**General idea about the project**

In view of the efforts of the countries of the world to help people with special needs and endeavor to support them in society in the best possible ways and facilitate their interaction with society and support it by all available means and starting from that the first goal of engineering sciences is to facilitate people's lives and achieve the welfare of societies, we offer this work to support those with special needs From deaf and dumb by facilitating their communication with society through a glove, it converts the sign language into a verbal language that is easy for all people to understand so that deaf and dumb can deal with all people without an obstacle of language that not all people can master but may be difficult for many of People to learn it. Therefore, and as a contribution from us in this field, we offer this work, which we hope will be a real beginning to merge people with special needs from the deaf and dumb completely into society without any obstacles.

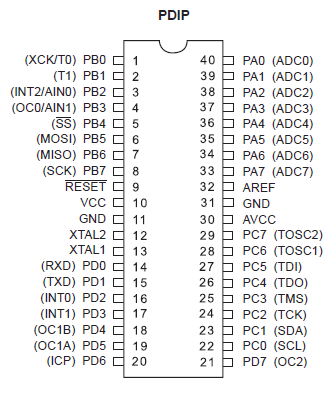
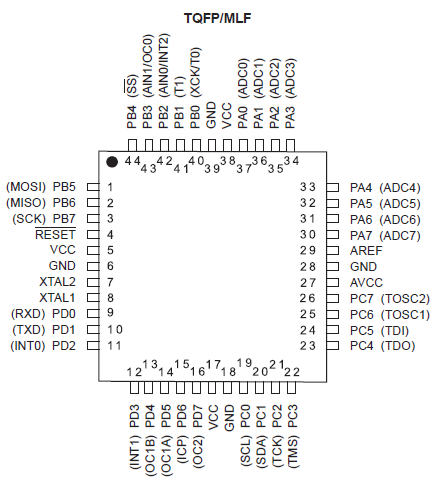
**Hardware Components**

* ATmega32 Micro-controller
* Flex Sensor
* HC05 Bluetooth Module
* Resistors
* Capacitors
* Regulator(5V)
* Battery
* Crystal Oscillator
* Glove

**ATmega32**

ATmega32 is a low-power CMOS 8-bit micro-controller based on the AVR enhanced RISC architecture.

**Pin Configurations**

**A Brief History of the AVR micro-controller**

The basic architecture of AVR was designed by two students of Norwegian Institute of Technology (NTH), Alf, Vegard, and then Was bought and developed by at Atmel in 1996.

You may ask what AVR stands for**;** AVR can have different meanings for different people! Atmel says that it is nothing more than a product name but it might stand for advanced virtual RISK, or Alf and Vegard RISK (the names of the AVR designers).

There are many kinds of AVR micro-controller with different properties, Except for AVR32, which is a 32-bit micro-controller, AVRs or all 8-bit micro-controller, meaning that the CPU can work on only 8 bits of data at a time. Data larger than 8bits has to be broken into 8bit pieces to be processed by the CPU.

One of the problems with the AVR micro-controllers is that they are not all 100% compatible in terms of software when going from one family to another family. to run programs written for ATtiny25 on an ATmega64, we must recompile the program and possibly change some register locations before loading it into the ATmega64. AVRs are generally classified into four broad groups: Mega, Tiny, special purpose, and classic.

**Micro-controller versus general purpose Microprocessor**

By Microprocessor is meant the General-Purpose Microprocessors such

as Intel's X86 Family (8086, 80286, 80386, ...) Or Motorola's PC Family.

\* These-Microprocessor Contain no Ram, no Rom, And no (I/O) Ports on The Chip Itself. For This-reason They are Commonly Referred to As General-Purpose Microprocessors.

\*A System Designer Using A General-Purpose Microprocessor Such as The Pentium Or the Power PC Must Add RAM, Rom, I/O Ports, And Timers Externally to Make Them Functional.

\*Although the Addition of External. Ram, Rom and(I/O) Ports Makes These Systems bulkier and Much More Expensive Advantage of Versatility, Enabling the Designer To-Decided on The Amount of Ram, Rom, and (I/O) Ports Needed to Fit the Task at Hand.

\*A Micro-controller Has A CPU (Microprocessor) In Addition to A Fixed Amount Of RAM, ROM, (I/O) Ports and Timer All on a Single Chip There are the Designer Cannot Add Any External Memory (I/O), Or Timer to It.

\* For Embedded Systems: Typically, The Micro-controller's ROM Is Burned with A Purpose for Specific Functions Needed for The System A Printer is an Example of An Embedded System Because the Processor Inside It Performs One Task Only.

\*A Pentium based PC (or any X86 PC) can be used for any number of applications such as word processor server, print server, bank teller terminal, video game player, network server or internet terminal. A PC can load and run software for a variety of applications.

\*The reason a PC can perform my tasks is that it has RAM memory and an operating system that loads the application software into Ram and lets the CPU run it.

\* X86 PC contains or is connected to various products such as the keyboard, printer, modem, disk controller, sound card, CD Rom driver, Mouse, and soon each one of these peripherals has a micro-controller inside it that performs only one task.

**Choosing a Micro-controller:**

There are five major 8-bit micro-controller they are**:** free scale semiconductors (formerly Motorola )68MC081 68 HC 11 Intel's 8051, Atmel's AVR, Zi log's Z8 and pic from microchip technology each of the above micro-controllers has a unique instruction set and register set.

They are not compatible with each other. programs written for one will not run on the others.

**Criteria for choosing a micro-controller:**

**1-** It must meet the task at hand efficiently and cost effectively. In analyzing the needs of a micro-controller-based project, we must first see Whether an 8-bit, 16-bit, or 32-bit micro-controller can best handle the computing needs of the task most effectively. Among other considerations in this category are**:**

(a) Speed what is the highest speed that the micro-controller supports?

(b) Packaging Does it come in a DIP (duel inline package) or a QFP (quad flat package) this is important in terms of space, assembling, and prototyping the end product.

(c) Power consumption is especially critical for battery powered products

(d) The amount of Ram and Rom on the chip.

(e) The number of I/O pins and the Timer on the chip.

(f) Ease of upgrade to higher- performance or lower- power- consumption versions.

(g) cost per unit. this is important in terms of the final cost of the product.

**2-** How easy it is to develop products around it. key considerations include the availability of an assembler, a debugger, a code-efficient c language compiler, an emulator, technical support, and both in-house and outside expertise.

**3-** It's ready availability in needed quantities both now and in the future.

**AVR Features**

**(1)** High-performance, low power AVR 8-bit micro-controller

8-bit**:** meaning that the CPU can work on only 8-bit of data at time.

Data larger than 8-bits has to be broken into 8-bit pieces to be processed

by the CPU.

**(2) ATmega32** is powerful, widely available, and comes in DIP packages.

**(3)** Speed grades**:** from 0 to 16 MHZ.

**(4)** Power consumption at 1 MHz, 3V, 25C

**-** Active**:** 1.1 mA

**-** Ideal mode**:** 0.35 mA

**-** Power down mode**:** <1mA.

**(5)** 32K code ROM, 2K Data RAM, and 1K EEPROM.

**(6)** 32 I/O Pins and 8 ADC, and 3 timers and endurance: 1000write/Erase cycles.

**(7)** The AVRwas one of the first micro-controllers to use on-chip Flashmemory for program storage. The Flash memory is ideal for fast development because flash memory can be erased in seconds compared to 20 minutes or more needed for the UV-EPROM.

**(8)** The cost is 55 EGP.

**(9)** Advanced RISC Architecture (reduced instruction set computer).

**Atmega32 ADC Features:**

The ADC peripheral of the ATmega32 has the following characteristics:

**(a)** It is a 10-bit ADC.

**(b)** It has 8 analog input channels, 7 differential input channels, and 2 differential input channels with the optional gain of 10x and 200x.

**(c)** The converted with binary data is held by two special function registerscalled ADCL (A/D Result Low) and ADCH (A/D Result High).

**(d)** Because the ADCH: ADCL registers give us 16 bits and the ADC data out is only 10 bits wide, 6 bits of the 16 are unused. we have the option of Making either the upper 6 bits or the lower 6 bits unused.

**(e)** We have three options for . can be connected to AVCC

(Analog ), internal 2.56V reference, or external AREF pin.

**(f)** The conversion time is dictated by the Crystal frequency connected to the XTAL pins (FOSC) and ADPS0:2 bits.

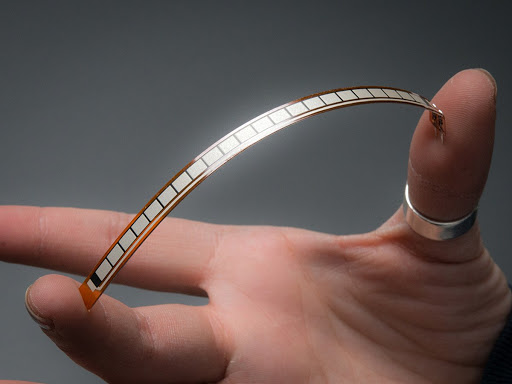
**FLEX SENSOR**

**Pin Configuration**

Flex sensor is a two-terminal device. The Flex sensor does not have polarized terminals like diode. So there is no positive and negative.

|  |  |
| --- | --- |
| **Pin Number** | **Description** |
| P1 | Usually connected to positive of power source. |
| P2 | Usually connected to ground. |

A simple flex sensor 4.5" in length. As the sensor is flexed, the resistance across the sensor increases. Patented technology by Spectra Symbol - they claim these sensors were used in the original [Nintendo Power Glove](http://en.wikipedia.org/wiki/Power_Glove" \t "_blank" \o "Nintendo power glove).



**FLEX SENSOR Features and Specifications**

* Operating  voltage of FLEX SENSOR: 0-5 V
* Can operate on LOW voltages
* Power rating: 0.5Watt (continuous), 1 Watt (peak)
* Life: 1 million
* Operating temperature: -45ºC to +80ºC
* Flat Resistance: 25K Ω
* Resistance Tolerance: ±30%
* Bend Resistance Range: 45K to 125K Ohms (depending on bend)

**ADC**

**ADC Device:**

* Analog \_ to\_ digital converters are among the most widely used devices for data acquisition.
* Digital computers use binary (discrete) values, but in the physical world everything is analog (continuous). temperature, pressure (wind or liquid), humidity, and velocity are a few examples of physical quantities that we deal with every day. A physical quantity is converted to electrical (voltage, current) signals using a device called a transducer.
* Transducer are also referred to as sensors. Sensors for temperature, velocity pressure, light, and many other natural quantities produce an output that is voltage (or current). therefore, we need an analog \_ to \_ digital converter to translate the analog signals to digital numbers so that the microcontroller can read and process them.

Display

ADCCC

Sensor

CPU

**Some of the major characteristics of the ADC**:

**Resolution:**

-The ADC has n \_bit resolution, where n can be 8,10,12,16, or even 24bits Higher \_ resolution ADCs provide a smaller step size.

-Step size is the smallest change that can be discerned by an ADC.

-Although the resolution of an ADC chip is decided at the time of its design, and can-not be changed, we can control the step size with the help of what is called .

-The voltage connected to this pin, analog with the resolution of the ADC chip, dictate the step size.

-For an 8\_ bit ADC, the step size is /256 because it is an 8\_bit ADC, and 2 to the power of 8 give us 256 steps.

-Resolution versus step size for ADC (= 5 V).

|  |  |  |
| --- | --- | --- |
| N\_ bit | Number of steps | Step size (mv) |
| 8 | 256 | 5/256 = 19.53 |
| 10 | 1024 | 5/1024 = 4.88 |
| 12 | 4096 | 5/4096 = 1.2 |
| 16 | 65,536 | 5/65,536 = 0.076 |

Note: = 5 v

-If the analog input range needs to be a to 4 Volt, is connected to 4 Volt

Step Size = /

Where n is number of bits

|  |  |  |
| --- | --- | --- |
| (V) | Vin Range (V) | Step Size (mv) |
| 5.00 | 0 to 5 | 5/256 = 19.53 |
| 4.0 | 0 to 4 | 4/256 = 15.62 |
| 3.0 | 0 to 3 | 3/256 = 11.71 |
| 2.56 | 0 to 2.56 | 2.56 / 256 = 10 |
| 2.0 | 0 to 2 | 2 / 256 = 7.81 |
| 1.28 | 0 to 1.28 | 1.28 / 256 = 5 |
| 1 | 0 to 1 | 1 / 256 = 3.90 |

Step Size is/ 256

-In some applications, we need the differential reference voltage where = (+) – (-) often the (-) pin is connected to ground and the (+) pin is used as the .

\* Conversion Time: Is major factor in judging an ADC.

- Is the time it takes the ADC to convert the analog input to a digital (binary) number.

- The conversion time is detected by the clock source connected to the ADC in addition to the method used for data conversion and technology used in the fabrication of the ADC chip such as MOS or TTL technology.

**Digital Data Output**:

-In an 8 \_ bit ADC we have an 8 \_ bit digital data output of D0 \_ D1, to calculate the output voltage, we use the following formula.

- = Vin / Step Size.

-is the digital data output (in decimal)

-Vin is the analog input voltage

-This data is brought out of the ADC chip either one bit at a time (serially), or in one chunk. using a parallel line of outputs.

**Types of ADC**:

**(1) Parallel ADC:**

-We have 8 or more pins dedicated to bringing out the binary data.

-The D0\_D7 data pins of the 8 \_ bit ADC provide an 8 \_ bit parallel data path between the ADC chip and the CPU

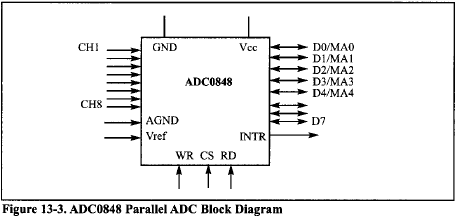
-In order to save pins, many 12 and 16 - bit ADCs use pins D0\_D7 to send

out the upper and lower bytes of the binary data.

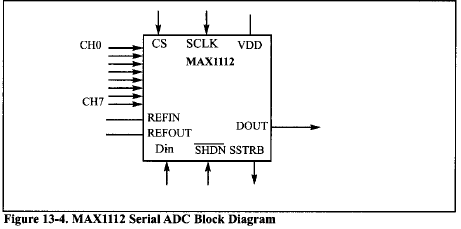
-Less time.

-One single read operation.

-ADC 0848 is an example of a parallel ADC with 8 \_ pins for the data output.



**(2) Serial ADC**:

* We have only one pin for data out, that means that inside the serial ADC, there is a parallel \_ in \_ serial \_out shift-register responsible for sending output the binary data one bit at a time.
* Widely used because it use-fewer pins their smaller packages take much less space on the printed circuit board.
* More CPU time is needed to get the converted data from the ADC because the CPU must get data one bit at a time
* MAX 1112 is an example of a serial ADC with a single pin for.
* 

***Method of converting****:*

Successive approximation ADC:

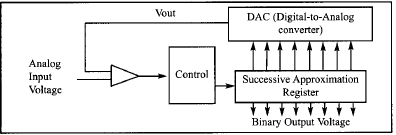
- It is a widely used method of converting an analog input to digital output.

**-It has three main components:**

**(1)** Successive approximation register (SAR).

**(2)** Comparator.

**(3)** Control unit.



Example:

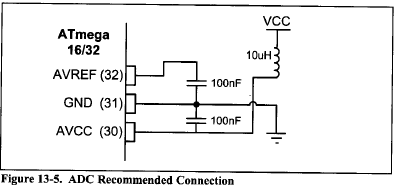
Assuming a step size of 10mV, the 80bit successive approximation ADC will go through the following steps to convert an input of 1 Volt:

1. It starts with binary 1000000.since 128\*10mV =1.28V is greater than the 1V input, bit 7 is cleared (dropped).
2. 01000000 gives us 64\*10mV =640mV and bit 6 kept since it is smaller than the 1V.
3. 01100000 gives us 96\*10mV=960mVand bit 5 is kept.
4. 01110000 gives us 112\*10mV =1120mV and bit 4 is dropped.
5. 01101000 gives us 108 \*10 mV = 1080mV bit 3 is dropped.
6. 01100100 gives us 100\*10mv =1000mv=1v and bit 3 is kept.
7. 011000110 gives us 102 \*10mv =1020mv and bit 1 is dropped.
8. 01100101 gives us 101\*10mv =1010mv and bit 0is dropped.

***ADC programming in the AVR:***

**AVR ADC hardware considerations:**

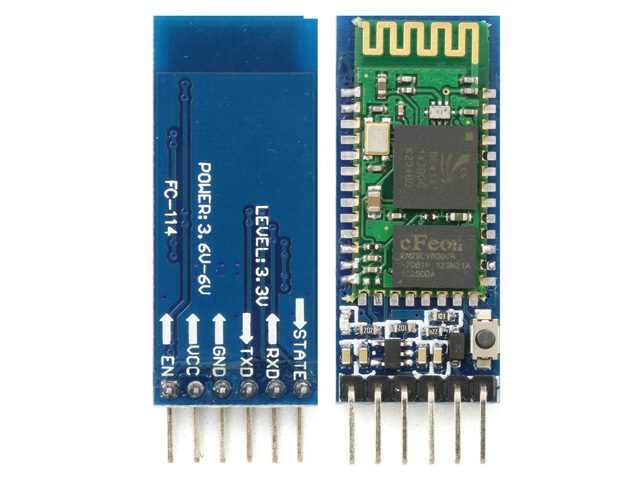
Decoupling AVCC from VCC: the AVCC pin provides the supply for analog ADC circuitry to get a better accuracy of AVR ADC we must provide a stable voltage source to the AVCC pin to achieve this, we provide ADC recommended connection.



**Connecting a capacitor between and GND.**

By connecting a capacitor between the AVREF pin and GND you can make the voltage more stable and increase the precision of ADC.

**HC-05 Bluetooth Module**

****

**Pin Configuration**

|  |  |  |
| --- | --- | --- |
| **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).  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-05”.

-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.

**HC-05 Technical Specifications:**

* Serial Bluetooth module for 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.

**Where to use HC-05 Bluetooth module:**

The **HC-05** 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-rate 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,

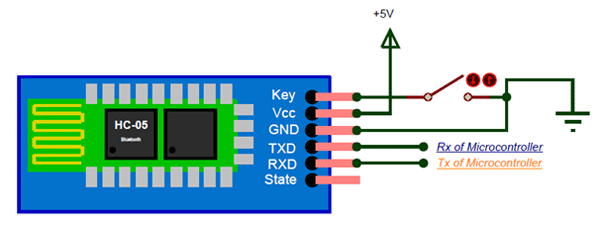
if you looking for a Wireless module that could transfer data from your computer or mobile phone to microcontroller or vice versa then this module might be the right choice for you.

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-05 Bluetooth module:**

The **HC-05** 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.



-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. The name password and other default parameters can be changed.

**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.

**USART**

**AVR Serial Port**

**Computers transfer data in two ways:**

**(1)** Parallel:

-Data transfers often eight or more lines (wire conductors) are used to transfer data to a device that is only a few feet away.

- Devices that use parallel transfers include printers and IDE hard disk.

**(2)** Serial:

- To transfer to a device located many meters away, the serial Method is used.

-The data is sent one bit at a time.

**Serial Communication Methods:**

**(1)** Synchronous: (Transfers a block of data characters at a time).

**(2)** A Synchronous: (Transfers a single byte at a time).

\*It's possible to write software to use either of these methods, But the programs can be tedious and long.

\*For this reason, Special IC chips are made by many manufacturers for serial data Communications.

\*These chips are commonly referred to as UART (universal asynchronous receiver - transmitter) and us UART (universal Synchronous – a Synchronous receiver -transmitter).

\* The AVR chip has a built-in us ART.

**Types of transmission:**

**(1)** **Simplex**: (Computer only sends data)

Receiver

Transmitter

**(2)** **Half duplex**: (Data is transmitted one way at a time)

Transmitter

Receiver

Receiver

Transmitter

**(3)**  **Full duplex**:(The data can go both ways at the same time)

-Full-duplex requires two wire conductors for the data lines

(In addition to the signal ground), one for transmission and one for reception.

Receiver

Transmitter

Receiver

Transmitter

**A synchronous serial communication and data Framing.**

-The data coming in a receiving end of the data line in a serial data transfer is all 0s and 1s.

-It's difficult to make sense of the data unless the sender and receiver agree on a set of rules, a protocol, on how the data is packed, how many bits constitute a character, and when the data begins and ends.

**Start and stop bits:**

- In a Synchronous method that each character is placed between start and stop bits, this is called the Framing.

-In data framing for asynchronous communications, the data such as ASCII characters are packed between a start bit and a stop bit.

- the star bit is always one bit, but the stop bit can be one or two bits.

- the start bit is always a 0 (low), and the stop bit is 1 (high).

**Data transfer rate:**

-The rate of data transfer in serial data communication is started in bps (bits per second) another widely used terminology for bps is Baud rate (the number of signal changes per second).

**Rx and Tx pins in the atmega32:**

-The atmega32 has two pins that are use specific for transmitting and receiving data serially.

- These two bins are called TX and RX and are part of the port D group

(PD0 and PD1).

**AVR serial port programming:**

-In the AVR microcontroller five registers are associated with the USART that we deal with in this project, They are:

**(1)** UDR (USART data register).

**(2)** UCSRA, UCSRB, UCSRC (USART control status register).

**(3)** UBRR (USART baud rate register).

**UBRR register and baud rate in the AVR:**

-It is used for programming of the baud rate in the AVR.

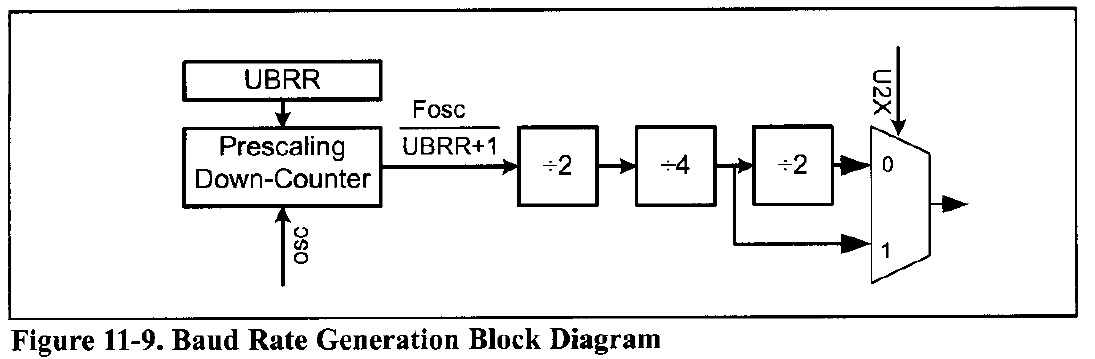
- for a given crystal frequency the value loaded into UBRR decides the baud rate.

- The relation between the value loaded into UBRR and the FOSC (Frequency of oscillator connected to XTAL 1 and EXTAL 2 pines)

Is dictated by the following Formula:

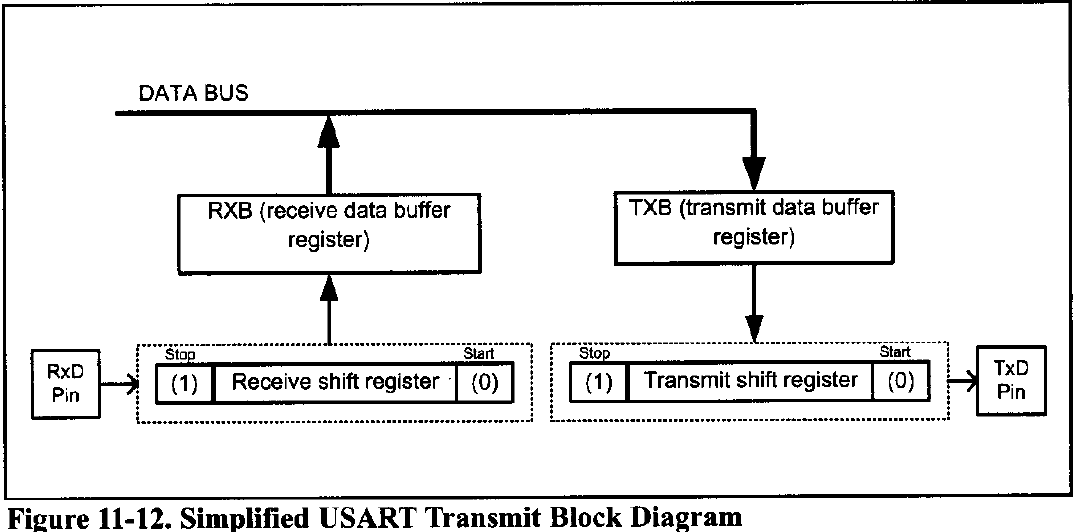
Desired Baud rate = FOSC / ( b(x+1) ).

X »» Is the value we load into the UBRR register



**UDR registers and USART data I/O in the AVR:**

The USART transmit data buffer register and USART receive data buffer register share the same I/O address, which is called USART data register or UDR. When you write data to UDR, it will be transferred to transmit data buffer register (TXB), And when you read data from UDR, it will return the contents of the receive data buffer register (RXB).

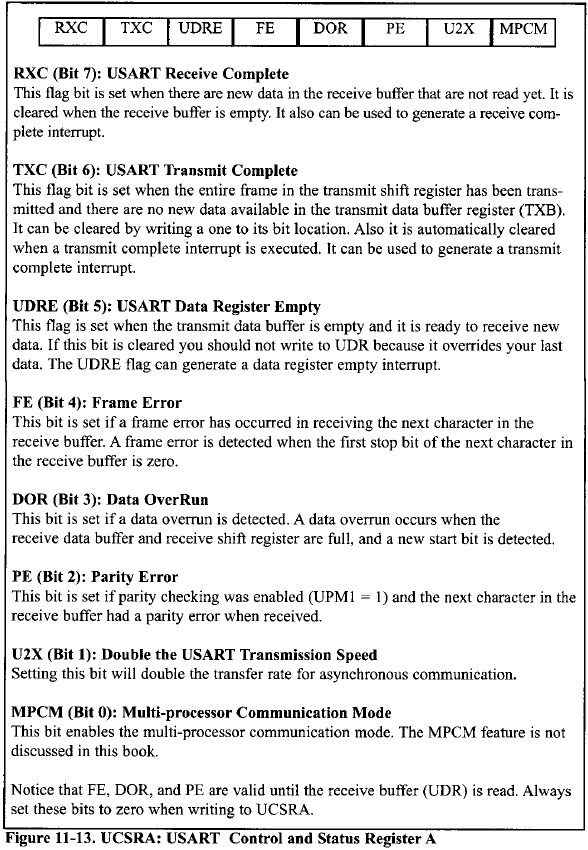


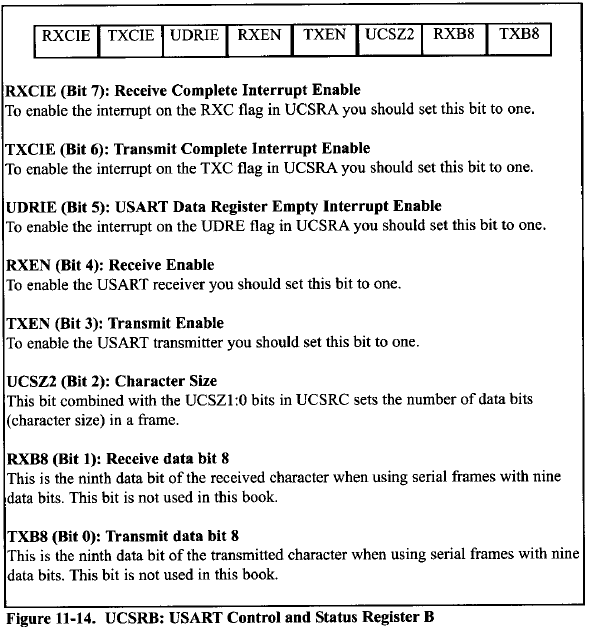
**UCSR Register and USART configurations in the AVR:**

-UCSRs are 8-bit control registers used for controlling serial communication in the AVR.

-There are three USART control status registers in the AVR.

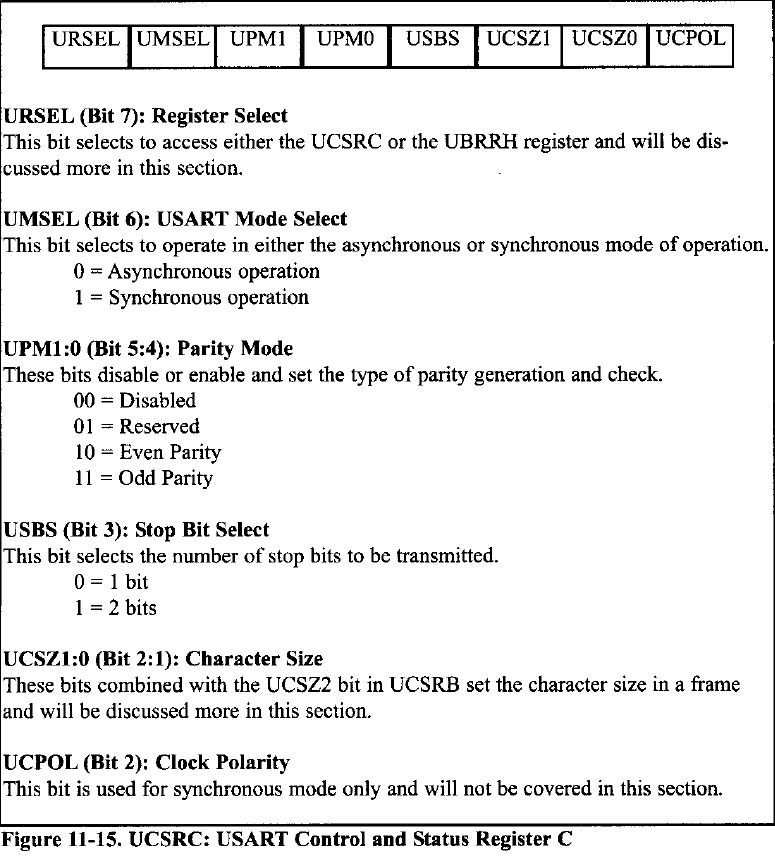
(UCSRA, UCSRB, UCSRC).



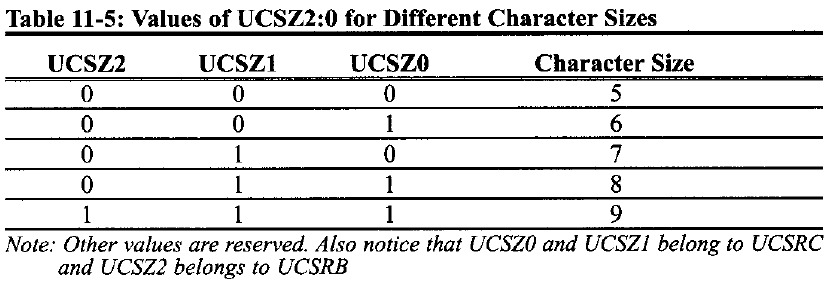


-Three of the UCSRB register bits are related to interrupt.

-We monitor (poll) UDRE flag bit to make sure, That the transmit data buffer is empty and it is ready to receive new data, By the same logic, we monitor the RXC flag to see if a byte of data has come in yet.



-To set the number of data bits (character size) in a frame you must set the value of the UCSZ1, UCSZ0 bits in the UCSRB and UCSZ2 bits in UCSRC.



**Programming the AVR to transfer data serially:**

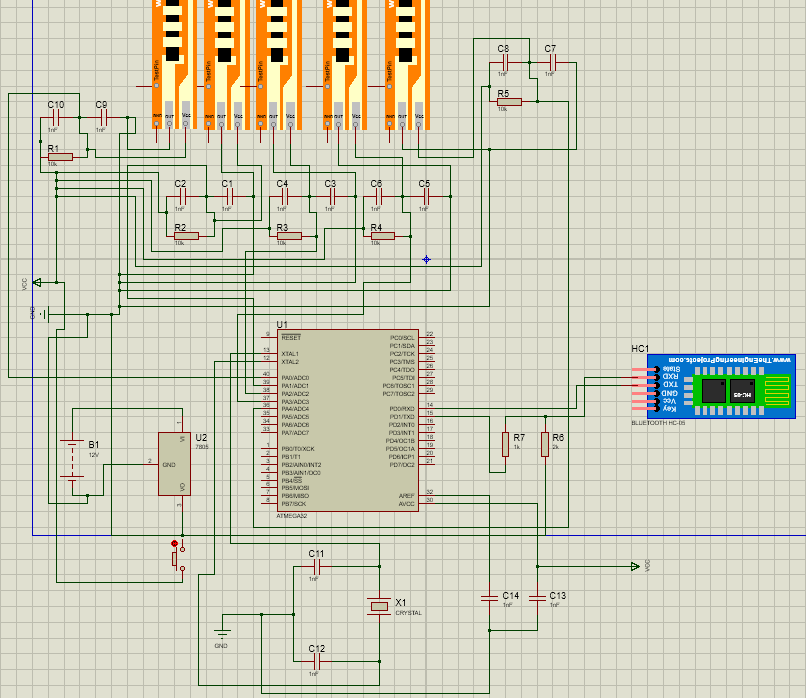
-In programming the AVR to transfer character bytes serially, the following steps must be taken:

1. The UCSRB Register is loaded with the value 08H, Enabling the USART transmitter. the transmitter will override normal port operation for the TXD pin when enabled.
2. The UCSRC register is loaded with the value 06 h, Indicating a synchronous with 8-bit data frame, no parity, and one stop bit.
3. The UBRR is loaded with certain value to set the baud rate for serial data transfer.
4. The character byte to be transmitted serially is written into the UDR register.
5. Monitor the UDRE bit of the UCSRA register to make sure UDR is ready for the next byte.

**(6)** To transmit the next to character, go to step4.

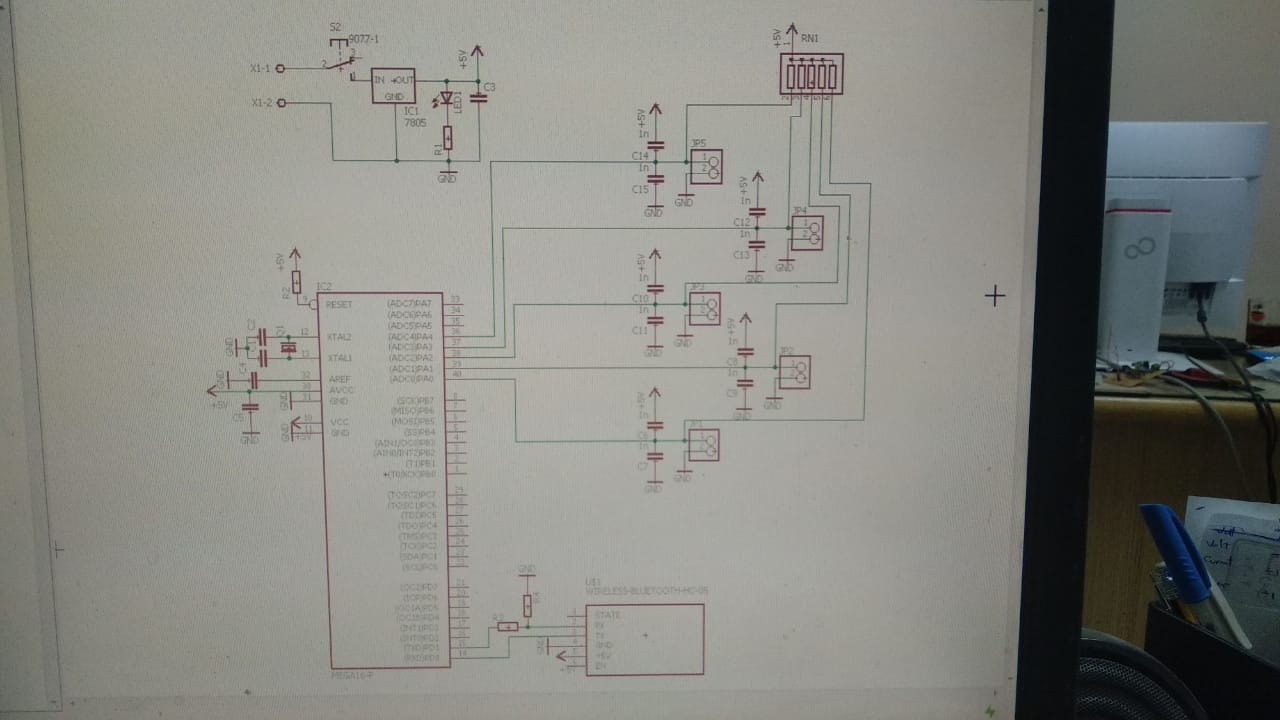
**Connections**

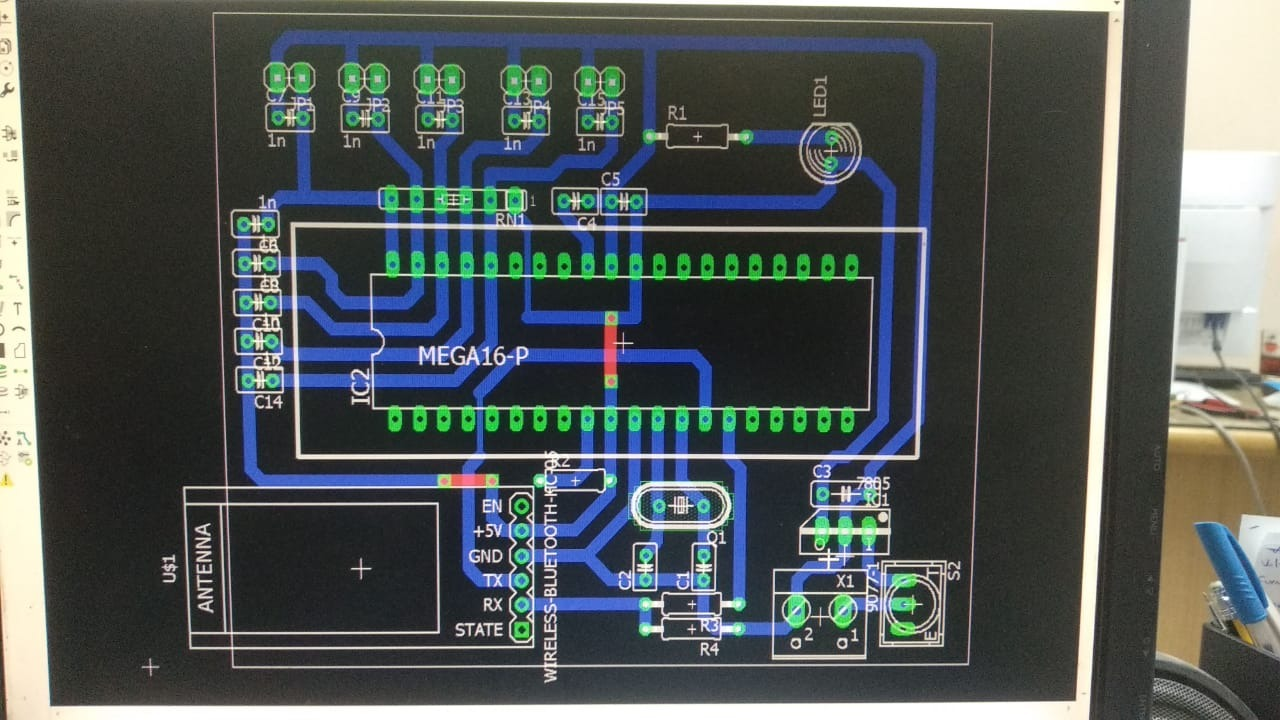
We used proteus for the below diagram

****

**PCB Design**

We used Eagle for designing the PCB layout



****

**AVR Code**

**#include "util/delay.h"**

**#include <avr/io.h>**

**#include <stdlib.h>**

**void ADC\_init(){**

**ADCSRA = 0X87;**

**ADCSRA |= (1 << ADSC);**

**while((ADCSRA & (1 << ADIF)) == 0);**

**}**

**void usart\_init(){**

**UCSRB = (1<<TXEN);**

**UCSRC = (1<<UCSZ1) | (1<<UCSZ0) | (1<<URSEL);**

**UBRRL = 0X4D;**

**}**

**void usart\_send(unsigned char *ch*){**

**while(!(UCSRA & (1<<UDRE)));**

**UDR = ch;**

**}**

**void UART\_String\_TX(char *string*[]){**

**usart\_send(' ');**

**for(int i=0; i<strlen(string); i++){**

**usart\_send(string[i]);**

**}**

**\_delay\_ms(1500);**

**}**

**int main(void)**

**{**

**PORTD = 0XFF; *//HC05***

**PORTA = 0X00; *//Sensors***

**int s0, s1, s2, s3, s4; *//sensor reading***

**usart\_init();**

**ADC\_init();**

**while(1)**

**{**

**ADMUX = 0XE0;**

**if(ADCH >= 50 && ADCH <= 100){**

**s0 = 1;**

**}else if(ADCH >= 110 && ADCH <= 140){**

**s0 = 2;**

**}else if(ADCH >= 145 && ADCH <= 200){**

**s0 = 3;**

**}**

**ADMUX = 0XE1;**

**if(ADCH >= 40 && ADCH <= 70){**

**s1 = 1;**

**}else if(ADCH >= 90 && ADCH <= 110){**

**s1 = 2;**

**}else if(ADCH >= 120 && ADCH <= 160){**

**s1 = 3;**

**}**

**ADMUX = 0XE2;**

**if(ADCH >= 40 && ADCH <= 70){**

**s2 = 1;**

**}else if(ADCH >= 100 && ADCH <= 120){**

**s2 = 2;**

**}else if(ADCH >= 130 && ADCH <= 180){**

**s2 = 3;**

**}**

**ADMUX = 0XE3;**

**if(ADCH >= 55 && ADCH <= 65){**

**s3 = 1;**

**}else if(ADCH >= 90 && ADCH <= 125){**

**s3 = 2;**

**}else if(ADCH >= 140 && ADCH <= 180){**

**s3 = 3;**

**}**

**ADMUX = 0XE4;**

**if(ADCH >= 115 && ADCH <= 140){**

**s4 = 1;**

**}else if(ADCH >= 145 && ADCH <= 160){**

**s4 = 2;**

**}else if(ADCH >= 170 && ADCH <= 195){**

**s4 = 3;**

**}**

**if(s1 == 1 && s0 != 1 && s2 != 1 && s3 != 1 && s4 != 1){**

**UART\_String\_TX("1");**

**}**

**if(s1 == 1 && s0 != 1 && s2 == 1 && s3 != 1 && s4 != 1){**

**UART\_String\_TX("2");**

**}**

**if(s1 == 1 && s0 != 1 && s2 == 1 && s3 == 1 && s4 != 1){**

**UART\_String\_TX("3");**

**}**

**if(s1 == 1 && s0 != 1 && s2 == 1 && s3 == 1 && s4 == 1){**

**UART\_String\_TX("4");**

**}**

**if(s1 == 1 && s0 == 1 && s2 == 1 && s3 == 1 && s4 == 1){**

**UART\_String\_TX("Thanks");**

**UART\_String\_TX("doctor");**

**UART\_String\_TX("Samir");**

**}**

**if(s1 != 1 && s0 == 1 && s2 != 1 && s3 != 1 && s4 != 1){**

**UART\_String\_TX("Ok");**

**}**

**if(s1 == 3 && s0 == 3 && s2 == 3 && s3 == 3 && s4 == 3){**

**UART\_String\_TX("0");**

**}**

**if(s1 == 1 && s0 == 1 && s2 != 1 && s3 != 1 && s4 != 1){**

**UART\_String\_TX("Bom");**

**}**

**if(s1 == 1 && s0 != 1 && s2 != 1 && s3 != 1 && s4 == 1){**

**UART\_String\_TX("I Love you");**

**}**

**if(s1 == 1 && s0 == 1 && s2 != 1 && s3 != 1 && s4 == 1){**

**UART\_String\_TX("Omar");**

**}**

**}**

**return 0;**

**}**

**Application**

Sign To Speech is an Android app that receives a text by Bluetooth, displays this text, and says it.

To develop our app, we used the [Java programming language](https://en.wikipedia.org/wiki/Java_(programming_language)) to develop the app and [Extensible Markup Language](https://en.wikipedia.org/wiki/XML) (XML) files to describe data resources. To help you develop your apps efficiently, Google offers an integrated development environment (IDE) called [Android Studio](https://developer.android.com/studio/index.html). It offers advanced features for developing, debugging, and packaging Android apps. Using Android Studio, you can develop for any Android-powered device, or create virtual devices that emulate any hardware configuration.

The minimum Android version is 4.0, which means it works in all current Android mobiles. App APK size is 1.6MB.

**XML**

Android layouts are written in extensible Markup Language, also known as XML. Much like HTML (or Hypertext *Markup Language*), XML is also a markup language. It was created as a standard way to encode data in internet-based applications. However, *unlike* HTML, XML is case-sensitive, requires each tag is closed properly, and preserves whitespace.

Android XML layouts are also part of a larger umbrella of Android files and components called resources.

**Resources** are the additional files and static content an application needs, such as animations, color schemes, layouts, menu layouts.

Each layout file must contain one (and only one!) root element. Linear Layouts, Relative Layouts, and Frame Layouts (see Root Views section below) may all be root elements. Other layouts may not be. All other XML elements will reside within this root object.

A View is simply an object from Android's built-in **View** class. It represents a rectangular area of the screen, and is responsible for displaying information or content, and event handling. Text, images, and buttons are all **Views** in Android.

A ViewGroup is a subclass of **View**, and is essentially an 'invisible container' that holds multiple **Views** or **ViewGroups** together, and defines their layout properties.

**Java**

Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language. Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the Oak name to Java.

## **Types of Java Applications**

There are mainly 4 types of applications that can be created using Java programming:

#### **1) Standalone Application**

Standalone applications are also known as desktop applications or window-based applications. These are traditional software that we need to install on every machine. Examples of standalone application are Media player, antivirus, etc. AWT and Swing are used in Java for creating standalone applications.

#### 

#### **2) Web Application**

An application that runs on the server side and creates a dynamic page is called a web application. Currently, [Servlet](https://www.javatpoint.com/servlet-tutorial), [JSP](https://www.javatpoint.com/jsp-tutorial), [Struts](https://www.javatpoint.com/struts-2-tutorial), [Spring](https://www.javatpoint.com/spring-tutorial), [Hibernate](https://www.javatpoint.com/hibernate-tutorial), [JSF](https://www.javatpoint.com/jsf-tutorial), etc. technologies are used for creating web applications in Java.

#### **3) Enterprise Application**

An application that is distributed in nature, such as banking applications, etc. is called enterprise application. It has advantages of the high-level security, load balancing, and clustering. In Java, [EJB](https://www.javatpoint.com/ejb-tutorial) is used for creating enterprise applications.

#### **4) Mobile Application**

An application which is created for mobile devices is called a mobile application. Currently, Android and Java ME are used for creating mobile applications.

****Why is Java the best programming language?****

* Java is easy to learn, the language has fluent English-like syntax.
* Java has rich API, xml parsing, database connection, networking, utilities and provides almost everything that a developer can expect.
* open source and available free.
* Very big community

# 

# **Bluetooth overview**

In order for Bluetooth-enabled devices to transmit data between each other, they 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 they 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 they can reconnect automatically during a future session as long as they're in range of each other and neither device has removed the bond.

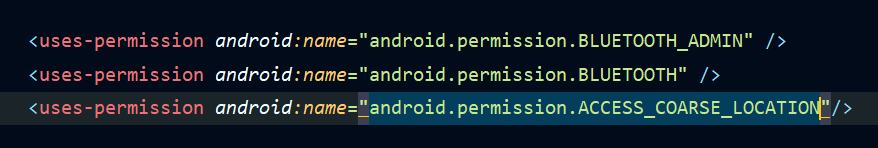
### **Bluetooth permissions**

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

The other permission that you must declare is **ACCESS\_COARSE\_LOCATION**. Your 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.

If you want your app to initiate device discovery or manipulate Bluetooth settings, you 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.

Declare the Bluetooth permission(s) in your application manifest file. For example:

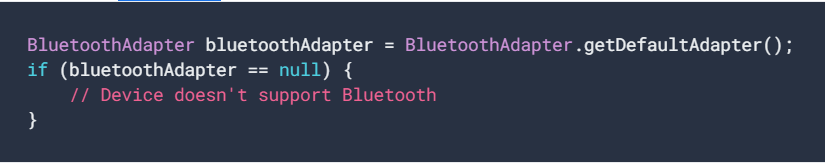


## **Set up Bluetooth**

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

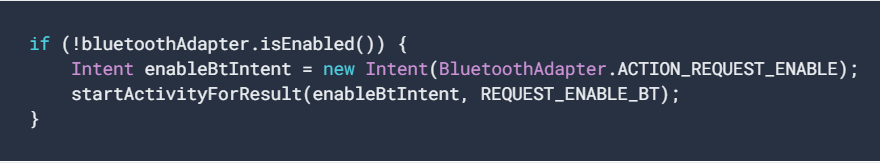
If Bluetooth isn't supported, then you should gracefully disable any Bluetooth features. If Bluetooth is supported, but disabled, then you can request that the user enable Bluetooth without leaving your application. This setup is accomplished in two steps, using the **[BluetoothAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter)*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

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())**[()](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 your application can interact with it using this object. If **[getDefaultAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#getDefaultAdapter())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#getDefaultAdapter()) returns **null**, then the device doesn't support Bluetooth. For example:

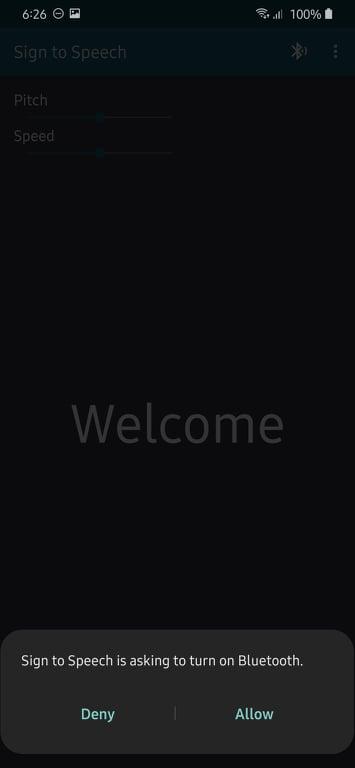


**2.Enable Bluetooth.**

Next, you need to ensure that Bluetooth is enabled. Call **[isEnabled](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#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,%20int))**[()](https://developer.android.com/reference/android/app/Activity#startActivityForResult(android.content.Intent,%20int)), 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 your application). For example:



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



## **Find devices**

Using the **[BluetoothAdapter](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter)**, you 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.

Note that 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 APIs require devices to be paired before an RFCOMM connection can be established. Pairing is automatically performed when you initiate an encrypted connection with the Bluetooth APIs.

### 

### **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, you can query all paired devices and get the name and MAC address of each device, as the following code snippet demonstrates:



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())**.

### **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, your application must register a BroadcastReceiver for the **[ACTION\_FOUND](https://developer.android.com/reference/android/bluetooth/BluetoothDevice#ACTION_FOUND)** intent.The following code snippet shows how you can register to handle the broadcast when devices are discovered:



## **Connect devices**

In order to create a connection between two devices, you 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 **[BluetoothSocket](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 **[BluetoothSocket](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.

### **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.

### **Connect as a server**

When you want to connect two devices, one must act as a server by holding an open **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)**. The purpose of the server socket is to listen for incoming connection requests and provide a connected **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** after a request is accepted. When the **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** is acquired from the **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)**, the **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)** can—and should—be discarded, unless you want the device to accept more connections.

To set up a server socket and accept a connection, complete the following sequence of steps:

1. Get a **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)** by calling **[listenUsingRfcommWithServiceRecord()](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#listenUsingRfcommWithServiceRecord(java.lang.String,%20java.util.UUID))**.  
   The string is an identifiable name of your service, which the system automatically writes to a new Service Discovery Protocol (SDP) database entry on the device. The name is arbitrary and can simply be your application name.

The Universally Unique Identifier (UUID) is also included in the SDP entry and forms the basis for the connection agreement with the client device. That is, when the client attempts to connect with this device, it carries a UUID that uniquely identifies the service with which it wants to connect. These UUIDs must match in order for the connection to be accepted.  
A UUID is a standardized 128-bit format for a string ID used to uniquely identify information. The point of a UUID is that it's big enough that you can select any random ID and it doesn't clash with any other ID. In this case, it's used to uniquely identify your application's Bluetooth service. To get a UUID to use with your application, you can use one of the many random UUID generators on the web, then initialize a **[UUID](https://developer.android.com/reference/java/util/UUID)** with **[fromString(String)](https://developer.android.com/reference/java/util/UUID#fromString(java.lang.String))**.

1. Start listening for connection requests by calling **[accept()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#accept())**.  
   This is a blocking call. It returns when either a connection has been accepted or an exception has occurred. A connection is accepted only when a remote device has sent a connection request containing a UUID that matches the one registered with this listening server socket. When successful, **[accept()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#accept())** returns a connected **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)**.
2. Unless you want to accept additional connections, call **[close()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#close())**.  
   This method call releases the server socket and all its resources, but doesn't close the connected **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** that's been returned by **[accept()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#accept())**. Unlike TCP/IP, RFCOMM allows only one connected client per channel at a time, so in most cases, it makes sense to call **[close()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#close())** on the **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)** immediately after accepting a connected socket.

Because the **[accept()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#accept())** call is a blocking call, it should not be executed in the main activity UI thread so that your application can still respond to other user interactions. It usually makes sense to do all work that involves a **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)** or **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** in a new thread managed by your application. To abort a blocked call such as **[accept](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#accept())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#accept()), call **[close](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#close())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket#close()) on the **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)** or **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** from another thread. Note that all methods on a **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)** or **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** are thread-safe.

Here's a simplified thread for the server component that accepts incoming connection



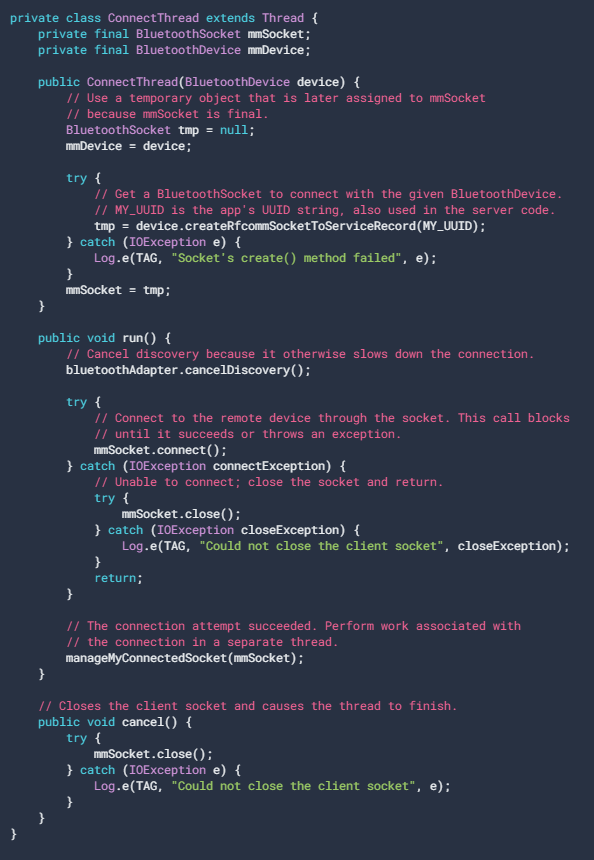
### **Connect as a client**

In order to initiate a connection with a remote device that is accepting connections on an open server socket, you must first obtain a **[BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice)** object that represents the remote device. To learn how to create a **[BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice)**, see [Finding Devices](https://developer.android.com/guide/topics/connectivity/bluetooth#FindingDevices). You must then use the **[BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice)** to acquire a **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** and initiate the connection.

The basic procedure is as follows:

1. Using the **[BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice)**, get a **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** by calling **[createRfcommSocketToServiceRecord(UUID)](https://developer.android.com/reference/android/bluetooth/BluetoothDevice#createRfcommSocketToServiceRecord(java.util.UUID))**.  
   This method initializes a **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)** object that allows the client to connect to a **[BluetoothDevice](https://developer.android.com/reference/android/bluetooth/BluetoothDevice)**. The UUID passed here must match the UUID used by the server device when it called **[listenUsingRfcommWithServiceRecord(String, UUID)](https://developer.android.com/reference/android/bluetooth/BluetoothAdapter#listenUsingRfcommWithServiceRecord(java.lang.String,%20java.util.UUID))** to open its **[BluetoothServerSocket](https://developer.android.com/reference/android/bluetooth/BluetoothServerSocket)**. To use a matching UUID, hard-code the UUID string into your application, and then reference it from both the server and client code.
2. Initiate the connection by calling **[connect](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect()). Note that this method is a blocking call.  
   After a client calls this method, the system performs an SDP lookup to find the remote device with the matching UUID. If the lookup is successful and the remote device accepts the connection, it shares the RFCOMM channel to use during the connection, and the **[connect](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect()) method returns. If the connection fails, or if the **[connect](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect()) method times out (after about 12 seconds), then the method throws an **[IOException](https://developer.android.com/reference/java/io/IOException)**.  
   Because **[connect](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#connect()) is a blocking call, you should always perform this connection procedure in a thread that is separate from the main activity (UI) thread.

Here is a basic example of a client thread that initiates a Bluetooth connection:



## **Manage a connection**

After you have successfully connected multiple devices, each one has a connected **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)**. This is where the fun begins because you can share information between devices. Using the **[BluetoothSocket](https://developer.android.com/reference/android/bluetooth/BluetoothSocket)**, the general procedure to transfer data is as follows:

1. Get the **[InputStream](https://developer.android.com/reference/java/io/InputStream)** and **[OutputStream](https://developer.android.com/reference/java/io/OutputStream)** that handle transmissions through the socket using **[getInputStream](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#getInputStream())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#getInputStream()) and **[getOutputStream](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#getOutputStream())**[()](https://developer.android.com/reference/android/bluetooth/BluetoothSocket#getOutputStream()), respectively.
2. Read and write data to the streams using **[read(byte[])](https://developer.android.com/reference/java/io/InputStream#read(byte%5B%5D))** and **[write(byte[])](https://developer.android.com/reference/java/io/OutputStream#write(byte%5B%5D))**.

There are, of course, implementation details to consider. In particular, you should use a dedicated thread for reading from the stream and writing to it. This is important because both the **[read(byte[])](https://developer.android.com/reference/java/io/InputStream#read(byte%5B%5D))** and **[write(byte[])](https://developer.android.com/reference/java/io/OutputStream#write(byte%5B%5D))** methods are blocking calls. The **[read(byte[])](https://developer.android.com/reference/java/io/InputStream#read(byte%5B%5D))** method blocks until there is something to read from the stream. The **[write(byte[])](https://developer.android.com/reference/java/io/OutputStream#write(byte%5B%5D))** method doesn't usually block, but it can block for flow control if the remote device isn't calling **[read(byte[])](https://developer.android.com/reference/java/io/InputStream#read(byte%5B%5D))** quickly enough and the intermediate buffers become full as a result. So, your main loop in the thread should be dedicated to reading from the **[InputStream](https://developer.android.com/reference/java/io/InputStream)**. A separate public method in the thread can be used to initiate writes to the **[OutputStream](https://developer.android.com/reference/java/io/OutputStream)**.

**Text To Speech**

Android allows you convert your text into voice. Not only you can convert it but it also allows you to speak text in variety of different languages.

Android provides **TextToSpeech** class for this purpose. In order to use this class, you need to instantiate an object of this class and also specify the initListener. Its syntax is given below.



In this listener, you have to specify the properties for TextToSpeech object , such as its language ,pitch e.t.c. Language can be set by calling **setLanguage()** method.

The method setLanguage takes an Locale object as parameter. The list of some of the locales available are :

1. **CHINESE**

### **ENGLISH**

### **FRENCH**

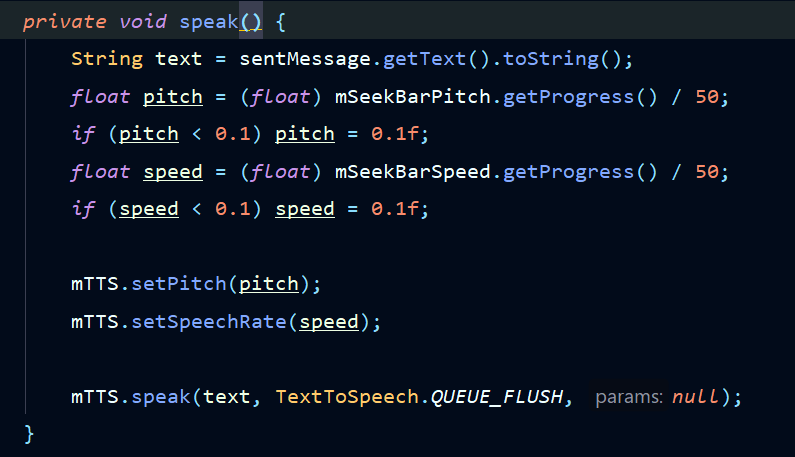
### **GERMAN**

### **ITALIAN**

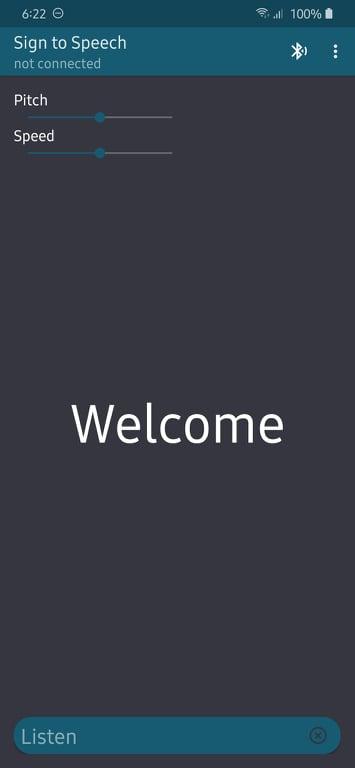
### **JAPANESE**

### **KOREAN**

Once you have set the language, you can call speak method of the class to speak the text. We will also adjust the pitch and speed of the spoken text using SeekBars.



So, after receiving the text by Bluetooth, the text is displayed in the center of the screen, and said.



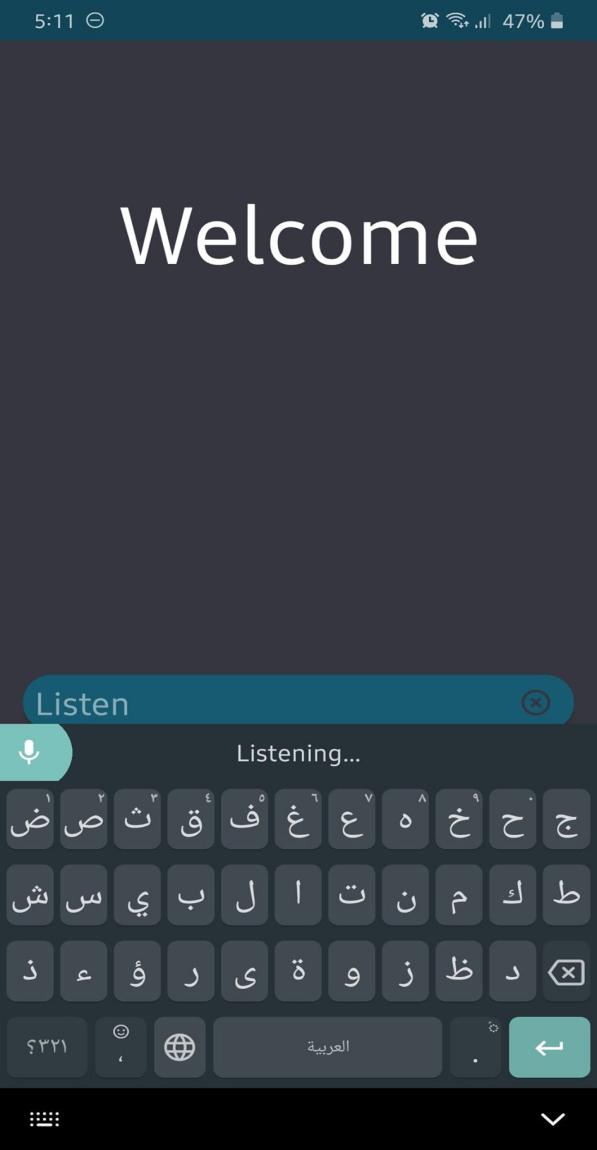
**Listening**

Now we solved the problem of speaking, but what about **listening**?

we used an app made by google which is Gboard(Google keyboad).

so how to use it inside our app:

* Download the app from Play Store.
* Make it the default keyboard for your mobile.
* In our app there’s a text field in the bottom of the screen, click inside it to open your keyboard.
* Click the record icon and your mobile now listening.
* The speech then is converted to text, and is shown inside the text field.
* The language you listen is the language selected in the keyboard.



**Resources**

* Mazidi AVR book
* [ATmega32 Datasheet](http://ww1.microchip.com/downloads/en/devicedoc/doc2503.pdf)
* <https://www.javatpoint.com/java-tutorial>
* <https://www.tutorialspoint.com/android/android_text_to_speech.htm>
* <https://developer.android.com/guide/topics/connectivity/bluetooth>
* <https://github.com/android/connectivity-samples/tree/master/BluetoothChat>
* <https://www.learnhowtoprogram.com/android/introduction-to-android/introduction-to-xml-and-android-layouts>
* <https://play.google.com/store/apps/details?id=com.google.android.inputmethod.latin&hl=en>
* <https://github.com/OmarShafei-coder/SignToSpeech>

**Names**

1. **Gehad Ahmed Mohamed**
2. **Omar Ahmed Ibrahim**
3. **Ahmed Abd Elnaby Yousef**
4. **Ali Omar Ali**
5. **Abdallah Mosaad**
6. **Mohamed Samir**
7. **Gamal Mahmoud**