

## CONTENTS

## PG .NO

---

1. ABSTRACT	02
2. INTRODUCTION TO EMBEDDED SYSTEMS	03
3. BLOCK DIAGRAM EXPLANATION	09
4. HARDWARE REQUIREMENTS	10
5. SOFTWARE REQUIREMENTS	45
6. PROGRAMMING CODE	57
7. UML DIAGRAMS	62
8. HARDWARE TESTING	66
9. OUTPUT	68
10. SOURCE CODE	69
11. BIBOLOGY	87

## **ABSTRACT**

This project IOT Liquid Level Monitoring system is a very innovative system which will inform the users about the level of liquid and will prevent it from overflowing. To demonstrate this the system makes use of 4 containers. For this the system uses ultrasonic sensors placed over the containers to detect the liquid level and compare it with the container's depth. The system makes use of AVR family microcontroller, LCD screen, Wifi modem for sending data and a buzzer. The system is powered by a 12V transformer.

The LCD screen is used to display the status of the level of liquid in the containers. Where as a web page is built to show the status to the user monitoring it. The web page gives a graphical view of the containers and highlights the liquid level in color in order to show the level of liquid. The LCD screen shows the status of the liquid level. The system puts on the buzzer when the level of liquid collected crosses the set limit. Thus this system helps to prevent the wastage of water by informing about the liquid levels of the containers by providing graphical image of the containers via a web page.

# Introduction

## What Is IoT?

The term *Internet of Things* (often abbreviated *IoT*) was coined more than ten years ago by industry researchers but has emerged into mainstream public view only more recently. Some claim the Internet of Things will completely transform how computer networks are used for the next 10 or 100 years, while others believe IoT is simply hype that won't much impact the daily lives of most people.

Internet of Things represents a general concept for the ability of network devices to sense and collect data from the world around us, and then share that data across the Internet where it can be processed and utilized for various interesting purposes.

Some also use the term *industrial Internet* interchangeably with IoT. This refers primarily to commercial applications of IoT technology in the world of manufacturing. The Internet of Things is not limited to industrial applications, however.

## What the Internet of Things Can Do for Us

Some future consumer applications envisioned for IoT sound like science fiction, but some of the more practical and realistic sounding possibilities for the technology include:

- receiving warnings on your phone or wearable device when IoT networks detect some physical danger is detected nearby
- self-parking automobiles

- automatic ordering of groceries and other home supplies
- automatic tracking of exercise habits and other day-to-day personal activity including goal tracking and regular progress reports

Potential benefits of IoT in the business world include:

- location tracking for individual pieces of manufacturing inventory
- fuel savings from intelligent environmental modeling of gas-powered engines
- new and improved safety controls for people working in hazardous environments

Network Devices and the Internet of Things

All kinds of ordinary household gadgets can be modified to work in an IoT system. Wi-Fi network adapters, motion sensors, cameras, microphones and other instrumentation can be embedded in these devices to enable them for work in the Internet of Things. Home automation systems already implement primitive versions of this concept for things like light bulbs, plus other devices like wireless scales and wireless blood pressure monitors that each represent early examples of IoT gadgets. Wearable computing devices like watches and glasses are also envisioned to be key components in future IoT systems.

The same wireless communication protocols like Wi-Fi and Bluetooth naturally extend to the Internet of Things also.

## **Issues Around IoT**

Internet of Things immediately triggers questions around the privacy of personal data. Whether real-time information about our physical location, or updates about our weight and blood pressure that may be accessible by our health care providers, having new kinds and more detailed data about ourselves streaming over wireless networks and potentially around the world is an obvious concern.

Supplying power to this new proliferation of IoT devices and their network connections can be expensive and logistically difficult. Portable devices require batteries that someday must be

replaced. Although many mobile devices are optimized for lower power usage, energy costs to keep potentially billions of them running remains high.

Numerous corporations and start-up ventures have latched onto the Internet of Things concept looking to take advantage of whatever business opportunities are available. While competition in the market helps lower prices of consumer products, in the worst case it also leads to confusing and inflated claims about what the products do.

IoT assumes that the underlying network equipment and related technology can operate semi-intelligently and often automatically. Simply keeping mobile devices connected to the Internet can be difficult enough much less trying to make them smarter. People have diverse needs that require an IoT system to adapt or be configurable for many different situations and preferences. Finally, even with all those challenges overcome, if people become too reliant on this automation and the technology is not highly robust, any technical glitches in the system can cause serious physical and/or financial damage.

## INTRODUCTION TO EMBEDDED SYSTEMS

### What is embedded system?

An Embedded System is a combination of computer hardware and software, and perhaps additional mechanical or other parts, designed to perform a specific function. An embedded system is a microcontroller-based, software driven, reliable, real-time control system, autonomous, or human or network interactive, operating on diverse physical variables and in diverse environments and sold into a competitive and cost conscious market.

An embedded system is not a computer system that is used primarily for processing, not a software system on PC or UNIX, not a traditional business or scientific application. High-end embedded & lower end embedded systems. High-end embedded system - Generally 32, 64 Bit Controllers used with OS. Examples Personal Digital Assistant and Mobile phones etc .Lower end embedded systems - Generally 8,16 Bit Controllers used with an minimal operating systems and hardware layout designed for the specific purpose.

### SYSTEM DESIGN CALLS:

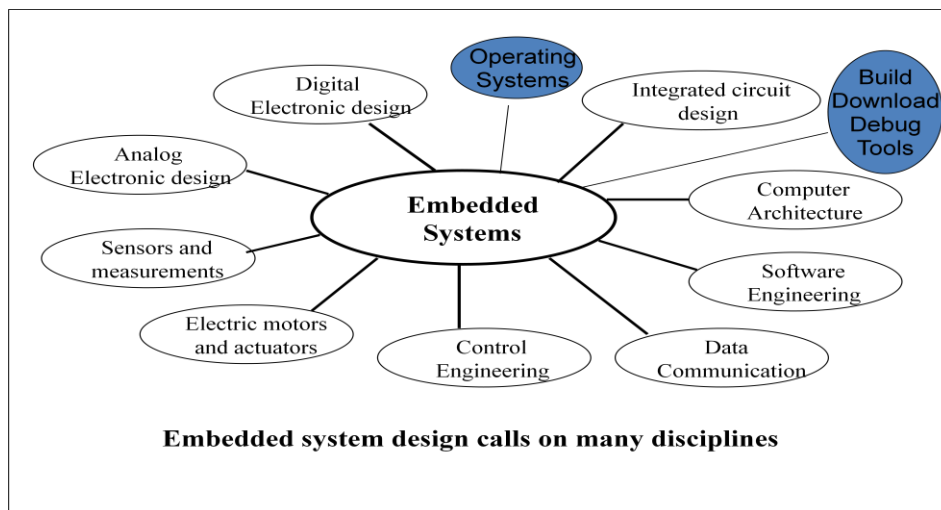


Figure 2(a): Embedded system design calls

## EMBEDDED SYSTEM DESIGN CYCLE:

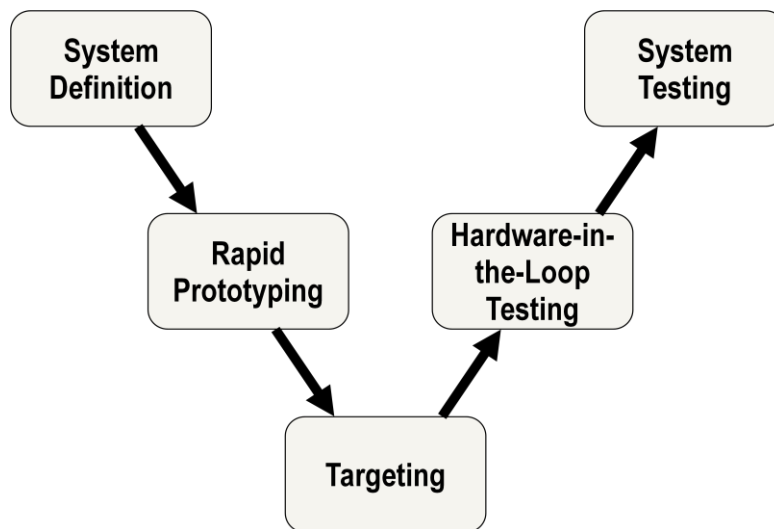


Figure:2(b) “V Diagram”

## Characteristics of Embedded System

- An embedded system is any computer system hidden inside a product other than a computer.
- They will encounter a number of difficulties when writing embedded system software in addition to those we encounter when we write applications.
  - Throughput – Our system may need to handle a lot of data in a short period of time.
  - Response–Our system may need to react to events quickly
  - Testability–Setting up equipment to test embedded software can be difficult.
  - Debugability–Without a screen or a keyboard, finding out what the software is doing wrong (other than not working) is a troublesome problem.
  - Reliability – embedded systems must be able to handle any situation without human intervention.
  - Memory space – Memory is limited on embedded systems, and you must make the software and the data fit into whatever memory exists.
  - Program installation – you will need special tools to get your software into embedded systems.

- Power consumption – Portable systems must run on battery power, and the software in these systems must conserve power.
- Processor hogs – computing that requires large amounts of CPU time can complicate the response problem.
- Cost – Reducing the cost of the hardware is a concern in many embedded system projects; software often operates on hardware that is barely adequate for the job.
- Embedded systems have a microprocessor/ microcontroller and a memory. Some have a serial port or a network connection. They usually do not have keyboards, screens or disk drives.

## **APPLICATIONS**

- 1) Military and aerospace embedded software applications
- 2) Communication Applications
- 3) Industrial automation and process control software
- 4) Mastering the complexity of applications.
- 5) Reduction of product design time.
- 6) Real time processing of ever increasing amounts of data.
- 7) Intelligent, autonomous sensors.

## **CLASSIFICATION**

- Real Time Systems.
- RTS is one which has to respond to events within a specified deadline.
- A right answer after the dead line is a wrong answer.

## **RTS CLASSIFICATION**

- Hard Real Time Systems
- Soft Real Time System

## **HARD REAL TIME SYSTEM**

- "Hard" real-time systems have very narrow response time.

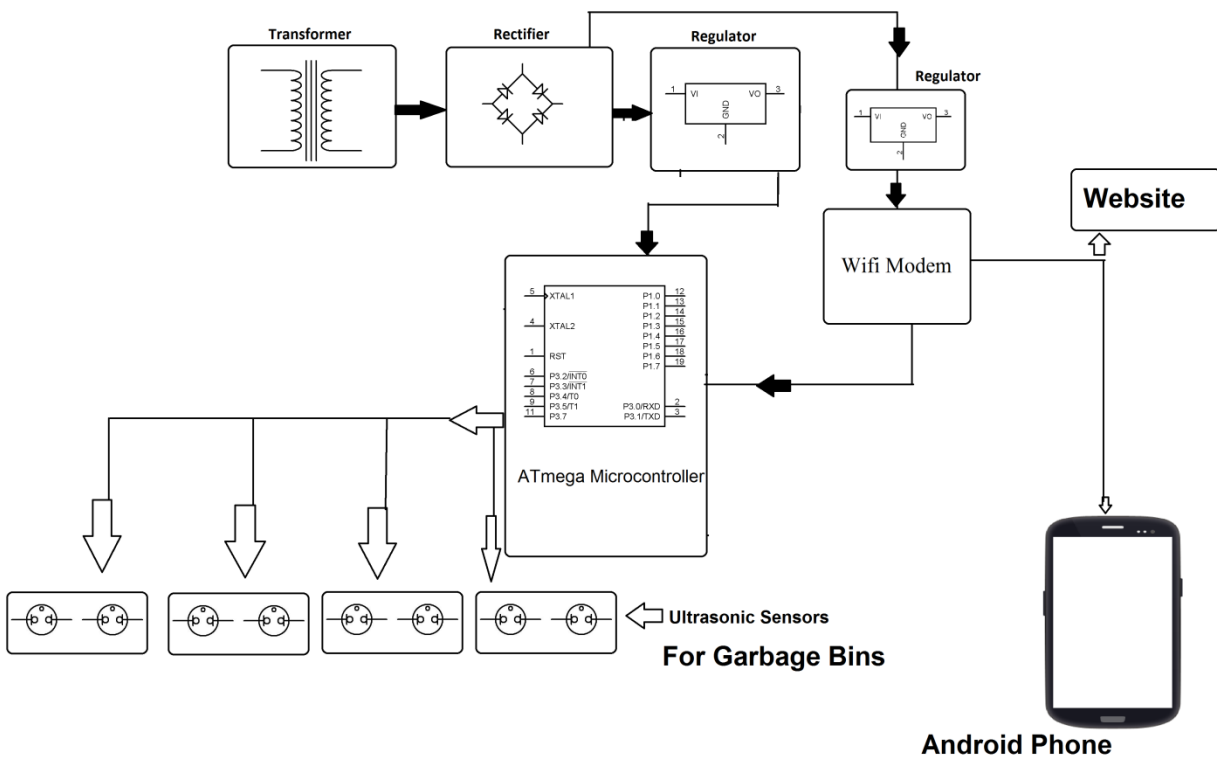


- Example: Nuclear power system, Cardiac pacemaker.

## SOFT REAL TIME SYSTEM

- "Soft" real-time systems have reduced constraints on "lateness" but still must operate very quickly and repeatably.
- Example: Railway reservation system – takes a few extra seconds the data remains valid.

## PROJECT BLOCK DIAGRAM



## 4. HARDWARE REQUIREMENTS

### HARDWARE COMPONENT:

1. TRANSFORMER
2. BRIDGE RECTIFIER
3. FILTER
4. VOLTAGE REGULATOR (7805)
5. MICROCONTROLLER
6. ULTRASONIC SENSORS
7. WIFI MODULE
8. LED
9. RESISTOR
10. CAPACITOR
11. PUSH BUTTON

# TRANSFORMER

Transformers convert AC electricity from one voltage to another with a little loss of power. Step-up transformers increase voltage, step-down transformers reduce voltage. Most power supplies use a step-down transformer to reduce the dangerously high voltage to a safer low voltage.



FIG 4.1: A TYPICAL TRANSFORMER

The input coil is called the primary and the output coil is called the secondary. There is no electrical connection between the two coils; instead they are linked by an alternating magnetic field created in the soft-iron core of the transformer. The two lines in the middle of the circuit symbol represent the core. Transformers waste very little power so the power out is (almost) equal to the power in. Note that as voltage is stepped down and current is stepped up.

The ratio of the number of turns on each coil, called the turn's ratio, determines the ratio of the voltages. A step-down transformer has a large number of turns on its primary (input) coil which is connected to the high voltage mains supply, and a small number of turns on its secondary (output) coil to give a low output voltage.

$$\text{TURNS RATIO} = (V_p / V_s) = (N_p / N_s)$$

Where,

$V_p$  = primary (input) voltage.

$V_s$  = secondary (output) voltage

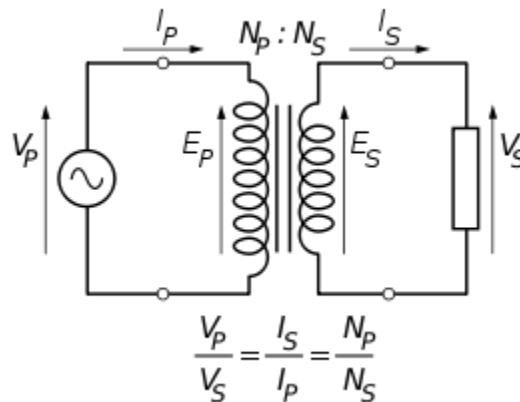
$N_p$  = number of turns on primary coil

$N_s$  = number of turns on secondary coil

$I_p$  = primary (input) current

$I_s$  = secondary (output) current.

## Ideal power equation



The ideal transformer as a circuit element

If the secondary coil is attached to a load that allows current to flow, electrical power is transmitted from the primary circuit to the secondary circuit. Ideally, the transformer is perfectly efficient; all the incoming energy is transformed from the primary circuit to the magnetic field and into the secondary circuit. If this condition is met, the incoming electric power must equal the outgoing power:

$$P_{\text{incoming}} = I_p V_p = P_{\text{outgoing}} = I_s V_s,$$

Giving the ideal transformer equation

$$\frac{V_s}{V_p} = \frac{N_s}{N_p} = \frac{I_p}{I_s}.$$

Transformers normally have high efficiency, so this formula is a reasonable approximation.

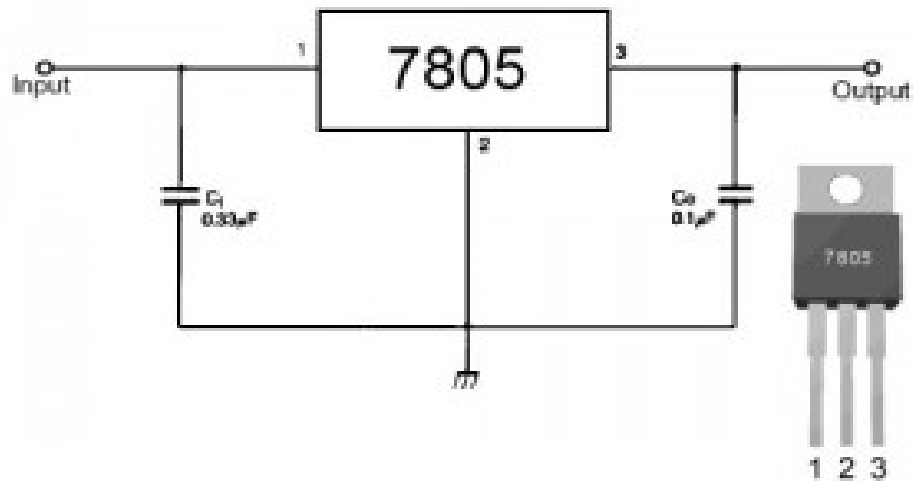
If the voltage is increased, then the current is decreased by the same factor. The impedance in one circuit is transformed by the *square* of the turns ratio. For example, if an impedance  $Z_s$  is attached across the terminals of the secondary coil, it appears to the primary circuit to have an impedance of  $(N_p/N_s)^2 Z_s$ . This relationship is reciprocal, so that the impedance  $Z_p$  of the primary circuit appears to the secondary to be  $(N_s/N_p)^2 Z_p$ .

## VOLTAGE REGULATOR 7805

### Features

- Output Current up to 1A.
- Output Voltages of 5, 6, 8, 9, 10, 12, 15, 18, 24V.
- Thermal Overload Protection.
- Short Circuit Protection.

- Output Transistor Safe Operating Area Protection.



## Description

The LM78XX/LM78XXA series of three-terminal positive regulators are available in the TO-220/D-PAK package and with several fixed output voltages, making them useful in a Wide range of applications. Each type employs internal current limiting, thermal shutdown and safe operating area protection, making it essentially indestructible. If adequate heat sinking is provided, they can deliver over 1A output Current. Although designed primarily as fixed voltage regulators, these devices can be used with external components to obtain adjustable voltages and currents.

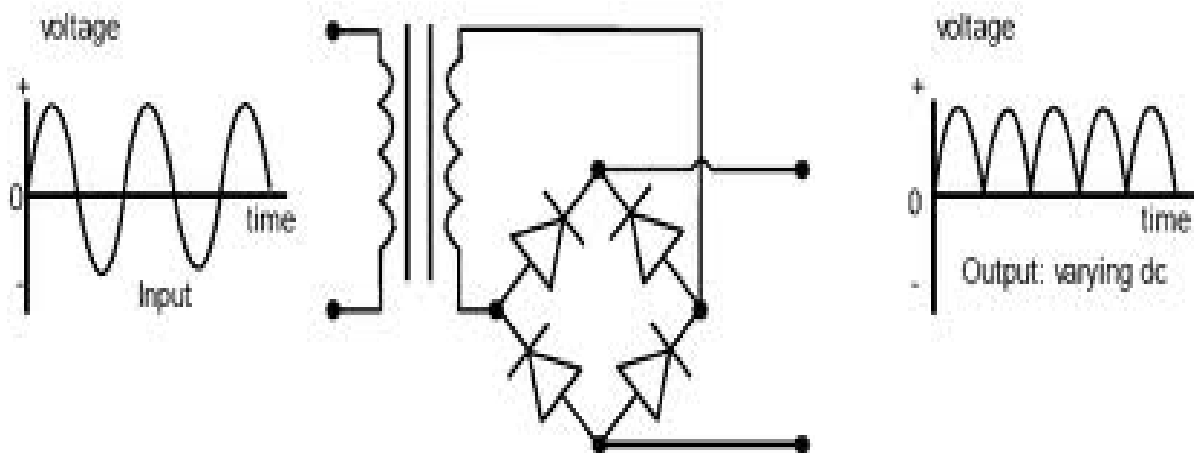
## Absolute Maximum Ratings

Parameter	Symbol	Value	Unit
Input Voltage (for $V_O = 5V$ to $18V$ ) (for $V_O = 24V$ )	$V_I$	35	V
	$V_I$	40	V
Thermal Resistance Junction-Cases (TO-220)	$R_{\theta JC}$	5	$^{\circ}C/W$
Thermal Resistance Junction-Air (TO-220)	$R_{\theta JA}$	65	$^{\circ}C/W$
Operating Temperature Range (KA78XX/A/R)	$T_{OPR}$	$0 \sim +125$	$^{\circ}C$
Storage Temperature Range	$T_{STG}$	$-65 \sim +150$	$^{\circ}C$

TABLE 4.2(b): RATINGS OF THE VOLTAGE REGULATOR

## RECTIFIER

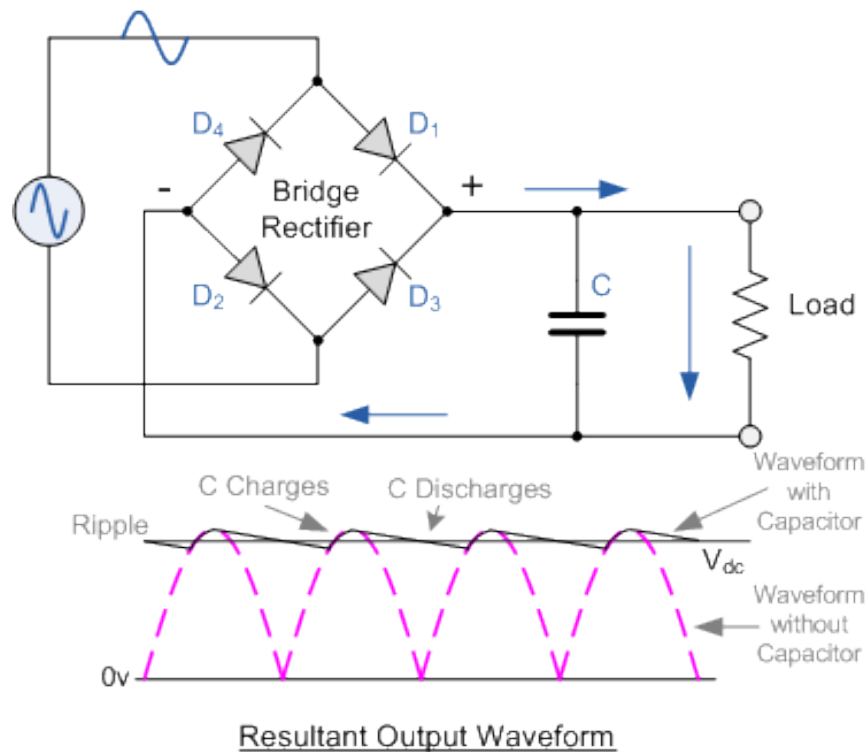
A rectifier is an electrical device that converts alternating current (AC), which periodically reverses direction, to direct current (DC), current that flows in only one direction, a process known as rectification. Rectifiers have many uses including as components of power supplies and as detectors of radio signals. Rectifiers may be made of solid state diodes, vacuum tube diodes, mercury arc valves, and other components. The output from the transformer is fed to the rectifier. It converts A.C. into pulsating D.C. The rectifier may be a half wave or a full wave rectifier. In this project, a bridge rectifier is used because of its merits like good stability and full wave rectification. In positive half cycle only two diodes (1 set of parallel diodes) will conduct, in negative half cycle remaining two diodes will conduct and they will conduct only in forward bias only.



## FILTER

Capacitive filter is used in this project. It removes the ripples from the output of rectifier and smoothens the D.C. Output received from this filter is constant until the mains voltage and load is maintained constant. However, if either of the two is varied, D.C. voltage received at this point changes. Therefore a regulator is applied at the output stage.

The simple capacitor filter is the most basic type of power supply filter. The use of this filter is very limited. It is sometimes used on extremely high-voltage, low-current power supplies for cathode-ray and similar electron tubes that require very little load current from the supply. This filter is also used in circuits where the power-supply ripple frequency is not critical and can be relatively high. Below figure can show how the capacitor charges and discharges.



## MICROCONTROLLER

### ATmega328:

The computer on one hand is designed to perform all the general purpose tasks on a single machine like you can use a computer to run a software to perform calculations or you can use a computer to store some multimedia file or to access internet through the browser, whereas the microcontrollers are meant to perform only the specific tasks, for e.g., switching the AC off

automatically when room temperature drops to a certain defined limit and again turning it ON when temperature rises above the defined limit.

There are number of popular families of microcontrollers which are used in different applications as per their capability and feasibility to perform the desired task, most common of these are 8051, AVR and PIC microcontrollers. In this we will introduce you with AVR family of microcontrollers.

### **History of AVR**

AVR was developed in the year 1996 by Atmel Corporation. The architecture of AVR was developed by Alf-Egil Bogen and Vegard Wollan. AVR derives its name from its developers and stands for Alf-Egil Bogen Vegard Wollan RISC microcontroller, also known as Advanced Virtual RISC.

AVR microcontrollers are available in three categories:

**Tiny AVR** – Less memory, small size, suitable only for simpler applications

**AVR** – These are the most popular ones having good amount of memory (up-to 256 KB), higher number of in-built peripherals and suitable for moderate to complex applications.

**Xmega AVR** – Used commercially for complex applications, which require large program memory and high speed?

### **Features:**

- RISC Architecture with CISC Instruction set
- Powerful C and assembly programming
- Scalable
- Same powerful AVR microcontroller core
- Low power consumption



- Both digital and analog input and output interfaces

**Description:**

The Atmel ATmega48/88/328 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega48/88/328 achieves throughputs approaching 1 MIPS per MHz allowing the system designed to optimize power consumption versus processing speed.

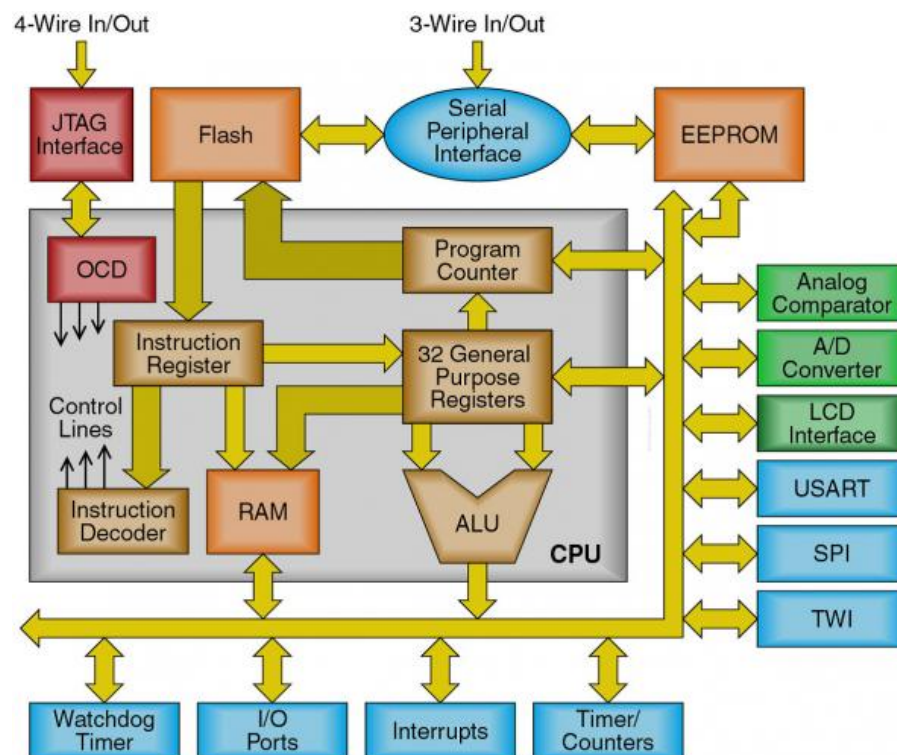
The Atmel ATmega48/88/328 provides the following features: 4K/8K/16K bytes of In-System Programmable Flash with Read-While-Write capabilities, 256/512/512 bytes EEPROM, 512/1K/1K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption.

The ATmega48, ATmega88 and ATmega328 differ only in memory sizes, boot loader support, and interrupt vector sizes. Table 2-1 summarizes the different memory and interrupts vector sizes for the three devices.

Device	Flash	EEPROM	RAM	Interrupt vector size
ATmega48	4Kbytes	256Bytes	512Bytes	1 instruction word/vector
ATmega88	8Kbytes	512Bytes	1Kbytes	1 instruction word/vector
ATmega168	16Kbytes	512Bytes	1Kbytes	2 instruction words/vector

ATmega88 and ATmega328 support a real Read-While-Write Self-Programming mechanism. There is a separate Boot Loader Section, and the SPM instruction can only execute from there. In ATmega48, there is no Read-While-Write support and no separate Boot Loader Section. The SPM instruction can execute from the entire Flash.

### Processor architecture:



AVR follows Harvard Architecture format in which the processor is equipped with separate memories and buses for Program and the Data information. Here while an instruction is being executed, the next instruction is pre-fetched from the program memory.

**ALU:** The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format.

**In-system reprogrammable flash program memory:** The ATmega48/88/328 contains 4K/8K/16K bytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as  $2K/4K/8K \times 16$ . For software security, the Flash Program memory space is divided into two sections, Boot Loader Section and Application Program Section in ATmega88 and ATmega328.

**EEPROM data memory:** The Atmel ATmega48 /88/328 contains 256/512/512 bytes of data EEPROM memory. It is organized as a separate data space e, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM Control Register.

**Program counter:** A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0. In computing, a program is a specific set of ordered operations for a computer to perform. An instruction is an order given to a computer processor by a program. Within a computer, an address is a specific location in memory or storage. A register is one of a small set of data holding places that the processor uses. Program counter is very important feature in the microcontrollers.

**RAM:** RAM stands for random access memory. This type of memory storage is temporary and volatile. You might have heard that if your system is working slowly you say that increase the RAM processing will increase. Let us understand in detail. Let us consider two cases to execute a

task first the complete task is executed at one place(A), second the task is distributed in parts and the small tasks are executed at different places(A,B C)and finally assembled. It is clear the work will be finished in second case earlier. The A, B, C basically represent different address allocation for temporary processing. This is the case with RAM also if you increase the RAM the address basically increases for temporary processing so that no data has to wait for its turn. On major importance of the RAM is address allocations. However the storage is temporary every time u boot your system the data is lost but when you turn on the system The BIOS fetch number of addresses available in the RAM. This memory supports read as well as write operations both.

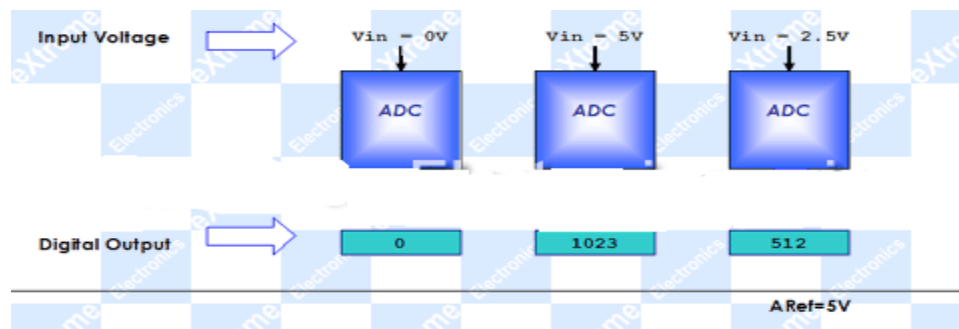
**Instruction execution section (IES).** It has the most important unit—instruction register and instruction decoder to control the flow of the instruction during the processing's.

**INPUT/OUTPUT PORTS:** To interact with the physical environment there are different input and output ports in every system like in PC we have VGA port to connect the monitor, USB port for flash memory connections and many more ports. Similarly ATMEGA 328 has its input and output ports with different configurations depending on the architecture like only input, only output and bi-directional input output ports. The accessing of this port is referred as input output interface design for microcontrollers. IT has analog input port, analog output port, digital input port ,digital output port, serial communication pins, timer execution pins etc.

**Analog Comparator & A/D converters:** The major question is that how a controller manage to detect variation of voltage in spite it could not understand the voltage but understand only digital sequence

Most of the physical quantities around us are continuous. By continuous we mean that the quantity can take any value between two extreme. For example the atmospheric temperature can take any value (within certain range). If an electrical quantity is made to vary directly in proportion to this value (temperature etc) then what we have is Analogue signal. Now we have we have brought a physical quantity into electrical domain. The electrical quantity in most case is voltage. To bring this quantity into digital domain we have to convert this into digital form. For this a ADC or analog to digital converter is needed. Most modern MCU including AVR's has an ADC on chip. An ADC converts an input voltage into a number. An ADC has a resolution. A 10 Bit ADC has a range of 0-1023. ( $2^{10}=1024$ ) The ADC also has a Reference voltage (ARef).

When input voltage is GND the output is 0 and when input voltage is equal to ARef the output is 1023. So the input range is 0-ARef and digital output is 0-1023.



- **Inbuilt ADC of AVR**

Now you know the basics of ADC let us see how we can use the inbuilt ADC of AVR MCU. The ADC is multiplexed with PORTA that means the ADC channels are shared with PORTA. The ADC can be operated in single conversion and free running mode. In single conversion mode the ADC does the conversion and then stop. While in free it is continuously converting. It does a conversion and then start next conversion immediately after that.

- **ADC Pre-scalar.**

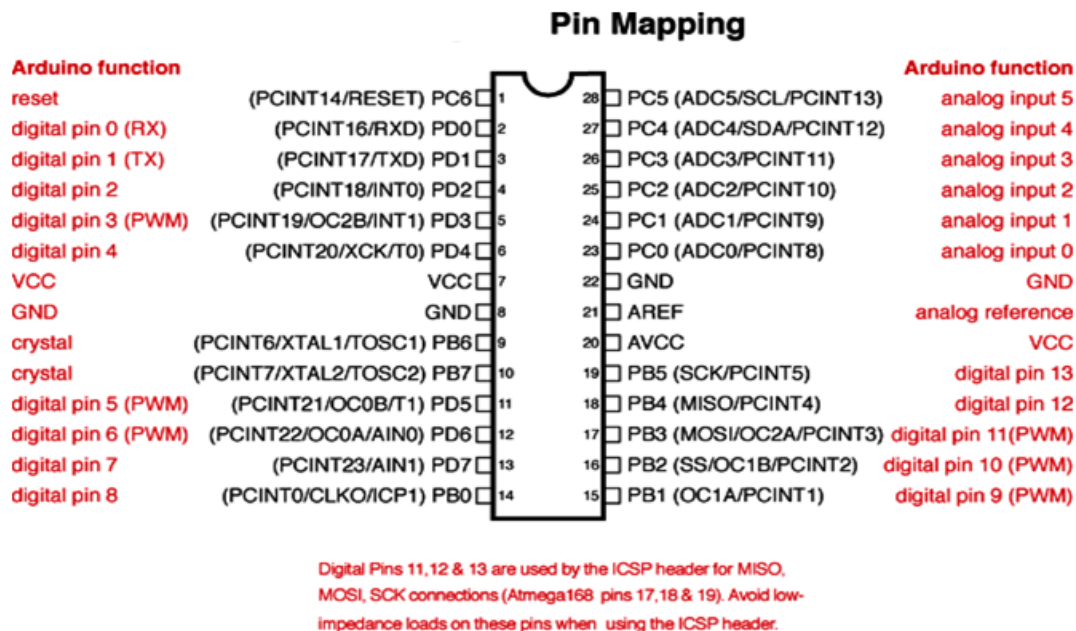
The ADC needs a clock pulse to do its conversion. This clock generated by system clock by dividing it to get smaller frequency. The ADC requires a frequency between 50 KHz to 200 KHz. At higher frequency the conversion is fast while a lower frequency the conversion is more accurate. As the system frequency can be set to any value by the user (using internal or external oscillators) (In board™ a 16MHz crystal is used). So the Pre-scalar is provided to produces

acceptable frequency for ADC from any system clock frequency. System clock can be divided by 2, 4, 16, 32, 64, 128 by setting the Pre-scalar.

## ▪ ADC Channels

The ADC in ATmega328 has 6 channels that mean you can take samples from eight different terminals. You can connect up to 8 different sensors and get their values separately.

## pin diagram & description



## PIN DESCRIPTION:

**VCC:** Digital supply voltage.

**GND:** Ground

## Port B (PB7:0) XTAL1/XTAL2/TOSC1/TOSC2

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source Capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are

activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running. Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7.6 is used as TOSC2.1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

### **Port C (PC5:0)**

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC5.0 output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

### **PC6/RESET:**

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset.

### **Port D (PD7:0):**

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

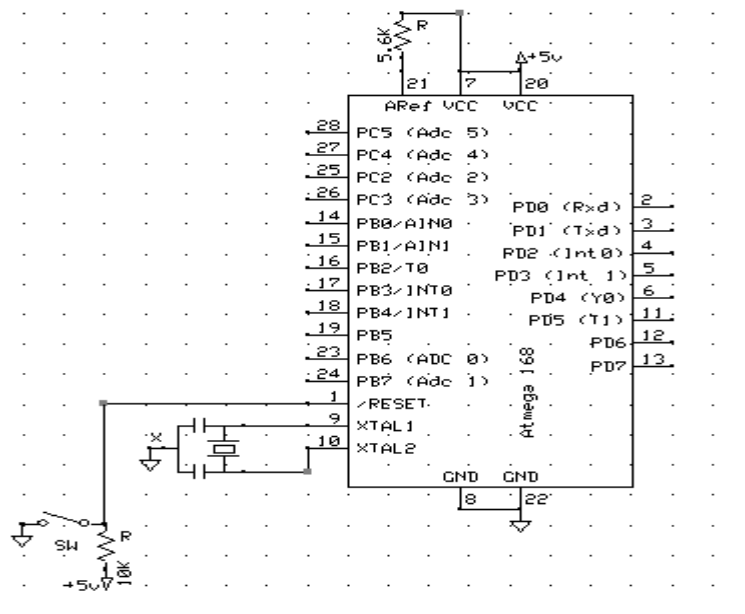
## AVCC:

AVCC is the supply voltage pin for the A/D Converter PC3:0, and ADC7:6. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC6.4 use digital supply voltage, VCC.

**AREF:** AREF is the analog reference pin for the A/D Converter.

## MINIMUM INTERFACE circuit for Atmega328 controller:

According the minimum interface discussed for the microcontrollers earlier, the minimum interface circuit for ATMEGA328 is:

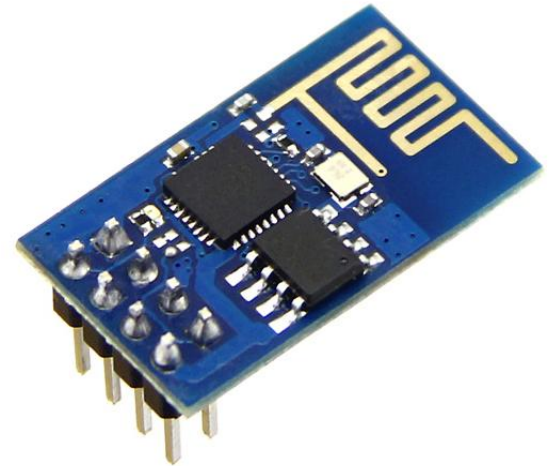




### Things to remember about ATMEGA328 controllers:

The ATMEGA controllers are strong controllers but you have to take some small points in mind always like:

- When you go for the programming of atmega328 consider the pin no. as configures in red color in Pin diagram shown before (like controllers pin number 2 is digital pin number 0 for input or output ,pin23 is analog pin A0 ). So you will address the pin according to that number.
- Use the proper pin for proper input output interface that analog input should be configured at analog pin analog output should be configured on PWM pins and likewise the digital inputs and outputs



### WIFIModem:

The ESP8266 WiFi Module is a self contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to your WiFi network. The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. Each ESP8266 module comes pre-programmed with an AT command set firmware. The ESP8266 module is an extremely cost effective board with a huge, and ever growing, community.

This module has a powerful enough on-board processing and storage capability that allows it to be integrated with the sensors and other application specific devices through its GPIOs with minimal development up-front and minimal loading during runtime. Its high degree of on-chip integration allows for minimal external circuitry, including the front-end module, is designed to occupy minimal PCB area. The ESP8266 supports APSD for VoIP applications and Bluetooth co-existence interfaces, it contains a self-calibrated RF allowing it to work under all operating conditions, and requires no external RF parts.

There is an almost limitless fountain of information available for the ESP8266, all of which has been provided by amazing community support. In the Documents section below you will find many resources to aid you in using the ESP8266, even instructions on how to transforming this module into an IoT (Internet of Things) solution!

Note: The ESP8266 Module is not capable of 5-3V logic shifting and will require an external Logic Level Converter. Please do not power it directly from your 5V dev board.

**Features:**

802.11 b/g/n

Wi-Fi Direct (P2P), soft-AP

Integrated TCP/IP protocol stack

Integrated TR switch, balun, LNA, power amplifier and matching network

Integrated PLLs, regulators, DCXO and power management units

+19.5dBm output power in 802.11b mode

Power down leakage current of <10uA

1MB Flash Memory

Integrated low power 32-bit CPU could be used as application processor

SDIO 1.1 / 2.0, SPI, UART

STBC, 1×1 MIMO, 2×1 MIMO

A-MPDU & A-MSDU aggregation & 0.4ms guard interval

Wake up and transmit packets in < 2ms

## ULTRASONIC SENSOR MODULE:



---

Here is a more easy use serial ultrasonic module. It will auto output the distance information via serial port after power on, you don't need to do any trigger and calculated, just need to read the serial pin and get the distance information.

Ultrasonic sensor provides a very low-cost and easy method of distance measurement. This sensor is perfect for any number of applications that require you to perform measurements between moving or stationary objects. Naturally, robotics applications are very popular but you'll also find this product to be useful in security systems or as an infrared replacement if so desired. You will definitely appreciate the activity status LED and the economic use of just one I/O pin.

The ultrasonic sensor measures distance using sonar; an ultrasonic (well above human hearing) pulse is transmitted from the unit and distance-to-target is determined by measuring the time required for the echo return. Output from the ultrasonic sensor is a variable-width pulse that corresponds to the distance to the target.

**Features:**

- Provides precise, non-contact distance measurements within a 2 cm to 3 m range
- Simple pulse in/pulse out communication
- Burst indicator LED shows measurement in progress
- 20 mA power consumption
- Narrow acceptance angle
- 3-pin header makes it easy to connect using a servo extension cable, no soldering required

**Key Specifications:**

- power supply :5V DC
- quiescent current : <15mA
- effectual angle: <15°
- ranging distance : 2cm – 350 cm
- resolution : 0.3 cm
- Output cycle : 50ms
- Baud Rate : 9600

**Output frame format (4Bytes)**

- 1st Byte : 0xFF (Start bit, Fixed value)
- 2nd Byte: H\_Data (High 8 bit of the distance)
- 3rd Byte: L\_Data (Low 8 bit of the distance)
- 4th Byte: Check Sum (0xFF+H\_DATA+L\_DATA=SUM, Check SUM is the low 8 bit of the SUM)

**Uses:**

Ultrasonic sensors are used to detect the presence of targets and to measure the distance to targets in many automated factories and process plants. Sensors with an on or off digital output are available for detecting the presence of objects, and sensors with an analog output which varies proportionally to the sensor to target separation distance are commercially available.

Because ultrasonic sensors use sound rather than light for detection, they work in applications where photoelectric sensors may not. Ultrasonic are a great solution for clear object detection and for liquid level measurement, applications that photo electrics struggle with because of target translucence. Target colour and/or reflectivity don't affect ultrasonic sensors which can operate reliably in high-glare environments.

Other types of transducers are used in commercially available ultrasonic cleaning devices. An ultrasonic transducer is affixed to a stainless steel pan which is filled with a solvent (frequently water or isopropanol) and a square wave is applied to it, imparting vibrational energy on the liquid.

**LED:**

A light-emitting diode (LED) is a semiconductor light source. LEDs are used as indicator lamps in many devices, and are increasingly used for lighting. When a light-emitting diode is forward biased (switched on), electrons are able to recombine with holes within the device, releasing energy in the form of photons.

This effect is called electroluminescence and the color of the light (corresponding to the energy of the photon) is determined by the energy gap of the semiconductor. An LED is often small in area (less than 1 mm<sup>2</sup>), and integrated optical components may be used to shape its radiation pattern. LEDs present many advantages over incandescent light sources including

lower energy consumption, longer lifetime, improved robustness, smaller size, faster switching, and greater durability and reliability.

### Types of LED'S

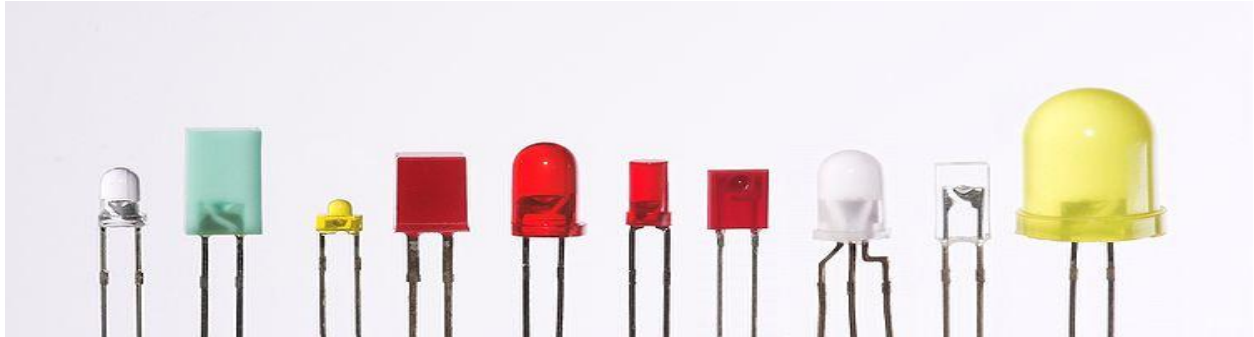


Fig 4.8(a): Types of LED

Light-emitting diodes are used in applications as diverse as replacements for aviation lighting, automotive lighting as well as in traffic signals. The compact size, the possibility of narrow bandwidth, switching speed, and extreme reliability of LEDs has allowed new text and video displays and sensors to be developed, while their high switching rates are also useful in advanced communications technology.

### Electronic Symbol:



4.8(b): symbol of LED

### White LED'S

Light Emitting Diodes (LED) have recently become available that are both white and bright, so bright that they seriously compete with incandescent lamps in lighting applications. They are still pretty expensive as compared to a GOW lamp but draw much less current and project a fairly well focused beam.

When run within their ratings, they are more reliable than lamps as well. Red LEDs are now being used in automotive and truck tail lights and in red traffic signal lights. You will be able to detect them because they look like an array of point sources and they go on and off instantly as compared to conventional incandescent lamps.

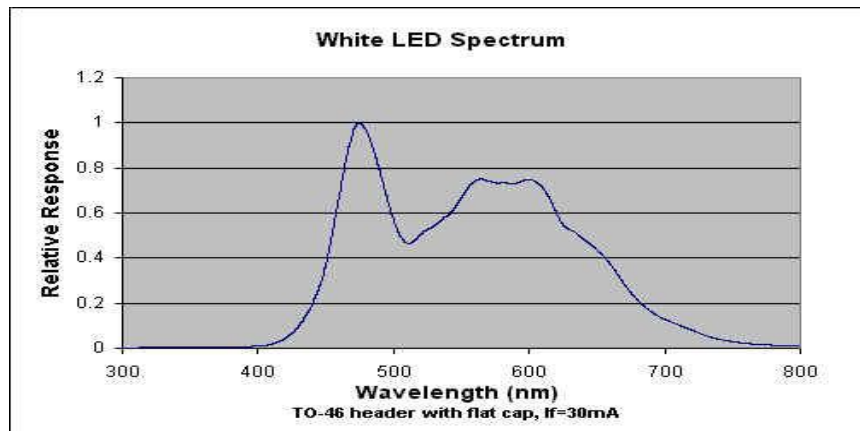


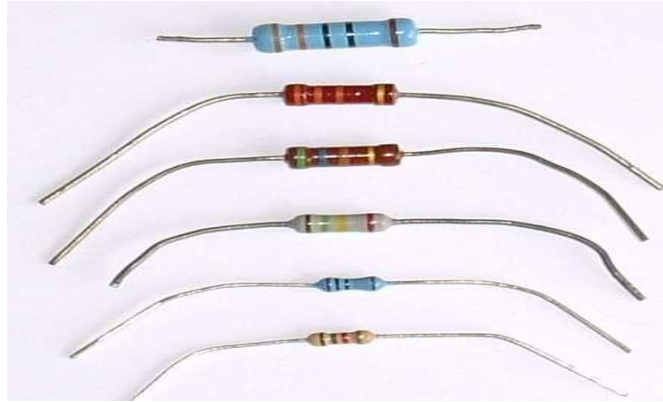
Fig 4.8(c): White LED spectrum

## RESISTORS

A resistor is a two-terminal electronic component designed to oppose an electric current by producing a voltage drop between its terminals in proportion to the current, that is, in accordance with Ohm's law:

$$V = IR$$

Resistors are used as part of electrical networks and electronic circuits. They are extremely commonplace in most electronic equipment. Practical resistors can be made of various compounds and films, as well as resistance wire (wire made of a high-resistivity alloy, such as nickel/chrome).



The primary characteristics of resistors are their resistance and the power they can dissipate. Other characteristics include temperature coefficient, noise, and inductance. Less well-known is critical resistance, the value below which power dissipation limits the maximum permitted current flow, and above which the limit is applied voltage. Critical resistance depends upon the materials constituting the resistor as well as its physical dimensions; it's determined by design.

Resistors can be integrated into hybrid and printed circuits, as well as integrated circuits. Size, and position of leads (or terminals) are relevant to equipment designers; resistors must be physically large enough not to overheat when dissipating their power.

A resistor is a two-terminal passive electronic component which implements electrical resistance as a circuit element. When a voltage  $V$  is applied across the terminals of a resistor, a current  $I$  will flow through the resistor in direct proportion to that voltage. The reciprocal of the constant of proportionality is known as the resistance  $R$ , since, with a given voltage  $V$ , a larger value of  $R$  further "resists" the flow of current  $I$  as given by Ohm's law:

$$I = \frac{V}{R}$$

Resistors are common elements of electrical networks and electronic circuits and are ubiquitous in most electronic equipment. Practical resistors can be made of various compounds and films, as well as resistance wire (wire made of a high-resistivity alloy, such as nickel-chrome). Resistors are also implemented within integrated circuits, particularly analog devices, and can also be integrated into hybrid and printed circuits.



The electrical functionality of a resistor is specified by its resistance: common commercial resistors are manufactured over a range of more than 9 orders of magnitude. When specifying that resistance in an electronic design, the required precision of the resistance may require attention to the manufacturing tolerance of the chosen resistor, according to its specific application. The temperature coefficient of the resistance may also be of concern in some precision applications. Practical resistors are also specified as having a maximum power rating which must exceed the anticipated power dissipation of that resistor in a particular circuit: this is mainly of concern in power electronics applications. Resistors with higher power ratings are physically larger and may require heat sinking. In a high voltage circuit, attention must sometimes be paid to the rated maximum working voltage of the resistor.

The series inductance of a practical resistor causes its behaviour to depart from ohms law; this specification can be important in some high-frequency applications for smaller values of resistance. In a low-noise amplifier or pre-amp the noise characteristics of a resistor may be an issue. The unwanted inductance, excess noise, and temperature coefficient are mainly dependent on the technology used in manufacturing the resistor. They are not normally specified individually for a particular family of resistors manufactured using a particular technology. A family of discrete resistors is also characterized according to its form factor, that is, the size of the device and position of its leads (or terminals) which is relevant in the practical manufacturing of circuits using them.

## Units

The ohm (symbol:  $\Omega$ ) is the SI unit of electrical resistance, named after Georg Simon Ohm. An ohm is equivalent to a volt per ampere. Since resistors are specified and manufactured over a very large range of values, the derived units of milliohm ( $1 \text{ m}\Omega = 10^{-3} \Omega$ ), kilohm ( $1 \text{ k}\Omega = 10^3 \Omega$ ), and megohm ( $1 \text{ M}\Omega = 10^6 \Omega$ ) are also in common usage.

The reciprocal of resistance  $R$  is called conductance  $G = 1/R$  and is measured in Siemens (SI unit), sometimes referred to as a mho. Thus a Siemens is the reciprocal of an ohm:  $S = \Omega^{-1}$ . Although the concept of conductance is often used in circuit analysis, practical resistors are always specified in terms of their resistance (ohms) rather than conductance.

## Theory of operation

### Ohm's law

The behaviour of an ideal resistor is dictated by the relationship specified in Ohm's law:

$$V = I \cdot R$$

Ohm's law states that the voltage (V) across a resistor is proportional to the current (I) passing through it, where the constant of proportionality is the resistance (R).

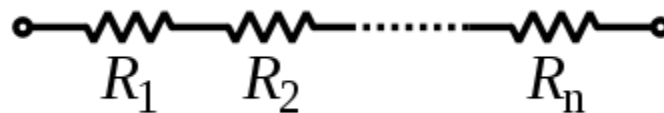
Equivalently, Ohm's law can be stated:

$$I = \frac{V}{R}$$

This formulation of Ohm's law states that, when a voltage (V) is present across a resistance (R), a current (I) will flow through the resistance. This is directly used in practical computations. For example, if a 300 ohm resistor is attached across the terminals of a 12 volt battery, then a current of  $12 / 300 = 0.04$  amperes (or 40 mill amperes) will flow through that resistor.

### Series and parallel resistors

In a series configuration, the current through all of the resistors is the same, but the voltage across each resistor will be in proportion to its resistance. The potential difference (voltage) seen across the network is the sum of those voltages, thus the total resistance can be found as the sum of those resistances:

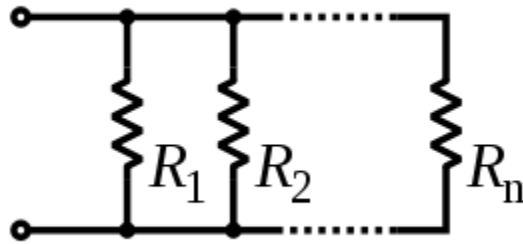


$$R_{eq} = R_1 + R_2 + \cdots + R_n$$

As a special case, the resistance of N resistors connected in series, each of the same resistance R, is given by NR.

Resistors in a parallel configuration are each subject to the same potential difference (voltage), however the currents through them add. The conductance of the resistors then add to determine

the conductance of the network. Thus the equivalent resistance ( $R_{eq}$ ) of the network can be computed:



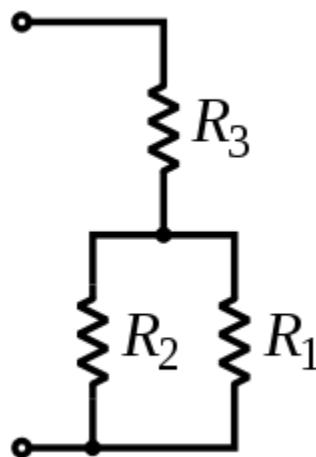
$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_n}$$

The parallel equivalent resistance can be represented in equations by two vertical lines "||" (as in geometry) as a simplified notation. For the case of two resistors in parallel, this can be calculated using:

$$R_{eq} = R_1 || R_2 = \frac{R_1 R_2}{R_1 + R_2}$$

As a special case, the resistance of N resistors connected in parallel, each of the same resistance R, is given by R/N.

A resistor network that is a combination of parallel and series connections can be broken up into smaller parts that are either one or the other. For instance,

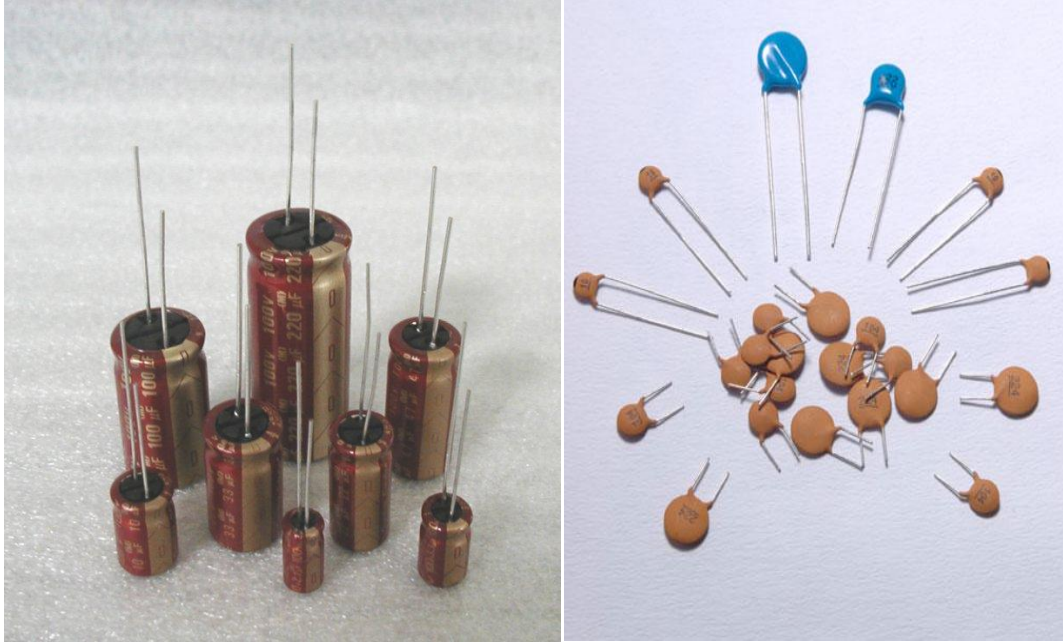


$$R_{eq} = (R_1 || R_2) + R_3 = \frac{R_1 R_2}{R_1 + R_2} + R_3$$

However, some complex networks of resistors cannot be resolved in this manner, requiring more sophisticated circuit analysis. For instance, consider a cube, each edge of which has been replaced by a resistor. What then is the resistance that would be measured between two opposite vertices? In the case of 12 equivalent resistors, it can be shown that the corner-to-corner resistance is  $\frac{5}{6}$  of the individual resistance. More generally, the Y- $\Delta$  transform, or matrix methods can be used to solve such a problem. One practical application of these relationships is that a non-standard value of resistance can generally be synthesized by connecting a number of standard values in series and/or parallel. This can also be used to obtain a resistance with a higher power rating than that of the individual resistors used. In the special case of N identical resistors all connected in series or all connected in parallel, the power rating of the individual resistors is thereby multiplied by N.

## **CAPACITORS**

A capacitor or condenser is a passive electronic component consisting of a pair of conductors separated by a dielectric. When a voltage potential difference exists between the conductors, an electric field is present in the dielectric. This field stores energy and produces a mechanical force between the plates. The effect is greatest between wide, flat, parallel, narrowly separated conductors.



An ideal capacitor is characterized by a single constant value, capacitance, which is measured in farads. This is the ratio of the electric charge on each conductor to the potential difference between them. In practice, the dielectric between the plates passes a small amount of leakage current. The conductors and leads introduce an equivalent series resistance and the dielectric has an electric field strength limit resulting in a breakdown voltage.

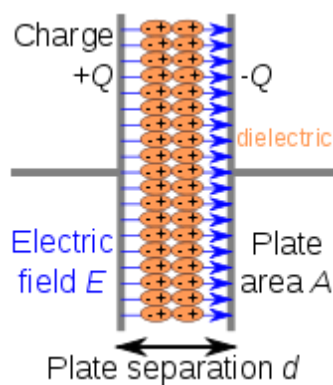
The properties of capacitors in a circuit may determine the resonant frequency and quality factor of a resonant circuit, power dissipation and operating frequency in a digital logic circuit, energy capacity in a high-power system, and many other important aspects.

A capacitor (formerly known as condenser) is a device for storing electric charge. The forms of practical capacitors vary widely, but all contain at least two conductors separated by a non-conductor. Capacitors used as parts of electrical systems, for example, consist of metal foils separated by a layer of insulating film.

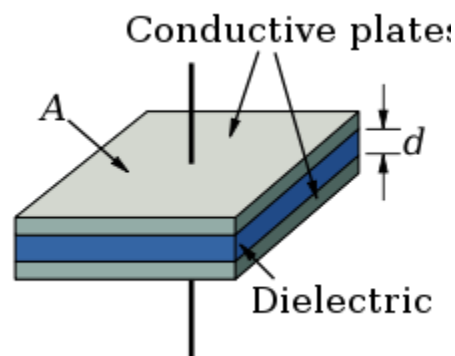
Capacitors are widely used in electronic circuits for blocking direct current while allowing alternating current to pass, in filter networks, for smoothing the output of power supplies, in the resonant circuits that tune radios to particular frequencies and for many other purposes.

A capacitor is a passive electronic component consisting of a pair of conductors separated by a dielectric (insulator). When there is a potential difference (voltage) across the conductors, a static electric field develops in the dielectric that stores energy and produces a mechanical force between the conductors. An ideal capacitor is characterized by a single constant value, capacitance, measured in farads. This is the ratio of the electric charge on each conductor to the potential difference between them.

The capacitance is greatest when there is a narrow separation between large areas of conductor, hence capacitor conductors are often called "plates", referring to an early means of construction. In practice the dielectric between the plates passes a small amount of leakage current and also has an electric field strength limit, resulting in a breakdown voltage, while the conductors and leads introduce an undesired inductance and resistance.



### Parallel plate model



Dielectric is placed between two conducting plates, each of area  $A$  and with a separation of  $d$ .

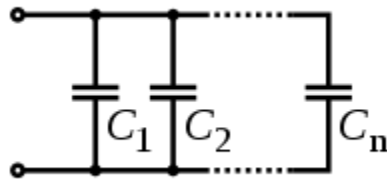
The simplest capacitor consists of two parallel conductive plates separated by a dielectric with permittivity  $\epsilon$  (such as air). The model may also be used to make qualitative predictions for other device geometries. The plates are considered to extend uniformly over an area  $A$  and a charge density  $\pm\rho = \pm Q/A$  exists on their surface. Assuming that the width of the plates is much greater than their separation  $d$ , the electric field near the centre of the device will be uniform with the magnitude  $E = \rho/\epsilon$ . The voltage is defined as the line integral of the electric field between the plates

$$V = \int_0^d E dz = \int_0^d \frac{\rho}{\epsilon} dz = \frac{\rho d}{\epsilon} = \frac{Qd}{\epsilon A}.$$

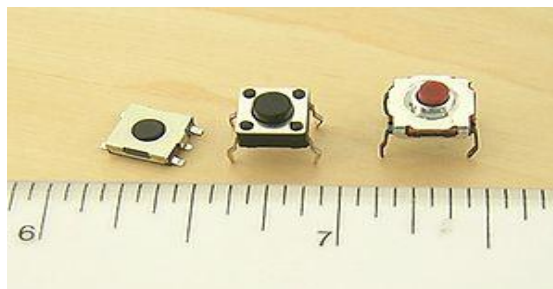
Solving this for  $C = Q/V$  reveals that capacitance increases with area and decreases with separation

$$C = \frac{\epsilon A}{d}.$$

The capacitance is therefore greatest in devices made from materials with a high permittivity.



## PUSH BUTTONS



A push-button (also spelled pushbutton) or simply button is a simple switch mechanism for controlling some aspect of a machine or a process. Buttons are typically made out of hard material, usually plastic or metal. The surface is usually flat or shaped to accommodate the human finger or hand, so as to be easily depressed or pushed. Buttons are most often biased switches, though even many un-biased buttons (due to their physical nature) require a spring to return to their un-pushed state. Different people use different terms for the "pushing" of the button, such as press, depress, mash, and punch.

**Uses:**

In industrial and commercial applications push buttons can be linked together by a mechanical linkage so that the act of pushing one button causes the other button to be released. In this way, a stop button can "force" a start button to be released. This method of linkage is used in simple manual operations in which the machine or process have no electrical circuits for control.

Pushbuttons are often color-coded to associate them with their function so that the operator will not push the wrong button in error. Commonly used colors are red for stopping the machine or process and green for starting the machine or process.

Red pushbuttons can also have large heads (mushroom shaped) for easy operation and to facilitate the stopping of a machine. These pushbuttons are called emergency stop buttons and are mandated by the electrical code in many jurisdictions for increased safety. This large mushroom shape can also be found in buttons for use with operators who need to wear gloves for their work and could not actuate a regular flush-mounted push button. As an aid for operators and users in industrial or commercial applications, a pilot light is commonly added to draw the attention of the user and to provide feedback if the button is pushed. Typically this light is included into the center of the pushbutton and a lens replaces the pushbutton hard center disk.

The source of the energy to illuminate the light is not directly tied to the contacts on the back of the pushbutton but to the action the pushbutton controls. In this way a start button when pushed will cause the process or machine operation to be started and a secondary contact designed into the operation or process will close to turn on the pilot light and signify the action of pushing the button caused the resultant process or action to start.



In popular culture, the phrase "the button" refers to a (usually fictional) button that a military or government leader could press to launch nuclear weapons.

**Push to ON button:**

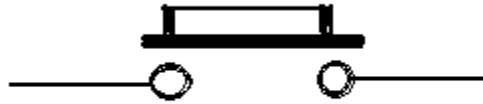


Fig 4.3(a): push on button

Initially the two contacts of the button are open. When the button is pressed they become connected. This makes the switching operation using the push button.

**DESCRIPTION:**

**POWER SUPPLY**

The circuit uses standard power supply comprising of a step-down transformer from 230V to 12V and 4 diodes forming a bridge rectifier that delivers pulsating dc which is then filtered by an electrolytic capacitor of about 470 $\mu$ F to 1000 $\mu$ F. The filtered dc being unregulated, IC LM7805 is used to get 5V DC constant at its pin no 3 irrespective of input DC varying from 7V to 15V. The input dc shall be varying in the event of input ac at 230volts section varies from 160V to 270V in the ratio of the transformer primary voltage V1 to secondary voltage V2 governed by the formula  $V1/V2=N1/N2$ . As  $N1/N2$  i.e. no. of turns in the primary to the no. of turns in the secondary remains unchanged V2 is directly proportional to V1. Thus if the transformer delivers 12V at 220V input it will give 8.72V at 160V. Similarly at 270V it will give 14.72V. Thus the dc voltage at the input of the regulator changes from about 8V to 15V because of A.C voltage variation from 160V to 270V the regulator output will remain constant at 5V.

The regulated 5V DC is further filtered by a small electrolytic capacitor of 10 $\mu$ F for any noise so generated by the circuit. One LED is connected of this 5V point in series with a current

limiting resistor of  $330\Omega$  to the ground i.e., negative voltage to indicate 5V power supply availability. The unregulated 12V point is used for other applications as and when required.

## **OPERATION EXPLANATION**

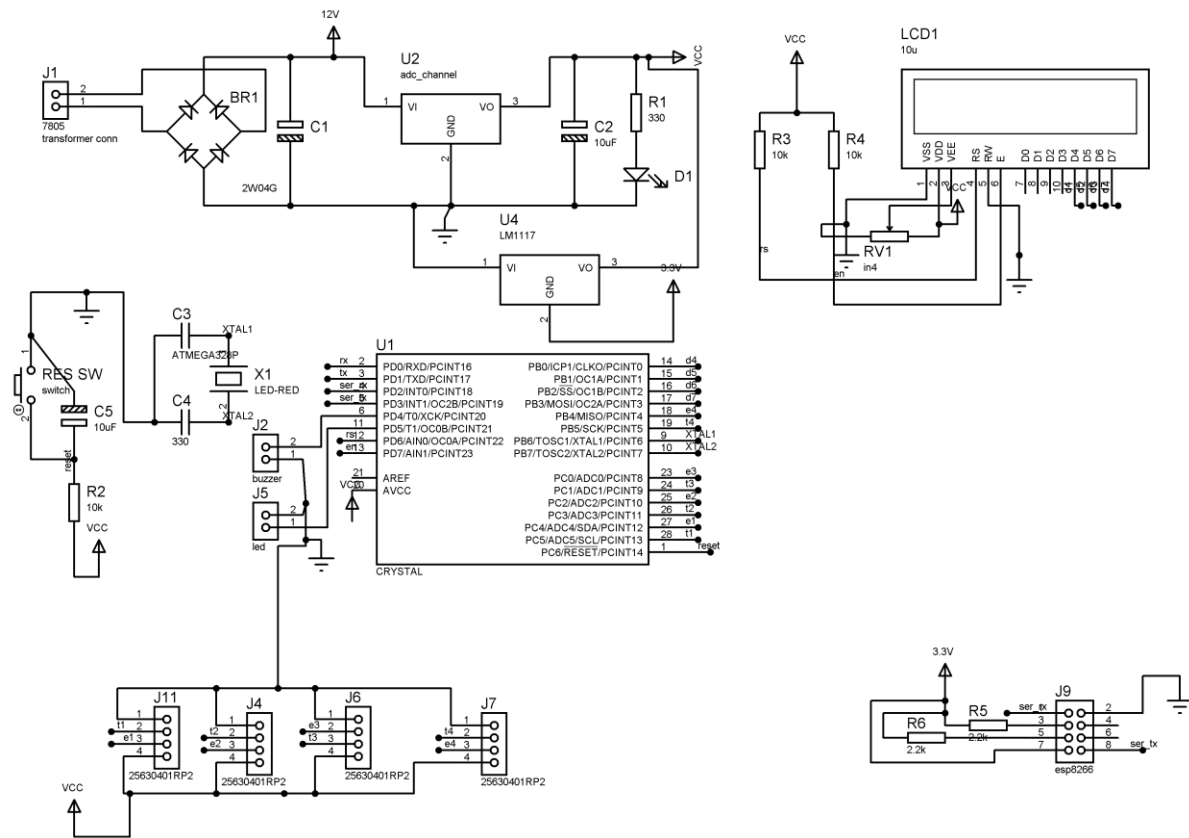
### **CONNECTIONS:**

The output of the power supply which is 5v is connected to the 40<sup>th</sup> Pin of microcontroller and GND is connected to its 20<sup>th</sup> Pin. Pin 21- 28 of port2 of micro controller is connected to D0-D7 data pins of LCD. Pin10 of Port3.0 of microcontroller is connected to ultrasonic module pin no.2. Pin 15, 16, 17 of Port0 of microcontroller is connected to RS, RW, EN pins of LCD.

### **WORKING:**

The project uses an ultrasonic sensor module comprising of 1 transmitter and 1 receiver. The transmitter can deliver 40 KHz ultrasonic sound while the maximum receiver is designed to accept only 40 KHz sound waves. The receiver ultrasonic sensor that is kept next to the transmitter shall thus be able to receive reflected 40 KHz, once the module faces any obstacle in front. Thus whenever any obstacles come ahead of the ultrasonic module it calculates the time taken from sending the signals to receiving them since time and distance are related for sound waves passing through air medium at 343.2m/sec. Upon receiving the signal MC program while executed displays the data i.e. the distance measured on a 16X2 LCD interfaced to the microcontroller in cms.

### Circuit Diagram





# SOFTWARE REQUIREMENTS

## INTRODUCTION TO AVR Studio 5

AVR Studio 5 is a free integrated development environment (IDE) for programming AVR microcontrollers offered by Atmel for Atmel's 8-bits, 32-bits and ARM Cortex-M families of AVR microcontrollers. AVR Studio 5 works with the WinAVR-gcc compiler and contains built-in support for AVR ISP programming.

It is a full software development environment with an editor, simulator, programmer, etc.

- It comes with its own integrated C compiler the **AVR GNU C Compiler**

(GCC). As such you do not need a third party C compiler.

- It provides a single environment to develop programs for both the 8-bits, 32-bits and ARM Cortex-M AVR series of microcontrollers.

- It also integrates fully QTouch studio.

- Provides support for several programmers including the STK500, AVR Dragon, etc.

## Installation

Requirements Windows 98/NT/2000/XP XP x64/VISTA/WIN 7 Internet Explorer 6.0 or later (Latest version is recommended) Recommended hardware: ~ Intel Pentium 200MHz processor or equivalent ~ 1024x768 screen (minimum 800x600 screen) ~ 256 MB memory ~ 100 MB free hard disk space

## CONCEPT OF COMPILER

Compilers are programs used to convert a High Level Language to object code. Desktop compilers produce an output object code for the underlying microprocessor, but not for other microprocessors. I.E the programs written in one of the HLL like 'C' will compile the code to run on the system for a particular processor like x86 (underlying microprocessor in the computer). For example compilers for Dos platform is different from the Compilers for Unix

platform. So if one wants to define a compiler then compiler is a program that translates source code into object code.

The compiler derives its name from the way it works, looking at the entire piece of source code and collecting and reorganizing the instruction. See there is a bit little difference between compiler and an interpreter. Interpreter just interprets whole program at a time while compiler analyses and execute each line of source code in succession, without looking at the entire program.

The advantage of interpreters is that they can execute a program immediately. Secondly programs produced by compilers run much faster than the same programs executed by an interpreter. However compilers require some time before an executable program emerges. Now as compilers translate source code into object code, which is unique for each type of computer, many compilers are available for the same language.

## **CONCEPT OF CROSS COMPILER**

A cross compiler is similar to the compilers but we write a program for the target processor on the host processors (like computer of x86). It means being in one environment you are writing a code for another environment is called cross development. And the compiler used for cross development is called cross compiler. So the definition of cross compiler is a compiler that runs on one computer but produces object code for a different type of computer.

## **KEIL CROSS COMPILER**

WinAVR<sup>TM</sup> (pronounced "whenever") is a suite of executable, open source software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform. It includes the GNU GCC compiler for C and C++.

WinAVR<sup>TM</sup> contains all the tools for developing on the AVR. This includes avr-gcc (compiler), avrdude (programmer), avr-gdb (debugger), and more! WinAVR is used all over the world from hobbyists sitting in their damp basements, to schools, to commercial projects.

WinAVR<sup>TM</sup> is comprised of many open source projects. If you feel adventurous, volunteers are always welcome to help with fixing bugs, adding features, porting, writing documentation and many other tasks on a variety of projects.

## Building an Application in AVR Studio

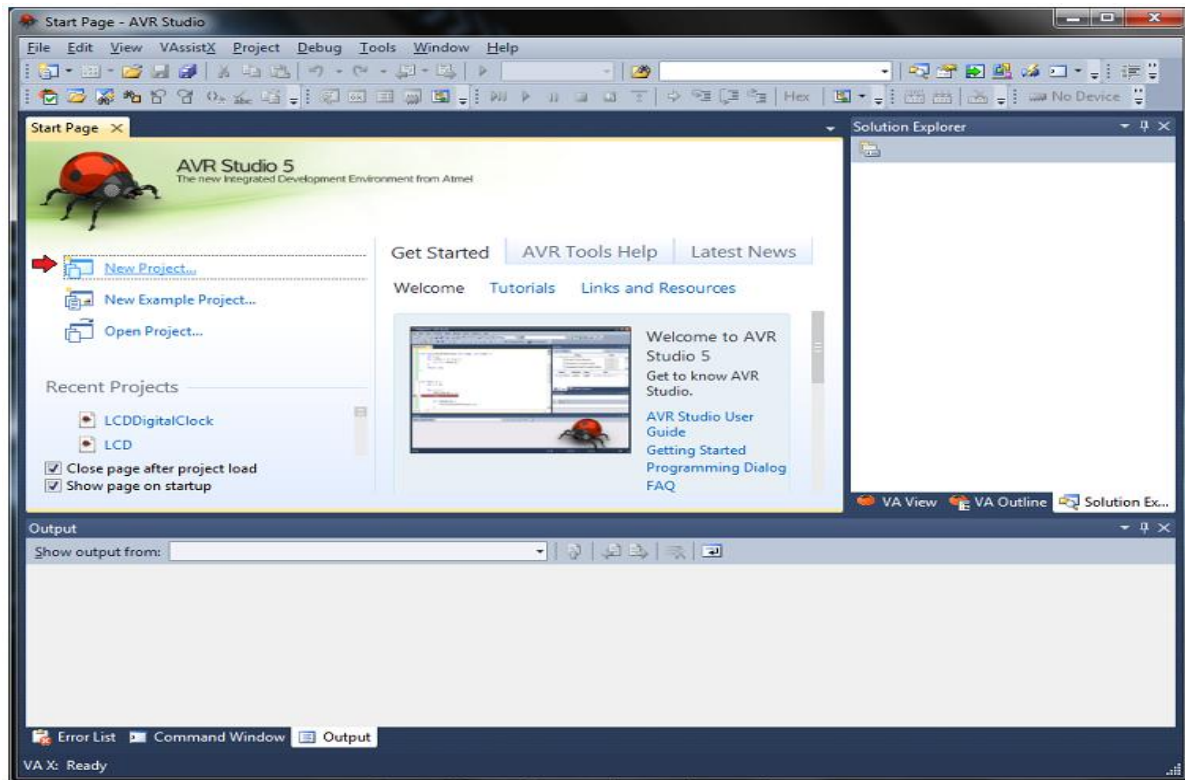
To build (compile, assemble, and link) an application in AVR Studio, you must:

1. Select Project
2. Select Build - Rebuild all target files or Build target.
3. AVR Studio compiles, assembles, and links the files in your project.

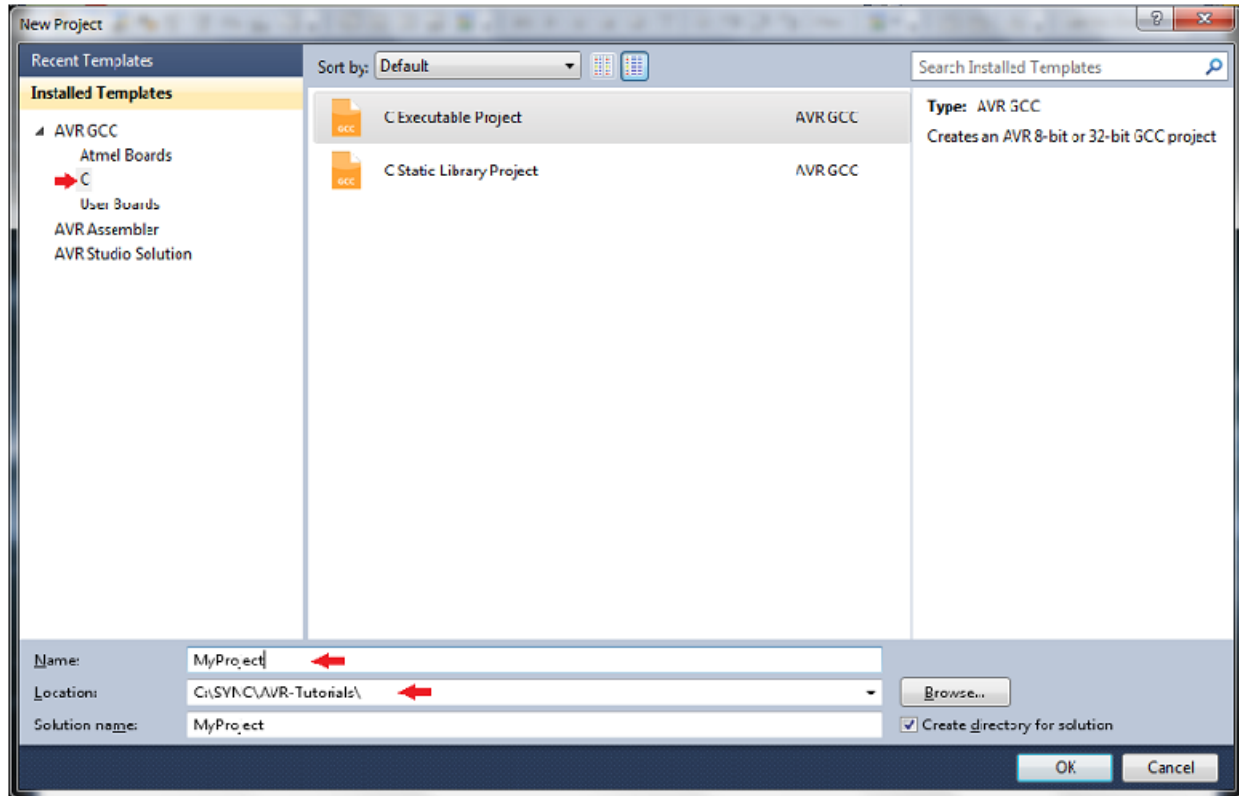
## Creating Your Own Application / New Project in AVR Studio

To create a new project in AVR Studio, you must:

1. Open AVR Studio 5

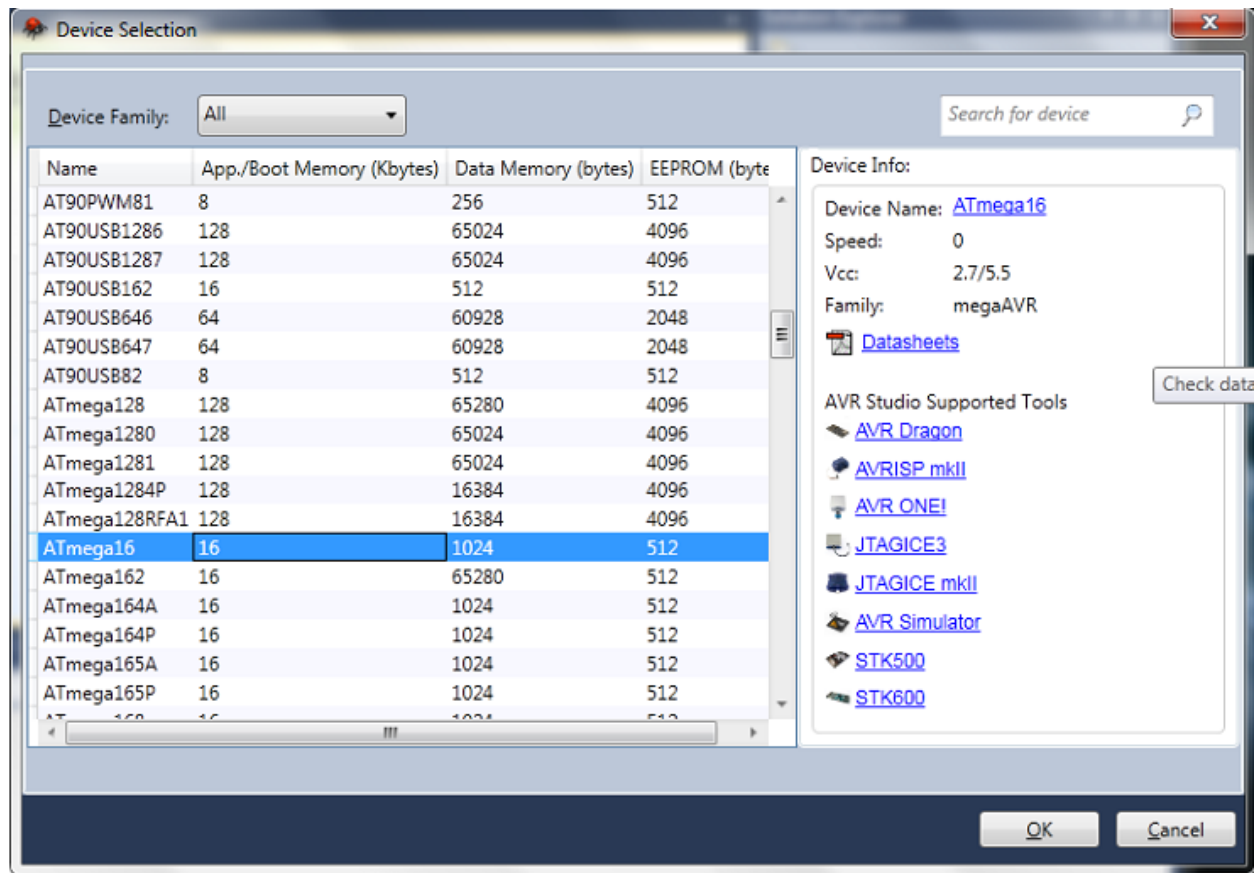


2. Select Project - New Project.

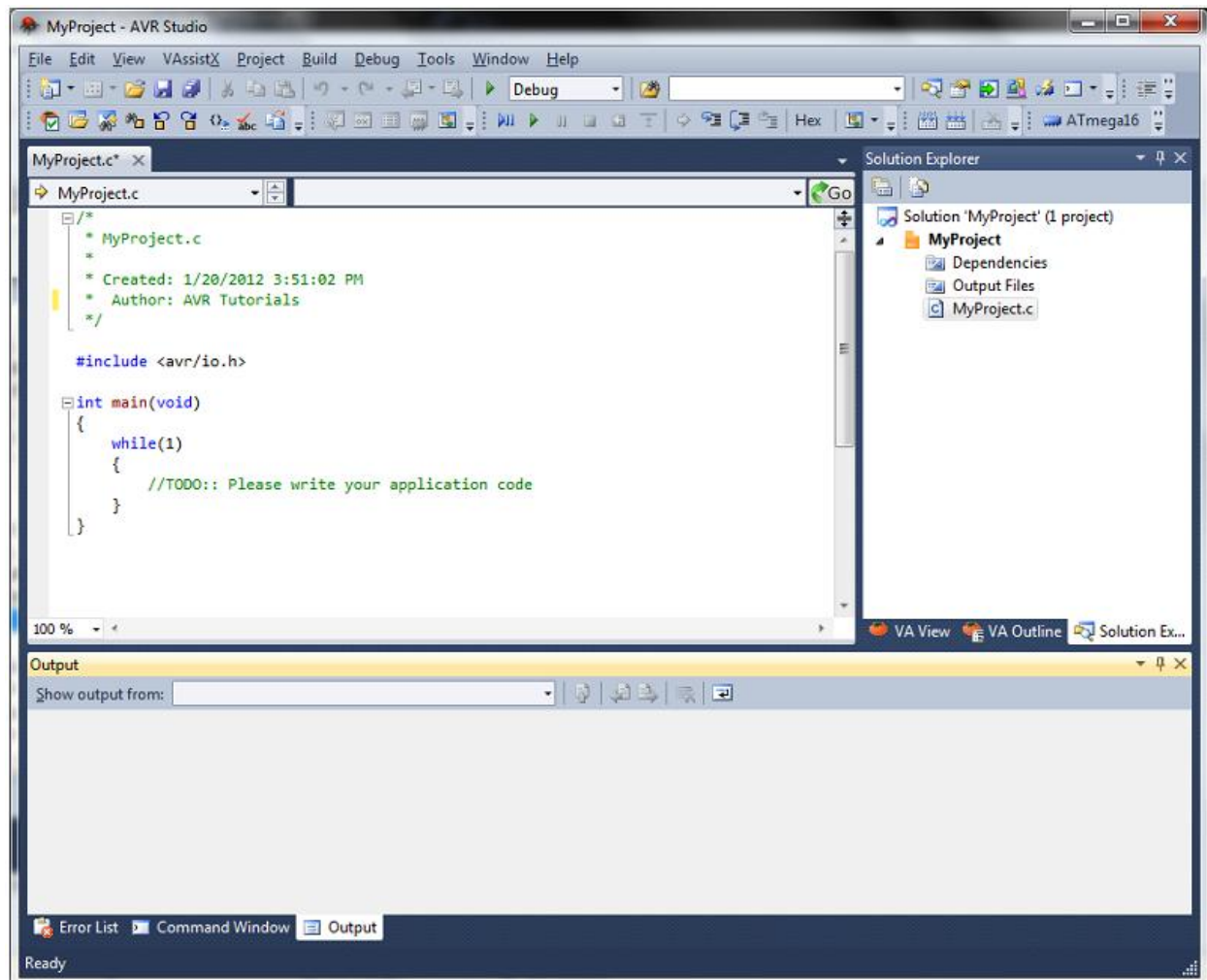


3. Write name of Project in the space given on the right hand side of label Name.
4. Similarly, write the location of where you want the project to be saved ( Browsing to the location and selecting it will also do)
5. Click on OK if you are sure of the above said steps have been done and are correctly done.
6. Below is the device selection screen for AVR Studio 5. Scroll down and select the microcontroller you will be using.





7. This is the AVR studio 5 editor where you type your C program. The editor starts your C program for you by providing you with the structure shown in the editor of the figure below.
8. Save the project once done with programming.
9. Build the target to generate the machine understandable hex code and other binaries.
10. If all the above steps have been followed correctly and after building the target no errors are coming then the necessary above said files get generated. Otherwise one has to debug the code.



## Debugging a Project in AVR Studio 5

The AVR Studio 5 Simulator has the following features:

- It supports software emulation of any real AVR device without actually connecting it.
- It gives access to all the peripherals of the real MCU but no external devices.
- So if you want to give external signals, you need to do it yourself, either by manually updating the registers or creating a stimuli file.

Now, let's take the following code example to explain the functionality of the AVR Simulator.

```
1    #include <avr/io.h>
2
3    int main(void)
```

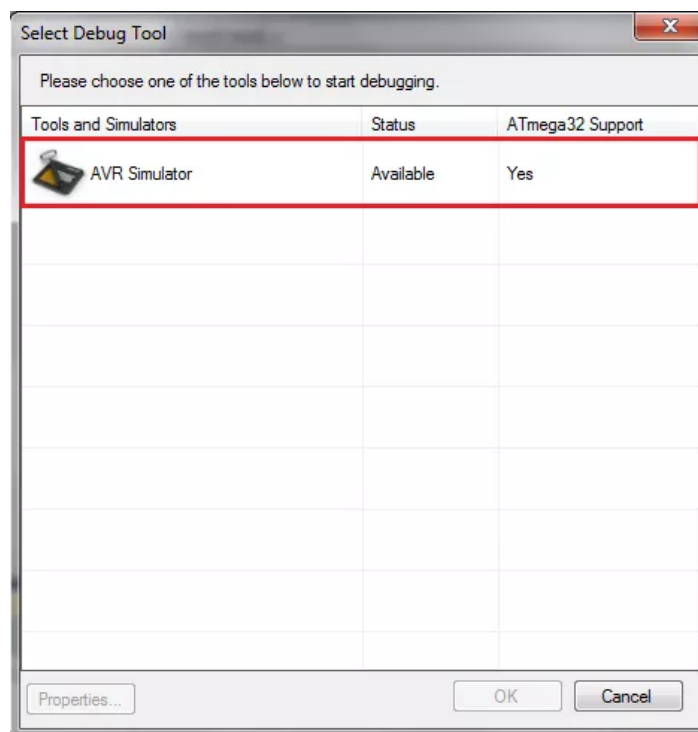
```

4    {
5        uint8_t counter;
6        DDRB = 0xFF;
7        while(1)
8        {
9            counter++; // insert breakpoint here <-----
10           PORTB = counter;
11        }
12    }

```

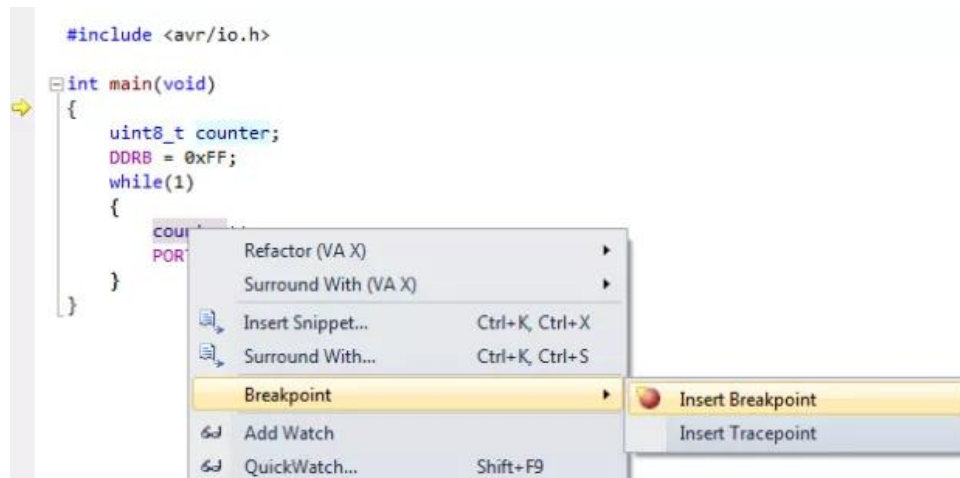
### Debugging the code using Simulator:

- Now, click on the **Debug** menu and then click on **Start Debugging and Break**. If initially no debugger is chosen, AVR Studio 5 will ask you to choose a **Debug Tool**. **AVR Simulator** is always an option there. Choose it and click **OK**.

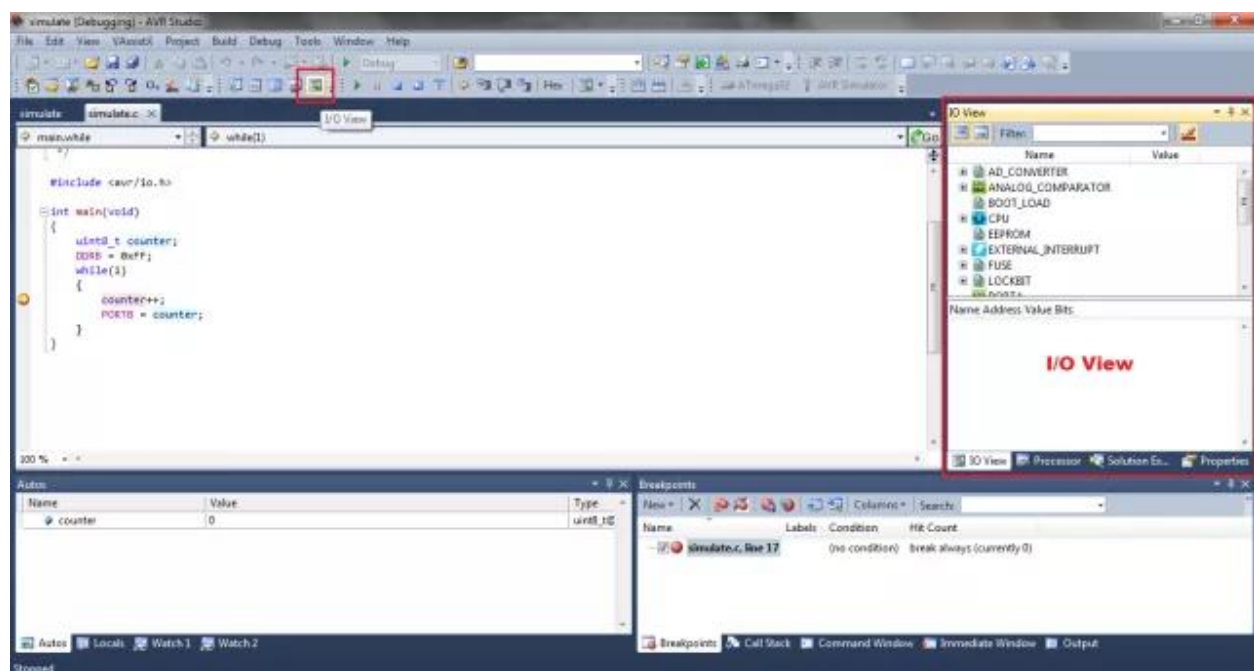


- After this, debugging starts and halts in the beginning of main(). You can see a yellow arrow mark determining the current executing line.

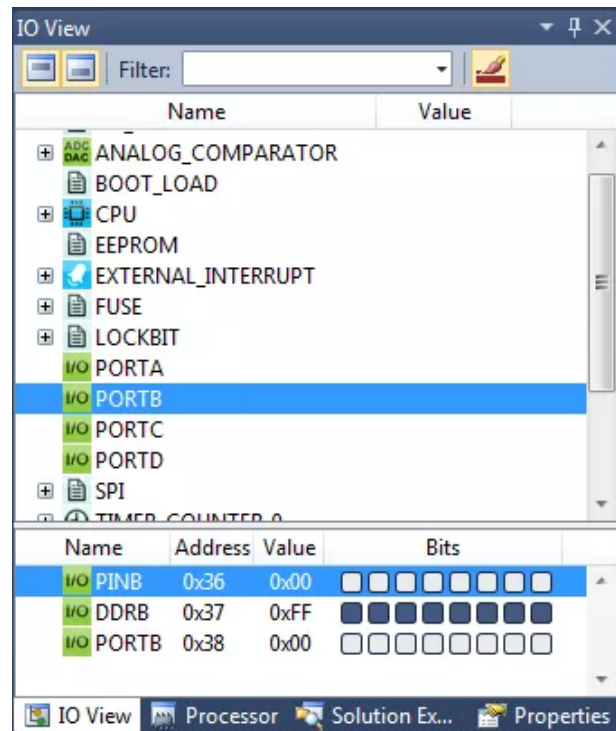
Let's place a breakpoint in the main and start execution. Highlight the variable counter in counter++, right click it, go to **Breakpoint** and then click on **Insert Breakpoint**



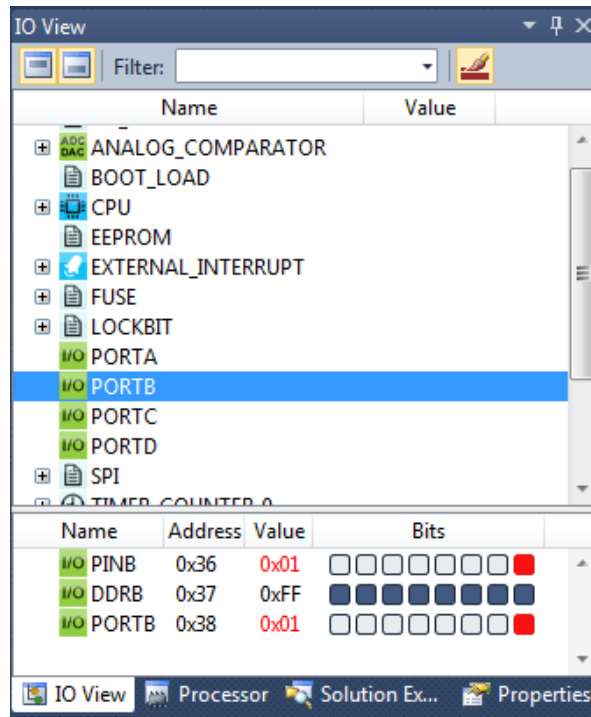
- Now press the **play** button (**F5**) or click on **Continue** from the **Debug** menu to run to the breakpoint.
- Now look at the affected registers in the **I/O view**. If you don't have the I/O View open, you can select it from the **Debug** toolbar or from the **Debug** windows menu.



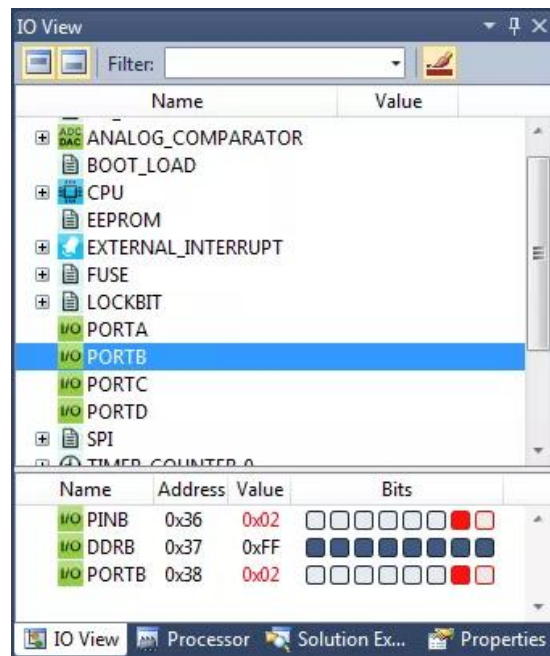
- All the peripheral features are mentioned over here. We can monitor any changes from the software and also manipulate the values to provide input.
- Now, since the counter changes the value of PORTB, scroll down in the I/O View and click on PORTB.

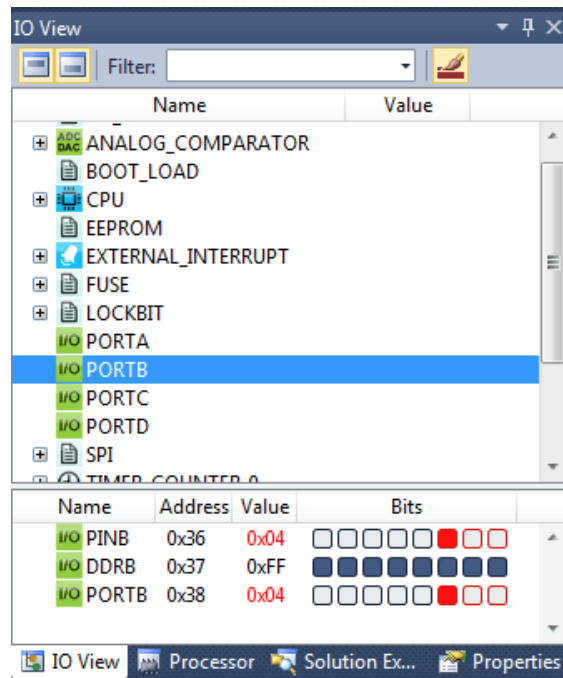
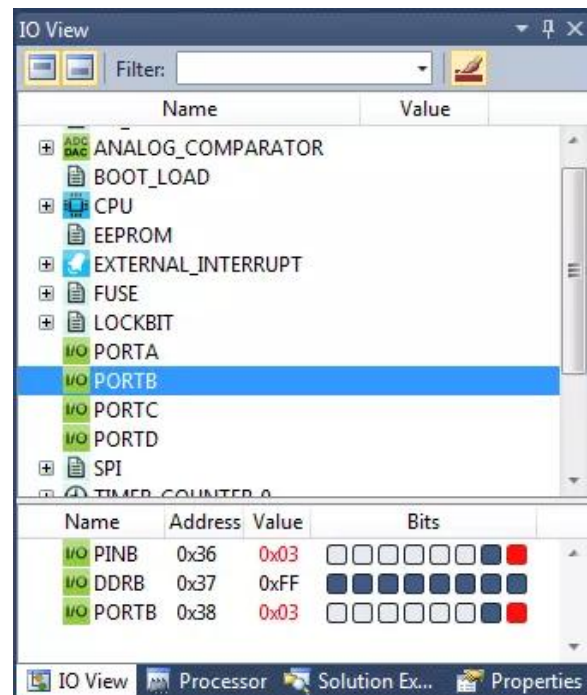


- Upon clicking PORTB, you can see the three registers assigned for PORTB operations, PINB, DDRB and PORTB. You can also view their current values.
- A solid block represents '1' whereas a blank block represents '0'.
- Since it the beginning of main(), we defined DDRB = 0xFF, all the blocks are filled. You can also look at its value there.
- Now, press the play button. The loop iterates once and stops at the breakpoint. You can see that values of PINB and PORTB have changed to 0x01. This is because after one iteration, counter = 1.

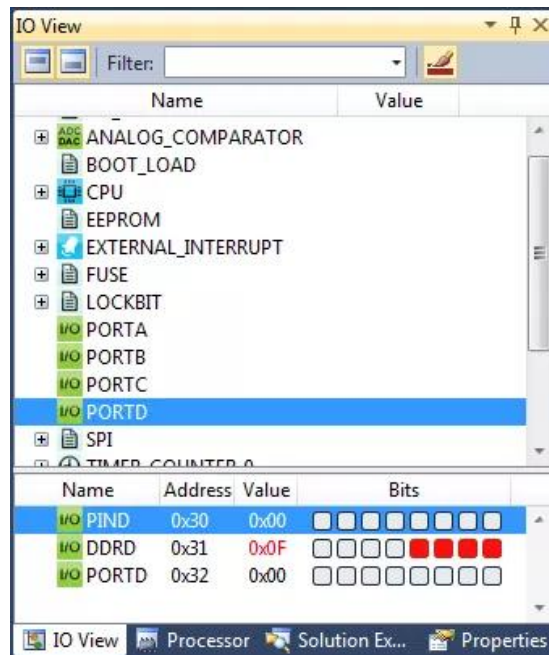


- The red block indicates that there has been a change in the value of the bit. If it's a solid red block, a change has been there from 0 to 1. If it's just a red outline, it's the other way round.
- Once again click on play. You will be able to see the following sequences.





- Now, if you want to change some other registers (apart from the ones changed by the code), simply click on the corresponding register and change its value.
- Say for example you want to change the value of DDRD. Click on PORTD and then give any value you want. You can also click on the corresponding bits to toggle the values.



So now we are done with the basics of AVR Studio 5. There are more advanced features of debugging in AVR Studio 5 which includes In-System Debugging which is a kind of runtime debugging unlike the software emulation that we learnt in this post.

## EMBEDDED C

Use of embedded processors in passenger cars, mobile phones, medical equipment, aerospace systems and defense systems is widespread, and even everyday domestic appliances such as dish washers, televisions, washing machines and video recorders now include at least one such device.

Because most embedded projects have severe cost constraints, they tend to use low-cost processors like the 8051 family of devices considered in this book. These popular chips have very limited resources available most such devices have around 256 bytes (not megabytes!) of RAM, and the available processor power is around 1000 times less than that of a desktop processor. As a result, developing embedded software presents significant new challenges, even for experienced desktop programmers. If you have some programming experience - in C, C++ or Java - then this book and its accompanying CD will help make your move to the embedded world as quick and painless as possible.



## Programming Code

USBasp is awesome USB based programmer for the AVR. In this tutorial we will see how to use AVRdude for burning hex files into AVR microcontroller using USBasp.

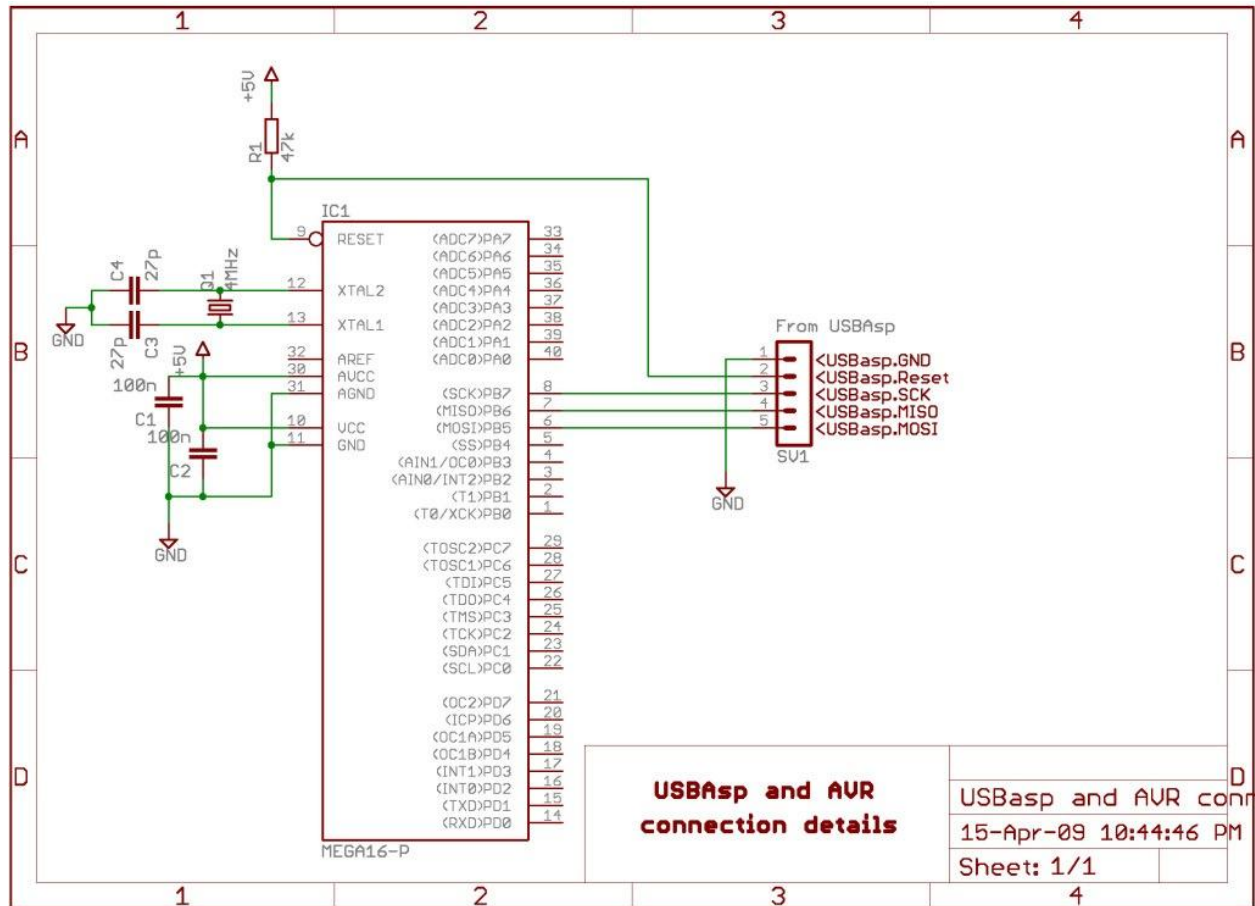
AVRdude is not readily available for windows installation. Windows version of AVRdude is bundled with WinAVR compiler

### Introduction:

In order to program any microcontroller you need the .HEX file. It is nothing but the machine code for the microcontroller. This file is generated by the corresponding assembler software, which converts assembly code into machine code. Assembly can be produced by third party cross compiler software or can be handwritten.

### Connections:

- Connect the USBasp to PC.
- Connect SPI programming pins of USBasp to the AVR microcontroller. Following figure shows sample schematic diagram, if you have different AVR, then connect MOSI,MISO, SCK, RESET and GND pins of that uC to corresponding pins of USBasp.



– Give +5V supply to the microcontroller.

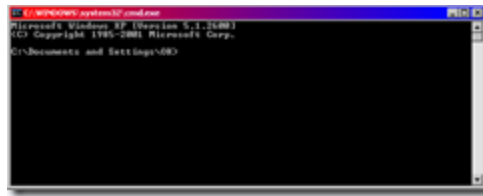
– **VIMP:** If you are burning a fresh microcontroller, close the Slow Serial Clock jumper of USBasp. Since many brand new microcontrollers are factory programmed for internal 1MHz oscillator. USBasp uses very high speed serial clock for faster programming. Thus you will have to specifically tell USBasp to use slow serial clock. This setting is done by above mentioned jumper.

**NOTE:** If you have uC which has internal oscillator enabled and after the programming you are not planning to change its fuse bits back to external clock setting, then you can skip the crystal.

## Executing AVRdude:

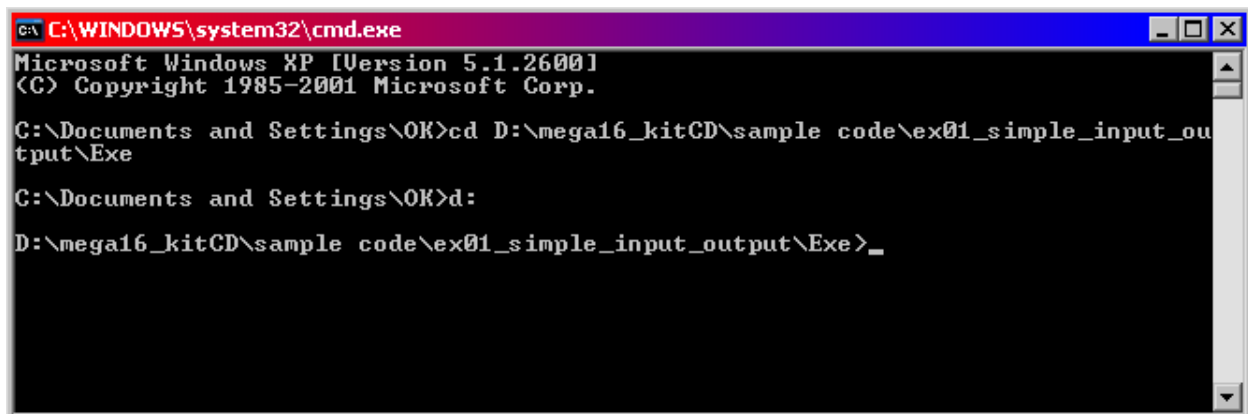
Fortunately AVRdude is command line tool, so that you can be very sure of what you are doing with your uC Or Unfortunately AVRdude is command line tool, so you will have to spend little time to get familiar with it

– Open the command prompt. (Press WinKey + R. Run dialogbox will appear. Type *cmd* and press enter.)



– Navigate to the directory where .hex file is located. For example :

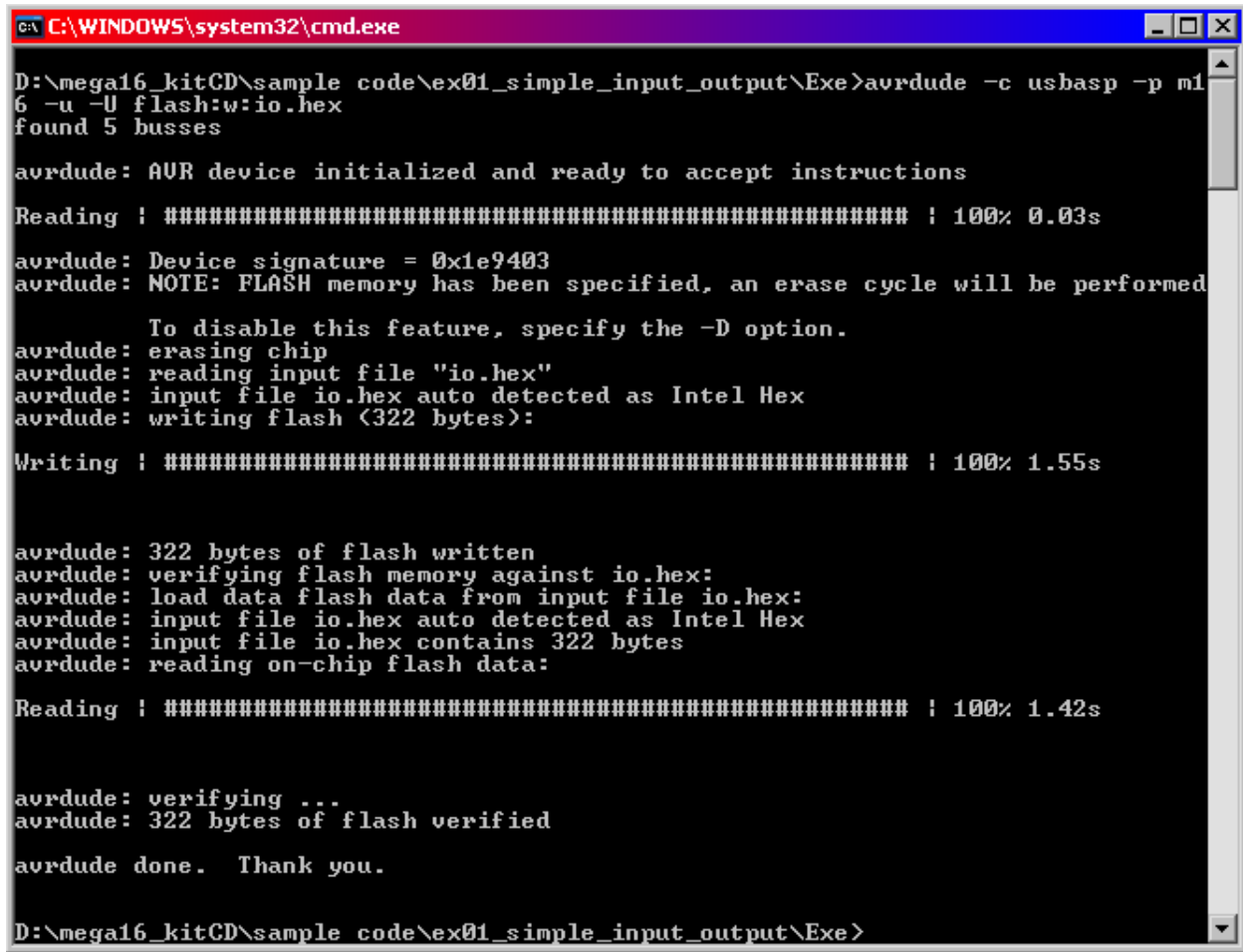
**>cd D:\mega16\_kitCD\sample code\ex01\_simple\_input\_output\Exe**



- To burn the hex file enter following command. Consider for example
- name of my hex file is *io.hex* :

**>avrdude -c usbasp -p m16 -u -U flash:w:io.hex**

You should see something like this :



```
C:\WINDOWS\system32\cmd.exe

D:\mega16_kitCD\sample code\ex01_simple_input_output\Exe>avrdude -c usbasp -p m16 -u -U flash:w:io.hex
found 5 busses

avrdude: AVR device initialized and ready to accept instructions

Reading : ##### : 100% 0.03s

avrdude: Device signature = 0x1e9403
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
        To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "io.hex"
avrdude: input file io.hex auto detected as Intel Hex
avrdude: writing flash (322 bytes):

Writing : ##### : 100% 1.55s

avrdude: 322 bytes of flash written
avrdude: verifying flash memory against io.hex:
avrdude: load data flash data from input file io.hex:
avrdude: input file io.hex auto detected as Intel Hex
avrdude: input file io.hex contains 322 bytes
avrdude: reading on-chip flash data:

Reading : ##### : 100% 1.42s

avrdude: verifying ...
avrdude: 322 bytes of flash verified

avrdude done. Thank you.

D:\mega16_kitCD\sample code\ex01_simple_input_output\Exe>
```

`avrdude -c usbasp -p m16 -u -U flash:w:io.hex`

-c : Indicates the programmer type. Since we are using the USBasp programmer, argument “usbasp” is mentioned.

-p : Processor. We are using ATmega16, hence “m16”. Note ATmega16 has two variants, one is “ATmega16L” (slow speed version) and “ATmega16” normal 16MHz version. However their device signature is same and hence you will have to use “m16” as parameter for both the AVRs. This applies to all AVRs having “L” variants.

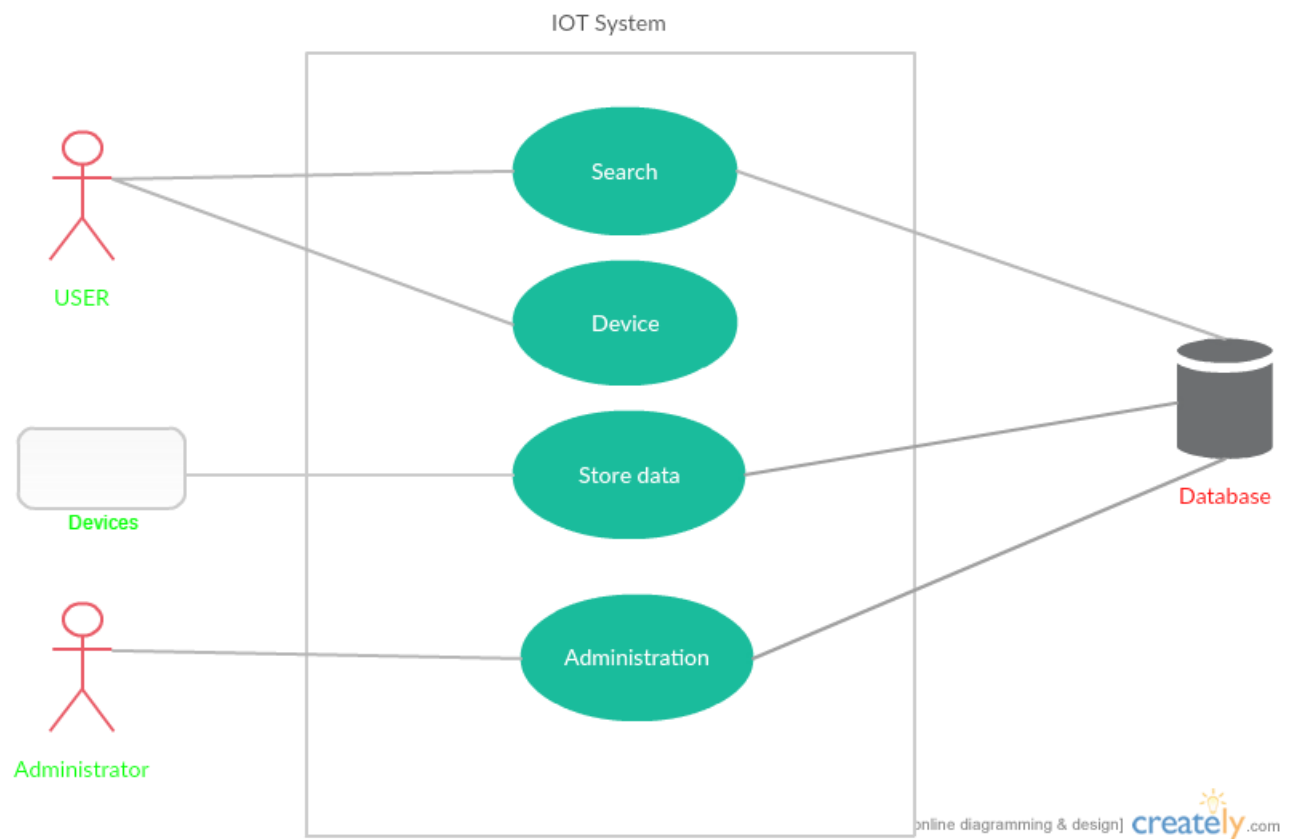
-u : Disables the default behavior of reading out the fuses three times before programming, then verifying at the end of programming that the fuses have not changed. Always use this option. Many times it happens that we forget to switch on the AVR's +5V power supply, then at the end of programming cycle, avrdude detects inconsistent fuses and tries to reprogram them. Since there is no power supply, fuses gets programmed incorrectly and entire microcontroller gets screwed up(means becomes useless). Thus always use this option.

-U : memtype:op:filename[:format]

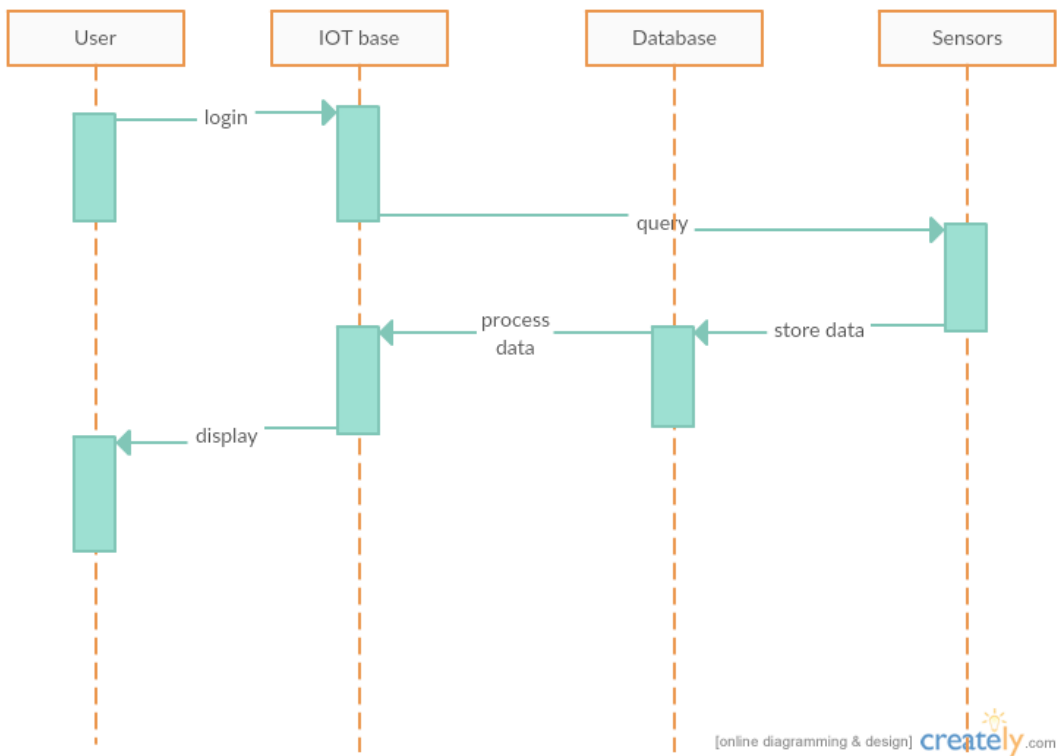
Perform a memory operation. Multiple '-U' options can be specified in order to operate on multiple memories on the same command-line invocation.

## UML DIAGRAMS:

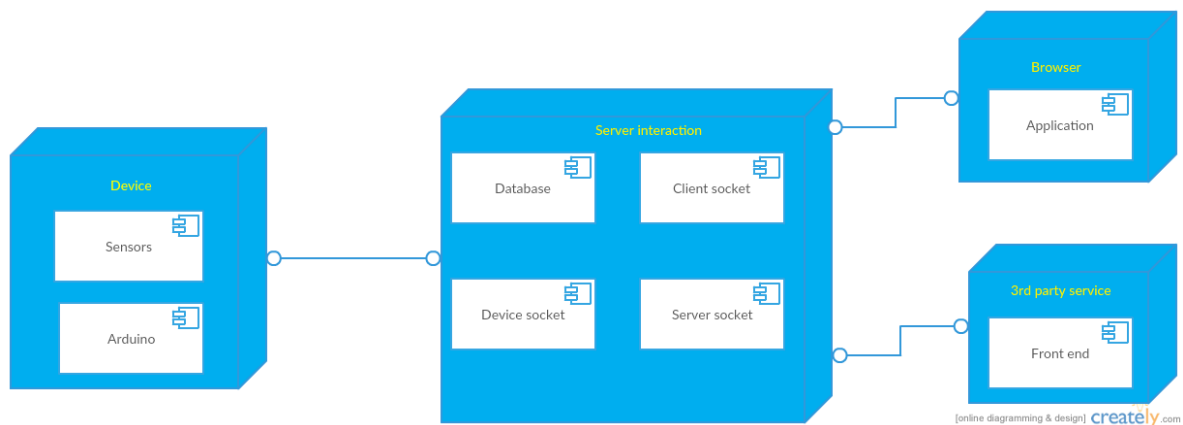
### USECASE DIAGRAM



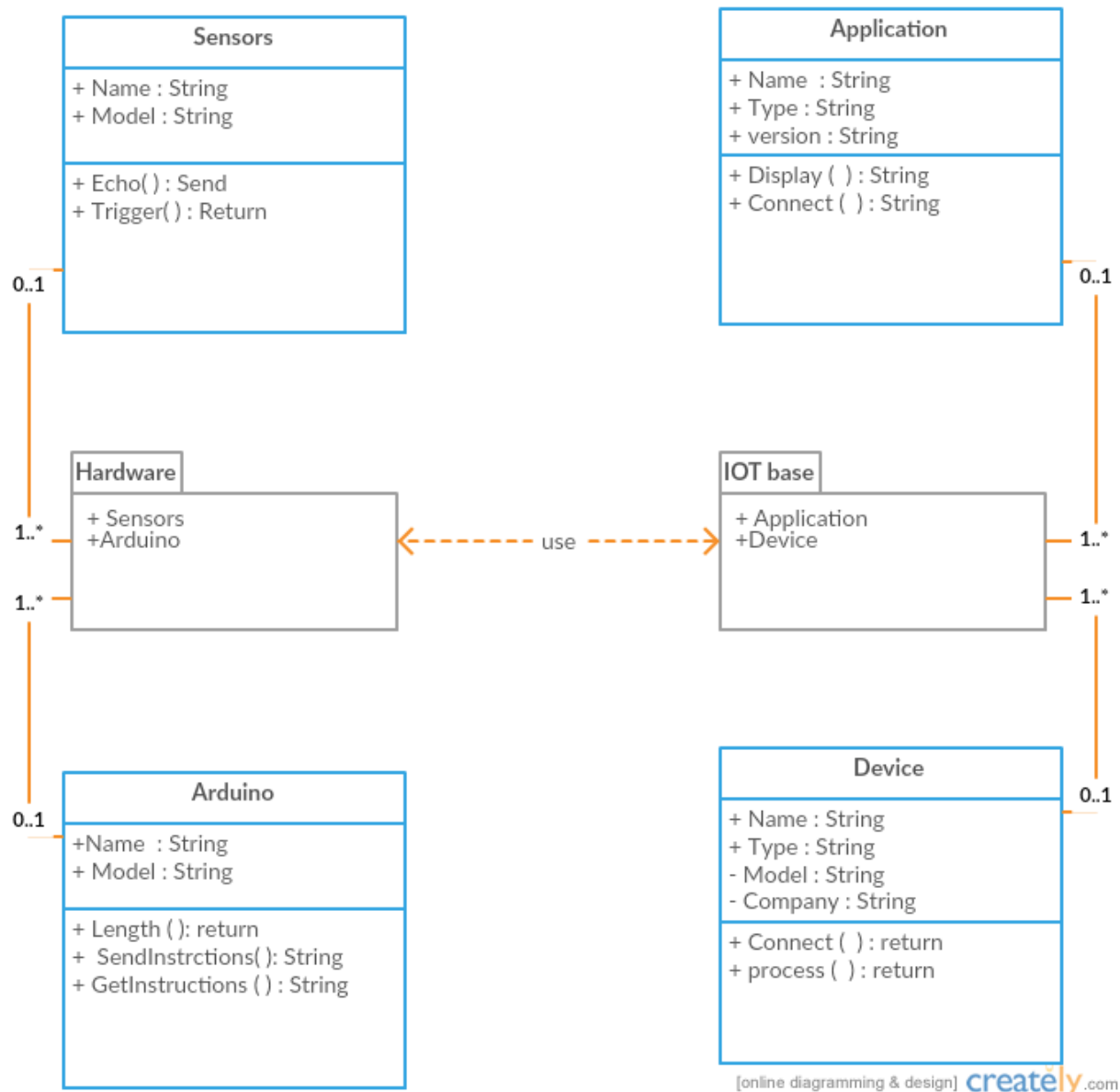
## SEQUENCE DIAGRAM :



## DEPLOYMENT DIAGRAM

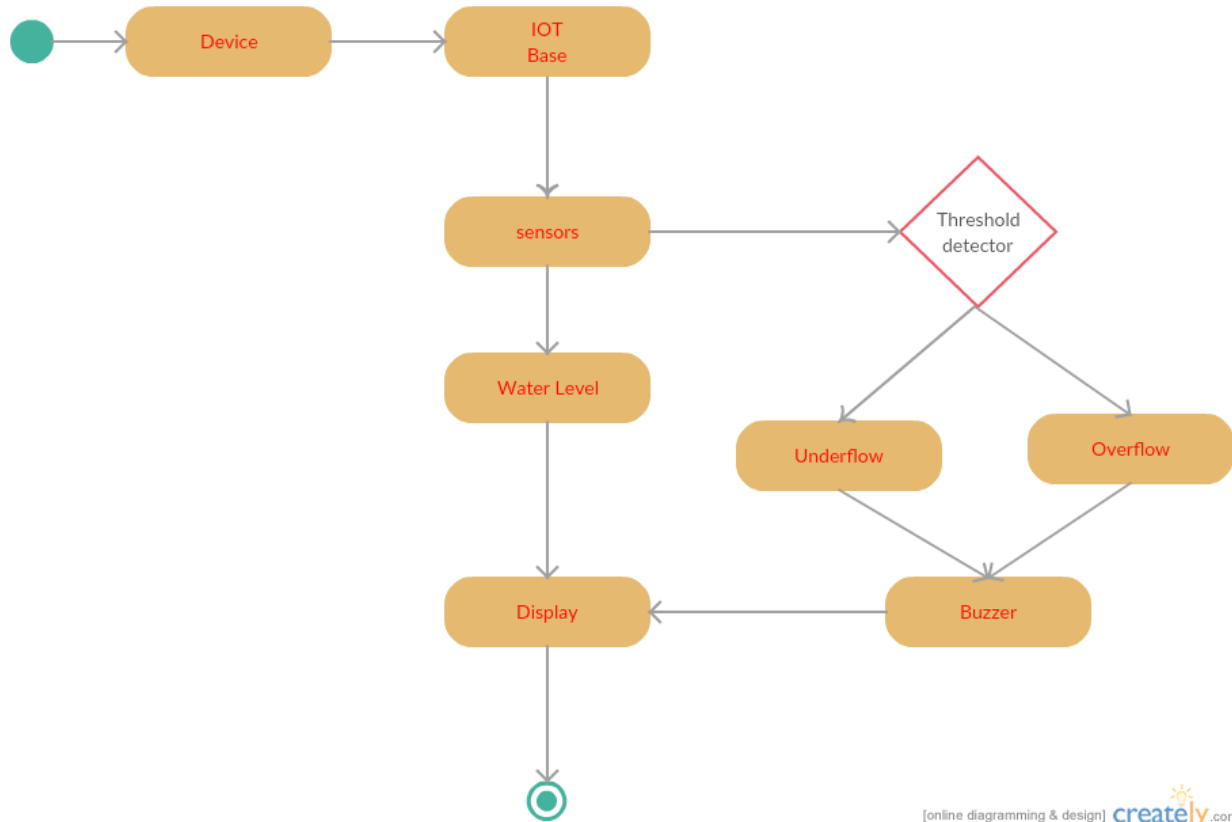


## CLASS DIAGRAM:





## ACTIVITY DIAGRAM:



[online diagramming & design] [creately.com](https://creately.com)

# **HARDWARE TESTING**

## **CONTINUITY TEST:**

In electronics, a continuity test is the checking of an electric circuit to see if current flows (that it is in fact a complete circuit). A continuity test is performed by placing a small voltage (wired in series with an LED or noise-producing component such as a piezoelectric speaker) across the chosen path. If electron flow is inhibited by broken conductors, damaged components, or excessive resistance, the circuit is "open".

Devices that can be used to perform continuity tests include multi meters which measure current and specialized continuity testers which are cheaper, more basic devices, generally with a simple light bulb that lights up when current flows.

An important application is the continuity test of a bundle of wires so as to find the two ends belonging to a particular one of these wires; there will be a negligible resistance between the "right" ends, and only between the "right" ends.

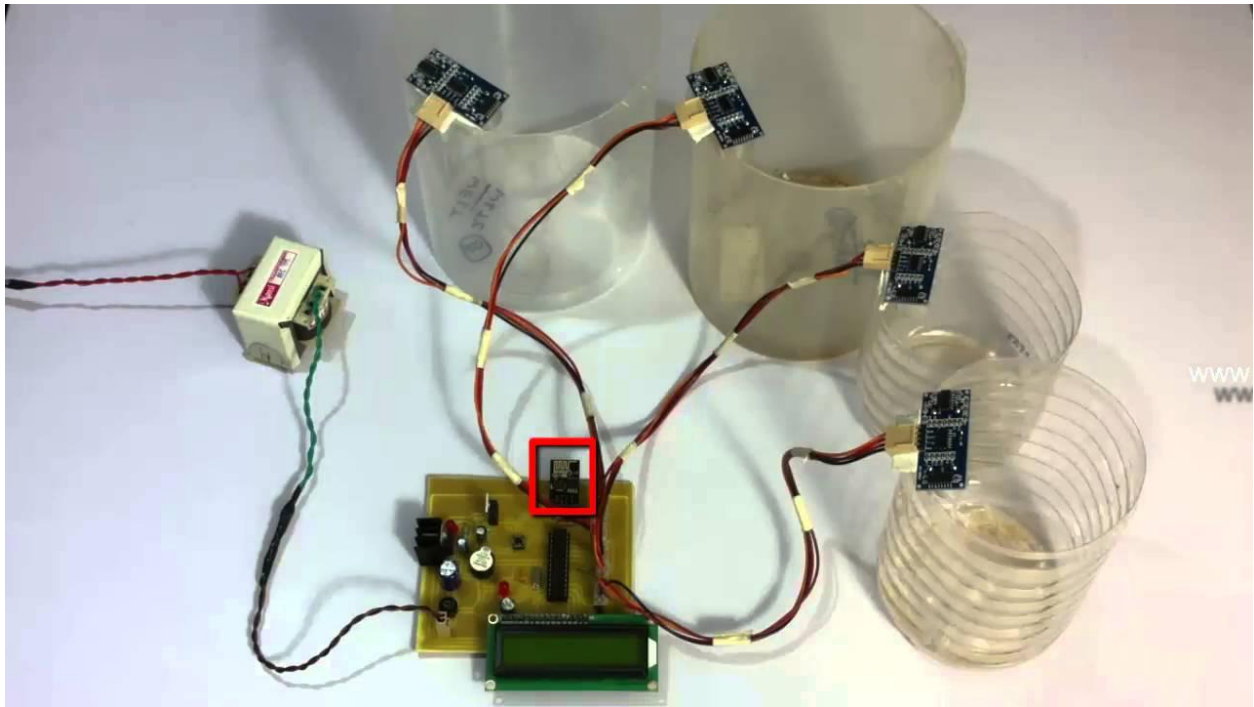
This test is the performed just after the hardware soldering and configuration has been completed. This test aims at finding any electrical open paths in the circuit after the soldering. Many a times, the electrical continuity in the circuit is lost due to improper soldering, wrong and rough handling of the PCB, improper usage of the soldering iron, component failures and presence of bugs in the circuit diagram. We use a multi meter to perform this test. We keep the multi meter in buzzer mode and connect the ground terminal of the multi meter to the ground. We connect both the terminals across the path that needs to be checked. If there is continuation then you will hear the beep sound.

## **POWER ON TEST:**

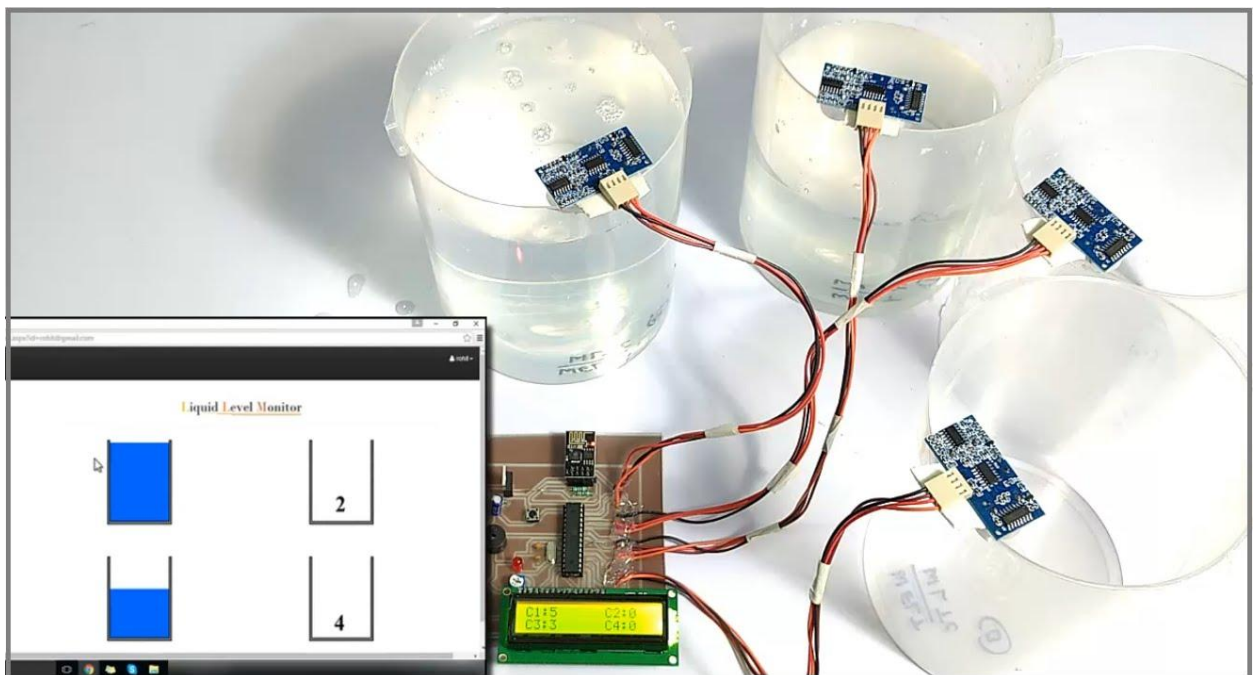
This test is performed to check whether the voltage at different terminals is according to the requirement or not. We take a multi meter and put it in voltage mode. Remember that this test is performed without microcontroller. Firstly, we check the output of the transformer, whether we get the required 12 v AC voltage.

Then we apply this voltage to the power supply circuit. Note that we do this test without microcontroller because if there is any excessive voltage, this may lead to damaging the controller. We check for the input to the voltage regulator i.e., are we getting an input of 12v and an output of 5v. This 5v output is given to the microcontrollers' 40<sup>th</sup> pin. Hence we check for the voltage level at 40<sup>th</sup> pin. Similarly, we check for the other terminals for the required voltage. In this way we can assure that the voltage at all the terminals is as per the requirement.

## OUTPUT



## LIQUID LEVEL MONITORING SYSTEM



## SOURCE CODE

```
#include <SoftwareSerial.h>
```

```
#include <LiquidCrystal.h>
```

```
#define FALSE false
```

```
int trigger1=A4;
```

```
int trigger2=A2;
```

```
int trigger3=A1;
```

```
int trigger4=12;
```

```
int echo1=A5;
```

```
int echo2=A3;
```

```
int echo3=A0;
```

```
int echo4=13;
```

```
int led=5;
```

```
int buzzer=4;
```

```
int distance=0;
```

```
int g1=0;
```

```
int g2=0;
```

```

int g3=0;

int g4=0; int time=0;

intupdate_st = 0;

intsetPoint1 = 0;

int setPoint2 = 0;

int setPoint3 = 0;

int setPoint4 = 0;


intlast_dist = 0,stability=0;

String str1="GET /Gbin.aspx?id=5463&gl=";

String str2="&g2=";

String str3="&g3=";

String str4="&g4=";

String str5=" HTTP/1.1\r\n";

```

SoftwareSerial esp8266(2,3); // make RX Arduino line is pin 2, make TX Arduino line is pin 3. //

This means that you need to connect the TX line from the esp to the Arduino's pin 2 // and the RX line from the esp to the Arduino's pin 3

```

LiquidCrystal lcd(6, 7, 8, 9, 10, 11);

String response = "";

void setcheckPoint();

```

```

intbinscan(int,int,int);

void setup()

{

Serial.begin(9600);

String response = "";

voidsetcheckPoint();

intbinscan(int,int,int);

void setup()

{

Serial.begin(9600);

esp8266.begin(9600); // your esp's baud rate might be different

lcd.begin(16, 2);


pinmode(trigger1,OUTPUT)

pinmode(trigger2,OUTPUT)

pinmode(trigger3,OUTPUT)

pinmode(trigger4,OUTPUT)

pinmode(echo1,INPUT)

pinmode(echo2,INPUT)

pinmode(echo3,INPUT)

pinmode(echo4,INPUT)

```

```

pinmode(led,OUTPUT);

pinmode(buzzer,OUTPUT);


lcd.clear();

lcd.print(" IOT Based");

lcd.setCursor(0,1);

lcd.print(" Liquid Monitor");


sendData("AT+RST\r\n",2000,DEBUG) //RESET MODULE

// sendData("AT+CWSAP=\"ESP_8266\", \"pwd\",5,3\r\n",10000,DEBUG);

sendData("AT+CWJAP=\"iotgarbage\", \"project1234\"\r\n",10000,DEBUG);

sendData("AT+CIFSR\r\n",5000,DEBUG); // get ipaddr


lcd.clear();

setcheckPoint();

// delay(1000);

lcd.clear();

}


void loop()

{

```



```

sensor(trigger1,echo1);

if(distance < 0 || distance >setPoint1 )

distance=0;

distance = 10-(((float)(distance)/(setPoint1))*10.0);

g1=distance+2;

// if(g1>=10) g1=10;

// if(g1<=0 || distance<=0) g1=0;

```

```

checkstatus(1,g1,0,0);

sensor(trigger2,echo2);

if(distance < 0 || distance > setPoint2 )

distance=0;

distance=10-(((float)(distance)/(setPoint2))*10.0);

g2=distance+2;

// if(g2>=10) g2=10;

// if(g2<=0 || distance<=0) g2=0;

checkstatus(2,g2,10,0);

```

```

sensor(trigger3,echo3);

if(distance < 0 || distance > setPoint3)

distance=0;

distance= 10-(((float)(distance))/(setPoint g3=distance+2;

// if(g3>=10) g3=10;

// if(g3<=0 || distance<=0) g3=0;

checkstatus(3,g3,0,1);


sensor(trigger4,echo4);

if(distance < 0 || distance > setPoint4 )

distance=0;

distance=10-(((float)(distance))/(setPoint4))*10.0);

g4=distance+2;

// if(g4>=10) g4=10;

// if(g4<=0 || distance<=0) g4=0;

checkstatus(4,g4,10,1);


// sendData("AT+CIPSTART=4,\"TCP\", \"www.nevemtech.com\",80\r\n\",5000,DEBUG);

// /* if(response[2] == 'b' && response[3] == 'u' && response[4] = is' && response[5] == 'y')
delay(2000); */

// sendData("AT+CIPSEND=4,100\r\n\",100,DEBUG);

```

```

// // if(response[2] == 'b' && response[3] == 'u' && response[4] == 's' && response[5] == 'y')
delay(5000);

//          sendData("GET          /Gbin.aspx?id=2190&g1=10&g2=0&g3=10&g4=0
HTTP/1.1\r\n",1000,DEBUG);

String str=str1+ g1 +str2+ g2 +str3+ g3 +str4+ g4 +str5;

sendData(str,100,DEBUG);

// if(response[2] == 'b' && response[3] == 'u' && response[4] == 's' && response[5] == 'y')
delay(5000);

sendData("AT+CWMODE=3\r\n",1000,DEBUG);

sendData("AT+CWAP=\"iotgarbage\", \"project1234\"\r\n",10000,DEBUG);

sendData("AT+CIFSR\r\n",5000,DEBUG); // get ip address

// delay(1000);

lcd.clear();

}

/* * Name: sendData

* Description: Function used to send data to E5P8266.

* Params: command - the data/command to send; timeout - the time to wait for a response; debug
- print to serial window?(true = yes, false = no)

* Returns: The response from the esp8266 (if there is a reponse)

*/

```

```
String sendData(string command, constint timeout, boolean debug)
```

```
{
```

```
response = "";
```

```
longint time = millis();
```

```
while( (time+timeout)>millis())
```

```
{
```

```
while(esp8266.available())
```

```
{
```

```
char c = esp8266.read();
```

```
}
```

```
}
```

```
if(debug)
```

```
{
```

```
Serial.print(response);
```

```
}
```

```
return response;
```

```
}
```

```
void sensor(inttriggerpin,intechopin)
```

```
{
```

```
digitalwrite(triggerpin,Low);
```

```

delaymicroseconds(2);

digitalwrite(triggerpin,HIGH);

delaymicroseconds(10);

digitalwrite(triggerpin,Low);

time=pulsein(echopin,HIGH);

}


voidcheckstatus(intbinNo,intl,intcol,int row)

{

if(1>=9)

{

tone(buzzer,50,5000);

digitalwrite(led,HIGH);

}

else

{

digitalwrite(led,Low);

}

lcd.setCursor(col,row);

lcd.print("B");

lcd.print(binNo);

```

```

lcd.print(":");

lcd.print(1);

}

void setcheckPoint()

{

setPoint1 = binscan(1,trigger1,echo1);

setPoint2 = binscan(2,trigger2,echo2);

setPoint3 = binscan(3,trigger3,echo3);

setPoint4 = binscan(4,trigger4,echo4);

}

int binscan(int binNo, int trigpin, int echopin)

{

last_dist = 0, stability=0;

sensor(trigpin, echopin);

last_dist = distance;

while(stability<4)

{

stability = 0;

for(int i=0; i<5; i++)

{

```

```

sensor(trigppin,echopin);

lcd.clear();

lcd.print(distance);

lcd.print("cm");

delay(200);

if(distance == last_dist)

} // last_dist = distance;

}

lcd.clear();

lcd.print("Bin ");

lcd.print(binNo);

lcd.print(" Empty");

}

    lcd.clear();

    lcd.print("Bin ");

    lcd.print(binNo);

    lcd.print(" Empty");

    lcd.setCursor(0,1);

    lcd.print("Bistance: ");

    lcd.print(distance);

```

```

        lcd.print("cm");

        delay(5000);

return distance;

}

intcheckstability(int t, int e, int s)

{

    for(inti=0;i<5;i++)

    {

        sensor(t,e);

        if(distance < 0 || distance > s )

            distance=0;

    }

}

```

## AURDINO

```

#include <iotgecko.h>

#define esp_baudrate 115200 // enter baud rate of your wifi module

// initialize the library with the esp8266(wifi module)

iotgecko gecko = iotgecko(esp_baudrate);

```



```

boolnotConected = true;

bool login = false;

String id = "Ykethansai@gmail.com" ; // iotgecko login id

String pass = "1617"; // iotgecko login password

String ssid = [WIFI_Name]; // SSID/name of your wifi router or wifi hotspot

String pass_key = [WIFI_Password]; //Wifi Password

constintno_of_containers = 4; // total number of containers to monitor

int container[no_of_containers];

int sensor1_value = 0;

int sensor2_value = 0;

int sensor3_value = 0;

int sensor4_value = 0;

//int sensor5_value = 0;

//int sensor6_value = 0;

//int sensor7_value = 0;

//int sensor8_value = 0;

void setup()

{

Serial.begin(115200);

Serial.println("iot liquid level monitoring");

delay(2000);

```

```

Serial.println("connectng to wifi");

Serial.print(ssid);

Serial.print("\t");

Serial.println(pass_key);

while(notConected)

{

if(gecko.GeckoConnect(ssid,pass_key)) //connect to wifi with given SSID and password

{

Serial.println("connceted to wifi...");

notConected = false;

}

else

{

Serial.println("can't connect to wifi");

}

delay(1000);

}

Serial.println("connecting to iotgecko.com");

while(!login)

{

if(gecko.GeckoVerify(id,pass)) //login to iotgecko.com with given ID and password

```

```

    {

Serial.println("connected succesfully");

login = true;

    }

else

    {

Serial.println("fail to connect");

    }

}

void loop()

{

// Write code for reading your sensor values here

// Convert sensor values in integer range min 0 to max 5

//Uncomment the lines depending upon the number of sensors

container[0] = sensor1_value;

container[1] = sensor2_value;

container[2] = sensor3_value;

container[3] = sensor4_value;

// container[4] = sensor5_value;

// container[5] = sensor6_value;

```

```

// container[6] = sensor7_value;

// container[7] = sensor8_value;

Serial.print("container 1 level is ");

Serial.println(container[0]);

Serial.print("container 2 level is ");

Serial.println(container[1]);

Serial.print("container 3 level is ");

Serial.println(container[2]);

Serial.print("container 4 level is ");

Serial.println(container[3]);

intiot_status = gecko.SendGParams(container,no_of_containers); //send data to iotgecko.com

if(iot_status == InvalidUserIdOrPassword)

{

Serial.println("Invalid UserId or Password");

while(1);

}

else if(iot_status == InvalidData)

{

Serial.println("Invalid Data");

while(1);

}

```

```

else if(iot_status == VALID)

{

Serial.println("data send succesfully.....");

}

else

{

Serial.println("connection lost");

while(!gecko.GeckoReconnect()) // reconnect to iotgecko.com

{

Serial.println("connction failed.....reconnecting");

delay(2000);

}

Serial.println("connected succesfully");

}

sensor1_value++;

sensor2_value++;

sensor3_value++;

sensor4_value++;

// sensor5_value++;

// sensor6_value++;

// sensor7_value++;

```

```
// sensor8_value++;

if((sensor1_value>5) || (sensor2_value>5) || (sensor3_value>5) || (sensor4_value>5))

{

    sensor1_value = sensor2_value = sensor3_value = sensor4_value = 0;

}

delay(1000);

}
```

# **BIBLIOGRAPHY**

## **TEXT BOOKS REFERED:**

1. “The ATMEGA Microcontroller and Embedded systems”
2. ATMEGA 328 Data Sheets.

## **WEBSITES**

- [www.atmel.com](http://www.atmel.com)
- [www.beyondlogic.org](http://www.beyondlogic.org)
- [www.wikipedia.org](http://www.wikipedia.org)
- [www.howstuffworks.com](http://www.howstuffworks.com)
- [www.alldatasheets.com](http://www.alldatasheets.com)