.widget.RecyclerView;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.content.Context;

import android.content.Intent;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.TextView;

import android.widget.Toast;

import java.lang.reflect.Array;

import java.util.ArrayList;

import java.util.List;

import java.util.Set;

public class MainActivity extends AppCompatActivity {

private BluetoothAdapter bluetoothAdapter;

private static final int REQUEST\_ENABLE\_BT = 1;

private View rootView;

private Helper helper;

private List<DeviceInfoDataSet> devices = new ArrayList<>();

private String[] array = new String[50];

private Set<BluetoothDevice> pairedDevices;

//private ListView listView;

private RecyclerView recyclerView;

private LinearLayoutManager linearLayoutManager;

private DeviceAdapter adapter;

private static final String TAG = "MainActivity";

private TextView refresh;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

rootView = getWindow().getDecorView().getRootView();

pairedDevices = bluetoothAdapter.getBondedDevices();

//listView = findViewById(R.id.lv\_paired\_devices);

recyclerView = findViewById(R.id.paired\_devices);

linearLayoutManager = new LinearLayoutManager(this);

refresh = findViewById(R.id.screenRefresh);

if (bluetoothAdapter == null) {

// Device doesn't support Bluetooth

helper.showSnackBar(rootView.getRootView(), "Device doesn't support Bluetooth");

}

getDevices();

enableBluetooth();

setAdapter();

}

public void showDevices(){

}

///////////////////////get paird devices/////////////////

public void getDevices() {

if (pairedDevices.size() > 0) {

// Loop through paired devices

for (BluetoothDevice device : pairedDevices) {

DeviceInfoDataSet deviceInfoDataSet = new DeviceInfoDataSet(device.getName(), device.getAddress());

Log.i(TAG, "device name: "+ device.getName());

devices.add(deviceInfoDataSet);

}

}

}

////////////////click event for acpt/ cancel bluetooth enable

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode == REQUEST\_ENABLE\_BT && resultCode == RESULT\_CANCELED) {

Toast.makeText(this, "please turn on your bluetooth", Toast.LENGTH\_SHORT).show();

//helper.showSnackBar(rootView, "please turn on your bluetooth");

enableBluetooth();

} else {

//Toast.makeText(this, "Bluetooth Turned on", Toast.LENGTH\_LONG).show();

helper.showSnackBar(rootView.getRootView(), "Bluetooth Turned on");

getDevices();

setAdapter();

}

}

////////////check bluetooth enable or not/////////////////////////

public void enableBluetooth() {

if (!bluetoothAdapter.isEnabled()) {

Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);

}else {

getDevices();

setAdapter();

}

}

public void setAdapter(){

if (devices.size()>0){

recyclerView.setLayoutManager(linearLayoutManager);

adapter = new DeviceAdapter(devices, this);

recyclerView.setAdapter(adapter);

}else {

recyclerView.setVisibility(View.GONE);

refresh.setVisibility(View.VISIBLE);

}

}

public void refresh(View view) {

Intent intent = new Intent(this, SplashScreen.class);

intent.putExtra("msg", "Restarting....");

startActivity(intent);

}

}

**Control Board Class:**

package subrota.shuvro.sendstringandroidaplication;

import androidx.appcompat.app.AppCompatActivity;

import androidx.recyclerview.widget.LinearLayoutManager;

import androidx.recyclerview.widget.RecyclerView;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothDevice;

import android.content.Context;

import android.content.Intent;

import android.os.Bundle;

import android.util.Log;

import android.view.View;

import android.widget.ArrayAdapter;

import android.widget.ListView;

import android.widget.TextView;

import android.widget.Toast;

import java.lang.reflect.Array;

import java.util.ArrayList;

import java.util.List;

import java.util.Set;

public class MainActivity extends AppCompatActivity {

private BluetoothAdapter bluetoothAdapter;

private static final int REQUEST\_ENABLE\_BT = 1;

private View rootView;

private Helper helper;

private List<DeviceInfoDataSet> devices = new ArrayList<>();

private String[] array = new String[50];

private Set<BluetoothDevice> pairedDevices;

//private ListView listView;

private RecyclerView recyclerView;

private LinearLayoutManager linearLayoutManager;

private DeviceAdapter adapter;

private static final String TAG = "MainActivity";

private TextView refresh;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

rootView = getWindow().getDecorView().getRootView();

pairedDevices = bluetoothAdapter.getBondedDevices();

//listView = findViewById(R.id.lv\_paired\_devices);

recyclerView = findViewById(R.id.paired\_devices);

linearLayoutManager = new LinearLayoutManager(this);

refresh = findViewById(R.id.screenRefresh);

if (bluetoothAdapter == null) {

// Device doesn't support Bluetooth

helper.showSnackBar(rootView.getRootView(), "Device doesn't support Bluetooth");

}

getDevices();

enableBluetooth();

setAdapter();

}

public void showDevices(){

}

///////////////////////get paird devices/////////////////

public void getDevices() {

if (pairedDevices.size() > 0) {

// Loop through paired devices

for (BluetoothDevice device : pairedDevices) {

DeviceInfoDataSet deviceInfoDataSet = new DeviceInfoDataSet(device.getName(), device.getAddress());

Log.i(TAG, "device name: "+ device.getName());

devices.add(deviceInfoDataSet);

}

}

}

////////////////click event for acpt/ cancel bluetooth enable

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode == REQUEST\_ENABLE\_BT && resultCode == RESULT\_CANCELED) {

Toast.makeText(this, "please turn on your bluetooth", Toast.LENGTH\_SHORT).show();

//helper.showSnackBar(rootView, "please turn on your bluetooth");

enableBluetooth();

} else {

//Toast.makeText(this, "Bluetooth Turned on", Toast.LENGTH\_LONG).show();

helper.showSnackBar(rootView.getRootView(), "Bluetooth Turned on");

getDevices();

setAdapter();

}

}

////////////check bluetooth enable or not/////////////////////////

public void enableBluetooth() {

if (!bluetoothAdapter.isEnabled()) {

Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(enableBtIntent, REQUEST\_ENABLE\_BT);

}else {

getDevices();

setAdapter();

}

}

public void setAdapter(){

if (devices.size()>0){

recyclerView.setLayoutManager(linearLayoutManager);

adapter = new DeviceAdapter(devices, this);

recyclerView.setAdapter(adapter);

}else {

recyclerView.setVisibility(View.GONE);

refresh.setVisibility(View.VISIBLE);

}

}

public void refresh(View view) {

Intent intent = new Intent(this, SplashScreen.class);

intent.putExtra("msg", "Restarting....");

startActivity(intent);

}

}

Device Adapter Class:

package subrota.shuvro.sendstringandroidaplication;

import android.content.Context;

import android.content.Intent;

import android.view.LayoutInflater;

import android.view.View;

import android.view.ViewGroup;

import android.widget.LinearLayout;

import android.widget.TextView;

import androidx.annotation.NonNull;

import androidx.recyclerview.widget.RecyclerView;

import java.util.ArrayList;

import java.util.List;

public class DeviceAdapter extends RecyclerView.Adapter<DeviceAdapter.ViewHolder>{

private List<DeviceInfoDataSet> devices = new ArrayList<>();

private Context context;

public DeviceAdapter(List<DeviceInfoDataSet> devices, Context context) {

this.devices = devices;

this.context = context;

}

@NonNull

@Override

public ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

View view = LayoutInflater.from(context).inflate(R.layout.device\_row, parent, false);

return new ViewHolder(view);

}

@Override

public void onBindViewHolder(@NonNull ViewHolder holder, int position) {

holder.name.setText(devices.get(position).getName());

holder.id.setText(devices.get(position).getId());

holder.deviceRow.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

Intent intent = new Intent(v.getContext(), ControlBoard.class);

intent.putExtra("id", devices.get(position).getId());

intent.putExtra("name", devices.get(position).getName());

context.startActivity(intent);

}

});

}

@Override

public int getItemCount() {

return devices.size();

}

public class ViewHolder extends RecyclerView.ViewHolder {

TextView name, id;

LinearLayout deviceRow;

public ViewHolder(@NonNull View itemView) {

super(itemView);

name = itemView.findViewById(R.id.device\_name);

id = itemView.findViewById(R.id.device\_id);

deviceRow = itemView.findViewById(R.id.deviceRow);

}

}

}

**Device Info Dataset Class:**

package subrota.shuvro.sendstringandroidaplication;

public class DeviceInfoDataSet {

private String name;

private String id;

public DeviceInfoDataSet(String name, String id) {

this.name = name;

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

Helper class:

package subrota.shuvro.sendstringandroidaplication;

import android.app.Activity;

import android.graphics.Color;

import android.view.View;

import android.widget.TextView;

import androidx.core.content.ContextCompat;

import com.google.android.material.snackbar.Snackbar;

public class Helper {

public static void showSnackBar(View v, String message) {

Snackbar.make(v, message, Snackbar.LENGTH\_SHORT)

.setAction("Action", null)

.show();

}

}

**Layout Files:**

**Splash\_screen.xml**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:gravity="center"

android:background="@drawable/splash"

tools:context=".SplashScreen">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:gravity="center"

android:textSize="24dp"

android:textColor="#ffffff"

android:text="Welcome"/>

</LinearLayout>

main activity xml:

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

tools:context=".MainActivity">

<androidx.recyclerview.widget.RecyclerView

android:visibility="visible"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:id="@+id/paired\_devices"

android:layout\_marginTop="10dp"

android:layout\_marginStart="15dp"

android:layout\_marginEnd="15dp"/>

<TextView

android:id="@+id/screenRefresh"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:gravity="center"

android:visibility="gone"

android:text="Refresh"

android:textColor="@color/black"

android:textSize="20dp"

android:layout\_gravity="center"

android:padding="20dp"

android:onClick="refresh"/>

</LinearLayout>

**Control\_board.xml:**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

xmlns:tools="http://schemas.android.com/tools"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

tools:context=".ControlBoard">

<androidx.cardview.widget.CardView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:elevation="5dp">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:text="Device Name"

android:textColor="@color/black"

android:textSize="18dp"

android:textStyle="bold"

android:padding="20dp"

android:maxLines="1"

android:id="@+id/controlBoardTitle"

android:ellipsize="end"/>

</androidx.cardview.widget.CardView>

<ScrollView

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical">

<TextView

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Message"

android:layout\_marginStart="15dp"

android:padding="3dp"

android:textColor="@color/black"

android:textSize="18dp"

android:textStyle="bold"

android:layout\_marginTop="10dp" />

<EditText

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:hint="message....."

android:ellipsize="start"

android:textSize="14dp"

android:textColor="@color/black"

android:gravity="top|left"

android:inputType="text|textMultiLine"

android:lines="10"

android:minLines="5"

android:scrollHorizontally="false"

android:scrollbars="vertical"

android:background="#CBCFD1"

android:layout\_margin="15dp"

android:padding="3dp"

android:id="@+id/controlBoardMessage"/>

<LinearLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:layout\_gravity="center"

android:gravity="center"

android:layout\_marginTop="5dp"

android:layout\_marginBottom="30dp"

android:orientation="vertical">

<Button

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Send"

android:background="@drawable/button\_shap"

android:textColor="@color/white"

android:textSize="20dp"

android:textStyle="bold"

android:padding="5dp"

android:gravity="center"

android:onClick="send"/>

</LinearLayout>

</LinearLayout>

</ScrollView>

</LinearLayout>

**Device\_row.xml:**

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

xmlns:tools="http://schemas.android.com/tools"

android:background="@drawable/card\_shape"

android:layout\_marginBottom="10dp"

android:id="@+id/deviceRow"

android:orientation="vertical">

<TextView

android:id="@+id/device\_name"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

tools:text="name"

android:textSize="16dp"

android:textColor="@color/black"

android:layout\_marginStart="15dp"

android:layout\_marginTop="5dp"

android:layout\_marginBottom="5dp"

android:layout\_marginEnd="15dp"/>

<TextView

android:id="@+id/device\_id"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

tools:text="name"

android:textSize="12dp"

android:textColor="@color/black"

android:layout\_marginStart="15dp"

android:layout\_marginBottom="5dp"

android:layout\_marginEnd="15dp"/>

</LinearLayout>