IoT
Internet of things

About the Company

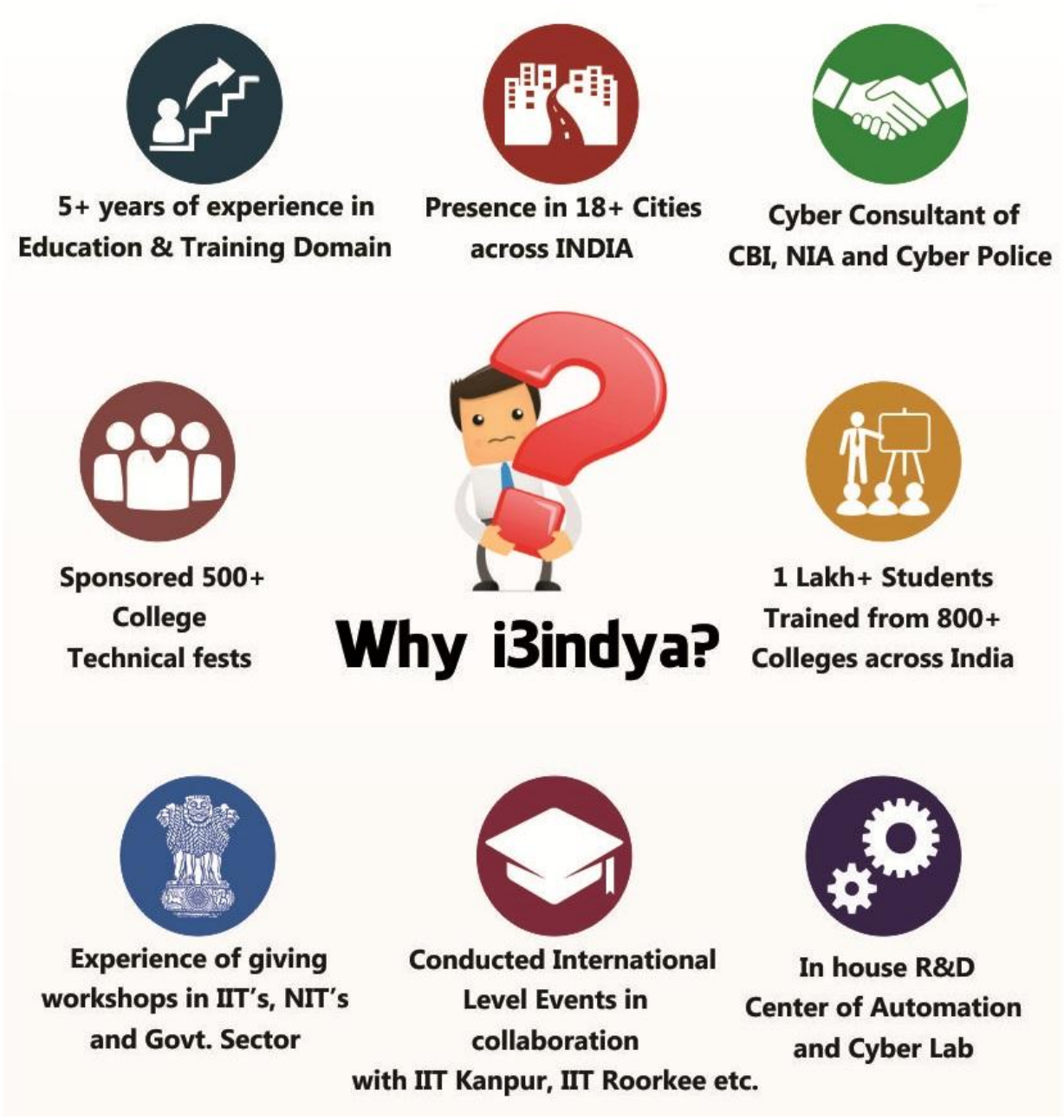
i3indya Technologies (An i3indya Group Company) with its foundation pillars as **Innovation, Information and Intelligence** is a Technical Training Service Company for **School, Colleges, Corporate** and **Government Sector**.

Since it's inception in **2009**, i3indya has expanded it's reach to **18+** cities across INDIA. We have trained more than **1,00,000+** students & professionals from **800+** School, Colleges, Government Organisation and Corporates.

Few of Our Institutional Customers are **IIT Bombay**, IIT Kanpur, IIT Madras, IIT Roorkee, NIT Warangal, **NIT Jalandhar**, **CBI**, NIA, **GAIL**, Infosys, **TCS**, DRDO,**Cyber Police** of Delhi, Chattisgarh, Jalandhar and many more…

At i3indya, we have a dedicated Research & Development Cell of **Cyber Forensics** and **Industrial Automation**, where experts have solved some of the most complex Cyber Security cases for Governments and MNC's and developed many Automated solutions for the Industry. Our research efforts have also dwelt into projects on Software Testing, Robotics kit Manufacturing, Website Testing, Circuit Designing and Secure Website Development.

Key Benefits of i3indya Technologies



Few i3indya Clients

Table of Contents

IoT (Internet of things)

Internet of Things (IOT) is a new era of computing technology. The vast network of devices connected to the Internet like smart phones and tablets and almost anything with a sensor on it like cars, machines in production plants, jet engines, oil drills, wearable devices, and more. Now these devices can talk with each other means it can collect and exchange data.

It is a Machine to machine (M2M) technology i.e. Machine to Machine, Machine to Man, Man to Machine, or Machine to Mobile. This technology made possible for machines to collect, exchange data and also take decision without human intervention. With this technology it is possible that

- Utilities and telcos that can predict and prevent service outages
- Airlines that can remotely monitor and optimize plane performance and
- Healthcare organizations that can base treatment on real-time genome analysis.

Future of IOT

Internet of Things can connect devices embedded in various systems to the internet. So they can be controlled from anywhere. This helps in collecting more data from more places. Businesses in the utilities, oil & gas, insurance, manufacturing, transportation, infrastructure and retail sectors can reap the benefits of IoT by making more informed decisions.

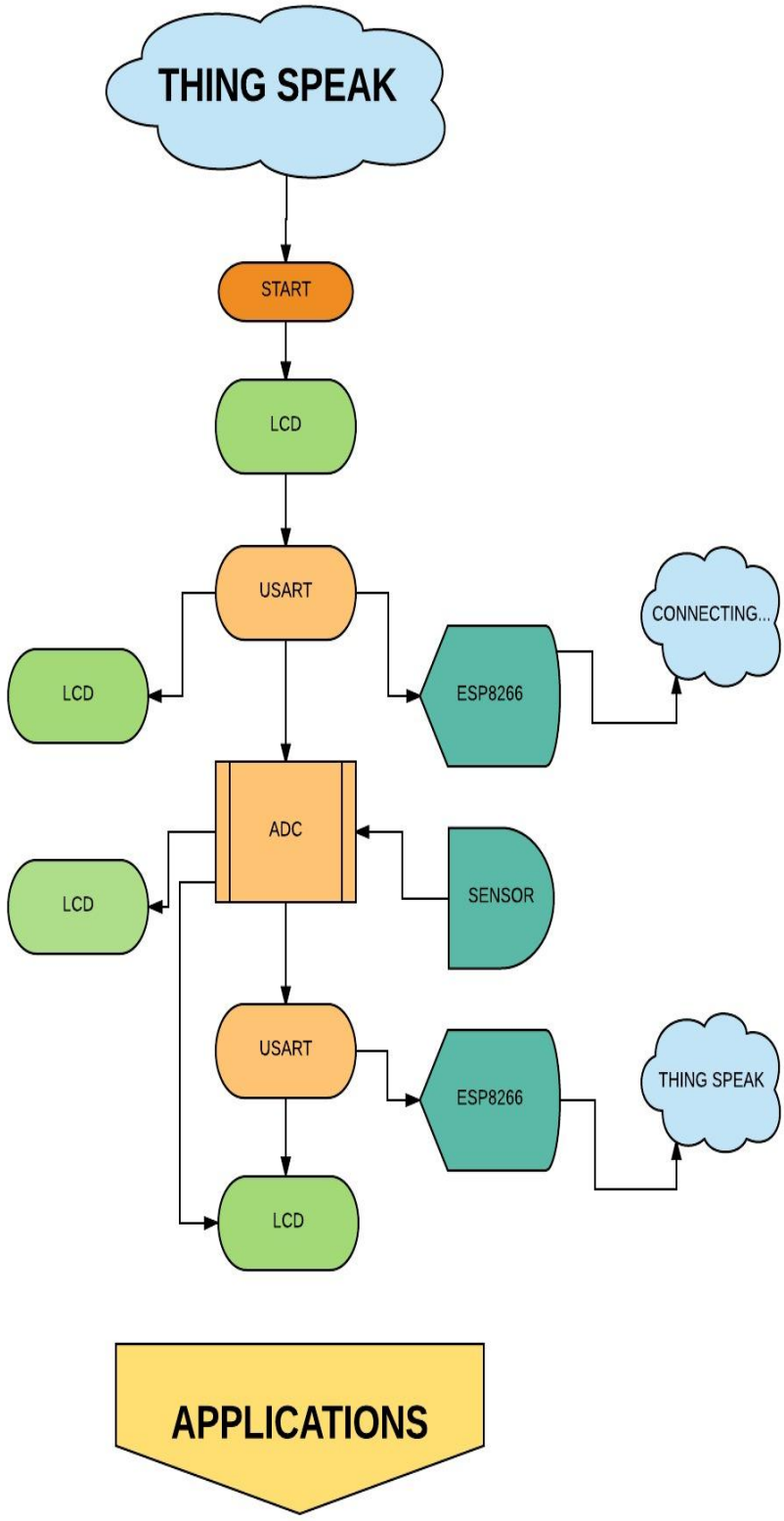
Now let’s discus how would our future look like in daily life with the help of IOT. People will have complete access to all products at home. For example, people would be able to switch off the motor or ac running at their home or the lights in car while sitting at their office. That's the kind of smartness IOT technology would bring it to the lifestyle of the people.

- Sensors in smart homes turn off utilities, close windows, monitor security, and report to house owners in real time.
- Vehicle mounted terminals automatically display the nearest parking space.
- Doctors would remotely monitor the conditions of their patients 24/7 by having the patients use devices at home instead of requiring the patients to stay at hospital.
- The power companies would read meters through telemetering systems instead of visiting houses.

Knowledge Requirement for IOT:

- Microcontrollers & ICs.
- ATmega8 Microcontroller
- Serial communication
- Introduction to ADC
- Introduction To WIFI
- Introduction to TCP/IP
- IOT Platform (think speak / IBM Watson / Local server)

Flow Chart:


THING SPEAK
START
LCD
USART
LCD
ESP8266
CONNECTING...
ADC
LCD
SENSOR
USART
ESP8266
THING SPEAK
LCD
APPLICATIONS

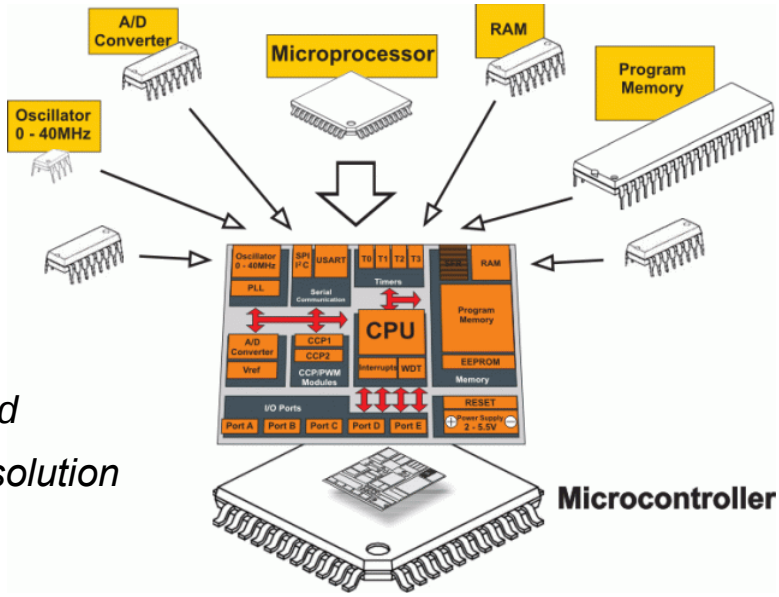
Microcontrollers & ICs

A microcontroller is a small computer on a single integrated circuit containing Microprocessor, RAM, EPROM and I/Os.

A **microprocessor** incorporates the functions of a computer's central processing unit (CPU) on a single integrated circuit (IC, or microchip).

A microcontroller has its configuration, in the same way as we borrow any computer or laptop of different configuration. Example I have my laptop configuration as follows:

- *Intel core2duo T6600 processor*
- *2.2GHz processing speed*
- *2 MB cache*
- *320 GB 7200 rpm hard disk*
- *4 GB DDR2 RAM*
- *512 MB dedicated ATI graphics card*
- *15" WLED screen with HD 720p resolution*



Like a laptop, the microcontroller has its own configuration. We are going to study about ATMEGA8L microcontroller. So, I will explain its configuration:

- *8 KB Flash memory*
- *1 KB RAM*
- *512 Byte EEPROM*
- *8MHz maximum speed*
- *28 pin IC*
- *and many others that we will come across later*

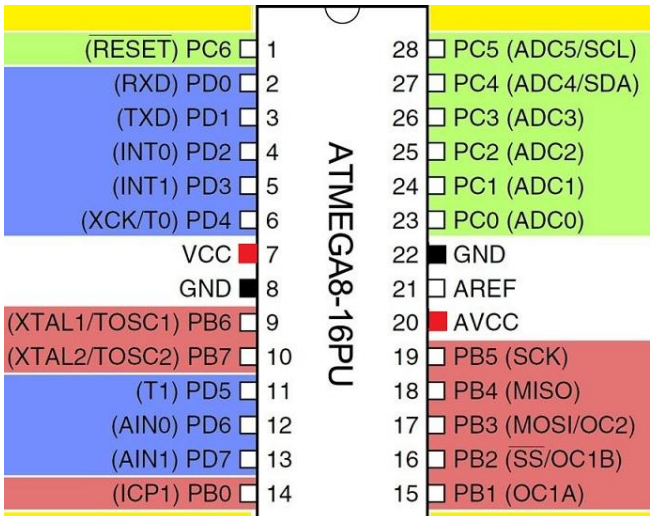
Introduction to Programming in ATMEGA8

ATmega8 microcontroller has 23 programmable input/output (I/O) pins which can be used for interfacing with external sensor. It is possible to configure them as input or output by setting a particular register value through programming.

This IC comes in 3 different packages, but we are using the popular 28-Pin PDIP package (Atmega8-16PU). Note that Atmega8 is available in 2 versions; ATmega8 and Atmega8L. Atmega8L is a low frequency version which works up to 8MHz frequency.

The 23 I/O ports of ATmega8 are organized into 3 groups:

- Port B (PB0 to PB7)
- Port C (PC0 to PC6)
- Port D (PD0 to PD7)



All of these I/O pins have secondary functions, which are shown in parenthesis on the pin out diagram shown here. Each of these registers are 8 bits wide, with each bit corresponding to a single pin (an exception

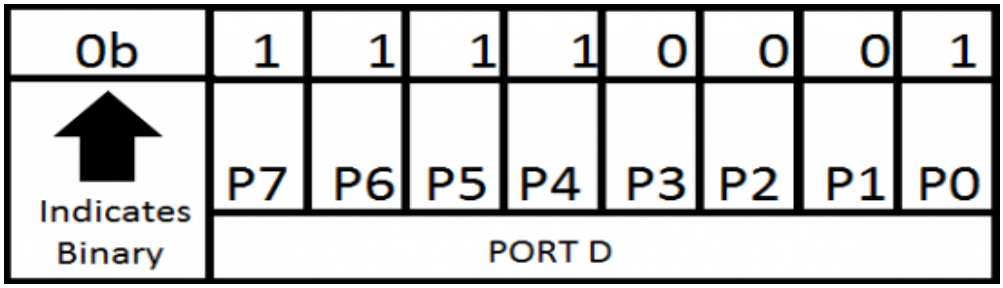
is bit 7 of the Port C register -PC6- most often used as the RESET pin, not an I/O). Registers used for reading and writing to the I/O ports are described below.

Register	Type	Description
DDRB	Read/Write	Port B Data Direction Register
PORTB	Read/Write	Port B Data Register
PINB	Read only	Port B Input Register
DDRC	Read/Write	Port C Data Direction Register
PORTC	Read/Write	Port C Data Register
PINC	Read only	Port C Input Register
DDRD	Read/Write	Port D Data Direction Register
PORTD	Read/Write	Port D Data Register
PIND	Read only	Port D Input Register

Let us consider the following code. What this means?

PORTD = 0b11110001;

The rightmost bit (least significant bit-LSB) represents pin 0 of port D (PD0) whilst the leftmost bit (most significant bit-MSB) represents pin 7. Here, the value is expressed in Binary. The same thing in Hexadecimal (Hex) is 0xF (0b means Binary / 0x means Hexadecimal)!



As per this, P1-P2-P3 are Inputs, and others (P0-P4-P5-P6-P7) are Outputs.

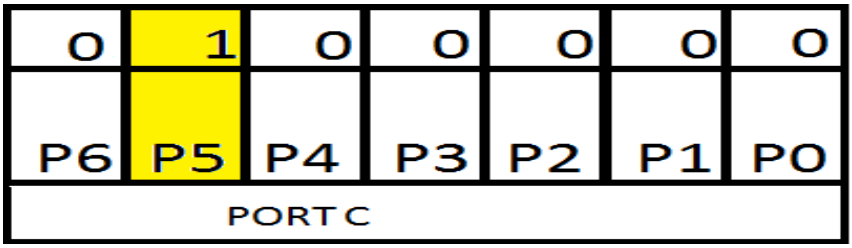
Output Port

In our first example (LEDTEST) one LED is connected to Pin 5 of Port C (PC5 → MCU Pin28), configured as an output port. Next, try to understand the code snippet included here.

```
#include <inttypes.h>
#include <avr/io.h>
#include <util/delay.h>

int main(){                          // The main function

DDRC = 0b11111111;                     // Set all the pins of PORTC as output
```

In this code, "DDRC = 0b11111111" sets all the pins of PORTC as outputs. But in our first example "DDRC |= (1<<5);" is used to set the Data Direction Register to output. Confused? Don't worry; infact both lines are doing the same thing. In both cases, a literal value is being assigned to PORTC.

- DDRC = 0b11111111 → Sets all the pins of PORT C as outputs
- DDRC |= (1<<5) → Shifts binary representation of 1 to left 5 times (i.e. 00000001 to 00100000), which means pin 5 of PORTC (PC5) is configured as an output port!



The C programming language includes a set of bitwise operators which apply logic operations to individual bits. These can be combined with _BVmacro to control individual pins. I will explain the Easy Coding Procedures immediately after the completion of the next part.

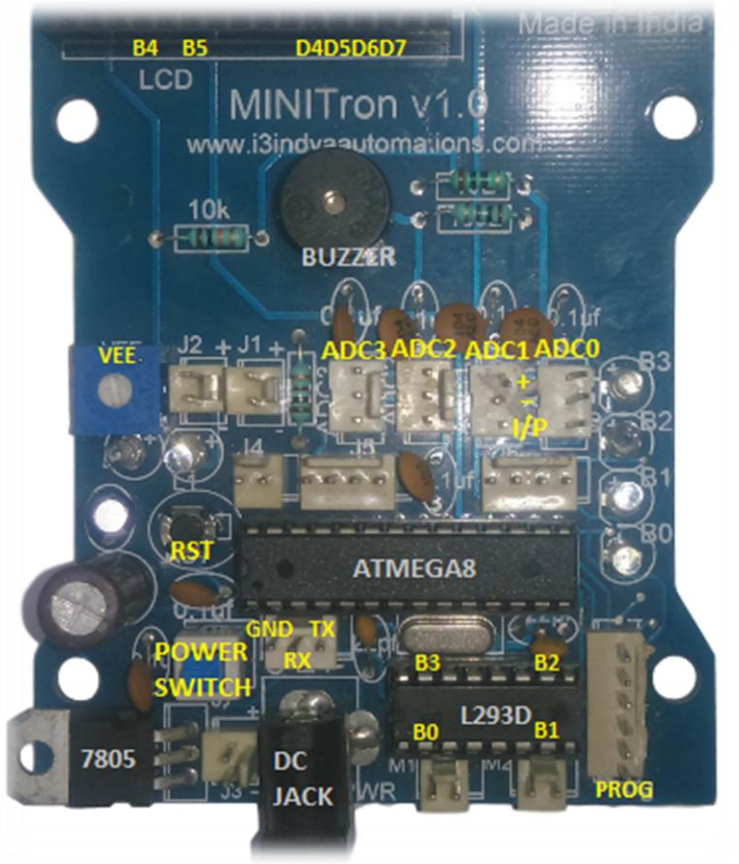
Input Port

Get ready to make an input connection with ATmega8. For this, take the breadboard and ensure that your first program (LEDTEST) is running well. Now remove the power supply and re-wire the breadboard as shown in the next circuit diagram. The work is very simple, because all you need is to add a pushbutton switch (at Pin 27) with the existing

circuit. Next, write-compile-build and burn the associated code to ATmega8 as before. If everything is OK, the LED at pin 28 starts blinking for 6 times when you push the pushbutton switch (S2) for a while. In this setup Pin 28 is output port, and Pin 27 is an input port. Code will be explicated in the forthcoming part, so stay tuned!

MINITron board

- Gives you a free hand on hardware as well as software
- Deals with register level programming
- You don't require shields like arduino board to work, basic components like L293D , led, lcd ports etc are already present



MINITron v1.0:

FEATURES :-

- Microcontroller: ATmega8.
- Operating Voltage: 5V.
- Input Voltage (recommended): 7-12V.
- Input Voltage (limits): 6-20V.
- Digital I/O Pins: 23

MINITron board Connections:

LCD

RS	:	PB4
RW	:	GND
EN	:	PB5
D4	:	PD4
D5	:	PD5
D6	:	PD6
D7	:	PD7

L293D

IN1	:	PB0
IN2	:	PB1
IN3	:	PB2

IN4	:	PB3

LED

B0	:	PB0
B1	:	PB1
B2	:	PB2
B3	:	PB3

BUZZER	:	PD2

ADC

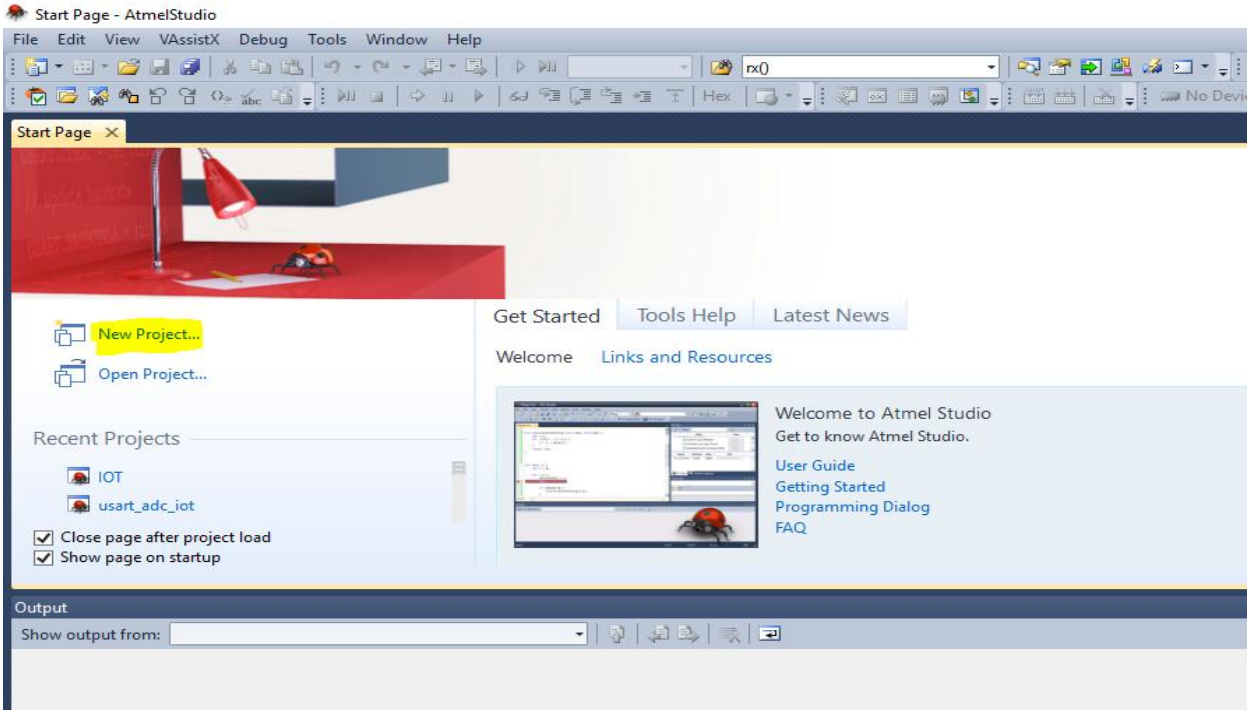
ADC0	:	PC0
ADC1	:	PC1
ADC2	:	PC2
ADC3	:	PC3
ADC4	:	PC4
ADC5	:	PC5
ADC6	:	PC6

USART

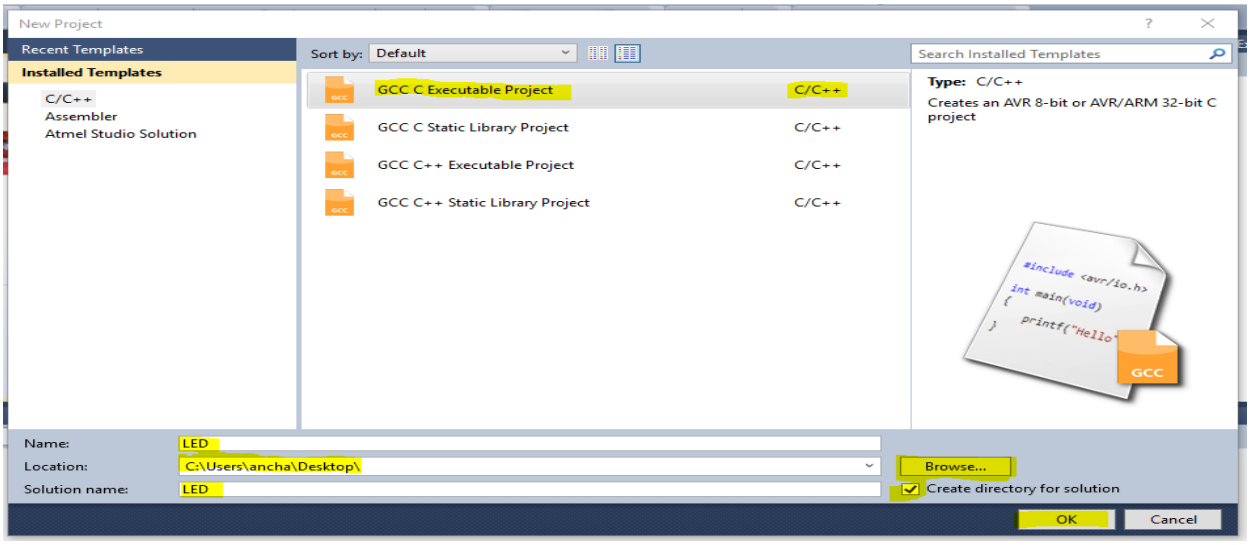
RX	:	PD0
TX	:	PD1

Some steps of program to blink LEDs using Atmel studio 6.0

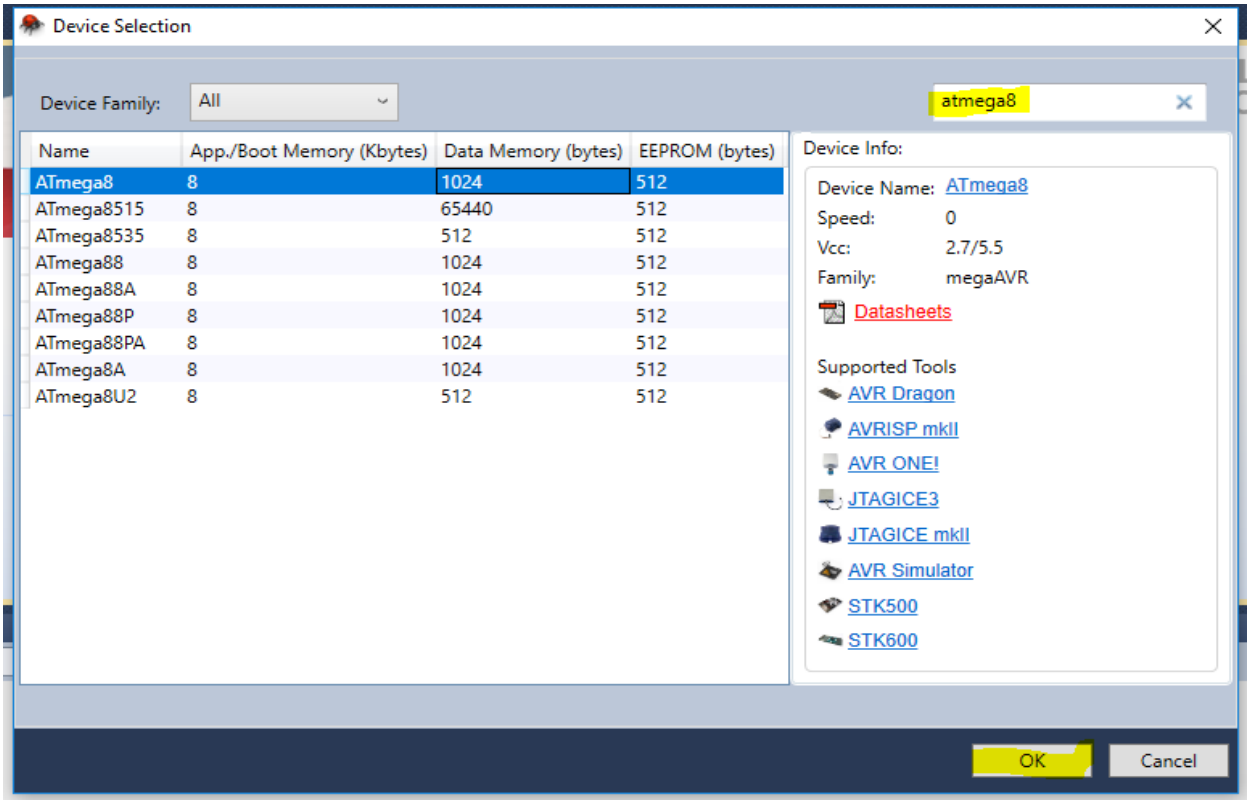
Step1. → New Project



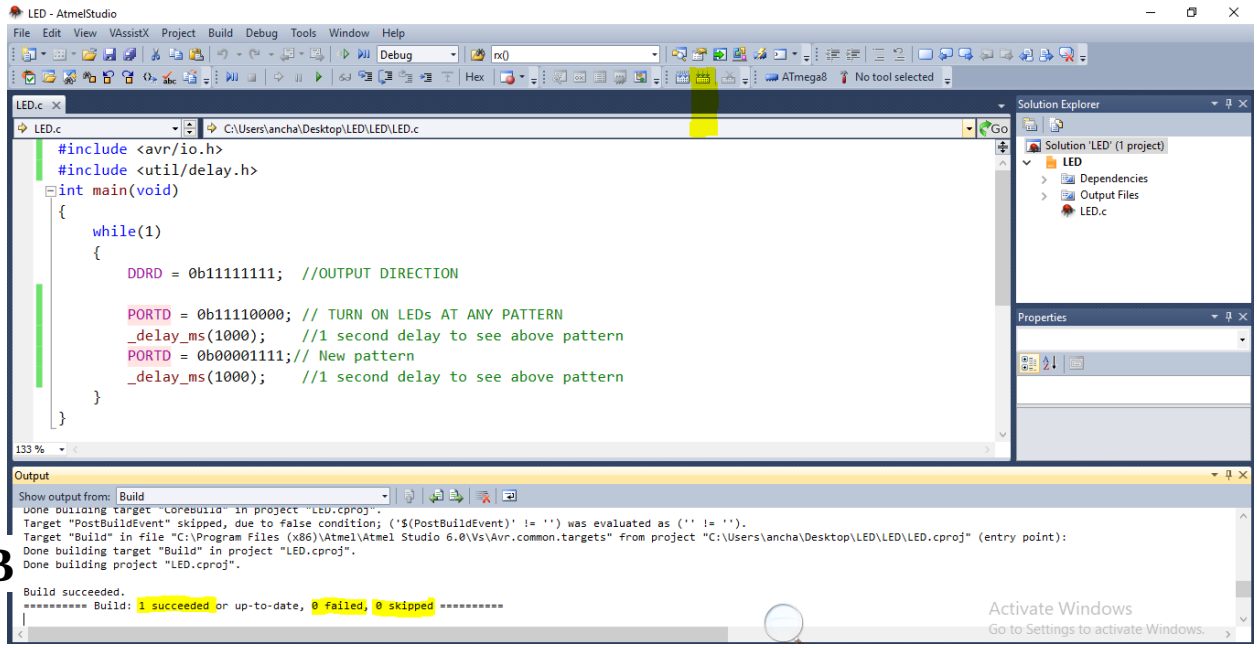
Step2. → GCC C Executable Project , Name , Location then ok.



Step3. → Select Microcontroller.



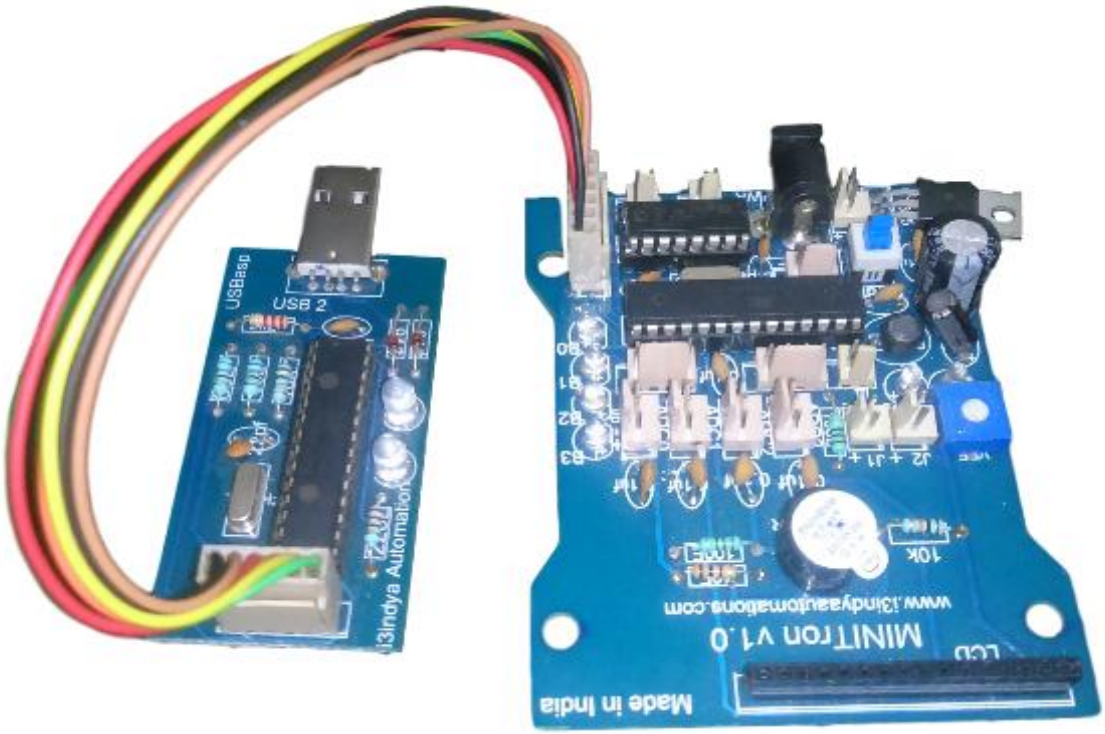
Step4. → Start coding form here:



B

AVR BURNER (US asp):

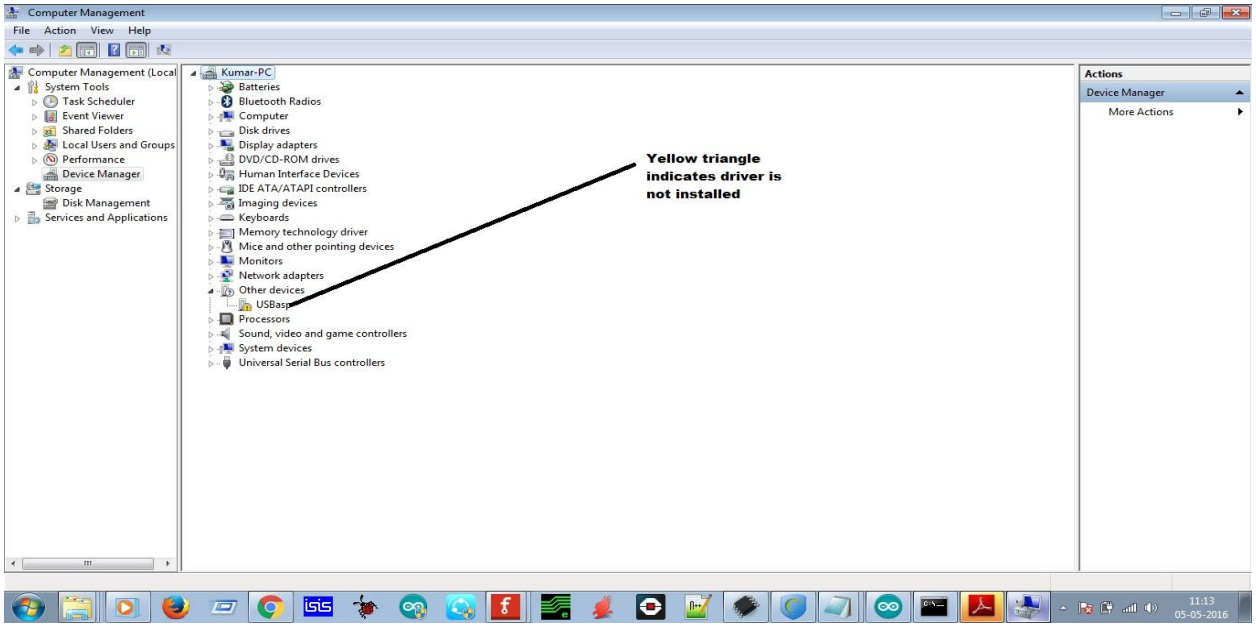
Interfacing between AVR loader and MINITron



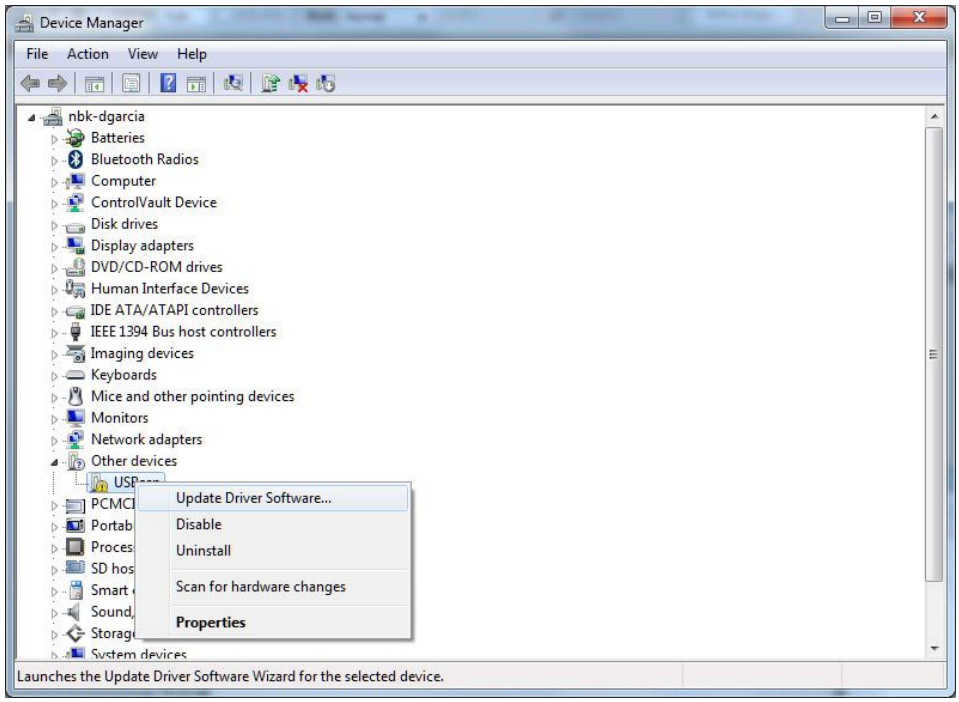
Driver software installation

There are some steps to follow-

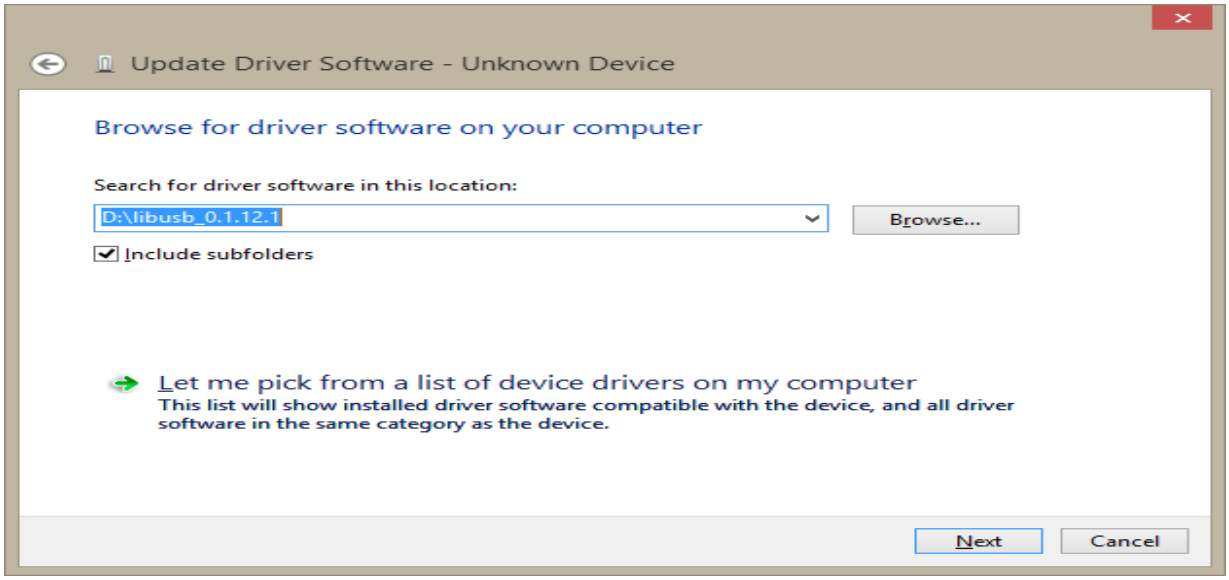
Step-1



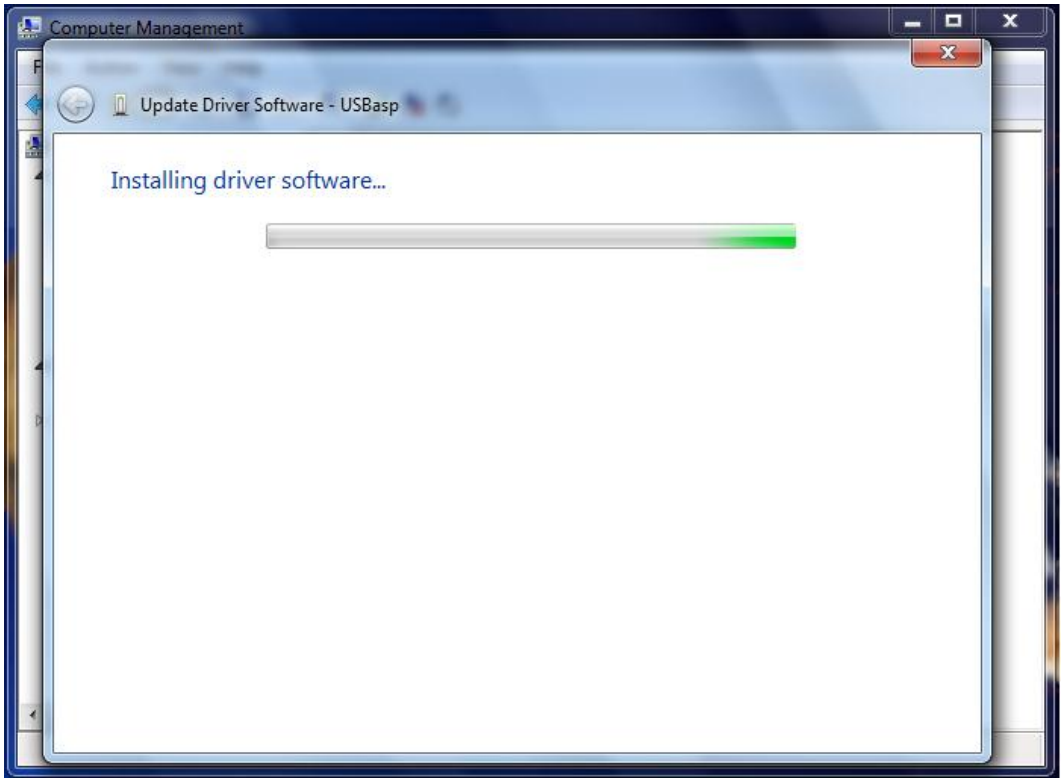
Step-2



Step-3


Update Driver Software - Unknown Device
Browse for driver software on your computer
Search for driver software in this location:
D:\libusb_0.1.12.1
Browse...
Include subfolders
Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.
Next
Cancel

Step-4


Computer Management
Update Driver Software - USBasp
Installing driver software...

Step-5

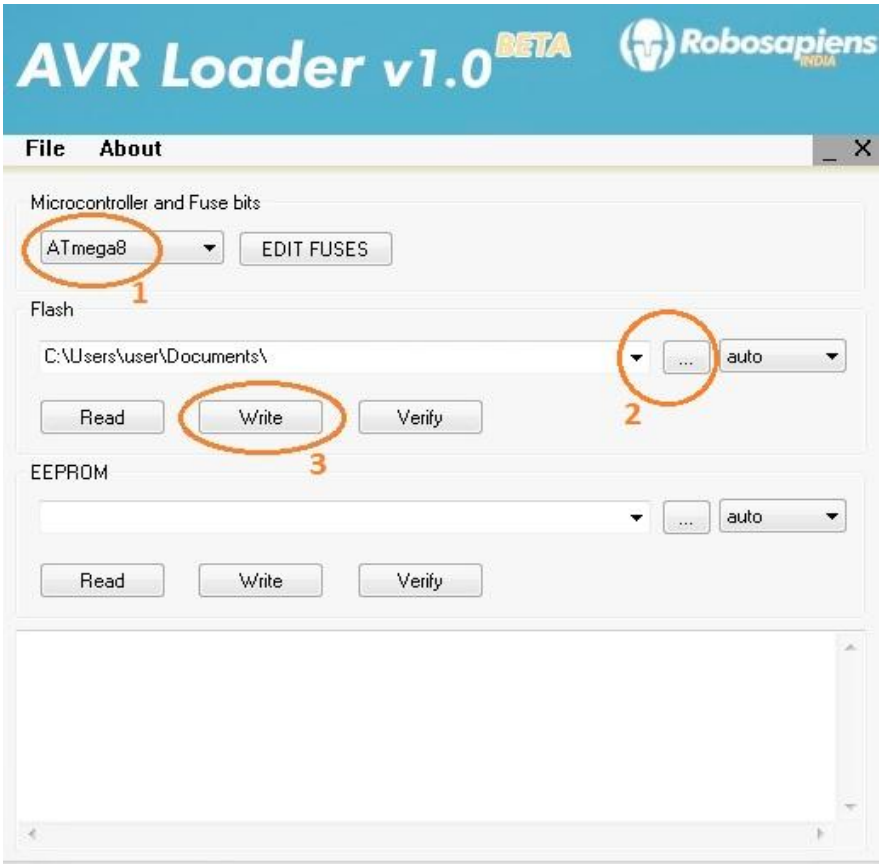

Device Manager
File
Action
View
Help
Windows Security
Windows can't verify the publisher of this driver software
Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.
Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.
See details

Step-6


Update Driver Software - USBasp
Windows has successfully updated your driver software
Windows has finished installing the driver software for this device:
USBasp
Close

Program burning to the controller

For this task you need one USBASP programmer. Just connect the output of the programmer (6 lines VCC, GND, RESET, SCK, MOSI, and MISO) to the respective pins of the unpowered target atmega8 (which you placed in the breadboard). The final thing needed is a program (burner tool) for flash burning! To burn the program you can use this simple and easy to use burner software. The "AVRLoader" can be downloaded from my Google Drive

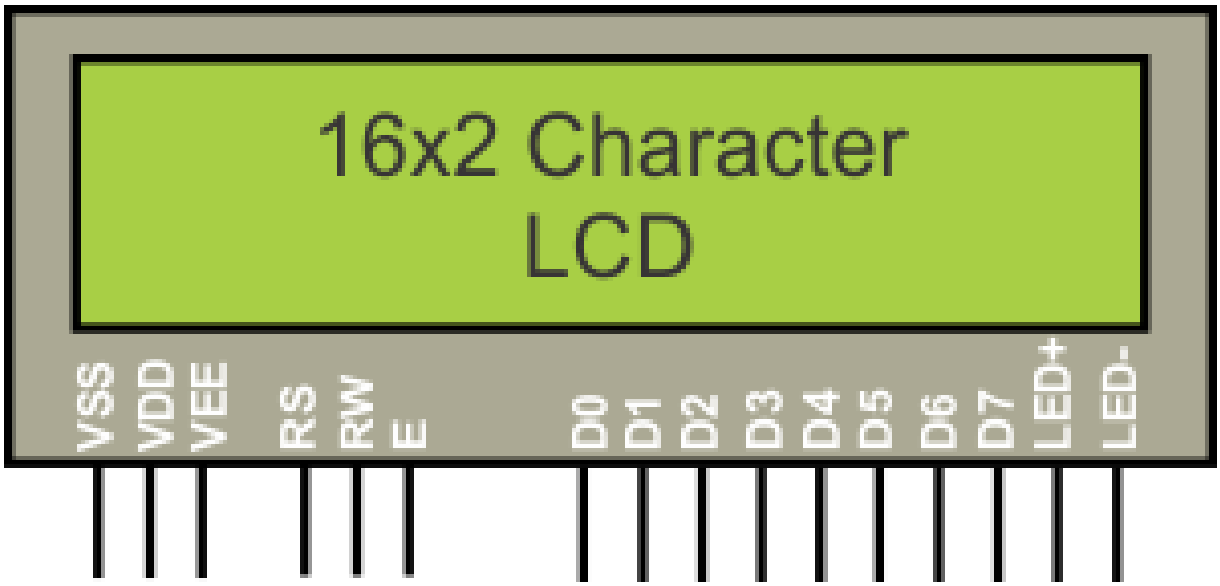


LCD INTERFACING

This section describes the operation modes of LCDs and how to program and interface LCDs.

16×2 LCD is named so because; it has 16 Columns and 2 Rows. There are a lot of combinations available like, 8×1, 8×2, 10×2, 16×1, etc. But the most used one is the 16*2 LCD; hence we are using it here.

All the above mentioned LCD display will have 16 Pins and the programming approach is also the same and hence the choice is left to you. Below is the **Pin out and Pin Description of 16x2 LCD Module**:



16X2 LCD:

Pin description and connection:

Sr. No	Pin No.	Pin Name	Pin Type	Pin Description	Pin Connection
1	Pin 1	Ground	Source Pin	This is a ground pin of LCD	Connected to the ground of the MCU/ Power source
2	Pin 2	VCC	Source Pin	This is the supply voltage pin of LCD	Connected to the supply pin of Power source
3	Pin 3	V0/VEE	Control Pin	Adjusts the contrast of the LCD.	Connected to a variable POT that can source 0-5V
4	Pin 4	Register Select	Control Pin	Toggles between Command/Data Register	Connected to a MCU pin and gets either 0 or 1. 0 -> Command Mode 1-> Data Mode
5	Pin 5	Read/Write	Control Pin	Toggles the LCD between Read/Write Operation	Connected to a MCU pin and gets either 0 or 1. 0 -> Write Operation 1-> Read Operation
6	Pin 6	Enable	Control Pin	Must be held high to perform Read/Write Operation	Connected to MCU and always held high.
7	Pin 7-14	Data Bits (0-7)	Data/Command Pin	Pins used to send Command or data to the LCD.	In 4-Wire Mode Only 4 pins (0-3) is connected to MCU In 8-Wire Mode All 8 pins(0-7) are connected to MCU
8	Pin 15	LED Positive	LED Pin	Normal LED like operation to illuminate the LCD	Connected to +5V
9	Pin 16	LED Negative	LED Pin	Normal LED like operation to illuminate the LCD connected with GND.	Connected to ground

- ***4-bit and 8-bit Mode of LCD:***

The LCD can work in two different modes, namely the 4-bit mode and the 8-bit mode. In **4 bit mode** we send the data nibble by nibble, first upper nibble and then lower nibble. For those of you who don't know what a nibble is: a nibble is a group of four bits, so the lower four bits (D0-D3) of a byte form the lower nibble while the upper four bits (D4-D7) of a byte form the higher nibble. This enables us to send 8 bit data.

Whereas **in 8 bit mode** we can send the 8-bit data directly in one stroke since we use all the 8 data lines.

Now you must have guessed it, Yes 8-bit mode is faster and flawless than 4-bit mode. But the major drawback is that it needs 8 data lines connected to the microcontroller. This will make us run out of I/O pins on our MCU, so 4-bit mode is widely used. No control pins are used to set these modes. It's just the way of programming that change.

- ***Read and Write Mode of LCD:***

As said, the LCD itself consists of an Interface IC. The MCU can either read or write to this interface IC. Most of the times we will be just writing to the IC, since reading will make it more complex and such scenarios are very rare. Information like position of cursor, status completion interrupts etc. can be read if required, but it is out of the scope of this tutorial.

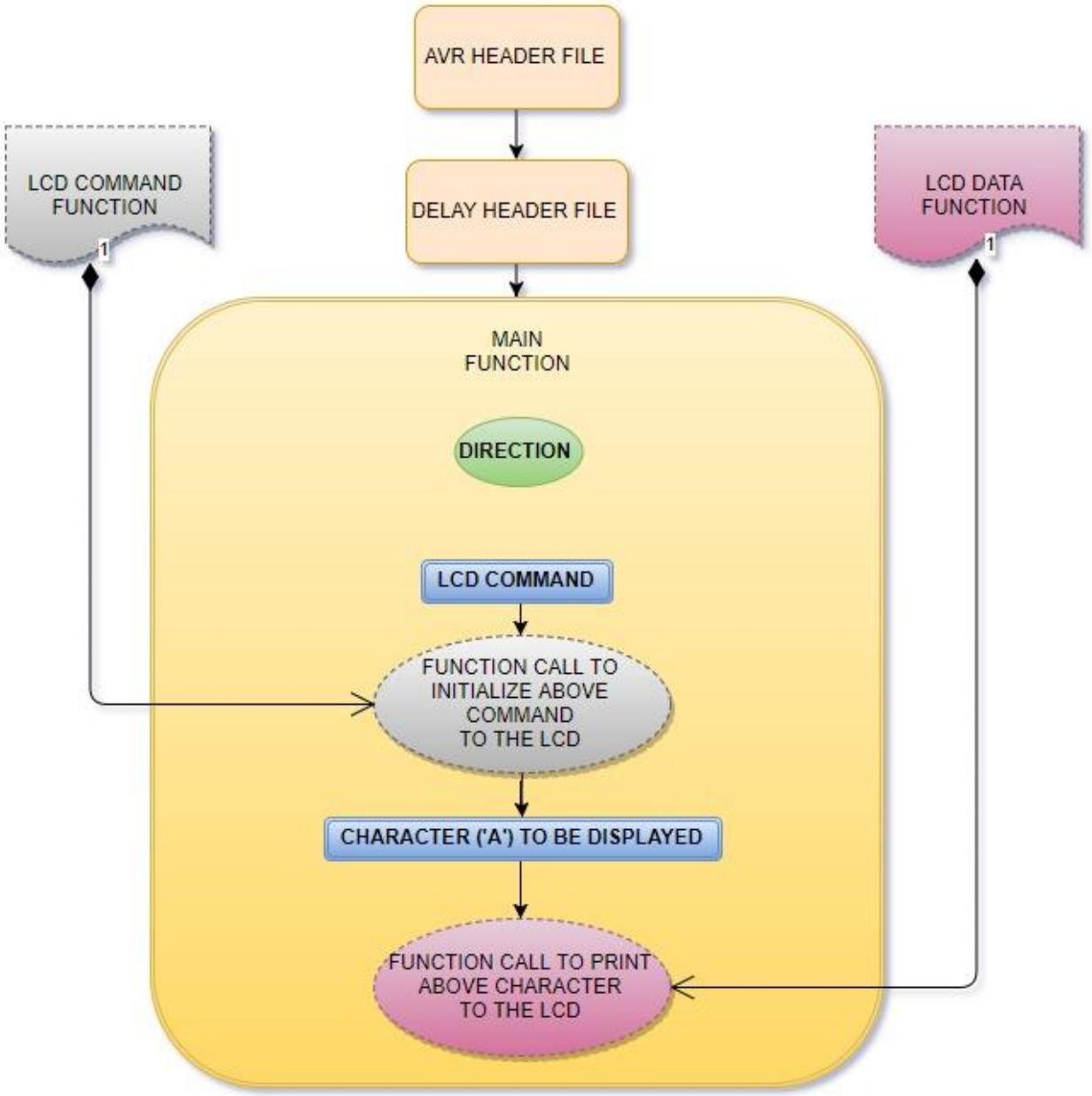
The Interface IC present in most of the LCD is **HD44780U;** in order to program our LCD we should learn the complete datasheet of the IC

LCD Commands

There are some preset commands instructions in LCD, which we need to send to LCD through some microcontroller. Some important command instructions are given below:

No.	Command	Hex value
1	Function Set: 8-bit, 1 Line, 5x7 Dots	0x30
2	Function Set: 8-bit, 2 Line, 5x7 Dots	0x38
3	Function Set: 4-bit, 1 Line, 5x7 Dots	0x20
4	Function Set: 4-bit, 2 Line, 5x7 Dots	0x28
5	Entry Mode	0x06
6	Display off Cursor off	0x08
7	Display on Cursor on	0x0E
8	Display on Cursor off	0x0C
9	Display on Cursor blinking	0x0F
10	Shift entire display left	0x18
11	Shift entire display right	0x1C
12	Move cursor left by one character	0x10
13	Move cursor right by one character	0x14

LCD programming flow chart



LCD PROGRAMMING IN 4 BIT MODE

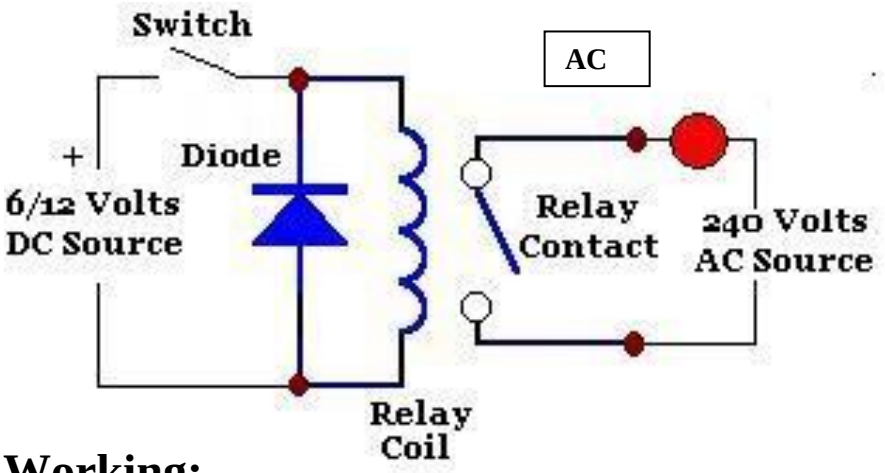
```
#define F_CPU 8000000UL                 // 8MHz frequency
#include <avr/io.h>                     // Avr header file
#include <util/delay.h>                 //Delay header file

void lcd_cmd(unsigned char cmd);        //Function for lcd commands
void lcd_data(unsigned char data);      // Function for lcd data
void lcd_init();                        // Function for lcd initialization



void lcd_data(unsigned char data)
{
        PORTD = data&(0xf0);            //Data in 4 bit mode
        PORTB |=(1<<PB4);               //RS = 1
        PORTB|=(1<<PB5);                //EN = 1
        _delay_ms(5);                   //To perform read and write operation
        PORTB&=~(1<<PB5);               //EN = 0

        PORTD = (data<<4)&(0xf0);       //For next 4 bit
        PORTB |=(1<<PB4);               //RS = 1
        PORTB|=(1<<PB5);                //EN = 1
        _delay_ms(5);                   //To perform read and write operation
        PORTB&=~(1<<PB5);               //EN = 0
}
```

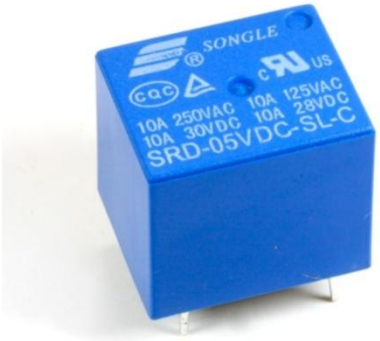
```
void lcd_cmd(unsigned char cmd)
{
        PORTD = cmd&(0xf0);             //Data in 4 bit mode
        PORTB &=~(1<<PB4);              //RS = 0
        PORTB|=(1<<PB5);                //EN = 1
        _delay_ms(5);                   //To perform read and write operation
        PORTB&=~(1<<PB5);               //EN = 0

        PORTD = (cmd<<4)&(0xf0);        //For next 4 bit
        PORTB &=~(1<<PB4);              //RS = 0
        PORTB|=(1<<PB5);                //EN = 1
        _delay_ms(5);                   //To perform read and write operation
        PORTB&=~(1<<PB5);               //EN = 0
}

void lcd_init()
{
        DDRD = DDRB =  0xff;            //Output direction for LCD
        lcd_cmd(0x02);                  //Return home
        lcd_cmd(0x28);                  //Function Set: 4-bit, 2 Line, 5x7 Dots
        lcd_cmd(0x0e);                  //Display on Cursor on
        lcd_cmd(0x01);                  //clear LCD
}
```

```
int main()
{
lcd_init();                    // initializing LCD function
lcd_data('W');                 //To display 'W' on LCD
lcd_data('E');                 //To display 'E' on LCD
lcd_data('L');                 //To display 'L' on LCD
lcd_data('C');                 //To display 'C' on LCD
lcd_data('O');                 //To display 'O' on LCD
lcd_data('M');                 //To display 'M' on LCD
lcd_data('E');                 //To display 'E' on LCD
return 0;
}
```

RELAY

A **relay** is an electromagnetic switch operated by a relatively small electric current that can turn on or off a much larger electric current.





Working:

A **relay** is an electromagnetic switch operated by a relatively small electric current that can turn on or off a much larger electric current. When a current flows through the coil, an electro-magnetic field is set up and triggers the switch for completing the circuit. With help of relays we can control high current and voltage rated device like home appliances, servo motors etc. with low voltage controller.

Introduction to Relay Driver IC

- **Relay driver circuit**

A Relay is an electro-magnetic switch that will be used whenever we want to use a low voltage circuit to switch a light bulb ON and OFF which is connected to 220V mains supply.

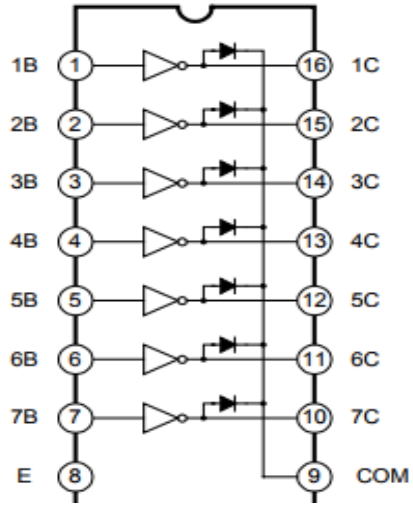
- **ULN2003A**

The ULN2003A is an array of seven NPN Darlington transistors capable of 500 mA, 50 V output. It features common-cathode fly back diodes for switching inductive loads. It can come in PDIP, SOIC, SOP or TSSOP packaging.

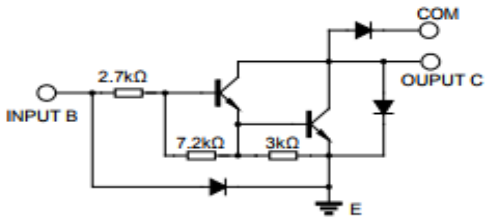
FEATURES

* 500mA rated collector current(Single output)
* High-voltage outputs: 50V
* Inputs compatibale with various types of logic.
* Relay driver application

LOGIC DIAGRAM

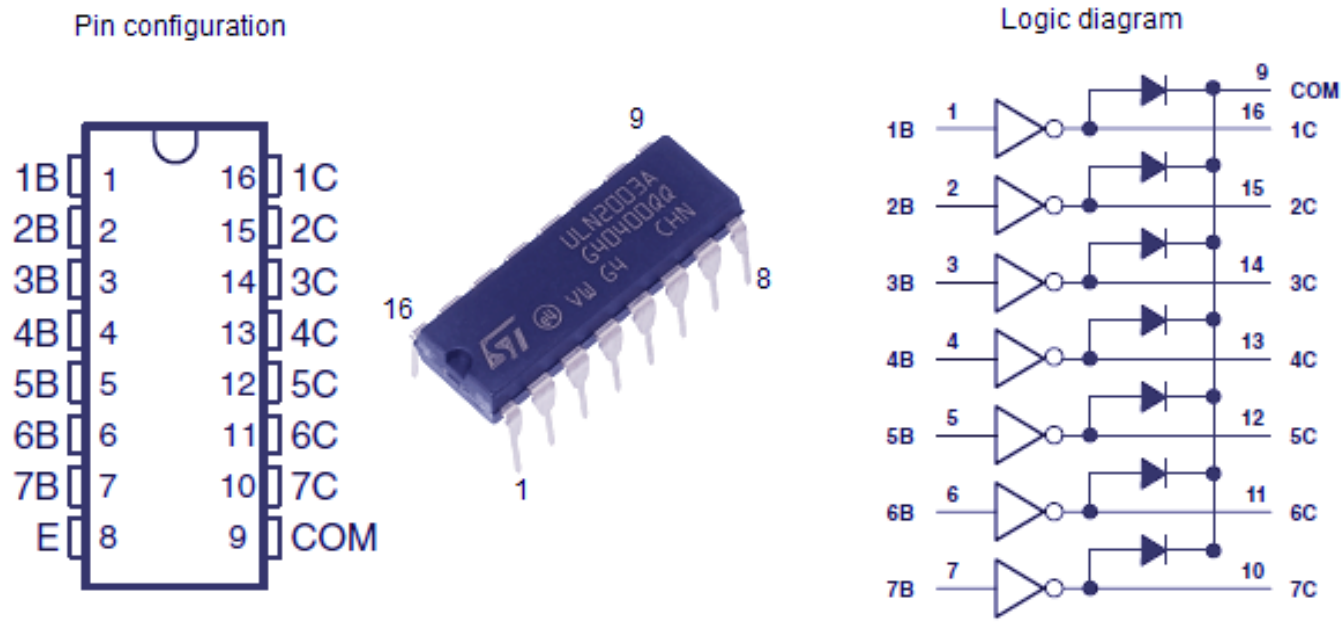


SCHEMATIC(EACH DARLINGTON PAIR)



ULN2003 is a high voltage and high current Darlington array IC. It contains seven open collector Darlington pairs with common emitters. A Darlington pair is an arrangement of two bipolar transistors.

ULN2003 belongs to the family of ULN200X series of ICs. Different versions of this family interface to different logic families. ULN2003 is for 5V TTL, CMOS logic devices. These ICs are used when driving a wide range of loads and are used as relay drivers, display drivers, line drivers etc. ULN2003 is also commonly used while driving Stepper Motors. Refer Stepper Motor interfacing using ULN2003.



ULN2003A driver IC pin configuration and internal logic diagram

www.circuitstoday.com

Each channel or Darlington pair in **ULN2003** is rated at 500mA and can withstand peak current of 600mA. The inputs and outputs are provided opposite to each other in the pin layout. Each driver also contains a suppression diode to dissipate voltage spikes while driving inductive loads. The schematic for each driver is given below:

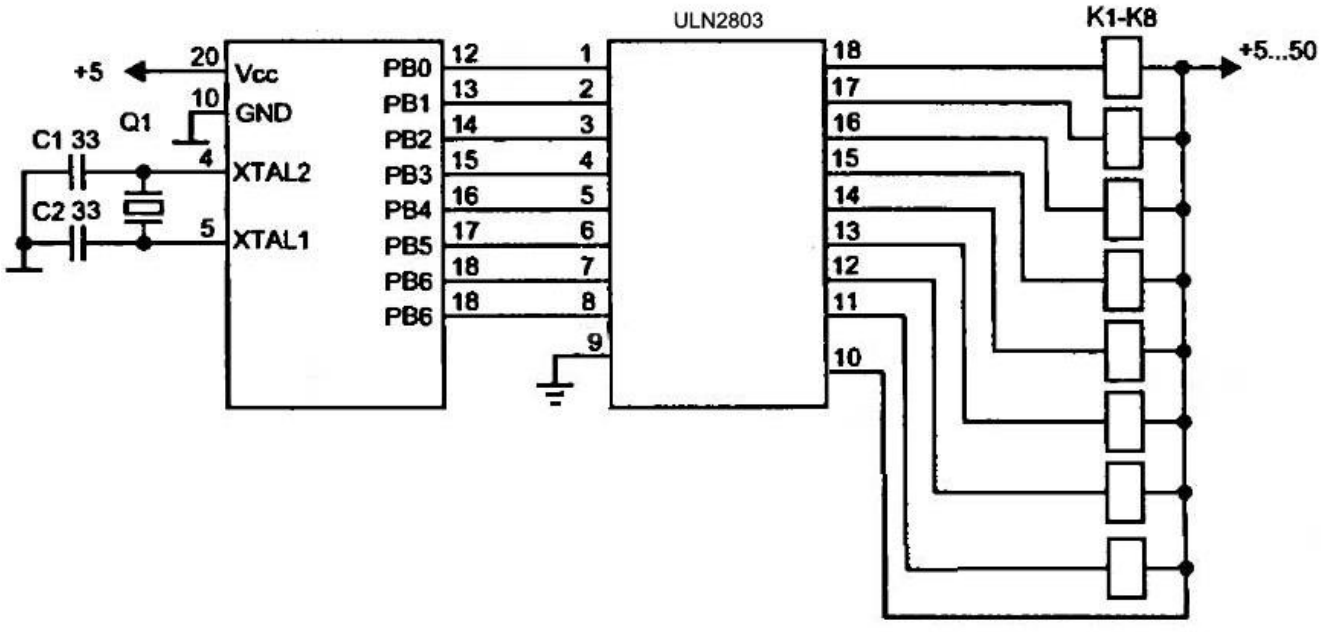
Pin No	Function	Name
1	Input for 1st channel	Input 1
2	Input for 2nd channel	Input 2
3	Input for 3rd channel	Input 3
4	Input for 4th channel	Input 4
5	Input for 5th channel	Input 5
6	Input for 6th channel	Input 6
7	Input for 7th channel	Input 7
8	Ground (0V)	Ground
9	Common freewheeling diodes	Common
10	Output for 7th channel	Output 7
11	Output for 6th channel	Output 6
12	Output for 5th channel	Output 5

13	Output for 4th channel	Output 4
14	Output for 3rd channel	Output 3
15	Output for 2nd channel	Output 2
16	Output for 1st channel	Output 1

- Pin Description:

- **Use of ULN 2803**

ULN2803 is a High voltage, high current Transistor Array IC used especially with Microcontrollers where we need to drive high power loads. Thick IC consists of a eight NPN Darlington connected transistors with common Clamp diodes for switching the loads connected to the output.

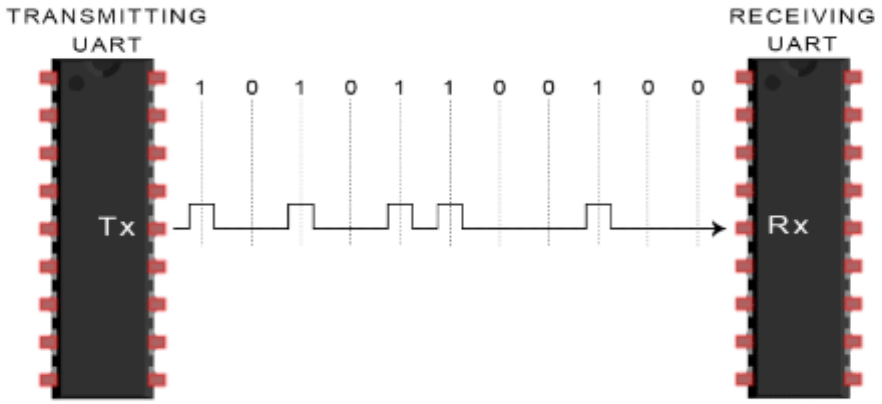


- **UART** stand for universal asynchronous Receiver Transmission. This is of the synchronous type i.e. the data bits are not with the clock pulses. In this method a single byte is transferred at a time.

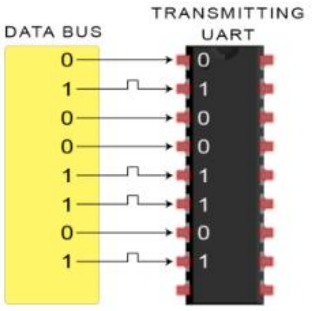
We can communicate between two Microcontrollers, between two PCs and between PCs and Microcontroller.

Basics of communication

- **Serial Communication: -** is the process of sending data one bit at a time.



- **Parallel Communication: -** is the process in which a number of data send and receive at a time.

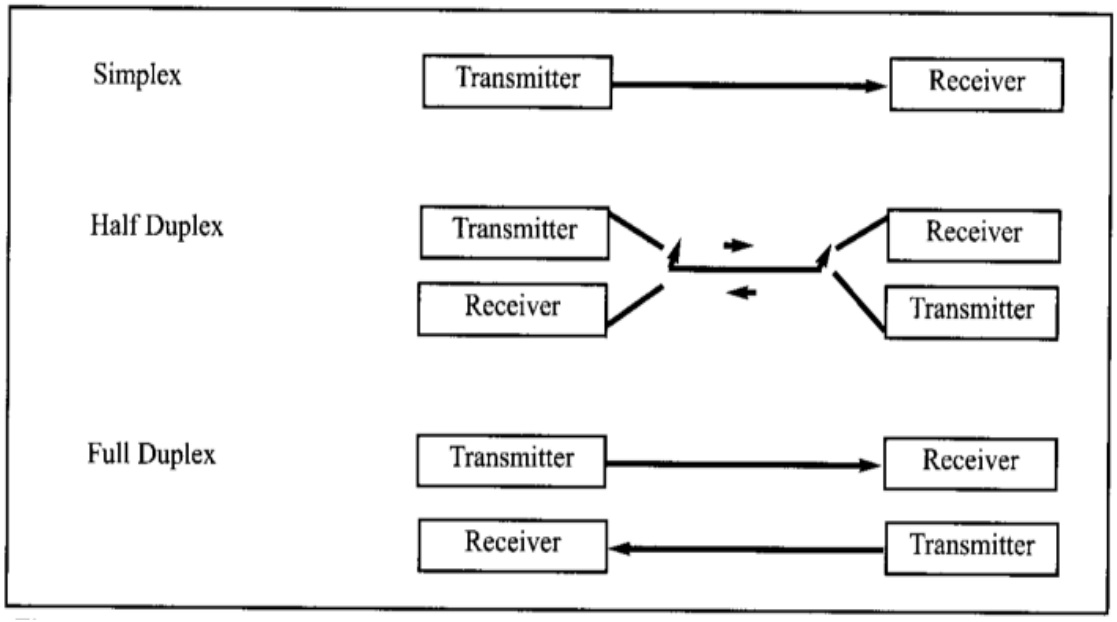


- **Advantages & Disadvantage of serial & parallel communication**

Serial communication	**Parallel communication**
1. Serial communication is used to transfer data at long distance	Parallel communication is used to transfer data at Short distance.
2. Slower	Faster
3. Cheaper as require less wire.	Costlier as require bunch of wire.
4. Example: Telephone	Example:- Printer

- **Mode of Communication:**
  - **Simplex:** In this method, we only send the data. Printer is an example of simplex communication.
  - **Half Duplex:** is used to communication where only... one side can talk at a time. Once one side has finished transmitting its data, the other side can respond. Only one node can talk at a time.
  - **Full Duplex:**

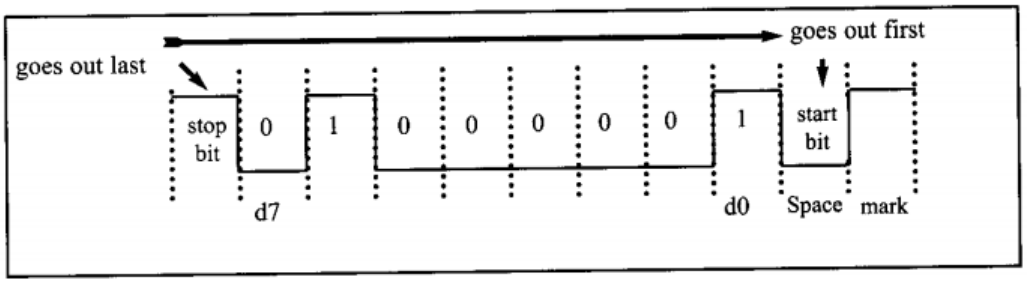
    This is used to describe communication where both sides are able to send and receive data at the same time. In these cases, there is no danger of a collision and therefore the transfer of data is completed much faster.



Terminology:

- **Baud rate:** Rate of data transfer in serial communication is called bps (bit per sec). However, rate of signal change in modem is known as baud rate. It is quite similar.

- **Data Framing:** In asynchronous data communication is widely used for character oriented transmission while block-orient data transfer is for synchronous method. In asynchronous method each character is placed between start and stop bit.
  - **START:** Always 0(LOW) bit.
  - **STOP:** Always 1(HIGH) bit. It can be of 2 bit.



- **Data framing**
  - **Voltage level converter:**

It is necessary to allow compatibility among data communication among various peripheral. As computer work on RS232 logic, in which 1 represented as -3 to -25v and 0 represented as +3 to +25. Whereas microcontroller work on TTL level, in this 1 represent as 2 to 5V and 0 represent as 0 to 0.8V.

- **Baud Rate:**

Rate of data transfer in serial communication is called baud rate. We need to have same baud rate to communicate between peripheral. We set the baud rate in **UBRR Resistor.**

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Table 11-3: Some PC Baud Rates in HyperTerminal

1,200
2,400
4,800
9,600
19,200
38,400
57,600
115,200

Desire BaudRate= F_CPU/(16(X+1))

=8MHZ/(16(X+1))

X= (500/Desire BaudRate)-1

- **Different value of Baud rate**

Table 11-4: UBRR Values for Various Baud Rates (Fosc = 8 MHz, U2X = 0)

Baud Rate	UBRR (Decimal Value)	UBRR (Hex Value)
38400	12	C
19200	25	19
9600	51	33
4800	103	67
2400	207	CF
1200	415	19F

UCSR Registers and USART Configurations

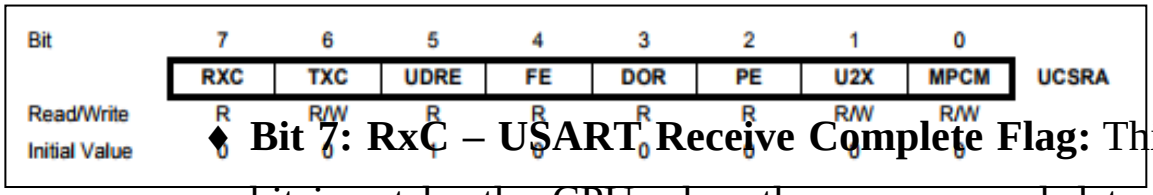
- **UDR: USART Data register (16-bit)**

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

The USART Transmit Data Buffer Register and USART Receive Data Buffer Registers share the same I/O address referred to as USART Data Register or UDR. The Transmit Data Buffer Register (TXB) will be the destination for data written to the UDR Register location. Reading the UDR Register location will return the contents of the Receive Data Buffer Register (RXB).

For 5-, 6-, or 7-bit characters the upper unused bits will be ignored by the Transmitter and set to zero by the Receiver.

- **UCSRA: USART Control and Status register A (8-bit)**



- **Bit 7: RxC – USART Receive Complete Flag:** This flag bit is set by the CPU when there are unread data in the Receive buffer and is cleared by the CPU when they receive buffer is empty. This can also be used to generate a Receive Complete Interrupt (see description of the RXCIE bit in UCSRB register).
- **Bit 6: TxC – USART Transmit Complete Flag:** This flag bit is set by the CPU when the entire frame in the Transmit Shift Register has been shifted out and there is no new data currently present in the transmit buffer (UDR). The TXC Flag bit is automatically cleared when a Transmit Complete Interrupt is executed, or it can be

cleared by writing a *one (yes, one and NOT zero)* to its bit location. The TXC Flag can generate a Transmit Complete Interrupt (see description of the TXCIE bit in UCSRB register).

- **Bit 5: UDRE – USART Data Register Empty:** The UDRE Flag indicates if the transmit buffer (UDR) is ready to receive new data. If UDRE is one, the buffer is empty, and therefore ready to be written. The UDRE Flag can generate a Data Register Empty Interrupt (see description of the UDRIE bit in UCSRB register). UDRE is set after a reset to indicate that the Transmitter is ready.
- **Bit 4: FE – Frame Error:** This bit is set if the next character in the receive buffer had a Frame Error when received (i.e. when the first stop bit of the next character in the receive buffer is zero). This bit is valid until the receive buffer (UDR) is read. The FE bit is zero when the stop bit of received data is one. Always set this bit to zero when writing to UCSRA.
- **Bit 3: DOR – Data Overrun Error:** This bit is set if a Data OverRun condition is detected. A Data OverRun occurs when they receive buffer is full (two characters), and a new start bit is detected. This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.

- **Bit 2: PE – Parity Error:** This bit is set if the next character in the receive buffer had a Parity Error when received and the parity checking was enabled at that point (UPM1 = 1). This bit is valid until the receive buffer (UDR) is read. Always set this bit to zero when writing to UCSRA.
- **Bit 1: U2X – Double Transmission Speed:** This bit only has effect for the asynchronous operation. Write this bit to zero when using synchronous operation. Writing this bit to one will reduce the divisor of the baud rate divider from 16 to 8 effectively doubling the transfer rate for asynchronous communication.
- **Bit 0: MPCM – Multi-Processor Communication Mode:** This bit enables the Multi-processor Communication mode. When the MPCM bit is written to one, all the incoming frames received by the USART Receiver that do not contain address information will be ignored. The Transmitter is unaffected by the MPCM setting. This is essential when the receiver is exposed to more than one transmitter, and hence must use the address information to extract the correct information.

- **UCSRB: USART Control and Status Register B (8-bit)**

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

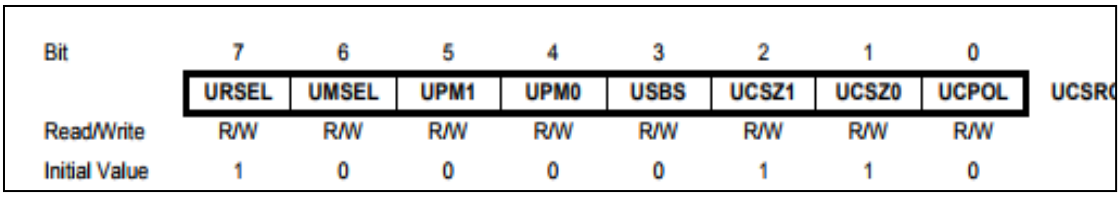
- **Bit 7: RXCIE – RX Complete Interrupt Enable:** Writing this bit to one enables interrupt on the RXC Flag. A USART Receive Complete interrupt will be generated only if the RXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the RXC bit in UCSRA is set. The result is that whenever any data is received, an interrupt will be fired by the CPU.
- **Bit 6: TXCIE – TX Complete Interrupt Enable:** Writing this bit to one enables interrupt on the TXC Flag. A USART Transmit Complete interrupt will be generated only if the TXCIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the TXC bit in UCSRA is set. The result is that whenever any data is sent, an interrupt will be fired by the CPU.
- **Bit 5: UDRIE – USART Data Register Empty Interrupt Enable:** Writing this bit to one enables interrupt on the UDRE Flag (remember – bit 5 in UCSRA?). A Data Register Empty interrupt will be generated only if the UDRIE bit is written to one, the Global Interrupt Flag in SREG is written to one and the UDRE bit in UCSRA is set. The result is that whenever

the transmit buffer is empty; an interrupt will be fired by the CPU.

- **Bit 4: RXEN – Receiver Enable:** Writing this bit to one enables the USART Receiver. The Receiver will override normal port operation for the RxD pin when enabled.
- **Bit 3: TXEN – Transmitter Enable:** Writing this bit to one enables the USART Transmitter. The Transmitter will override normal port operation for the TxD pin when enabled.
- **Bit 2: UCSZ2 – Character Size:** The UCSZ2 bits combined with the UCSZ1:0 bits in UCSRC register sets the number of data bits (Character Size) in a frame the Receiver and Transmitter use. More information given along with UCSZ1:0 bits in UCSRC register.
- **Bit 1: RXB8 – Receive Data Bit 8:** RXB8 is the ninth data bit of the received character when operating with serial frames with nine data bits. It must be read before reading the low bits from UDR.
- **Bit 0: TXB8 – Transmit Data Bit 8:** TXB8 is the ninth data bit in the character to be transmitted when operating with serial frames with nine data bits. It must be written before writing the low bits to UDR.

- **UCSRC: USART Control and Status Register C (8-bit)**

The UCSRC register can be used as either UCSRC, or as UBRRH register. This is done using the URSEL bit.



Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

- **Bit 7: URSEL – USART Register Select:** This bit selects between accessing the UCSRC or the UBRRH Register. It is read as one when reading UCSRC. The URSEL must be one when writing the UCSRC.
- **Bit 6: UMSEL – USART Mode Select:** This bit selects between Asynchronous and Synchronous mode of operation.

UMSEL	Mode
0	Asynchronous Operation
1	Synchronous Operation

- **Bit 5:4: UPM1:0 – Parity** Mode: This bit helps you enable/disable/choose the type of parity.

UPM1	UPM0	Parity Mode
0	0	Disabled
0	1	Reserved
1	0	Enabled, Even Parity
1	1	Enabled, Odd Parity

- **Bit 3: USBS – Stop Bit Select:** This bit helps you choose the number of stop bits for your frame.

USBS	Stop Bit(s)
0	1-bit
1	2-bit

- **Bit 2:1: UCSZ1:0 – Character Size:** These two bits in combination with the UCSZ2 bit in UCSRB register helps choosing the number of data bits in your frame.

UCSZ2	UCSZ1	UCSZ0	Character Size
0	0	0	5-bit
0	0	1	6-bit
0	1	0	7-bit
0	1	1	8-bit
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Reserved
1	1	1	9-bit

- **Bit 0: UCPOL – Clock Polarity:** This bit is used for Synchronous mode only.

Programming

Let's make the structure our code – first initialize UART, then read from it, and then write to it.

- **Initializing UART**

Initializing UART involves the following steps:

1. Setting the Baud Rate.
2. Setting Data Size (5/6/7/8/9 bits)
3. Enabling Reception/ Transmission (In the TXEN and RXEN bits in UCSRB)
4. Setting Stop Bits.

```
// function to initialize UART
void uart_init (void)
{
  UBRRL = 0x33;                    // set baud rate (0x33 correspond to 9600 BaudRate)
 UCSRB|= (1<<TXEN)|(1<<RXEN);              // enable receiver and transmitter
 UCSRC|= (1<<UCSZ0)|(1<<UCSZ1) (1<<URSEL);   // 8bit data format
   }
```

- **Explanation:**

Firstly set the baud rate is as per the calculations in the UBBRL resistor according to Frequency and BaudRate of device.

Then we enable the TxEN and RxEN are set to 1 to enable UART transmission and reception.

The Bits UCSZ0 and UCSZ1 are used to select the number of data bits to be transmitted. As per the table, we have set the two bits to transmit/receive 8 bits.

One thing to note is that UCSRC and UBRRH share the same I/O location in the memory. So, the bit that controls the selection/switching of the registers is the URSEL Bit in the UCSRC register. This creates a lot of confusion, and if this bit is not handled properly, UART doesn't work! So pay attention to the code initialization.

If URSEL is zero during a write operation, the UBRRH value will be updated. If URSEL is one, the UCSRC setting will be updated. This means that while URSEL bit is 0, and even if we are addressing UCSRC, UBBRH will be addressed, which would in turn result in setting up of wrong, non standard baud rates, and everything getting messed up! Since the default value of URSEL is 0, it is safe to first use UBRRH, and then use the UCSRC Register.

- **Using the USART of AVR Microcontroller: Reading and Writing Data:**

Till now we have seen the basics of RS232 communication, the function of level converter and the internal USART of AVR micro. After understanding the USART of AVR we have also written a easy to use function to initialize the USART. That was the first step to use RS232. Now we will see how we can actually send/receive data via rs232. As this tutorial is intended for those who are never used USART we will keep the things simple so as to just concentrate on the "USART"

part. Of course after you are comfortable with usart you can make it more usable my using interrupt driven mechanism rather than "polling" the usart.

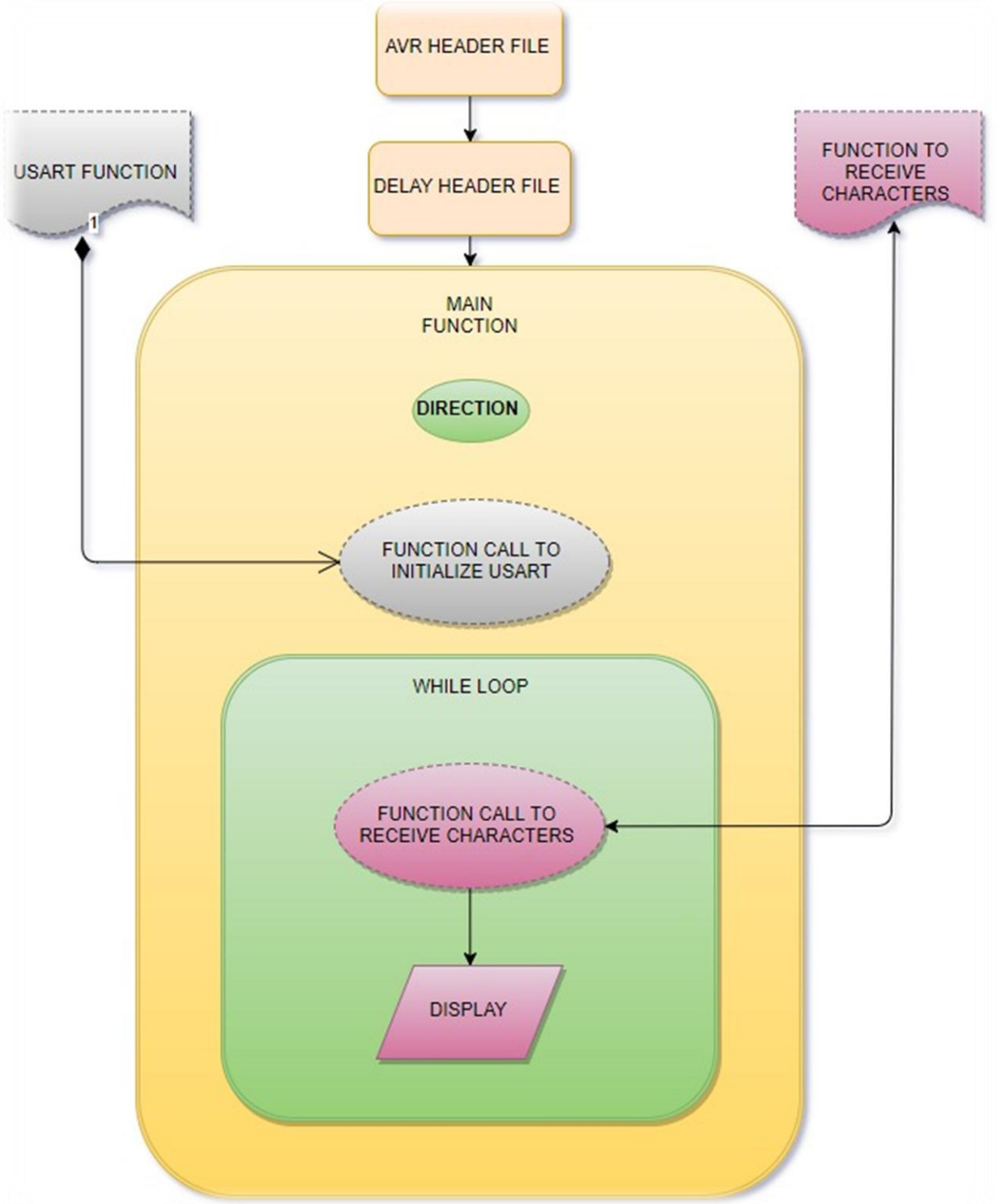
So let’s get started! In this section we will learn about polling method, we will make two functions:-

USARTReadChar (): To read the data (char) from the USART buffer.

USARTWriteChar (): To write a given data (char) to the USART.

These two functions will demonstrate the use of USART in the most basic and simplest way. After that you can easily write functions that can write strings to USART.

FLOW CHART OF USART TO RECEIVE ANY CHARACTER

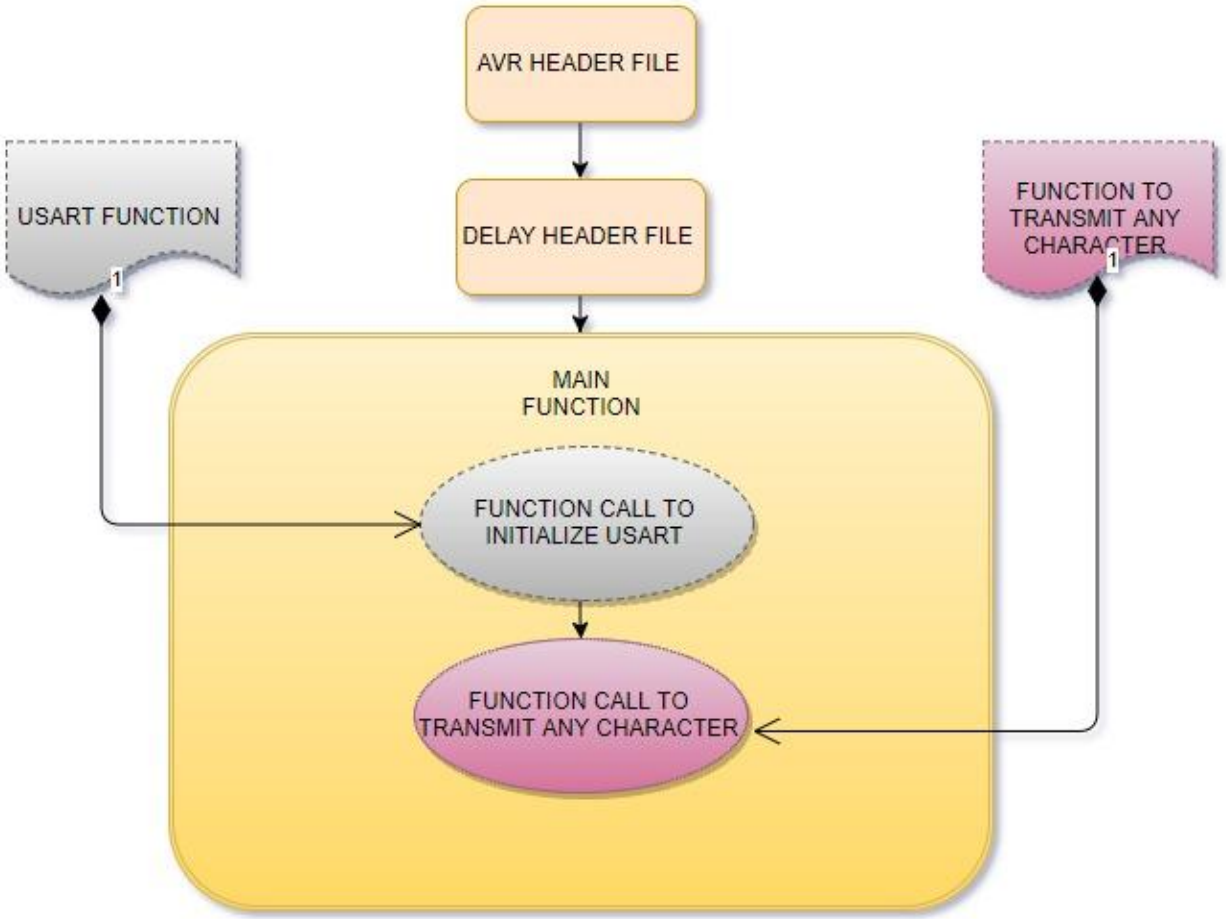


- **Reading from the USART: USARTReadChar() Function:**

This function will help you read data from the USART. For example if you use your PC to send data to your micro the data is automatically received by the USART of AVR and put in a buffer and bit in a register (UCSRA) is also set to indicate that data is available in buffer. It's now your duty to read this data from the register and process it, otherwise if new data comes in the previous one will be lost. So that wait until the RXC bit (bit no 7) in UCSRA is SET and then read the UDR register of the USART.

```
char USARTReadChar()
{
//Wait until a data is available
while (!(UCSRA & (1<<RXC)))
  {
    //Do nothing
  }
  //Now USART has got data from host
  //and is available is buffer
  return UDR;
}
```

- **Writing to the USART: USARTWriteChar() Function:**



This function will help you write a given character data to the USART. Actually we write to the buffer of USART and the rest is done by USART that means it automatically sends the data over RS232 line. One thing we need to keep in mind is that before writing to USART buffer we must first check that the buffer is free or not. It it's not we simply wait until it is free. If it's not free it means that USART is still busy sending some other data and once it finishes it will take the new data from buffer and start sending it.

Please not that the data held in the buffer is not the current data which the USART is busy sending. USART reads data from the buffer to its shift register which it starts sending and thus the buffer is free for your data. Every time the USART gets it data from buffer and thus making it empty it notifies this to the CPU by telling "**U**SART **D**ata **R**egister Ran **E**mpty" (UDRE). It does so by setting a bit (UDRE bit no 5) in UCSRA register.

So in our function we first wait until this bit is set (by the USART), once this is set we are sure that buffer is empty and we can write new data to it.

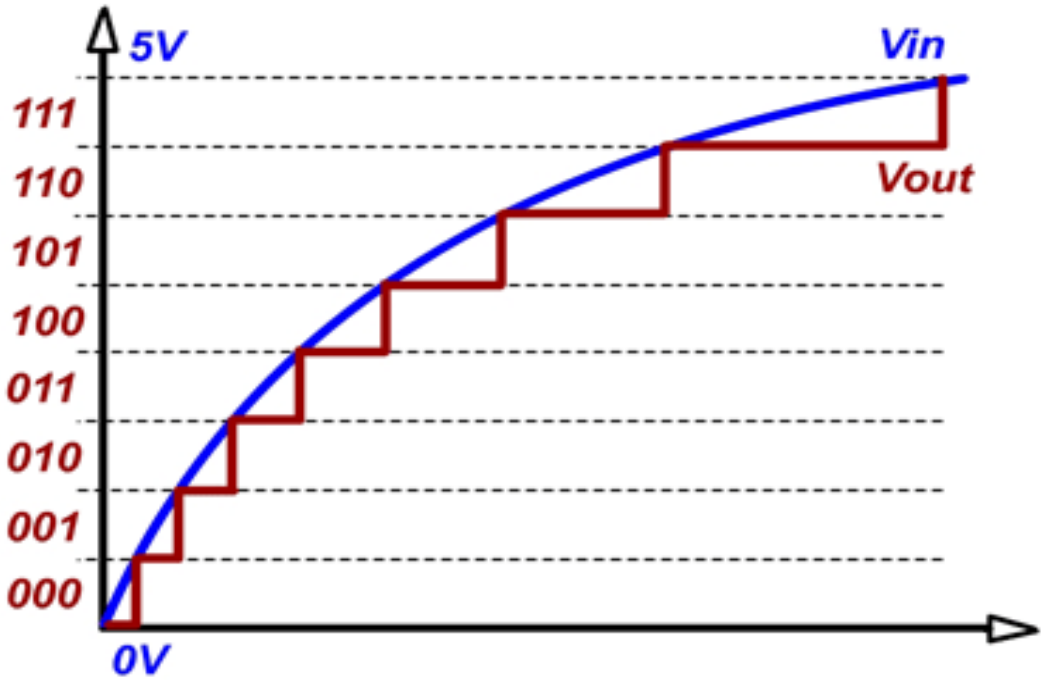
```
void USARTWriteChar (char data)

{

  //Wait until the transmitter is ready

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

  {

      //Do nothing

  }      //Now write the data to USART buffer

  UDR=data; }
```

Example:

```
#define F_CPU 8000000UL                                  // 8Mhz frequency
#include <avr/io.h>
#include <util/delay.h>



void tt(unsigned char t)                                  //function to transmit a character
{
        while(!(UCSRA&(1<<UDRE)));                        //wait until UDR is empty
        UDR=t;                                            //transmit character
}
void u_init()                                             //function to initialize USART
{
        UBRRL=51;                                         // 9600 BAUD RATE
        UCSRB|=(1<<TXEN);                                 // transmit enable
        UCSRC|=(1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);          //setting character size
}

int main(void)
{
        u_init();                                         // initializing USART
        tt('A');                                          // transmit ‘A’
}
```

ANALOG TO DIGITAL CONVERTER (ADC)

Analog-to-digital converter is a system that converts an **analog** signal, such as a sound, temperature, pressure, voltage, picked up by a sensor



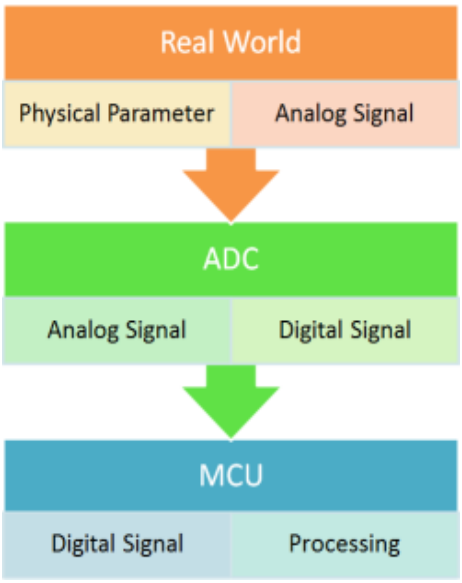
and convert into a **digital** signal.

For example, the temperature inside a boiler is around 800°C. During its light-up, the temperature never approaches directly to 800°C. If the ambient temperature is 400°C, it will start increasing gradually to 450°C, 500°C and thus reaches 800°C over a period of time. This is an analog data.

Now, we must process the data that we have received. But analog signal processing is quite inefficient in terms of accuracy, speed and desired output. Hence, we convert them to digital form using an Analog to Digital Converter (ADC).

- **Signal Acquisition Process**

  - In the **Real World**, a sensor senses any physical parameter and converts into an equivalent analog electrical signal.
  - For efficient and ease of signal processing, this analog signal is converted into a digital signal using an **Analog to Digital Converter (ADC)**.
  - This digital signal is then fed to the **Microcontroller (MCU)** and is processed accordingly.



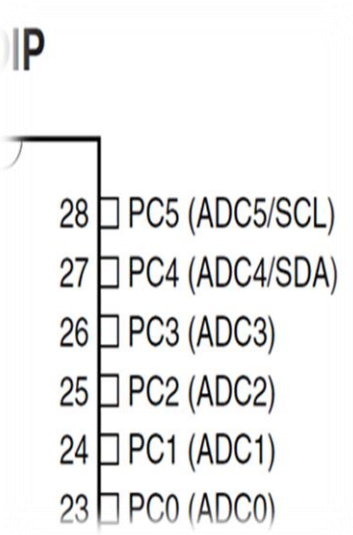
- **Interfacing ADC Sensors**

In general, sensors provide with analog output, but a MCU is a digital one. Hence we need to use ADC. For simple circuits, comparator op-amps can be used. But even this won't be required if we use a MCU. We can straightaway use the inbuilt ADC of the MCU. In ATMEGA8/16/32, PORTC contains the ADC pins.

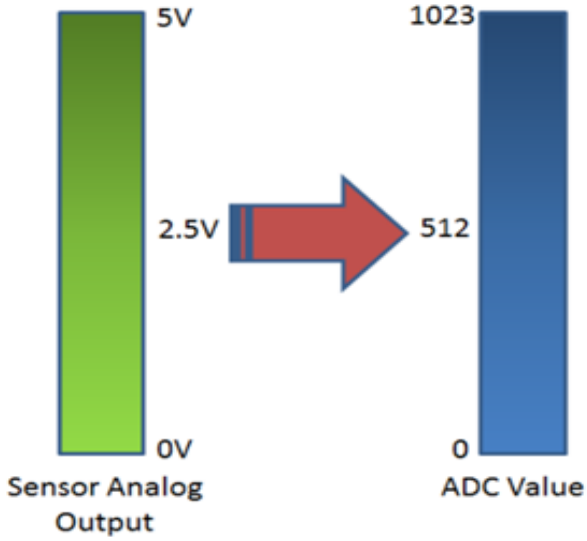
The ADC of the AVR

The AVR features inbuilt ADC in almost all its MCU. In ATMEGA8/16, PORTA contains the ADC pins. Some other features of the ADC are as follows:

- 10-bit Resolution
- 0.5 LSB Integral Non-linearity
- ±2 LSB Absolute Accuracy
- 13µs - 260µs Conversion Time
- Up to 15 kSPS at Maximum Resolution
- 6 Multiplexed Single Ended Input Channels
- 2 Additional Multiplexed Single Ended Input Channels (TQFP and QFN/MLF Package on
- Optional Left Adjustment for ADC Result Readout
- 0 - V_{CC} ADC Input Voltage Range
- Selectable 2.56V ADC Reference Voltage
- Free Running or Single Conversion Mode
- Interrupt on ADC Conversion Complete
- Sleep Mode Noise Canceler



- **7 channels:** imply that there are 7 ADC pins are multiplexed together. We can easily see that these pins are located across PORTC (PC0…PC6).
- **10 bit resolution:** implies that there are 2^10 = 1024 steps (as described below).



Methodology

Suppose we use a 5V reference. In this case, any analog value in between 0 and 5V is converted into its equivalent ADC value as shown above. The 0-5V range is divided into 2^10 = 1024 steps. Thus, a 0V input will give an ADC output of 0, 5V input will give an ADC output of 1023, whereas a 2.5V input will give an ADC output of around 512. This is the basic concept of ADC.
To those whom it might concern, the type of ADC implemented inside the AVR MCU is of Successive Approximation type.

- **Techniques and Resistor**

  - ♦ ADC Prescaler
  - ♦ ADC Registers – ADMUX, ADCSRA, ADCH, ADCL and SFIOR

ADC Prescaler

ADC operates within a frequency range of 50 kHz to 200 kHz. But the CPU clock frequency is much higher. So to achieve it, frequency division must take place. The prescaler acts as this division factor. It produces desired frequency from the external higher frequency. There are some predefined division factors – 2, 4, 8, 16, 32, 64, and 128. For example, a prescaler of 64 implies F_ADC = F_CPU/64. For F_CPU = 8MHz, F_ADC = 8M/64 = 250 kHz.

ADC Registers

- ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADMUX Register

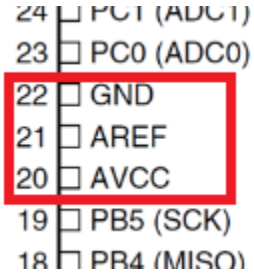
♦ Bits 7:6 – REFS1:0 – Reference Selection Bits –

These bits are used to choose the reference voltage. The following combinations are used.

Reference Voltage Selection

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

The ADC needs a reference voltage to work upon. For this we have three pins AREF, AVCC and GND. We can supply our own reference voltage across AREF and GND. For this, **choose the first option**. Apart from this case, we can either connect a capacitor across AREF pin or ground it to prevent from noise, or we may choose to leave it unconnected.



If we want to use the VCC (+5V), choose **the second option.** Or else, **choose the last option** for internal Vref.

♦ Bit 5 – ADLAR – ADC Left Adjust Result –

Make it '1' to Left Adjust the ADC Result. We will discuss about this a bit later.

- **Bits 4:0 – MUX3:0 – Analog Channel and Gain Selection Bits** – There are 7 ADC channels (PC0…PC6). Which one do we choose? Choose any one! It doesn't matter. How to choose? We can choose it by setting these bits. Since there are 4 bits, it consists of 2^4 = 16 different conditions as follows. However, we are concerned only with the first 7 conditions. Initially, all the bits are set to zero.

To initialize ADMUX, we write

- ADMUX = (1<<REFS0);

- **ADCSRA – ADC Control and Status Register A**

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 7 – ADEN – ADC Enable** –

As the name says, it enables the ADC feature. Unless this is enabled, ADC operations cannot take place across PORTC i.e. PORTC will behave as GPIO pins.

- **Bit 6 – ADSC – ADC Start Conversion** –

Write this to '1' before starting any conversion. This 1 is written as long as the conversion is in progress, after which it returns to zero. Normally it takes 13 ADC clock pulses for this operation. But when we call it for the first time, it takes 25 as it performs the initialization together with it.

- **Bits 2:0 – ADPS2:0 – ADC Prescaler Select Bits** –

The prescaler (division factor between XTAL frequency and the ADC clock frequency) is determined by selecting the proper combination from the following.

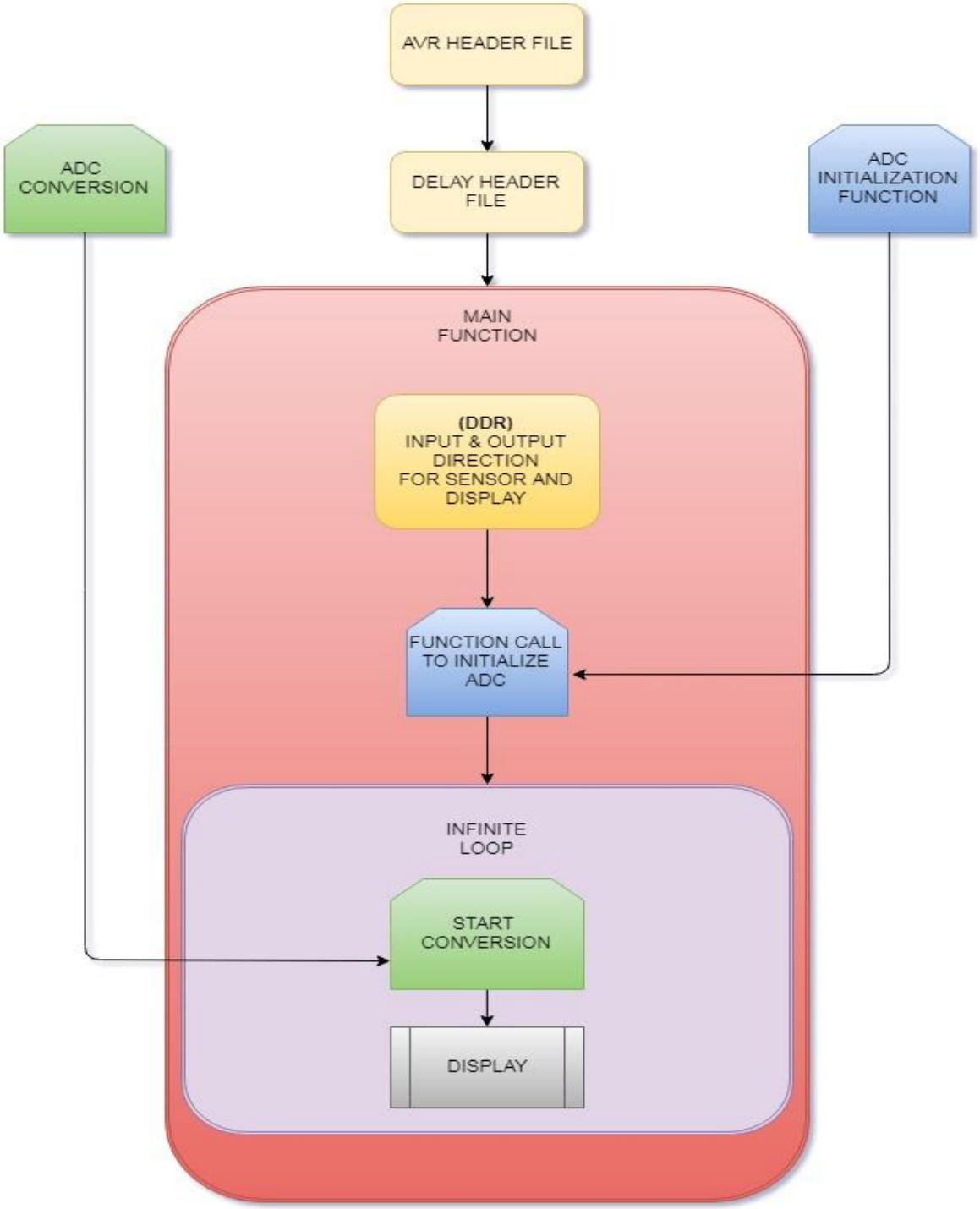
ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Assuming XTAL frequency of 8MHz and the frequency range of 50 kHz-200 kHz, we choose a prescaler of 64.

Thus, F_ADC = 8M/64 = 125 kHz.

Thus, we initialize ADCSRA as follows.

```
ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1);  // prescaler = 64
```



ADC FLOW CHART:

Programming

- **ADC Initialization**

  Let's initialize the adc as per our requirement and configuration.

Code:

```
void adc_init()
{
   // AREF = AVcc
   ADMUX = (1<<REFS0);

   // ADC Enable and prescaler of 64
   // 8000000/64 = 125000
   ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1);
}
```

Reading ADC Value

The following code segment reads the value of the ADC.

Code:

```
uint16_t adc_read(uint8_t ch)
{
  // select the corresponding channel 0~6
  // ANDing with '6' will always keep the value
  // of 'ch' between 0 and 6
  ch &= 0b0000111;  // AND operation with 6
  ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing


  // start single convertion
  // write '1' to ADSC
  ADCSRA |= (1<<ADSC);
```

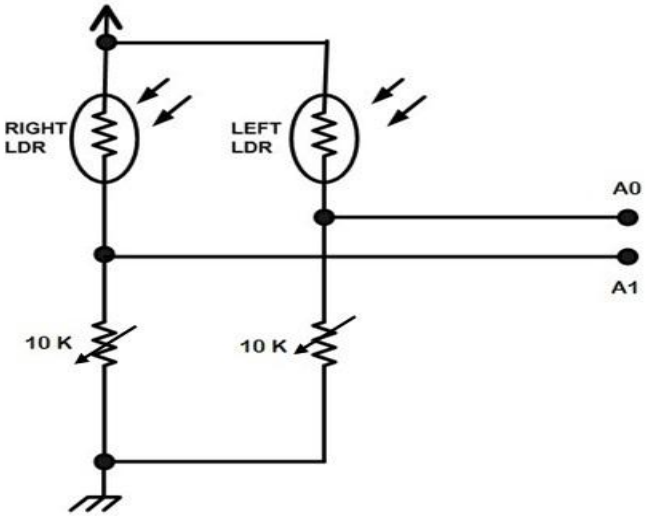
```
// wait for conversion to complete
// ADSC becomes '0' again
// till then, run loop continuously
while(ADCSRA & (1<<ADSC));
return (ADC); }
```

EXAMPLE:

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

#include <util/delay.h>

void adc()

{

        DDRC = 0;                       // input direction for sensor

        DDRB = 0xff;                    //output direction for display

        ADCSRA = 0x87;                  //to enable adc & select ck/128

        ADMUX = 0xc0;                   // 2.56V  Vref

}

int main(void)

{

        adc();                          //initialize adc

        while(1)

        {

                ADCSRA|=(1<<ADSC);              // start conversion

                while(!(ADCSRA&(1<<ADIF)));     //wait for conversion to finish

                PORTB=ADC;                      //analog value on PORTB

        }

}
```

INTRODUCTION TO LDR

LDR stand for **light dependent resistor.** It senses the intensity of light and accordingly changes its resistance. The resistance decreases exponentially as the light intensity increases and resistance increases as light intensity decreases correspondingly we calibrate the LCD value.



Working:

Note that since the ADC returns values in between 0 and 1023, for dark conditions, the value should be low (below 100 or 150) whereas for bright conditions, the value should be quite high (above 900).

INTRODUCTION TO WIFI

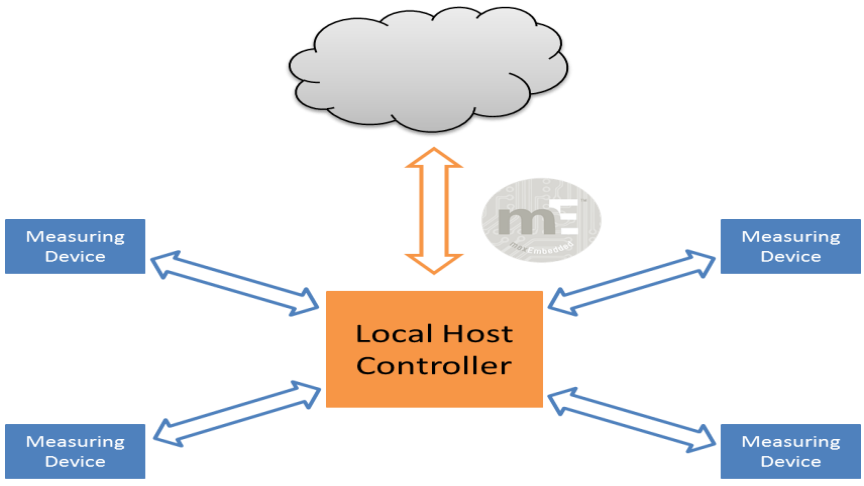
Wi-Fi stands for wireless fidelity. This means you can access or connect to a network using radio waves, without needing to use wires. We are using wifi in our daily life use in our mobile phone, offices etc. Now we are going to study about a wifi module, through which we can connect internet to any device. This module is known as **ESP2866.**

Introduction to ESP 8266 module

ESP8266 is an UART to Wi-Fi module, cheap and easy way to connect any small microcontroller wirelessly to Internet. We can now send sensor data directly from your device to the cloud without sending it through a local host.

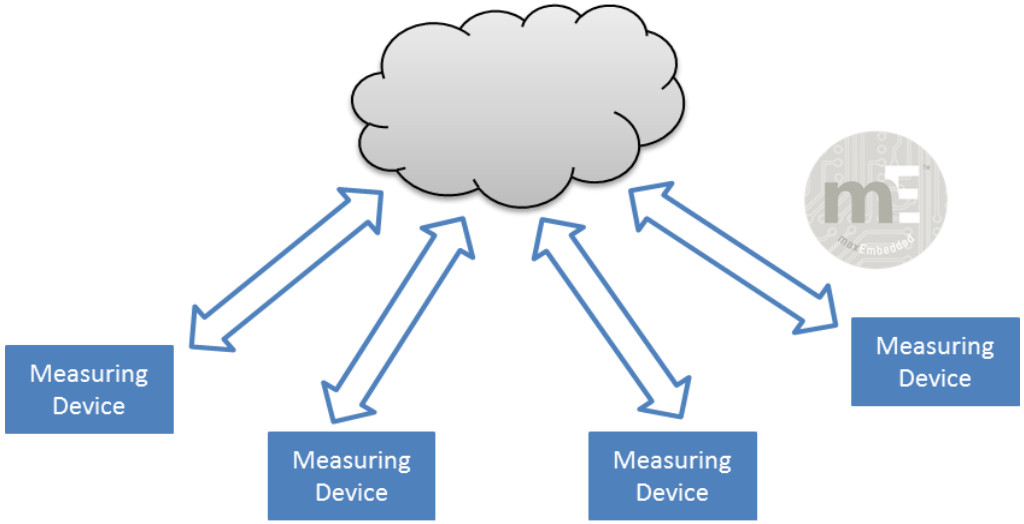
Let's take two scenarios for our understanding:

Scenario 1: WE have a local host (say a Microcontroller) connected

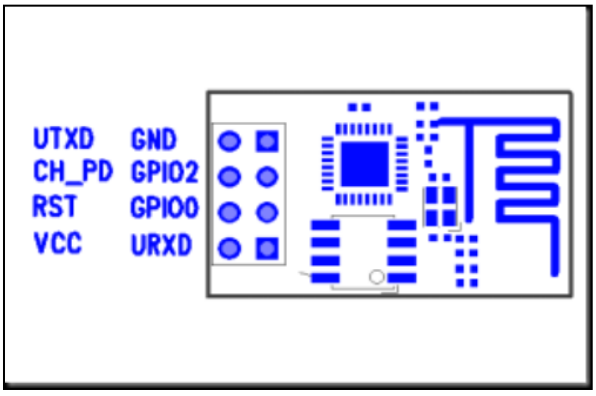


to the internet. That's it – no other device has internet connection. Say if your fridge wants to talk to the internet, then it needs to connect to your local host first, which then sends that data to the internet. What a pain!

Scenario 2: Now take another instance in which our fridge wants to talk to internet. In that we use the ESP8266 to connect the fridge to the internet. Now your fridge can directly talk to the internet without the need for a local host.



- **Pin Description of WIFI module(ESP2866)**



1. **Vcc**: 3.3V, up to 300 mA
2. **GND**: Ground
3. **Tx**: UART Tx of the module, to be connected to the Rx of the microcontroller
4. **Rx**: UART Rx of the module, to be

connected to the Tx of the microcontroller

5. **RST**: Reset/Restart, pull to GND to restart
6. **CH_PD**: Chip enable, used for flash boot.
7. **GPIO0**: pulled low for update mode
8. **GPIO2**: -

- **TTL Conversion**

We used a USB to TTL converter for connectivity between PC & WIFI module. The converter has 3.3v output which is used to power the ESP2866. It has also a +5v supply out. Care has to be taken with power supply as the unit is to be powered at 3.3V and using 5V will probably kill it.

- **Connection between ESP2866 and TTL**

WIFI Module		***USB-TTL***
Vcc	:	3.3v
Gnd	:	Gnd
Tx	:	Rx
Rx	:	Tx
CH_PD	:	connected to 3.3v to enable chip firmware boot.
RST	:	connected to 3.3v through 3k3 resistors.

Introduction to AT command

AT commands are instructions used to control a modem. AT is the abbreviation of Attention. We also require AT command to configure our ESP2866 module. So, it can communicate between microcontroller and internet.

Note that the starting "AT" is the prefix that informs the modem about the start of a command line. It is not part of the AT command name. For example, D is the actual AT command name in ATD.

- ✓ AT
  OK
- ✓ ATE0
  OK
- ✓ AT+CWLAP (list the available wifi connections)
  OK
- ✓ AT+CWJAP=”wifi name”,”password” (Connect to the wifi)
  CONNECTED
  OK
- ✓ AT+CIPMUX=1
  OK
- ✓ AT+CIPSTART=4,"TCP","184.106.153.149",80 (This is to connect with web page)
  4, CONNECT
  OK
- ✓ AT+CIPSEND=4,44 (4 is a channel and 44 is the number of characters)
  OK
  > GET /update?key=4DW2G8J5JKXZ8C9L&field1=88 (string we have to pass from here to update DATA on server)
  Recv 44 bytes
  SEND OK
- ✓ AT+CIPCLOSE=4 (To close TCP connection)
  4,CLOSED OK

INTRODUCTION TO TCP/IP

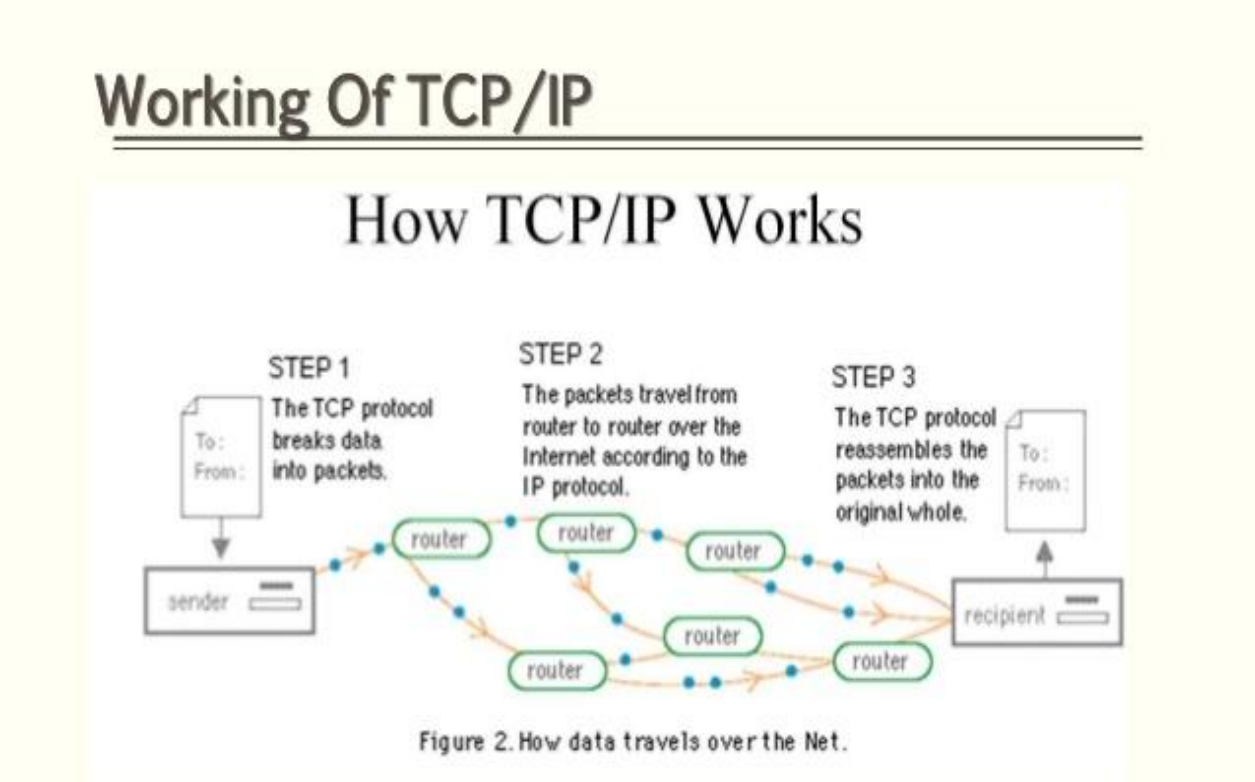
TCP stands for Transmission Control Protocol whereas **IP** Internet Protocol and refers to a number of protocols or set of instruction.

The IP (Internet Protocol) handles the address part of each packet so that it gets to the right destination. The **IP** is used by TCP to transport them from one network to another. Think of IP as a sort of high-way that allows other protocols to get on and find their way to other computers. TCP are the trucks on the highway, and the loads they are carrying are protocols such as HTTP, FTP and more. Here is explanation about how IP works:-

- Your computer has an IP. When you launch a browser and try to load a page, the request is sent out with your computer's IP address as the return address (where the webpage should be sent).
- The request goes looking for the website's address (IP). When it finds it, the website (as a virtual device) is now talking to your computer... sending it's data to compile the website.
- You see on your browser. So, you're device needs an IP so the website knows where to send data, and the website needs an IP so your computer knows where to get the data.

TCP is a connection-oriented protocol, which means a connection is established and maintained until the application programs at each end have finished exchanging messages. . Here is explanation about how TCP works:-

- It determines how to break application data into packets that networks can deliver sends packets to and accepts packets from the network layer.
- Manages flow control of data packet.
- It is meant to provide error-free data transmission—handles retransmission of dropped or garbled packets as well as acknowledgement of all packets that arrive.



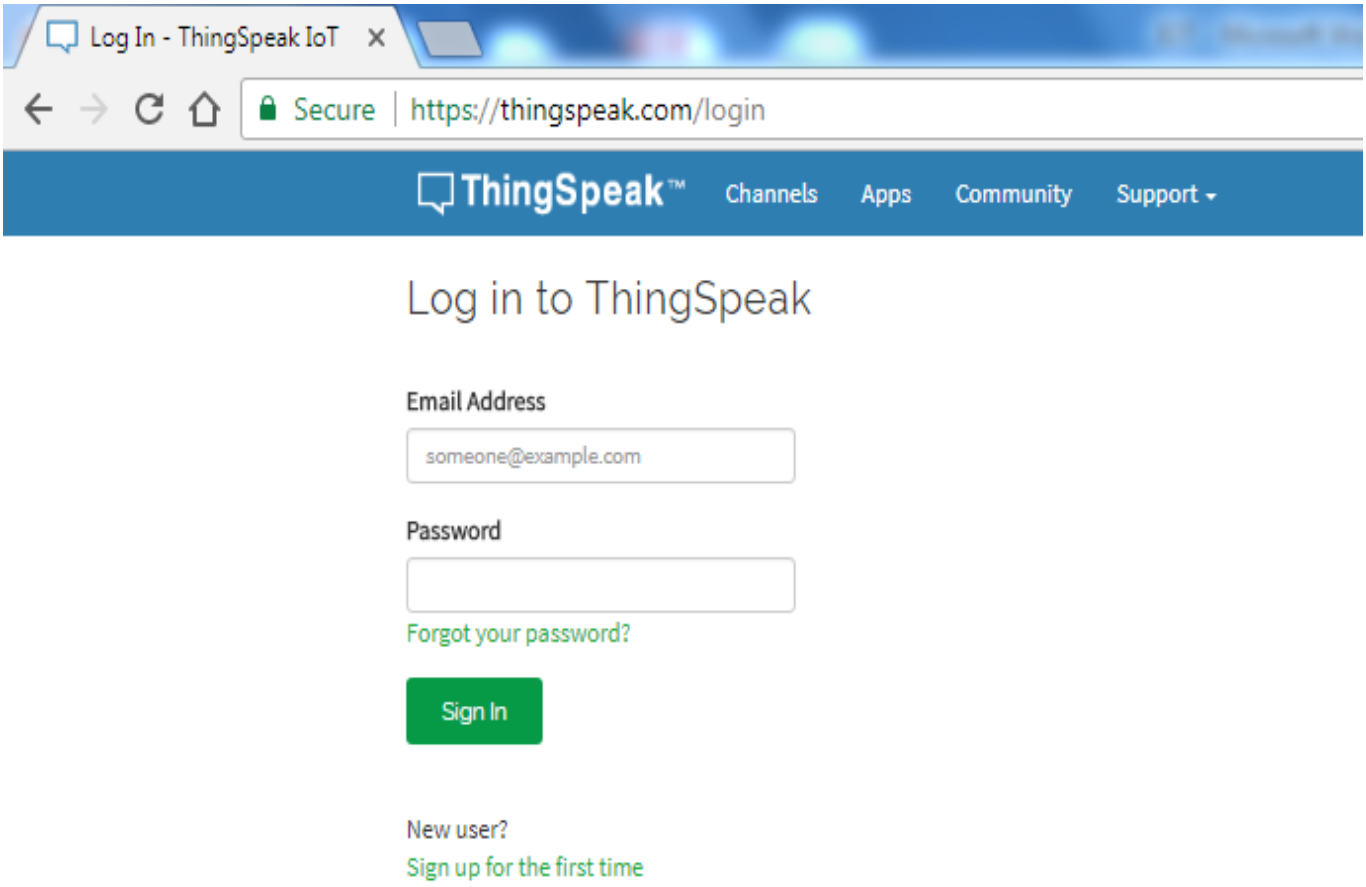
Note: How Internet works?

Remember, IP is required to connect all networks. TCP is a mechanism that allows us to transfer data safely, and HTTP which utilizes TCP to transfer its data, is a specific protocol used by Web servers and clients.

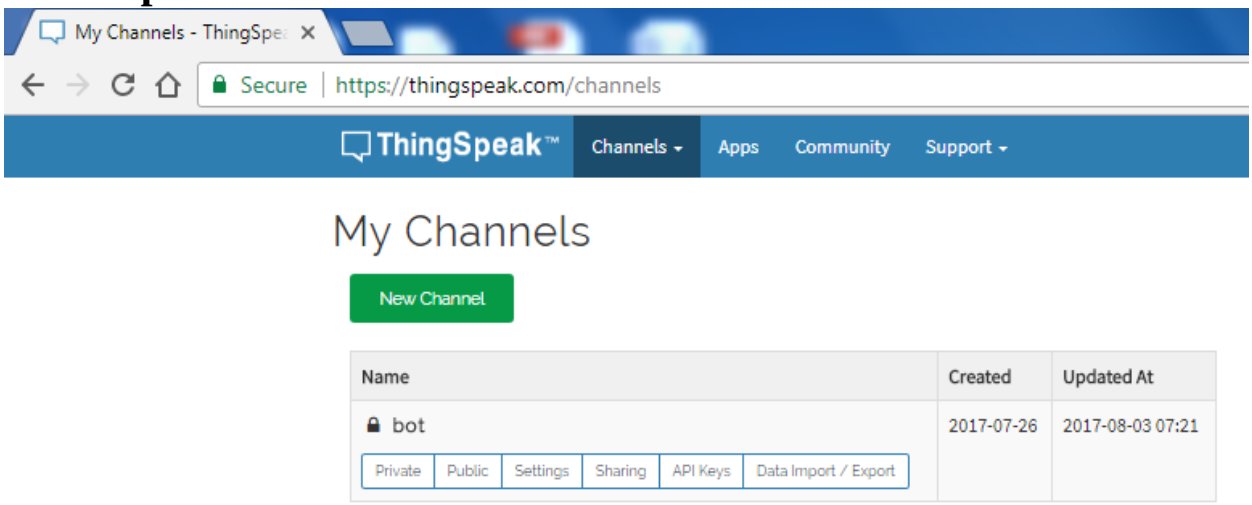
IOT Platform (thing speak)

- Firstly we have to make an account on website
- EX: thingspeak.com
- There are some steps given below:

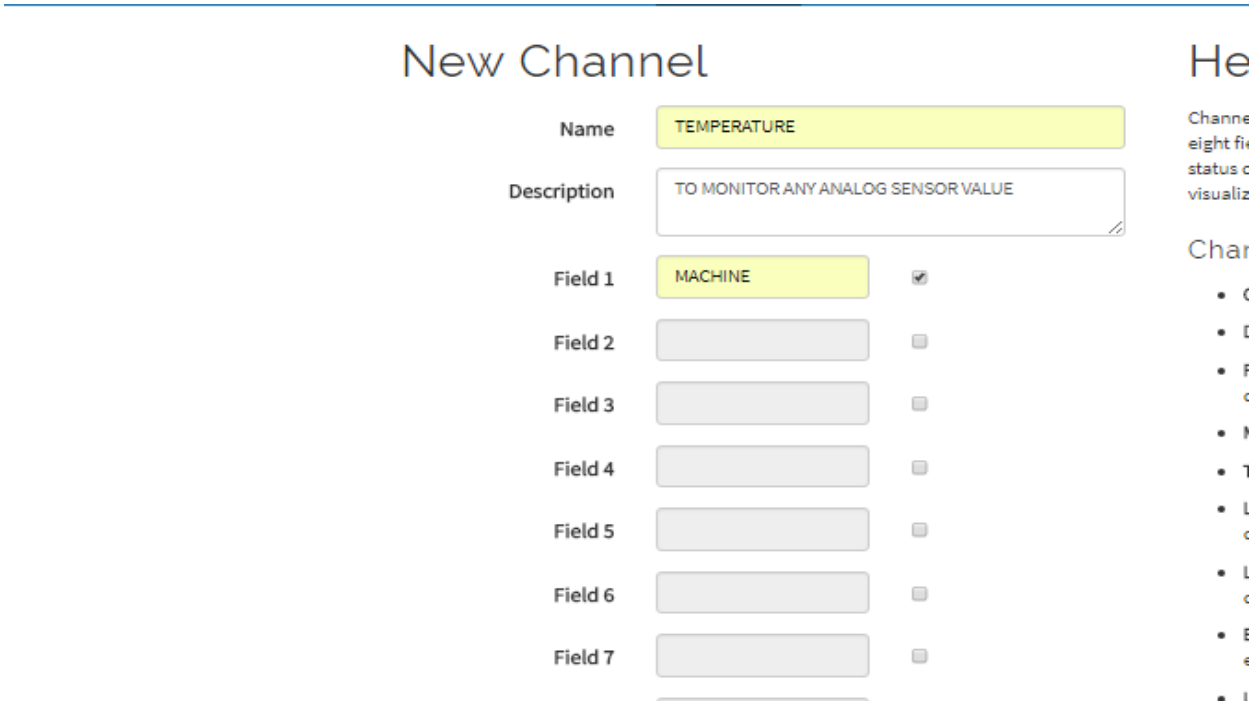
Step 1 → Create an account on “thingspeak.com”.



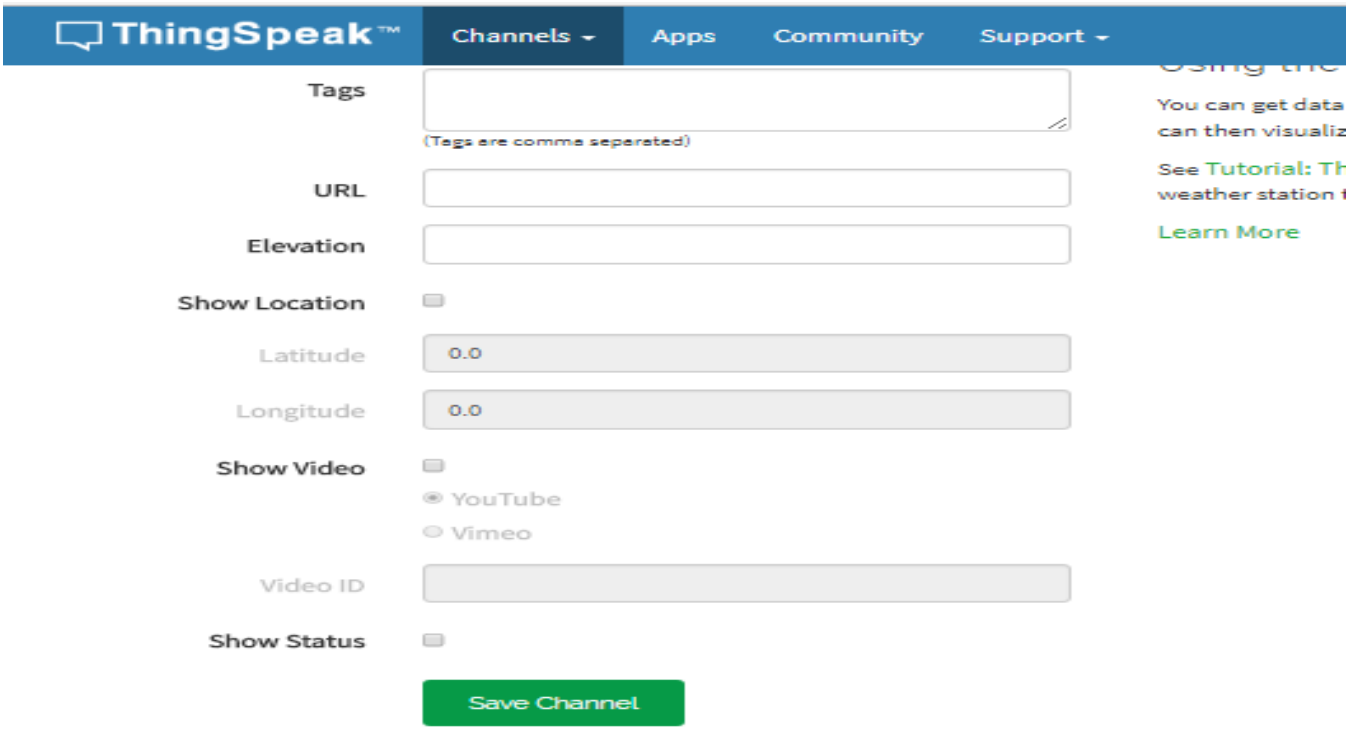
Step 2 → Click on new channel".

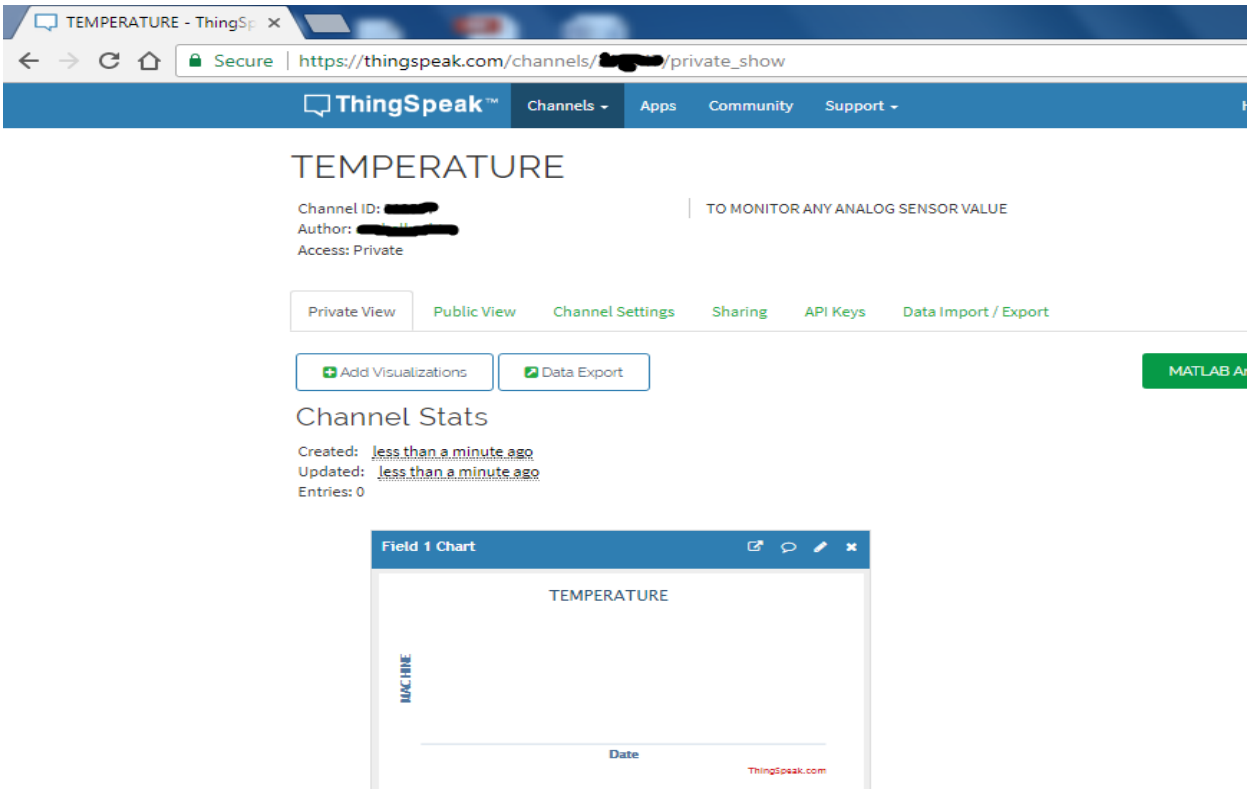


Step 3 → Fill the information.

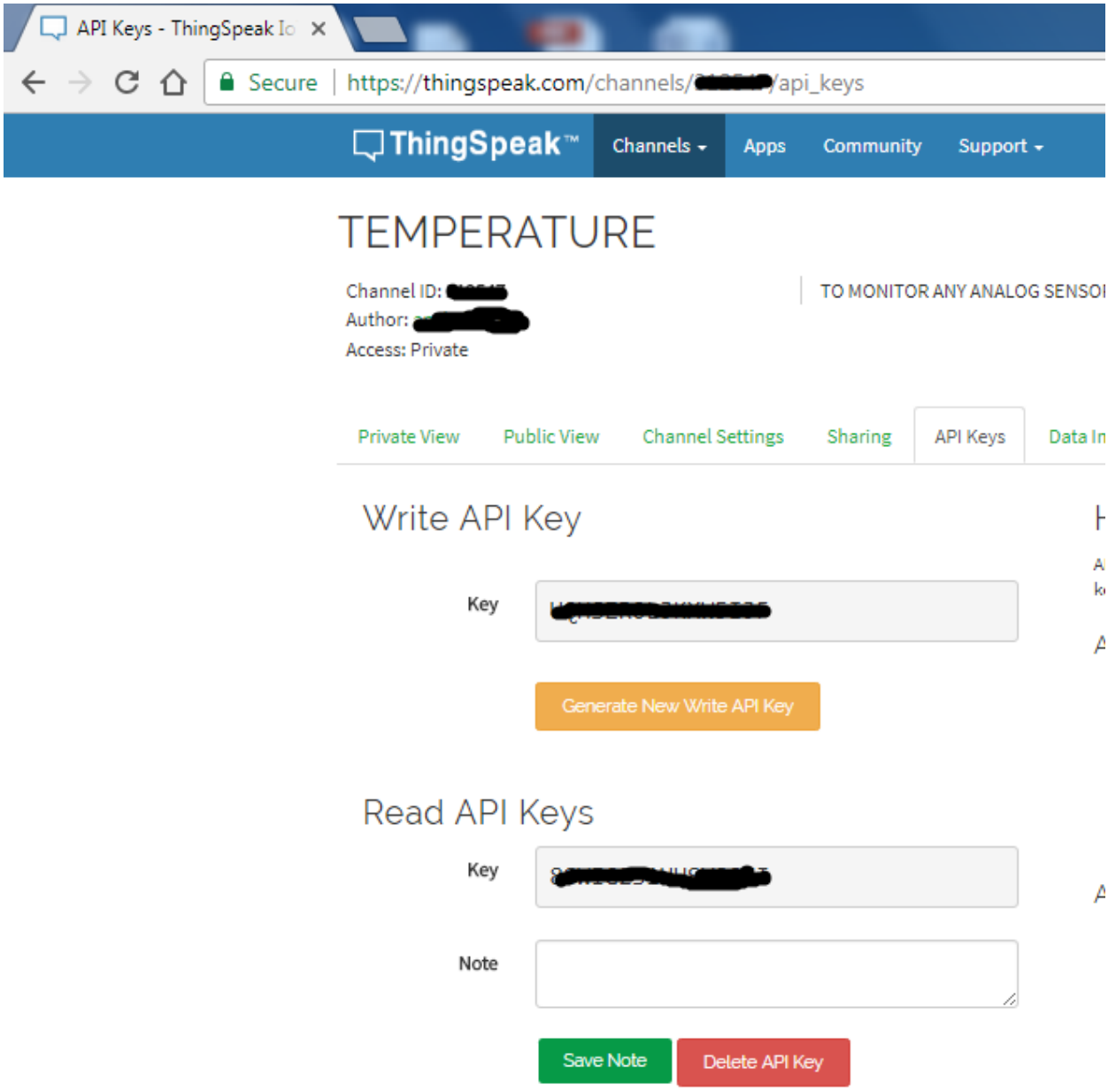


Step 4 → save channel.

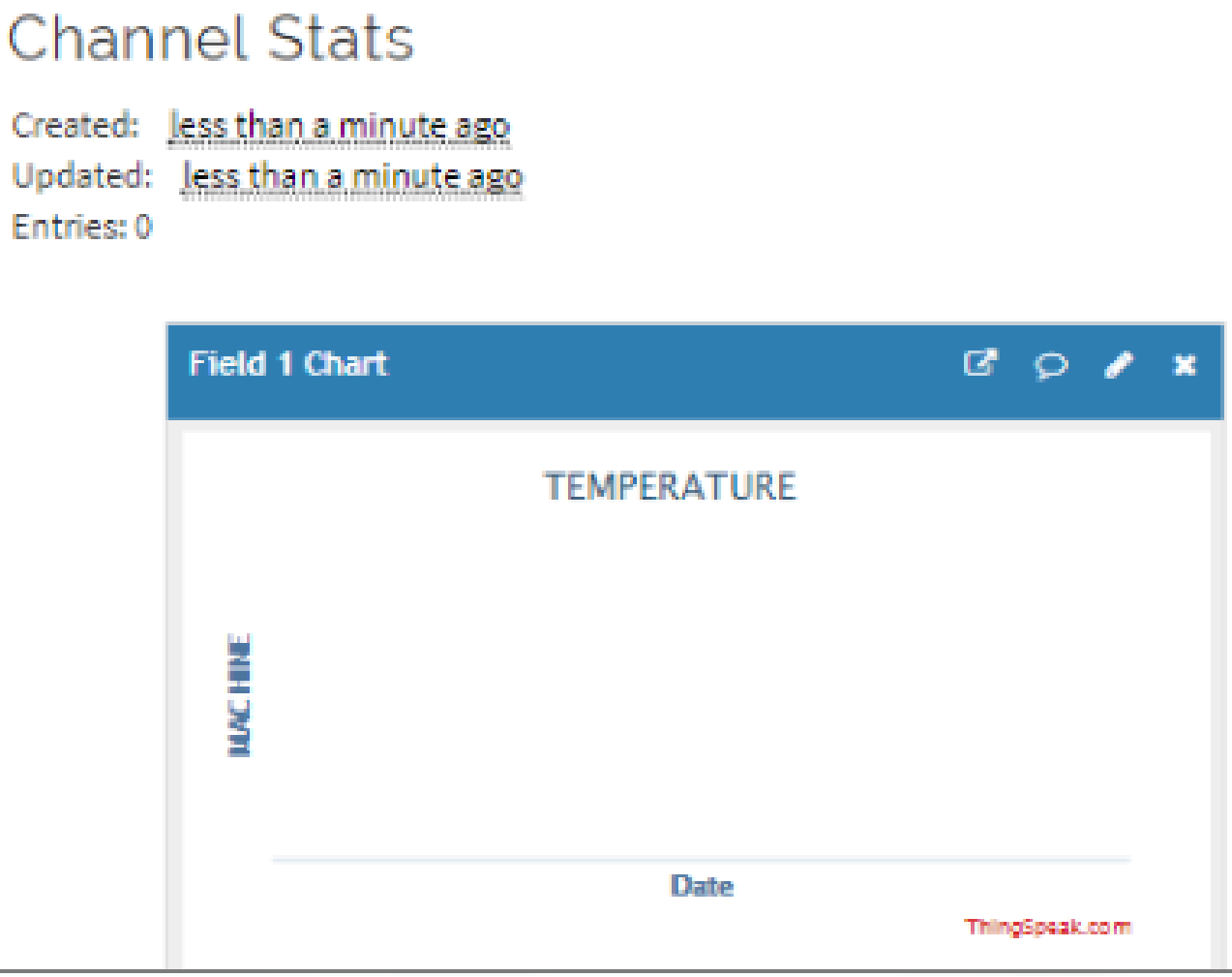




Step 5 → "Write API Key".



Step 6 → your current output is:



By these steps we can make an IOT platform on thing speak and after use of our microcontroller we can easily make IOT projects. Some IOT projects are:

- **IOT BASED SMART HOME.**
- **IOT based Smart Lighting Intelligent and weather adaptive lighting in street lights for efficient use of power.**
- **A low cost smart irrigation control system**