HEWLETT-PACKARD EDUCATION SERVICES

Embedded Systems & Robotics

Basic

http://www.hpesindia.com

Contents

Section A

Digital Electronics

Chapter 1 Basics

1.1 What is a Digital system?

In most general terms, this system's behavior is sufficiently explained by using only two of its states can be Voltage(more than x volts or less?),distance covered(more than 2.5 km or less?],true-false or weight of an elephant(will my weighing machine withstand it?))

Note that although in every case, the all the intermediate states **ARE POSSIBLE AND DO EXIST,** our point of interest are such that we don't require their explicit description**.** In electronic systems we mostly deal with Voltage levels as digital entities.

1.2 Assigning States

There is no specific fixed definition of logic levels in electronics. Most commonly used level designation is the one used in CMOS and TTL (transistor transistor logic) families:

Logic high –> designated as '1'

Logic low –> designated as '0'

Where high and low are actually 'higher' and 'lower' with respect to a reference voltage level (ideally taken as 2.5V)

GOOGLY: Why assign '0'and '1' and not 'a 'and 'b', 'x' and 'y,'cat 'and 'dog'?
ANS: Computational ease!

1.3 Number Systems in digital electronics

1. **Binary:** Only '0' and '1'.
2. **Hexadecimal:** 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Dec	Hex	Bin
00	0	0000
01	1	0001
02	2	0010
03	3	0011
04	4	0100
05	5	0101
06	6	0110
07	7	0111
08	8	1000
09	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

1.4 Types of Digital Circuits

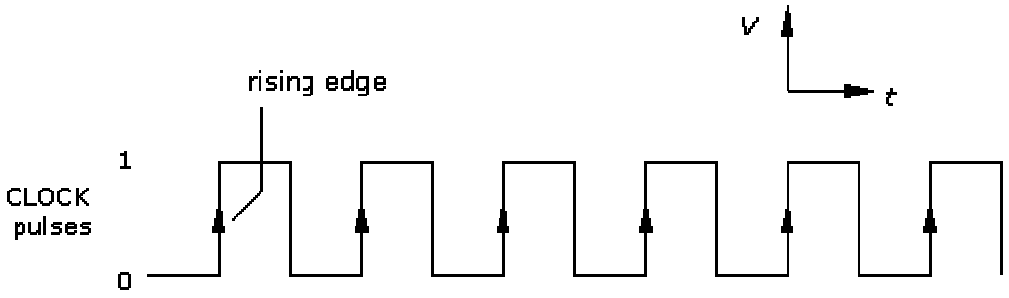
Combinatorial Circuits: Inthese circuits, the past states are immaterial and the output depends only upon the present state. Example logic gates

Sequential circuits: In these circuits, the next state is completely determined by the past states. Hence these follow a predictable structure and essentially require a timing device. Ex. counters, flip flops.

1.5 Clock: Building block of a sequential circuit

A clock is simply alternate high and low states of voltage with time i.e. essentially a square wave. Important terms related to clock are its duty cycle and its frequency:

Duty cycle: It is the ratio of $T_h$and T_h+T_l $D = \frac{Th}{Tl+Th}$



1.6 Logic Gates: Building block of a combinatorial circuitry

These are essentially combinatorial circuits used to implement logical Boolean operations like AND, NAND, OR, XOR and NOT. NOT and NAND are called universal gates as any other gate can be formed using either of them!

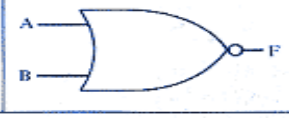
Name	Graphic Symbol	Algebraic Function	Truth Table
AND		F = A · B or F = AB	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		F = A + B	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or F = A′	A | F 0 | 1 1 | 0
NAND		$F = (\overline{AB})$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (\overline{A + B})$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Figure 1: Table of Logic Gates

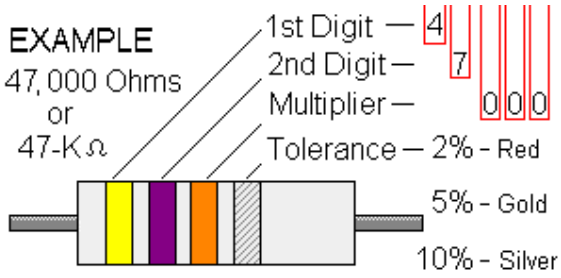
1.7 Practical CircuitingElements

1.7.1 Resistor:

A color scheme is followed to give the specifications of a resistor. The table for color code is shown below:

BLACK		0
BROWN		1
RED		2
ORANGE		3
YELLOW		4
GREEN		5
BLUE		6
VIOLET		7
GRAY		8
WHITE		9

Figure 2: Table of Resistance



The 1st two bands specify the 2 digits of the resistor value whereas the 3rd band specifies the multiplier in terms of the power to which 10 is raised and multiplied to the 2 digits.

The tolerance tells the possible % variation of the resistor value about the value indicated by bands.

1.7.2 Capacitor:

The 2 types of capacitors we frequently use in circuits are ceramic and electrolytic capacitors. While ceramic capacitors do not have a fixed polarity; electrolytic capacitors should be connected in their specified polarities only else they might blow off! This polarity is usually provided on the side of the capacitors 'corresponding leg.

Figure 4: Electrolytic cap with –ve polarity leg seen

Figure 3: Ceramic cap with value 15x104 pF

1.7.3 Breadboard:

This is the base used for setting up the circuit. This has embedded metal strips in it that form a grid of connections inside its body. This allows us to take multiple connections from a single point without any need of soldering/disordering as in PCBs. It is always a good habit to test the circuit on breadboard before making it on a PCB.

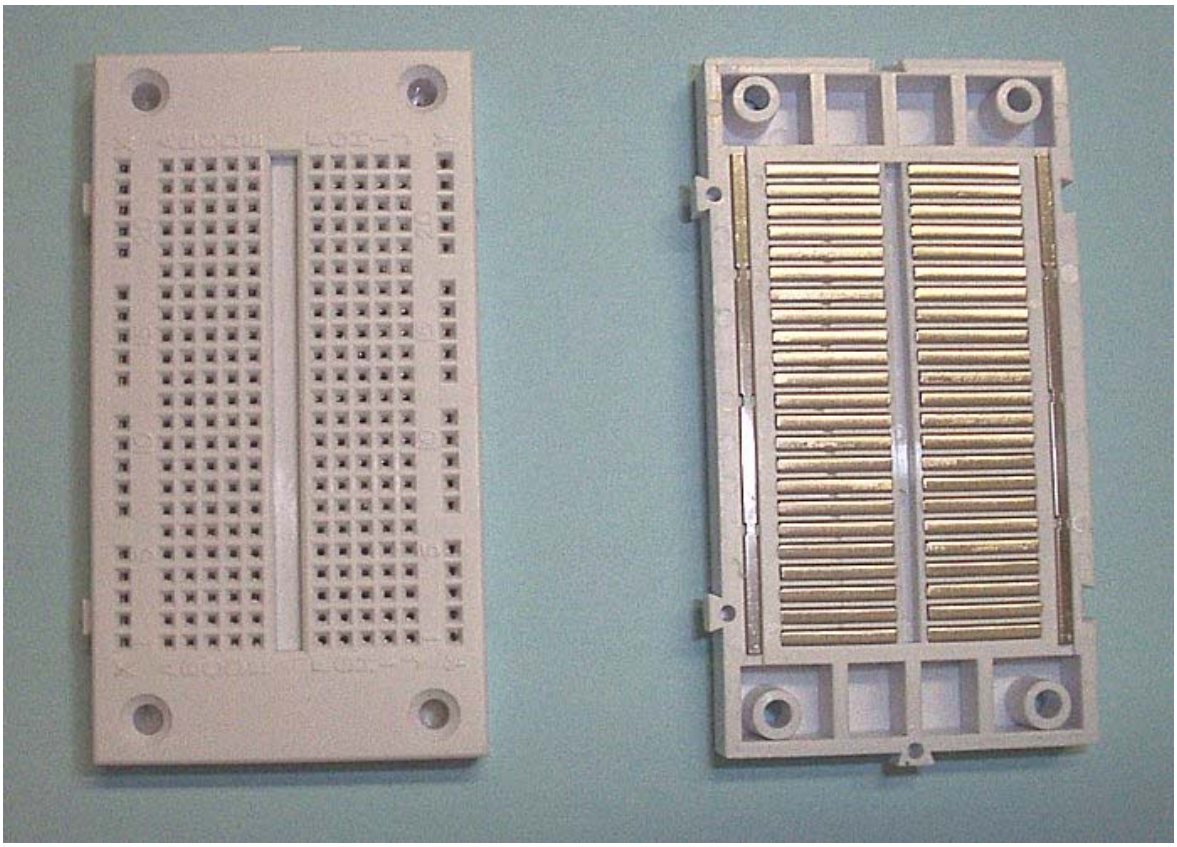


Figure 5: Top view showing the connecting holes. Bottom view shows the contact metal strips

1.7.4 Integrated Circuits (IC)

ICs or Integrated Circuits are packaged circuits designed for some fixed purpose. An IC has its fixed IC name/number that can be used to get catalog of its functions and pin configuration.ICs come in various sizes and packages depending upon the purpose.

NOTE: Numbering scheme of IC pins will be explained in the lab session. Different ICs may have different number of pins.

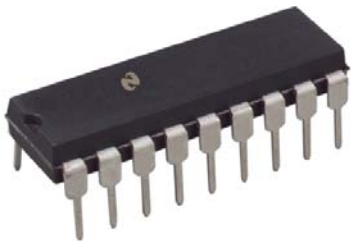


Figure 6: IC

1.7.5 LED

LED (Light Emitting Diode) is frequently used to display the outputs at various stages of the circuit. It is essentially a Diode with the energy released in the form of photons due to electron transitions falling in the visible region. Hence normal diode properties apply to it.

It glows only in fwd bias mode i.e. with p junction connected to +ve voltage and n junction to negative.

Diodes are essentially low power devices. The current through the LED should be less than 20mA.Hence always put a 220 ohm resistor in series with the LED.

Never forget that LEDs consume a significant amount of power of the outputs of the ICs (CMOS based).Hence it is advisable to only use them for checking the voltage level (high or low) and then remove them.

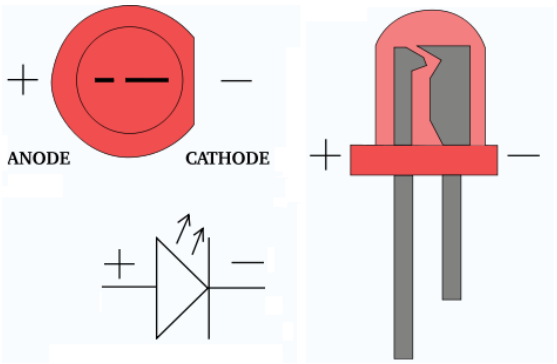


Figure 7: LEDs

Chapter 2 Some Integrated Circuits and Implementation

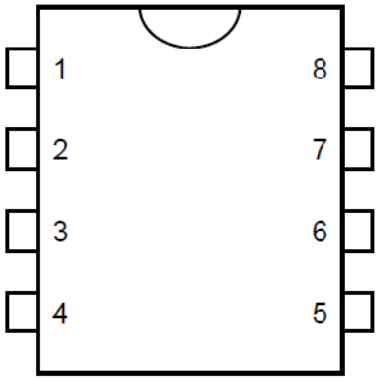
2.1 555

555 is an IC used to generate a clock .The two attributes of a clock are

- Frequency
- Duty cycle.

Both of these can be changed using this IC, however the duty cycle is always <50%.

There are two modes in which 555 can run.



1 - GND
2 - Trigger
3 - Output
4 - Reset
5 - Control voltage
6 - Threshold
7 - Discharge
8 - Vcc

Figure 8: Pin Configuration of 555

2.1.1 Monostable mode

As the name suggests; in this mode the output is stable in only one (mono) state i.e. 'off' state. Thus it can stay only for a finite time, if **triggered,** to the other state i.e. 'on' state. This time can be set choosing appropriate values of resistances in the formula:

T=1.1 xR1 xC1

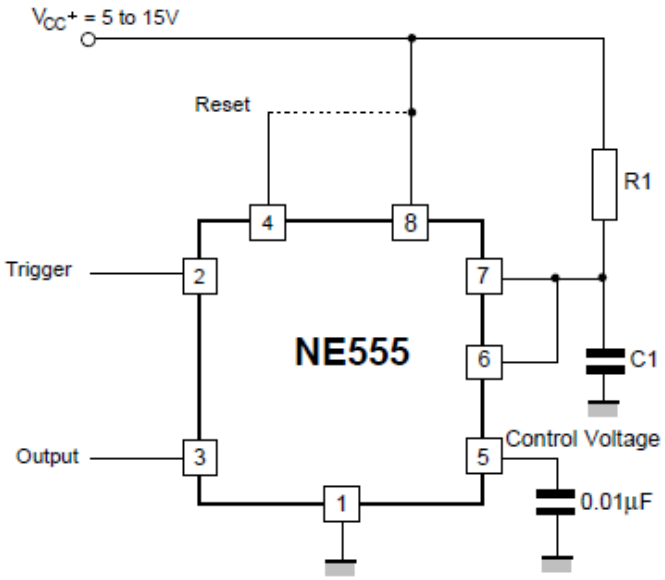


Figure 9: 555 in monostable mode

2.1.2 Astable mode

In this mode; the output is stable neither in 'high' state nor in 'low ' state. Hence it oscillates from one state to another giving us a square wave or clock. We can set the clock frequency and Duty cycle D by the formulae:

$$F=\frac{1.44}{(R1+2R2)C1} \qquad D=\frac{(R1+R2)}{(R1+2R2)}$$

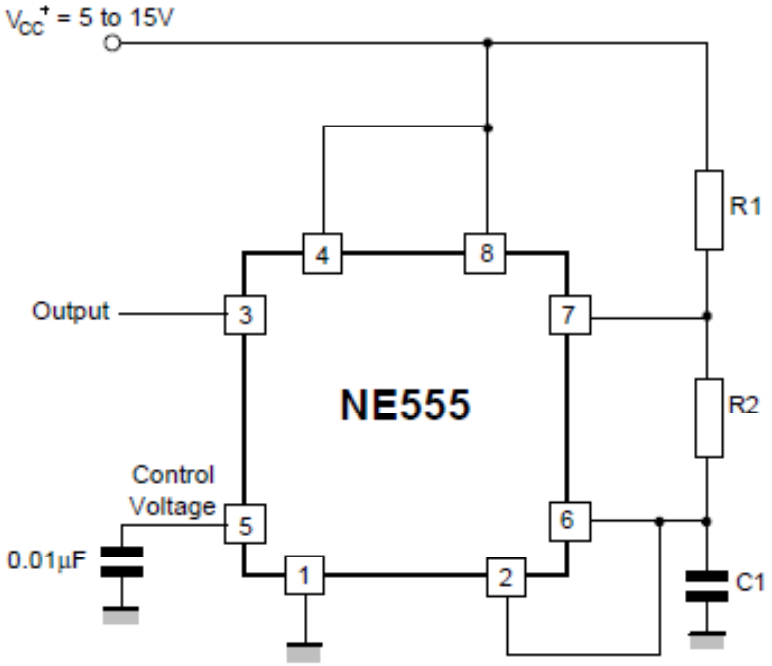


Figure 10: 555 in astable mode

NOTE: Capacitor C2 is just to filter the noise and its value can be suitably chosen to be 0.01µF. It can also be neglected.

2.2 4029 counter

With the clock made, we are ready to count the number of pulses passed into the circuit. Note that any kind of counting requires a **memory** (you got to know that you have just counted '3' to go to '4'!)**.** Hence 4029 can also be used as a memory element that remembers its immediate previous state.

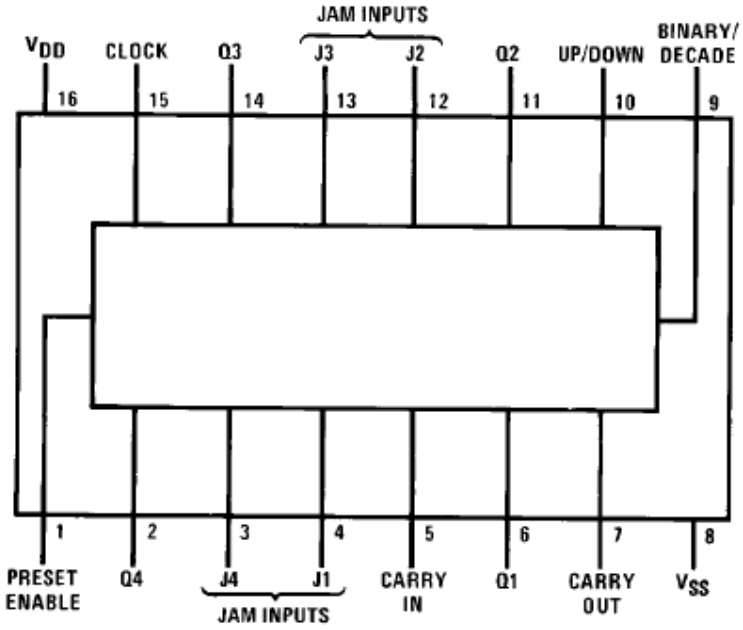


Figure 11: 4029 pin configuration

2.2.1 Pin Description

PIN No.	Name	Pin function	Connection reqd.
1	Parallel load	If given high; loads the value of Parallel bit into the o/p bits	Gnd
2	Output Bit 3	Most significant bit of o/p	To pin 6 of 7447
3	Parallel input bit 3	Most significant bit of parallel i/p	Input
4	Parallel input bit 0	Least significant bit of parallel i/p	Input
5	Clock enable bar	Low on this pin enables counting as per the clock received	Gnd
6	Output Bit 0	Least significant bit of parallel o/p	To pin 7 of 7447
7	TC bar	Output bit that gives a low when the count is complete. Can be used to signal the end of counting.	None if you don't want to use it
8	Gnd	Needed for powering	Gnd
9	Binary/Hex bar	To choose b/w binary and hexadecimal modes	low for count 0-15 high for count 0-9
10	Up/Down bar count	To choose b/w up counting and down counting modes	Low for down count High for up count
11	Output bit 1	2^{nd} bit of o/p	To pin 1 of 7447
12	Parallel input bit 1	2^{nd} bit of i/p	Input
13	Parallel input bit 2	3^{rd} bit of i/p	Input
14	Output bit 2	3^{rd} bit of o/p	To pin 2 of 7447
15	Clock pulse	Clock pulse is given here	Clock from 555
16	V_{dd}	Needed for powering	+5 V

2.3 7447: BCD to 7 segment display decoder

For displaying the number in the counter output on a seven segment display (i.e. 7 LEDs making up a figure of '8' as in a general calculator. See fig.)we need to decode the 4 bits and match them to the 7 pins for lighting the LEDs corresponding to the number. This work is done by 7447.

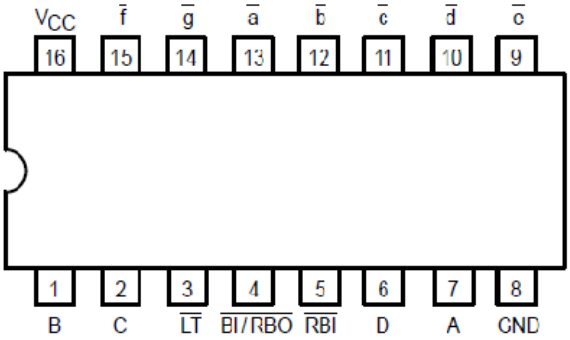
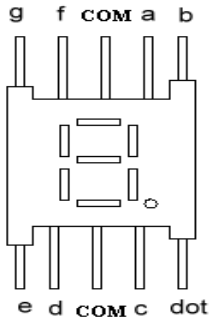
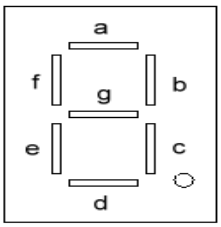


Figure 12: 7447 pin configuration





Seven-Segment Display

2.3.1 Pin Description

PIN no.	Name	Function	Connection reqd.
1	i/p B	2nd bit(O1) of 4029's o/p	To O1 of 4029
2	i/p C	3rd bit(O2) of 4029's o/p	To O2 of 4029
3	Lamp Test bar	Used to check that all LEDs of 7 seg are working.	High for normal fn Low to glow all LEDs
4	BI /RBI	Kept high to allow normal function	Kept high
5	RBI	Blanks '0' from being displayed	Kept high
6	i/p D	Most significant bit(O3) of 4029's o/p	To O2 of 4029
7	i/p A	3rd bit(O2) of 4029's o/p	To O2 of 4029
8	Gnd	For power	Connected to gnd
9-15	a-g as per the fig	The o/p pins to 7segment display	To 7 seg display
16	Vcc	For power	Connected to +5 V

NOTE:

- The COM pins are to be connected to Vcc via 220 ohm resistor. Why resistor is required??
- The dot pin is just for display of decimal point and essentially only makes the upper and lower sides distinguishable from each other for a single display. without the asymmetry produced by dot how will we be able to see which side is upper and which is lower?

2.4 LDR(Light Dependent Resistor)

Light Dependent Resistor (LDR) or photo resistor is a device that acts like a resistance and its resistance varies with the intensity of light incident on it. In this device, if photons of sufficient energy fall on it, the resistance drops drastically as the electrons in the semiconductor are able to jump from the valence band to the conduction band and are available for conduction. The LDRs used are mostly responsive to visible light. The resistance might drop from as high as 1MΩ in the dark to 1 kΩ in bright light.

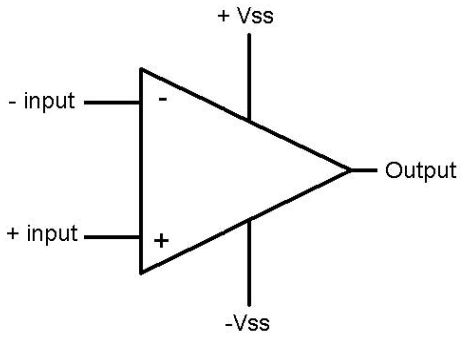
Figure 13: LDRs. The coiled portion is responsive to light

2.5 Operational Amplifier (Opamp)

Opamp is a very important device used in everyday electronics .It is essentially a differential amplifier with a very high gain of the order of 10^5!By differential amplifier I mean that it amplifies the difference of 2 signals and gives the output.

Opamp equation:

$$V_{out} = A(V_+ - V_-)$$ where A is the gain of the order 10^5.



Ironically, this high gain in open loop makes it impossible to use it as a general purpose differential amplifier directly.

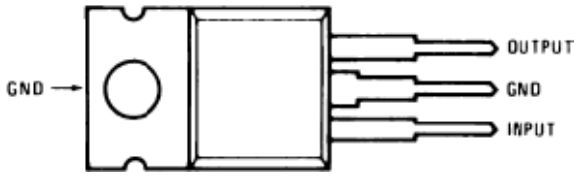
GOOGLY: If $(V_+ - V_-) = 0.005V$; Vss = 12V what will be the output??

2.5.1 Opamp as a comparator

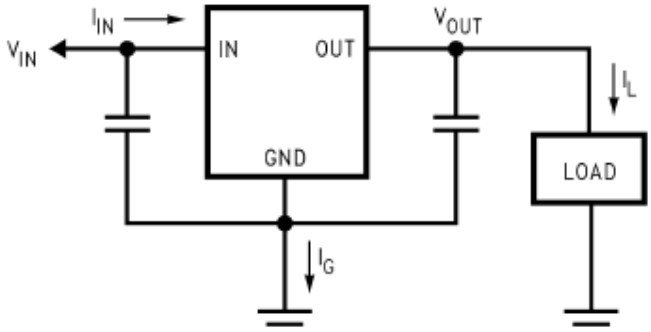
Simplest use of Opamp is as a comparator. It can be used to convert an analog signal to a digital signal defined by a fixed threshold. Set V- as the threshold voltage say 2.5 V and apply the analog signal to be digitized at V+ .What will be the output? Well if you have worked out the googly; this should be a piece of cake!

2.6 7805 Voltage Regulator

7805 voltage regulator is used to get +5 V output out of a higher voltage supply (7.5V-20V).We use adapter's supply to generate +5V here. Connect the gnd and +12V of adapter to the pins as shown and get +5V directly as an output out of the 3rd pin. Current up to 0.5 A can be obtained from this regulator without any significant fall in voltage level.



NOTE: Use 2 capacitors of value say 0.1µF to filter the noise in the input and output of regulator's supply as shown.

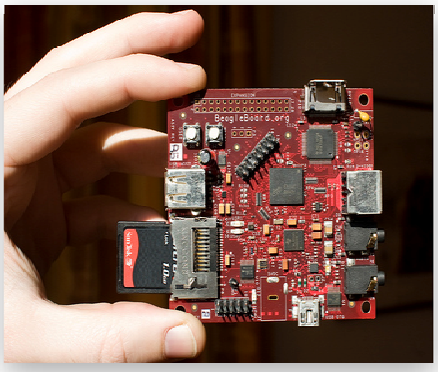


*Section **B***

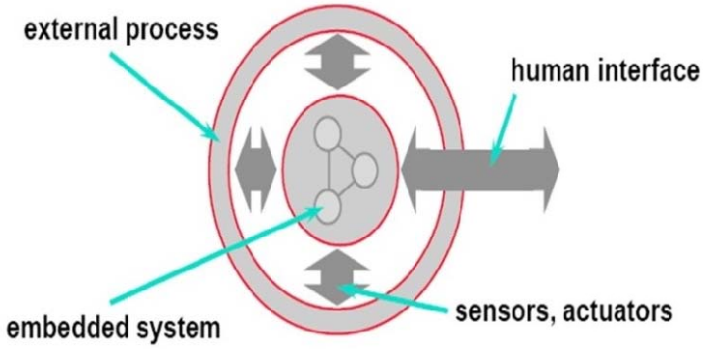
Embedded Systems

Chapter 3 Introduction to Embedded Systems

Embedded system is a scaled down computer system which is designed to perform a specific task/operation. Unlike a general purpose computer system which is used for a variety of tasks, like playing music, games, surfing internet etc. The term embedded tells that whole system is embedded into an appliance. A single chip contains both hardware and software (technically, firmware). It is designed to perform operations which minimize (or even completely avoid) need of human control.



Basic Flow diagram for embedded computer systems can be shown as:



- Various ways we can Define……
  - An **Embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints.
    or
  - An **Embedded system** is a Software program on a H/W chip designed for a specific purpose and can also contain some mechanical moving parts.
    - As features get on added daily definition itself is evolving.
      or
- An **Embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing.

  or
- An **Embedded system** is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device.

Some important things to note about embedded systems:

1. Once an embedded hardware is programmed for a certain task, it is used forever for the same task. Changing the firmware afterwards is not possible.
2. Such systems are limited in computational resources like memory, CPU processing speed, I/O facilities but are still capable of performing the task given to them very efficiently.
3. Embedded systems can also be having a reduced functionality version of operating system called RTOS (Real Time Operating System) for highly specialized applications.
4. Is a system built to perform its duty, completely or partially independent of human intervention?
5. Is specially designed to perform a few tasks in the most efficient way.
6. Interacts with physical elements in our environment, viz. controlling and driving a motor, sensing temperature, etc.

3.1 Applications

An embedded system can be defined as a control system or computer system designed to perform a specific task. Examples:

1. Pen drives (for controlling the communication between P.C. and Flash Chip and also the small LED!)
2. Hard disks(again for the same purpose)
3. Mouse(Reads and Interprets the Sensors and send final result to P.C.),Keyboards
4. Printers: Ever opened a printer for installing ink cartridge? Then you must have seen the printed head. There are motors to control the print head and the paper movement. Your P.C. is not directly connected to them but there is built in MCU of printer to control all these. Your P.C. just sends the data (pixels) through the communication line (USB or parallel).But the MCU used here is fairly fast and has lots of RAM.
5. Automobiles
6. Calculators, Electronic wending machines, Electronic weighing scales, Phones(digital with LCD and phonebook)
7. Cell phones
8. Security System
9. Alarm system
10. Automobile system
11. Digital Camera
12. Environment monitoring systems (using sensors and actuators)

Embedded systems are often required to perform real time operations. By Real time operations, we mean that, operations where delay of even a few milliseconds could be dangerous. Some real time systems may be:

1. Sensor system in Nuclear Plants
2. Flight control systems
3. Automobile Braking system and engine controlling systems

These are situations where we need very accurate timing and control. Failure in such situations may cause great loss.

3.2 Embedded System Types

- **Non Real-time Embedded** system is one in which there is no deadline, even if fast response or high performance is desired or preferred.

- **Real time Embedded Systems:**Where deadline is to be met.
  Soft real-time embedded systems.
  Hard real-time embedded systems.

- Following are some applications of Embedded Sys. but not realtime.
  - Security Systems.
  - Mobile and PDA.
  - Alarm System.
  - Auto mobile system.
  - Digital Camera.

- Examples of Real-time Embedded systems.
  - Sensor system in Nuclear plants.
  - Missile defence system.
  - Flight control system.
  - Anti-collision system in Auto mobiles.

Chapter 4 Introduction to Microcontrollers

4.1 What is Microcontroller?

- **What is a Microcontroller?**

A Microcontroller is a programmable digital processor with necessary peripherals. Both microcontrollers and microprocessors are complex sequential digital circuits meant to carry out job according to the program / instructions. Sometimes analog input/output interface makes a part of microcontroller circuit of mixed mode(both analog and digital nature). A microcontroller can be compared to a Swiss knife with multiple functions incorporated in the same IC.

- **Fig. 4.1 A Microcontroller compared with a Swiss knife**

- **Microcontrollers Vs Microprocessors**

1. A microprocessor requires an external memory for program/data storage. Instruction execution requires movement of data from the external memory to the microprocessor or vice versa. Usually, microprocessors have good computing power and they have higher clock speed to facilitate faster computation.
2. A microcontroller has required on-chip memory with associated peripherals. A microcontroller can be thought of a microprocessor with inbuilt peripherals.
3. A microcontroller does not require much additional interfacing ICs for operation and it functions as a standalone system. The operation of a microcontroller is multipurpose, just like a Swiss knife.
4. Microcontrollers are also called embedded controllers. A microcontroller clock speed is limited only to a few tens of MHz Microcontrollers are numerous and many of them are application specific.

- **Development/Classification of microcontrollers (Invisible)**

Microcontrollers have gone through a silent evolution (invisible). The evolution can be rightly termed as silent as the impact or application of a microcontroller is not well known to a common user, although microcontroller technology has undergone significant change since early 1970's. Development of some popular microcontrollers is given as follows.

Intel 4004	4 bit (2300 PMOS trans, 108 kHz)	1971
Intel 8048	8 bit	1976
Intel 8031	8 bit (ROM-less)	.
Intel 8051	8 bit (Mask ROM)	1980
Microchip PIC16C64	8 bit	1985
Motorola 68HC11	8 bit (on chip ADC)	.
Intel 80C196	16 bit	1982
Atmel AT89C51	8 bit (Flash memory)	.
Microchip PIC 16F877	8 bit (Flash memory + ADC)	.

- **Development of microprocessors (Visible)**

  Microprocessors have undergone significant evolution over the past four decades. This development is clearly perceptible to a common user, especially, in terms of phenomenal growth in capabilities of personal computers. Development of some of the microprocessors can be given as follows.

Intel 4004	4 bit (2300 PMOS transistors)	1971
Intel 8080 8085	8 bit (NMOS) 8 bit	1974
Intel 8088 8086	16 bit 16 bit	1978
Intel 80186 80286	16 bit 16 bit	1982
Intel 80386	32 bit (275000 transistors)	1985
Intel 80486 SX DX	32 bit 32 bit (built in floating point unit)	1989
Intel 80586 I MMX Celeron II III IV	64 bit	1993 1997 1999 2000
Z-80 (Zilog)	8 bit	1976
Motorola Power PC 601 602 603	32-bit	1993 1995

4.2 Basic Architectures of Microcontrollers

We use more number of microcontrollers compared to microprocessors. Microprocessors are primarily used for computational purpose, whereas microcontrollers find wide application in devices needing real time processing / control.

Application of microcontrollers is numerous. Starting from domestic applications such as in washing machines, TVs, air conditioners, microcontrollers are used in automobiles, process control industries, cell phones, electrical drives, and rob+otics and in space applications.

- **Microcontroller Chips**

Broad Classification of different microcontroller chips could be as follows:

- Embedded (Self -Contained) 8 - bit Microcontroller
- 16 to 32 Microcontrollers
- Digital Signal Processors

- **Features of Modern Microcontrollers**
  - Built-in Monitor Program
  - Built-in Program Memory
  - Interrupts
  - Analog I/O
  - Serial I/O
  - Facility to Interface External Memory
  - Timers

- **Internal Structure of a Microcontroller**

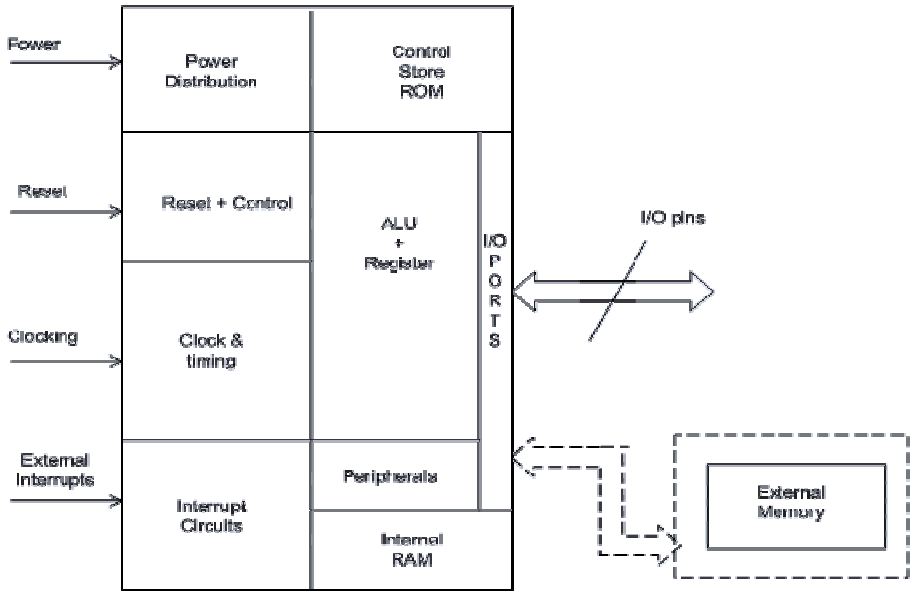


Fig. 4.2 Internal Structure of a Microcontroller

- **Harvard vs. Princeton Architecture**

At times, a microcontroller can have external memory also (if there is no internal memory or extra memory interface is required). Early microcontrollers were manufactured using bipolar or NMOS technologies. Most modern microcontrollers are manufactured with CMOS technology, which leads to reduction in size and power loss. Current drawn by the IC is also reduced

considerably from 10mA to a few micro Amperes in sleep mode(for a microcontroller running typically at a clock speed of 20MHz).
Many years ago, in the late 1940's, the US Government asked Harvard and Princeton university's to come up with a computer architecture to be used in computing distances of Naval artillery shell for defense applications. Princeton suggested computer architecture with a single memory interface. It is also known as Von Neumann architecture after the name of the chief scientist of the project in Princeton University John Von Neumann (1903 - 1957 Born in Budapest, Hungary).

Harvard suggested a computer with two different memory interfaces, one for the data / variables and the other for program / instructions. Although Princeton architecture was accepted for simplicity and ease of implementation, Harvard architecture became popular later, due to the parallelism of instruction execution.

- **Princeton Architecture (Single memory interface)**

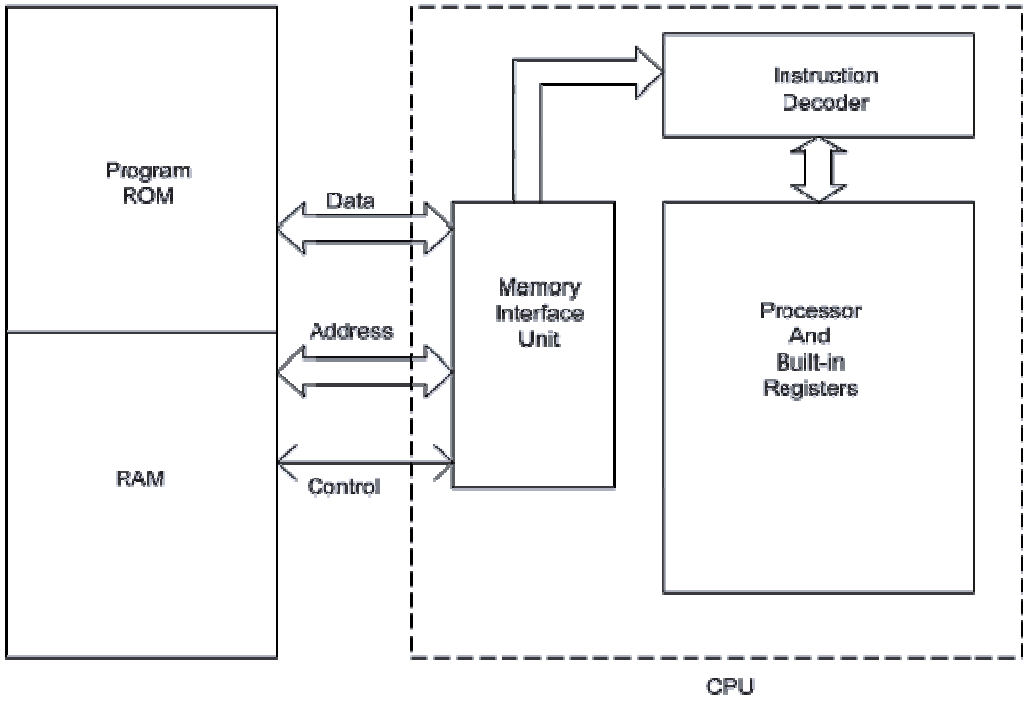


Fig. 4.3 Princeton Architecture

Example: An instruction "Read a data byte from memory and store it in the accumulator" is executed as follows: -

Cycle 1 - Read Instruction

Cycle 2 - Read Data out of RAM and put into Accumulator

- **Harvard Architecture (Separate Program and Data Memory interfaces)**

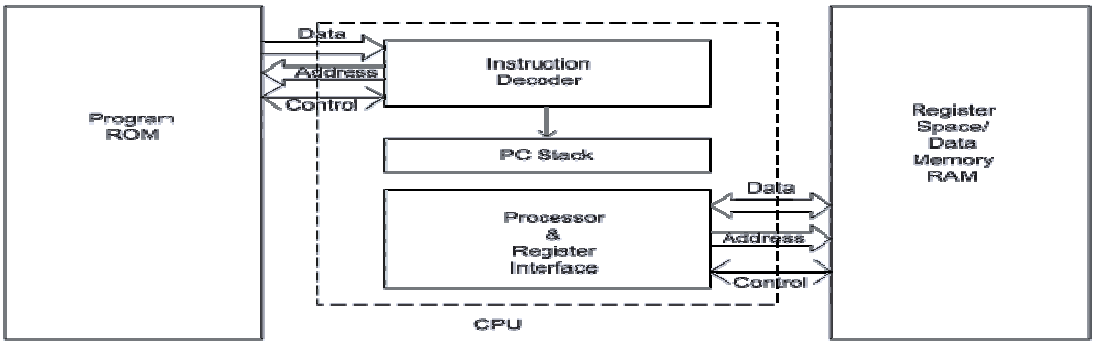


Fig. 4.4 Harvard Arcitecture

The same instruction (as shown under Princeton Architecture) would be executed as follows:

Cycle 1

Complete previous instruction
- Read the "Move Data to Accumulator" instruction

Cycle 2
Execute "Move Data to Accumulator" instruction
- Read next instruction

Hence each instruction is effectively executed in one instruction cycle, except for the ones that modify the content of the program counter. For example, the "jump" (or call) instructions takes 2 cycles. Thus, due to parallelism, Harvard architecture executes more instructions in a given time compared to Princeton Architecture.

4.3 Digital Integrated Circuits (ICs)

You must be knowing about Digital Integrated Circuits (ICs)right ? For example:

- 7404: Hex Inverter
- 7408: Quad 2-input AND gate
- 7410: Triple 3-input NAND Gate
- 7432: Quad 2-input OR Gate
- 7457: 60:1 Frequency divider

There are AND, XOR, NAND, NOR, OR logic gate ICs, Counters, Timers, Seven Segment Display Drivers and much more. Just check out 7400 Series and 4000 Series of Integrated Circuits.

Now let's take Quad 2 input AND gate IC. It has 4 AND gates, each having 2 pins for input and 1 pin for output. The truth table or the function table of each gate is fixed. This is as follows,

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

Similarly all the Integrated circuits have their function tables and input and output pins fixed. You cannot change the function and no input pin act as output and vice versa. So whenever you want to design some circuit you first have to get the output as a function of inputs and then design it using gates or whatever the requirement is.

So once a circuit is built we cannot change its function! Even if you want to make some changes again you have to consider all the gates and components involved. Now if you are designing any circuit which involves change of the function table every now and then you are in trouble ! For example if I want to design an Autonomous Robot which should perform various tasks and I don't just want to fix the task. Suppose I make it to move in a path then I want to change the path ! How to do that?

Here comes the use of Microcontrollers! Now if I give you an Integrated Circuit with 20 pins and tell you that you can make any pin as output or input also you can change the function table by programming the IC using your computer! Then your reactions will be wow! That's nice :-) That's what the most basic function of a microcontroller is. It has set of pins called as PORT and you can make any pin either as output or input. After configuring pins you can program it to perform according to any function table you want. You can change the configuration or the function table as many times you wants.

There are many Semiconductor Companies which manufactures microcontrollers. Some of them are:

- Intel
- Atmel
- Microchip
- Motorola

We will discuss about Atmel Microcontrollers commonly known as AVR in this section.

Question: How a microcontroller works?

Answer: Well I cannot go into lot of details about the working because it is a vast topic in itself. I can just give an overview.

Microcontroller consists of an Microprocessor (CPU that is Central processing Unit) which is interfaced to RAM (Random Access Memory) and Flash Memory (one your pen drive has !). You feed your program in the Flash Memory on the microcontroller. Now when you turn on the microcontroller, CPU accesses the instruction from RAM which access your code from Flash. It sets the configuration of pins and start performing according to your program.

Question: How to make the code?

Answer: You basically write the program on your computer in any of the high level languages like C, C++, and JAVA etc. Then you compile the code to generate the machine file. Now you will ask what this machine file is. All the machines understand only one language, 0 & 1 that is on and off. Now this 0 & 1 both corresponds to 2 different voltage levels for example 0 volt for 0 logic and +5 volt for 1 logic. Actually the code has to be written in this 0, 1 language and then saved in the memory of the microcontroller. But this will be very difficult for us ! So we write the code in the language we understand (C) and then compile and make the machine file (.hex). After we make this machine file we feed this to the memory of the microcontroller.

Question: How to feed the code in the flash of Microcontroller?

Answer: Assuming you have the machine file (.hex) ready and now you want to feed that to the flash of the microcontroller. Basically you want to make communication between your computer and microcontroller. Now computer has many communication ports such as Serial Port, Parallel Port and USB (Universal Serial Bus).

Lets take Serial Port, it has its own definition that is voltage level to define 0 & 1 (yeah all the data communication is a just collection of 0 &1) Serial Port's protocol is called as UART (Universal Asynchronous Receiver & Transmitter) Its voltage levels are :-12 volt for 0 logic and +12 volt for 1 logic.

Now the voltage levels of our microcontroller are based on CMOS (Complementary Metal Oxide Semiconductor) technology which has 0 volt for 0 logic and +5 volt for 1 logic.

Two different machines with 2 different ways to define 0 & 1 and we want to exchange information between them. Consider microcontroller as French and Computer's Serial Port as an Indian person (obviously no common language in between!) If they want to exchange information they basically need a mediator who knows both the language. He will listen one person and then translate to other person. Similarly we need a circuit which converts CMOS (microcontroller) to UART (serial port) and vice versa. This circuit is called as programmer. Using this circuit we can connect computer to the microcontroller and feed the machine file to the flash.

- The **AVR** (Advanced Virtual Risc) is a Modified Harvard architecture 8-bit RISC single chip microcontroller (µC) which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage.

4.4 Processor Type & Memory Structures

- **Micro-coded and hard-coded processors:**

The implementation of computer architecture can be broadly achieved in two ways. A computer is a complex sequential digital circuit with both combinational and sequential circuit components. In a micro-coded processor, each instruction is realized by a number of steps that are implemented using small subroutines. These subroutines are called micro-codes stored within the instruction decode unit. Hence, a micro-coded processor can be called a processor within a processor.

Micro-coded processor:

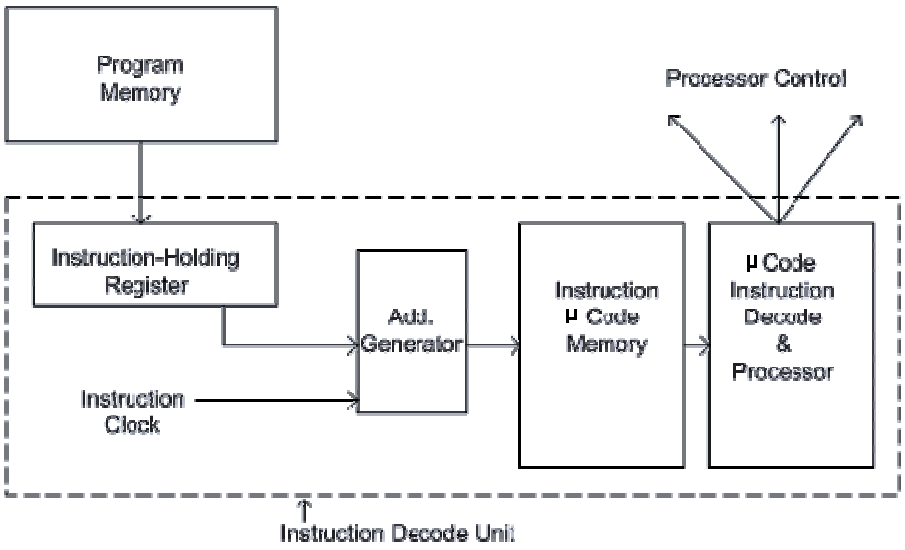


Fig.4.5 Architecture of a Micro-Coded Processor

Let us take an example. The instruction "Move Acc, Reg" can be executed in the following steps.

1. Output address to the data memory
2. Configure the internal bus for data memory value to be stored in accumulator.
3. Enable bus read.
4. Store the data into the accumulator.
5. Compare data read with zero or any other important condition and set bits in the STATUS register.
6. Disable data bus.

Each step of the instruction is realized by a subroutine (micro-code). A set of bits in the instruction points to the memory where the micro-code for the instruction is located.

Advantages: -

1. Ease of fabrication.
2. Easy to debug.

Disadvantage: -

1. Program execution takes longer time.

Hard coded processor:

Each instruction is realized by combinational and/or sequential digital circuits. The design is complex, hard to debug. However, the program execution is faster.

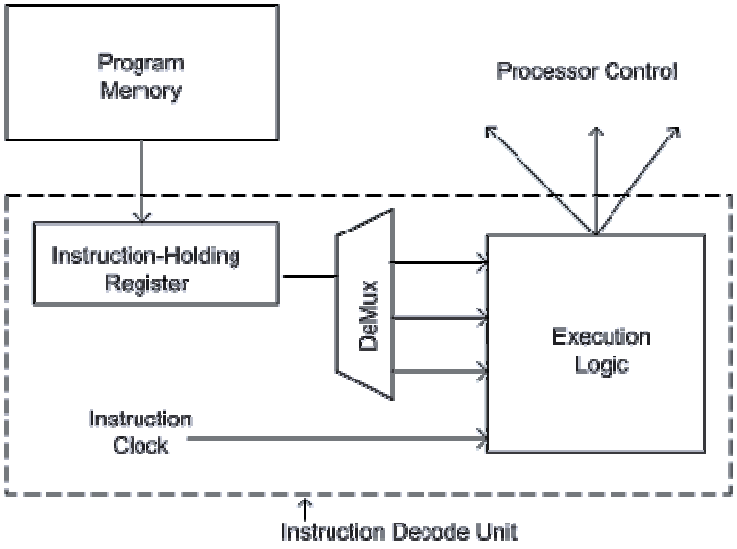


Fig 4.6 Architecture of a Hard - Coded Processor

- **Memory Type**

In a microcontroller, two types of memory are found. They are, program memory and data memory respectively. Program memory is also known as 'control store' and 'firm ware'. It is non-volatile i.e, the memory content is not lost when the power goes off. Non-volatile memory is also called Read Only Memory(ROM). There are various types of ROM.

1. **Mask ROM:** Some microcontrollers with ROM are programmed while they are still in the factory. This ROM is called Mask ROM. Since the microcontrollers with Mask ROM are used for specific application, there is no need to reprogram them. Some times, this type of manufacturing reduces the cost for bulk production.

2. **Reprogrammable program memory (or) Erasable PROM (EPROM):**Microcontrollers with EPROM were introduced in late 1970's. These devices are electrically programmable but are erased with UV radiation. The construction of a EPROM memory cell is somewhat like a MOSFET but with a control and float semiconductor as shown in the figure.

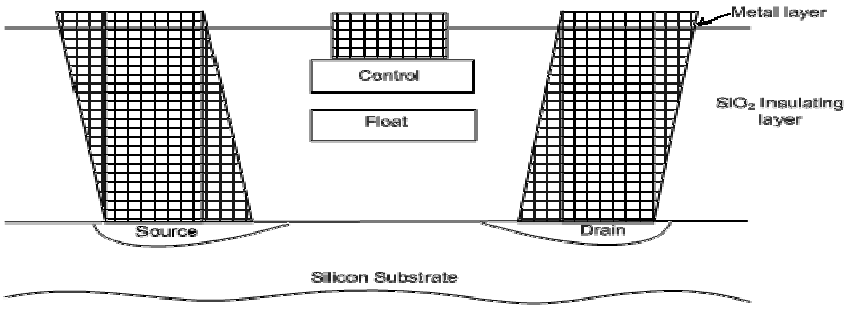


Fig 4.7 Structure of an EPROM

In the unprogrammed state, the 'float' does not have any charge and the MOSFET is in the OFF state. To program the cell, the 'control' above the 'float' is raised to a high enough potential such that a charge leaks to the float through SiO_2 insulating layer. Hence a channel is formed between 'Source' and 'Drain' in the silicon substrate and the MOSFET becomes 'ON'. The charge in the 'float' remains for a long time (typically over 30 years). The charge can be removed by exposing the float to UV radiation. For UV erasable version, the packaging is done in a ceramic enclosure with a glass window.

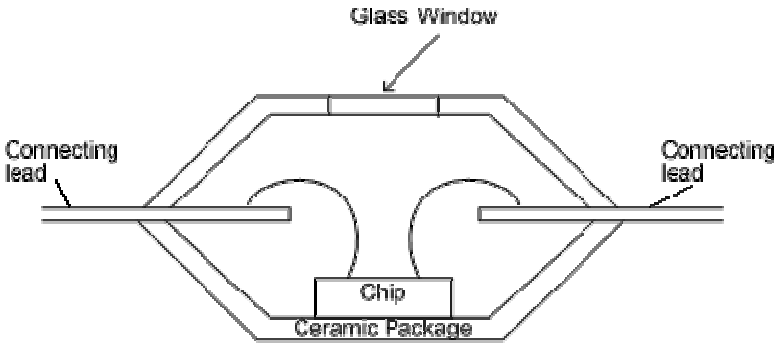


Fig 4.8 UV erasable version of an EPROM

Usually, these versions of micro controllers are expensive.

3. **OTP EPROM**: One time programmable (OTP) EPROM based microcontrollers do not have any glass window for UV erasing. These can be programmed only once. This type of packaging results in microcontroller that have the cost 10% of the microcontrollers with UV erase facility(i.e., 1/10th cost).

4. **EEPROM**: (Electrically Erasable Programmable ROM): This is similar to EPROM but the float charge can be removed electrically.

5. **FLASH (EEPROM Memory)**: FLASH memory was introduced by INTEL in late 1980's. This memory is similar to EEPROM but the cells in a FLASH memory are bussed so that they can be erased in a few clock cycles. Hence the reprogramming is faster.

4.5 Organization of Data Memory

- **DATA Memory**

Data memory can be classified into the following categories

- Bits
- Registers
- Variable RAM
- Program counter stack

Microcontroller can have ability to perform manipulation of individual bits in certain registers(bit manipulation). This is a unique feature of a microcontroller, not available in a microprocessor.

Eight bits make a byte. Memory bytes are known as file registers.

Registers are some special RAM locations that can be accessed by the processor very easily.

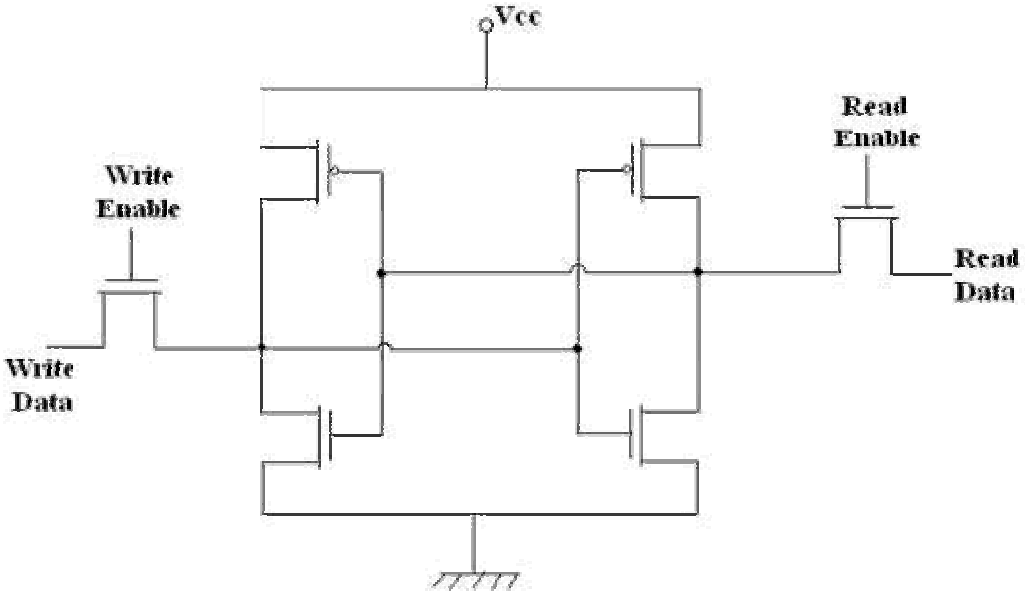


Fig 4.9 Static RAM (SRAM) memory cell

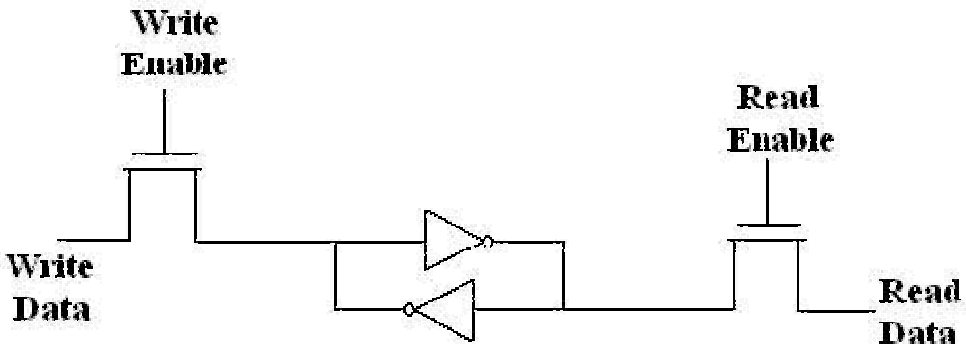


Fig 4.10 SRAM memory cell equivalent

The figure of a single SRAM cell is shown above. This consists of two CMOS inverters connected back to front, so as to form a latch.

Processor stacks store/save the data in a simple way during program execution. Processor stack is a part of RAM area where the data is saved in a Last In First Out (LIFO) fashion just like a stack of paper on a table. Data is stored by executing a 'push' instruction and data is read out using a 'pop' instruction.
I/O Registers: In addition to the Data memory, some special purpose registers are required that are used in input/output and control operations. These registers are called I/O registers. These are important for microcontroller peripheral interface and control applications.

- **Hardware interface registers (I/O Space)**

As we already know a microcontroller has some embedded peripherals and I/O devices. The data transfer to these devices takes place through I/O registers. In a microprocessor, input /output (I/O) devices are externally interfaced and are mapped either to memory address (memory mapped I/O) or a separate I/O address space (I/O mapped I/O).

In a microcontroller, two possible architectures can be used i.e., Princeton(Von Neumann) architecture and Harvard architecture.

- **I/O Register space in Princeton architecture**

In Princeton architecture we have only one memory interface for program memory (ROM) and data memory (RAM). One option is to map the I/O Register as a part of data memory or variable RAM area. This architecture is simple and straight forward. This is called memory mapped I/O. Alternatively a separate I/O register space can be assigned.

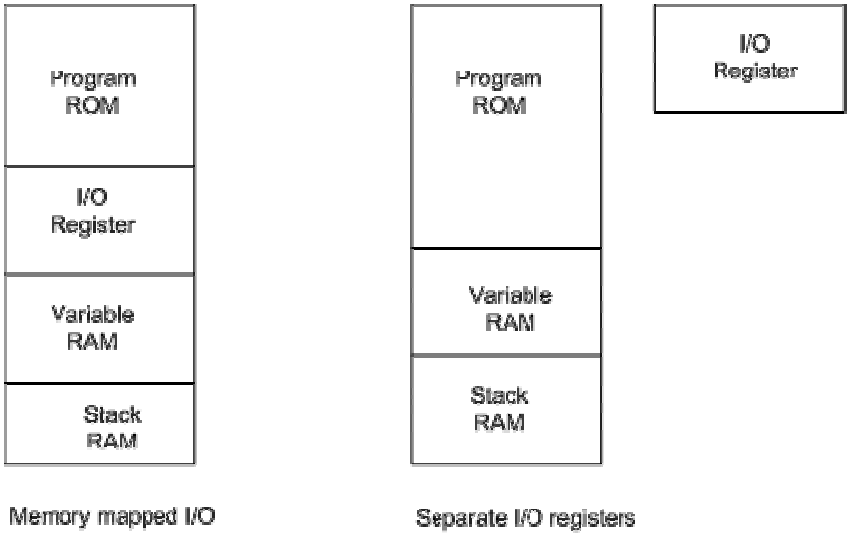


Fig 4.11 I/O Registers in Princeton Architecture

The drawback of memory mapped I/O is that a program which wrongly executed may overwrite I/O registers.

- **I/O Register space in Harvard architecture**

These are the following options available for I/O register space in Harvard Architecture.

1. I/O registers in program ROM.
2. I/O registers in register space (Data Memory area).
3. I/O registers in separate space.

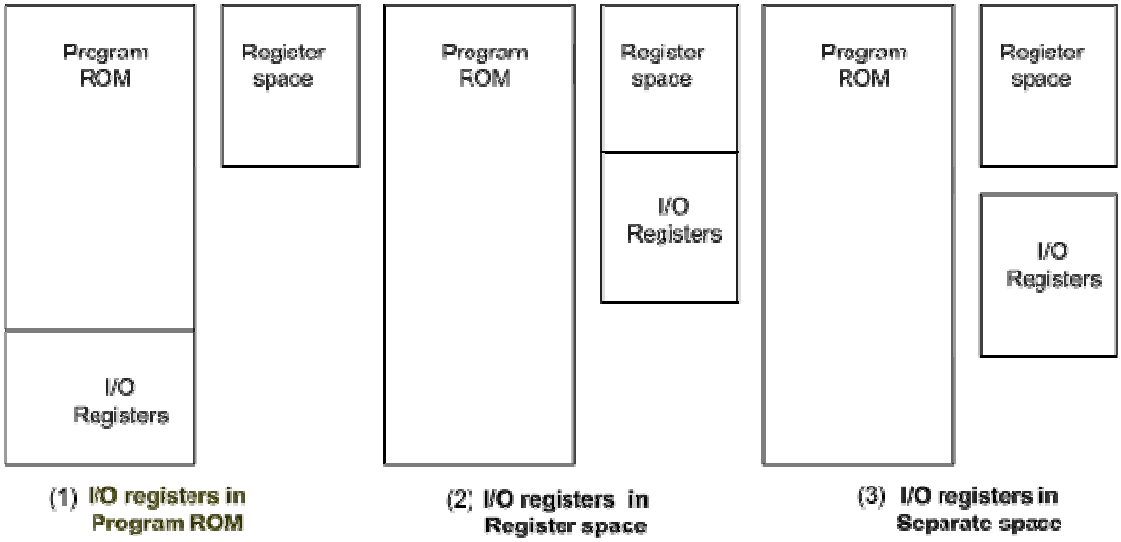


Fig 4.12Organization of I/O registers in Harvard Architecture

The first option is somewhat difficult to implement as there is no means to write to program ROM area. It is also complicated to have a separate I/O space as shown in (3). Hence the second option where I/O registers are placed in the register space is widely used.

4.6 Compiler / IDE (Integrated Development Environment)

Atmel Microcontrollers are very famous as they are very easy to use. There are many development tools available for them. First of all we need an easy IDE for developing code. I suggest beginners to use CVAVR (Code Vision AVR) Evaluation version is available for free download from the website. It has limitation of code size. It works on computers with Windows platform that is Windows XP & Vista.

Some famous compilers/development tools supporting Windows for Atmel Microcontrollers are:

- WINAVR (AVRGCC for Windows)
- Code Vision AVR (CVAVR)
- AVR Studio (Atmel's free developing tool)

AVRGCC is a very nice open source compiler used by most of the people.

4.7 Programmer

Programmer basically consists of two parts:

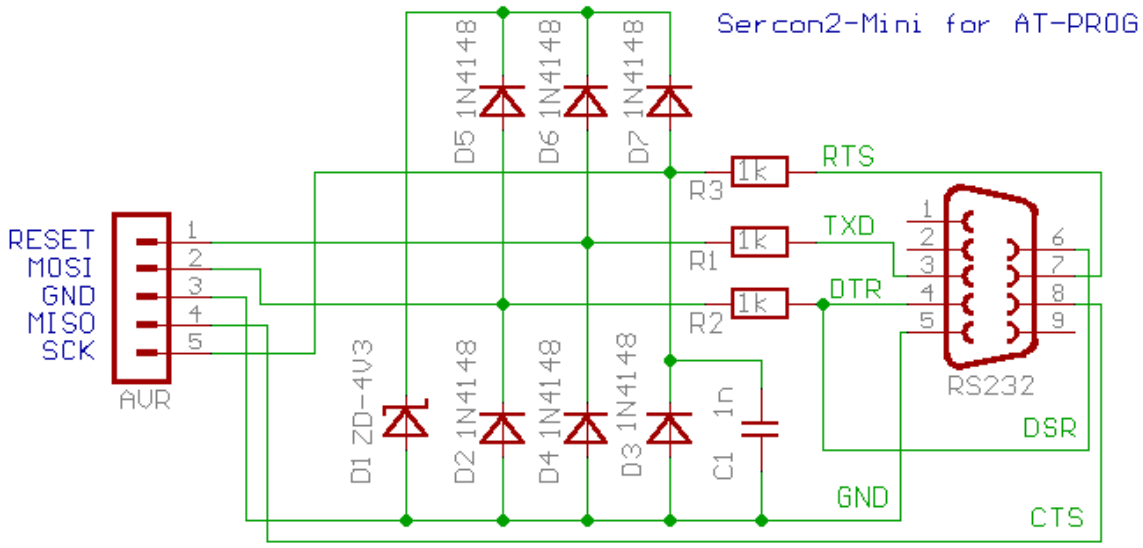
- Software (to open .hex file on your computer)
- Hardware (to connect microcontroller)

Hardware depends on the communication port you are using on the computer (Serial, Parallel or USB). I suggest beginners to use Serial Programmer as it is very easy to build. Software for that is Pony Prog. Some famous Windows (XP, Vista) programmers are:

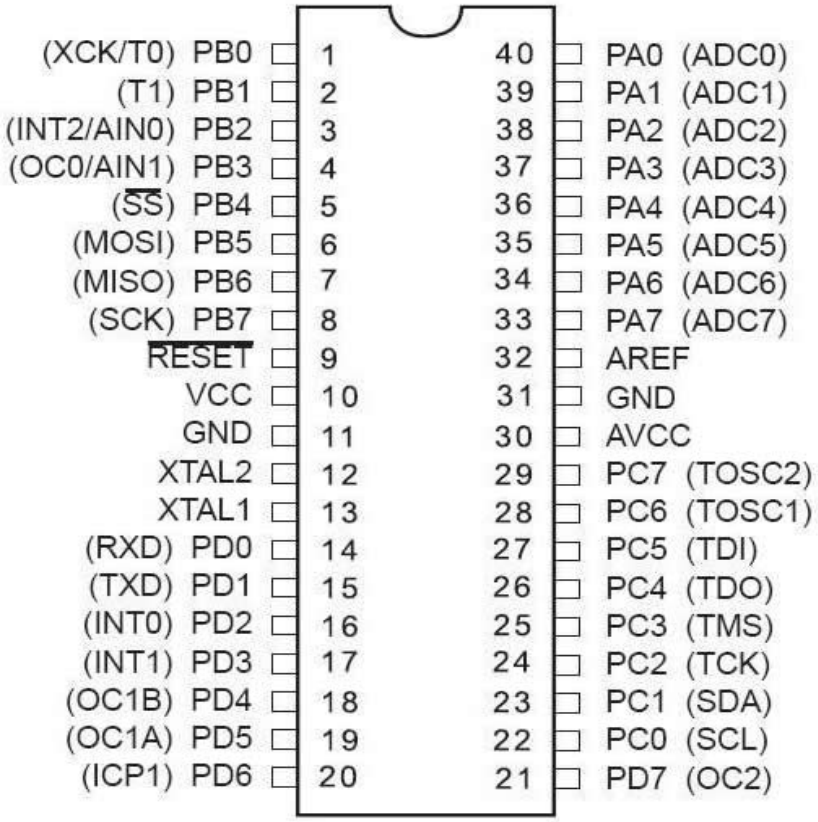
- Pony Prog(Serial, Parallel)
- AVRdude(supports many hardwares)
- AVRStudio (supports Atmel's hardware)
- ATProg (Serial)
- USB-ASP (USB)

4.8 How to use Serial Programmer's Circuit (Hardware)

This is the easiest programmer circuit to make. You just have to get Serial Port connector and three 1K resistors and you are done! Circuit Diagram is attached.



Now open the datasheet of Atmega you are using (I am using Atmega-16). Go to the pin configuration and find the following pins and connect the programmer. Programming is done through SPI (Serial Peripheral Interface) which involves MISO, MOSI and SCK pins. RESET is used to reset the chip. 0 volt on this pin will reset the chip and for normal running it should be pulled up to +5V.



- MOSI (Master Out Slave In)
- MISO (Master In Slave Out)
- SCK (Serial Clock)
- RESET
- GND (Ground)

Now connect the power supplies that are Vcc and GND to the micro controller.

Vcc = +5V and GND = 0V

Do not forget to connect Reset to Vcc with a 1K/10K resistor for pulling up.

That is it we are ready with the hardware.

Note: - The Trainer Board has On-Board Serial Programmer.

- **Software**

As I told you there are two parts of a programmer, hardware and software. We can build this hardware as it is very easy with just 3-4 components. Now we need software which support this hardware and can communicate with micro controller using this circuit. There are 2 good softwares for Windows. They are,

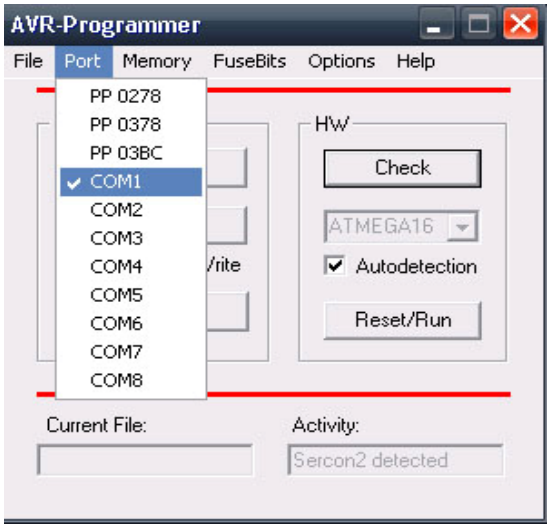
- Pony Prog
- At-Prog

Both of them support Serial as well as Parallel port, but I have always preferred serial port because it has only 9 pins, hence a smaller connector is required.

Now let us discuss about them.

- **At-Prog**

I consider it as simplest programmer ever. The circuit we discussed is actually based on this programmer. Download the folder from the website, unzip it. It has an executable file named **at-prog.exe** double click to execute it.

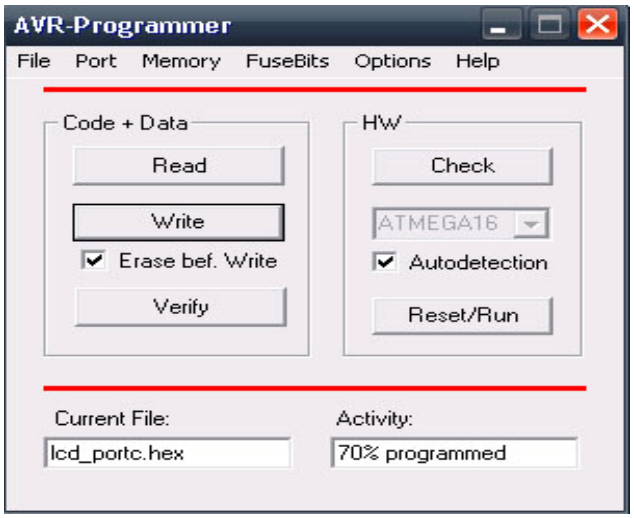


Go to **PORT** and select the address of serial port (by default COM1). Once you click that **Sercon2 detected** should appear in the **Activity** tab. That is the name of the programmer circuit.

Turn on the power supply and connect the circuit to the micro controller.

Click on **Check**, it should show **OK** in the **Activity** tab and device name (ATMEGA16) will automatically come in the tab below check option. This is the autodetect option of this programmer which auto detects the device name connected by its signature.

Now just open the device file from **File**--->**Open** and click on **Write**. It will write and verify the program. Done! So simple isn't it? :)

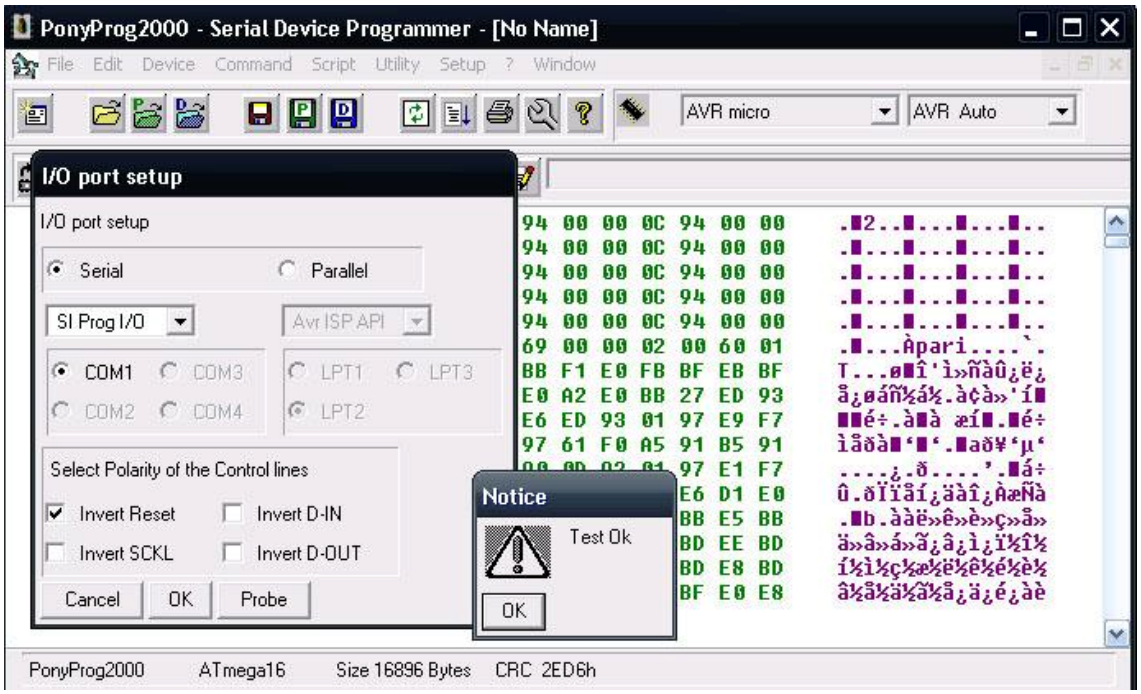


- **Pony Prog**

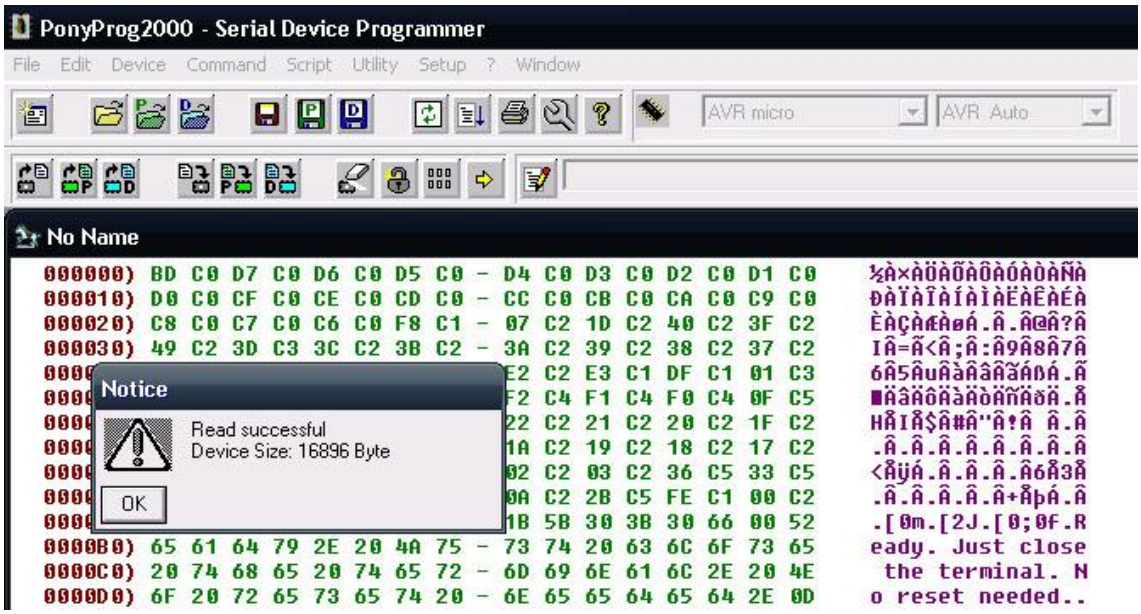
One of the most commonly used programmers on Windows. Download it from the website and install it. Now let's set it up for our hardware.

First select **AVR micro** and **AVR-Auto** (you can also specify device name, **Atmega16**) in the chip options (last two drop down tabs)

Go to the **Setup**---->**Interface-Setup**. Then do the settings as shown in the picture below. Then connect the circuit to the microcontroller, turn on the power supply. Now click on **Probe**. You should get **Test OK** message. If not, check your connections again.

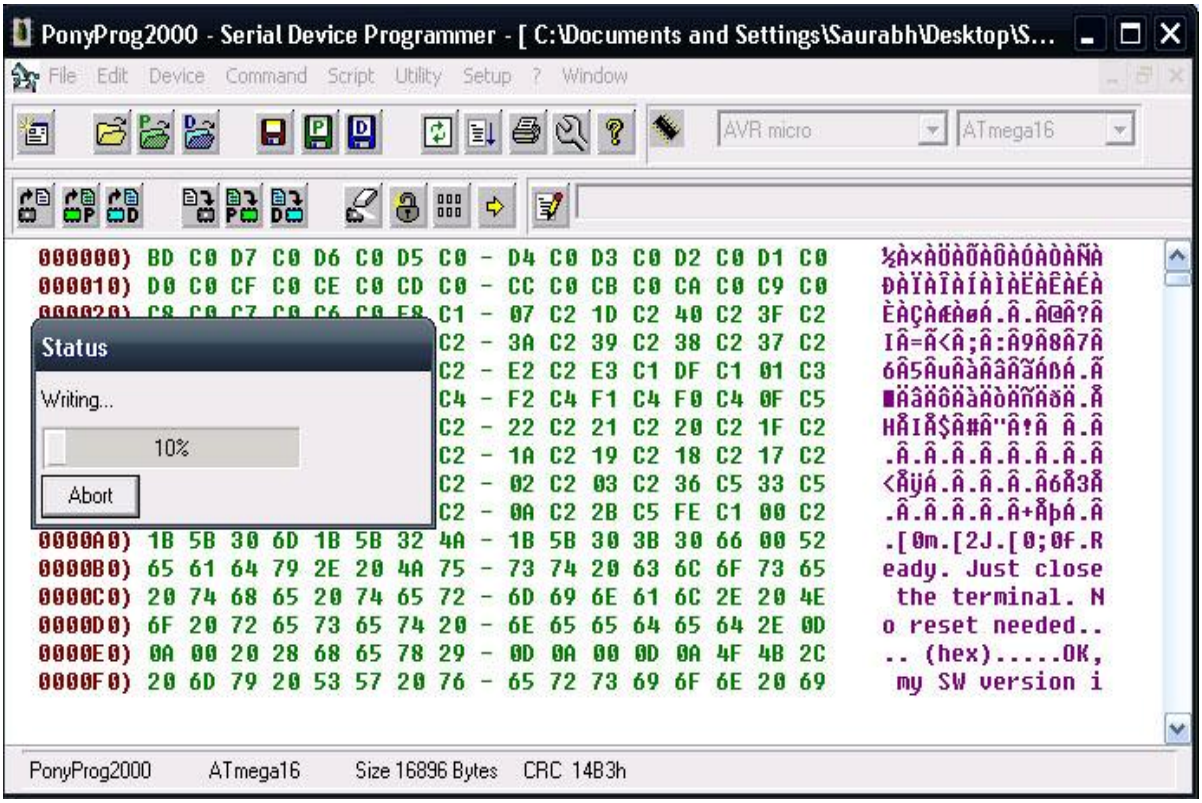


Now let's read the Microcontroller. Go to **Command**---->**Read All**. It should start reading the signature and the flash memory. You should get **Read Successful** message after that.



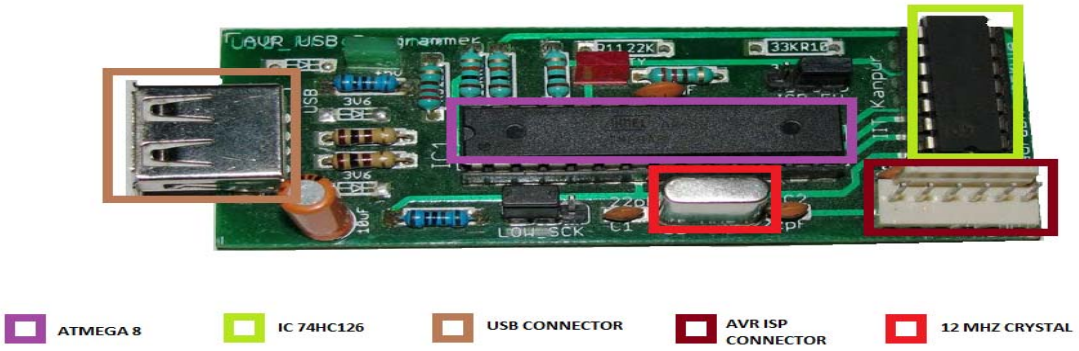
So we are done with the settings and testing's. Everything is working fine :) Now just open the hex file from **File**---->**Open-Device-File**

Go to **Command**---->**Write-Program (Flash)**. It will start writing and then verifying the code. Congratulations :) you did it!



4.9 USB Programmer

USBasp is a USB in-circuit programmer for Atmel AVR controllers. It simply consists of an ATMega48 or an ATMega8 and a couple of passive components. The programmer uses a firmware-only USB driver; no special USB controller is needed.



Chapter 5 Code Vision AVR (CVAVR)

An IDE has following functions:

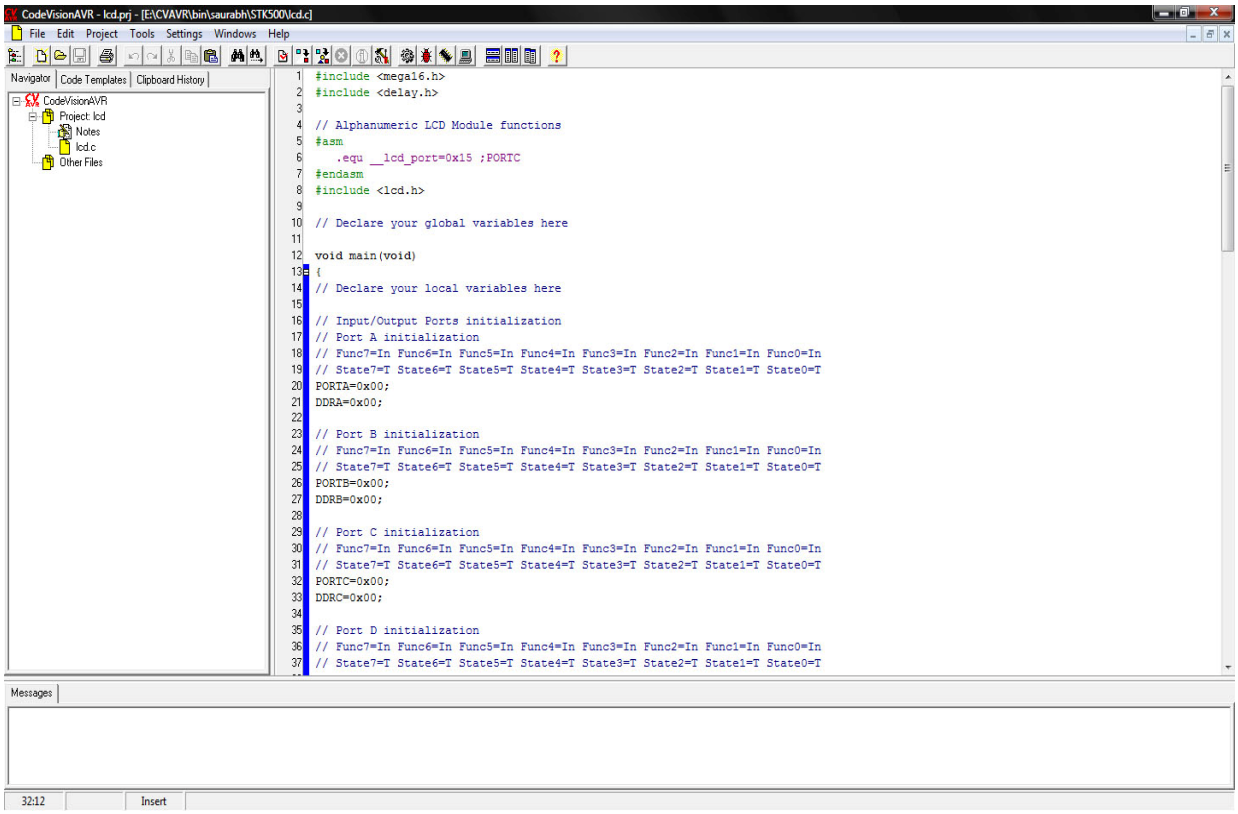
- Preprocessing
- Compilation
- Assembly
- Linking
- Object Translation
- Text Editor

If we just use compiler and linker independently we still need to get a text editor. So combining everything will actually mess things up. So the best way is to get Software which has it all. That's called an Integrated Development Environment, in short IDE.

I consider **Code-Vision-AVR** to be the best IDE for getting started with AVR programming on Windows XP, Vista. It has a very good Code Wizard which generate codes automatically ! You need not mess with the assembly words. So in all my tutorials I will be using CVAVR. You can download evaluation version for free which has code size limitation but good enough for our purpose.

For all my examples I will be using **Atmega-16** as default microcontroller because it very easily available and is powerful enough with sufficient number of pins and peripherals we use. You can have a look on the datasheet of **Atmega-16** in the datasheet section.

Let's take a look on the software. The main window looks like following,



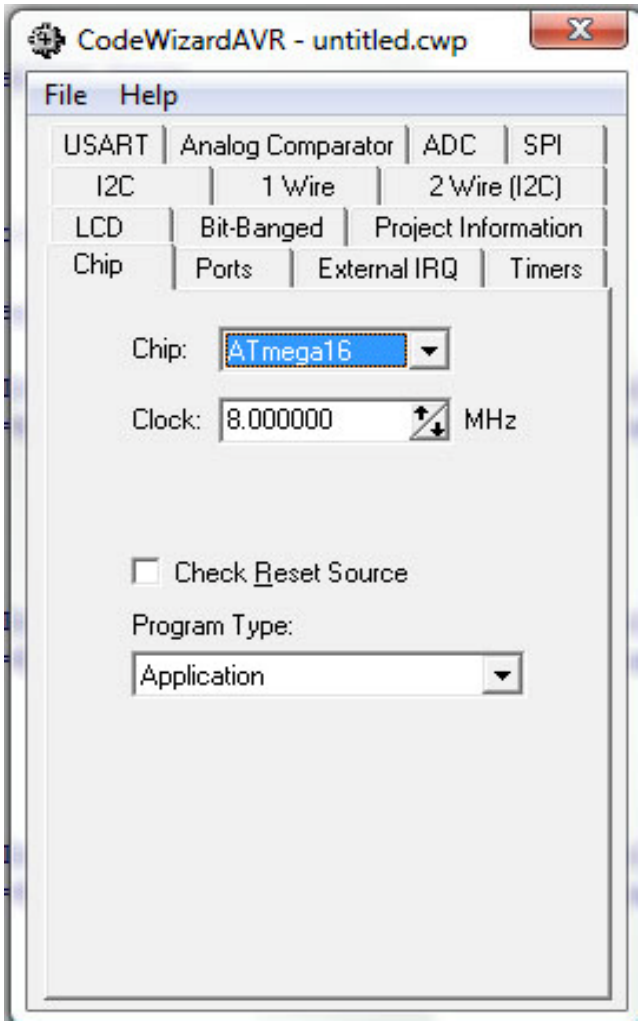
Now click on **File ---> New --->Project**

A pop up window will come asking whether you want to use Code Wizard AVR, obviously select yes because that is the reason we are using CVAVR !

Now have a look on this Wizard. It has many tabs where we can configure PORTS, TIMERS, LCD, ADC etc. I am explaining some of them

5.1 CHIP:

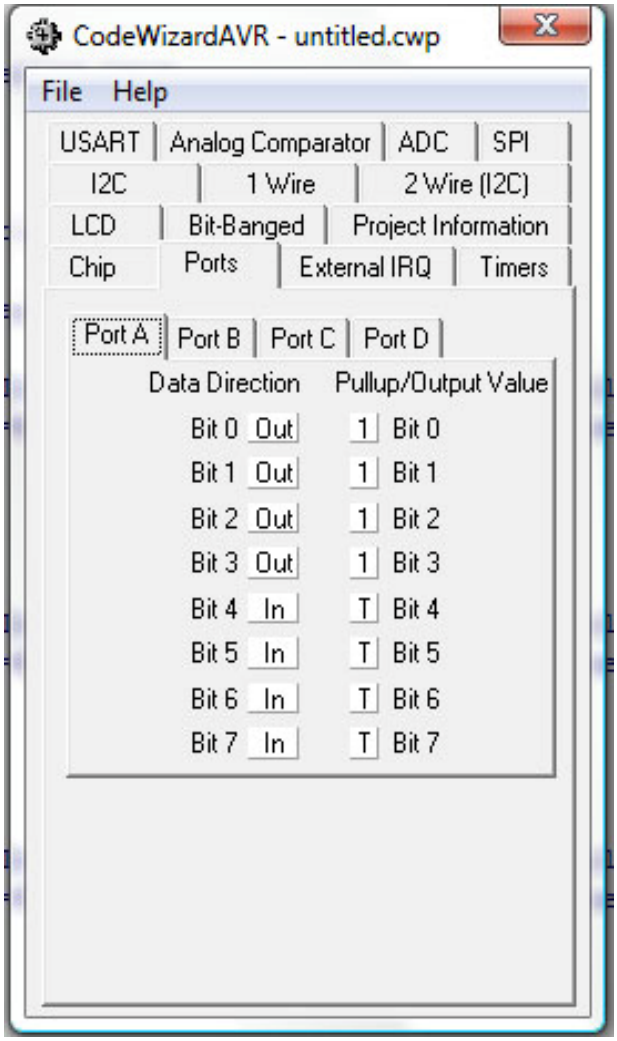
Select the chip for which you are going to write the program. Then select the frequency at which Chip is running. By default all chips are set on Internal Oscillator of 1 MHz so select 1 MHz if that is the case. If you want to change the running clock frequency of the chip then you have to change its fuse bits (I will talk more about this in fuse bits section).



5.2 PORT:

PORT is usually a collection of 8 pins.

From this tab you can select which pin you want to configure as output and which as input. It basically writes the DDR and PORT register through this setting. Registers are basically RAM locations which configure various peripherals of microcontroller and by changing value of these registers we can change the function it is performing. I will talk more about registers later. All the details are provided in the datasheet.



So you can configure any pin as output or input by clicking the box.

For Atmega-16 which has 4 Ports we can see 4 tabs each corresponding to one Port. You can also set initial value of the Pins you want to assign. or if you are using a pin as input then whether you want to make it as pull-up or tristated, again I will talk in details about these functions later.

Similarly using this code wizard you can very easily configure all the peripherals on the Atmega.

Now for generating code just go to **File ----> Generate, Save and Exit** (of the code wizard)

Now it will ask you name and location for saving three files. Two being project files and one being the .C file which is your program. try to keep same names of all three files to avoid confusion. By default these files are generated in **C:\CVAVR\bin**

The generated program will open in the text editor. Have a look it has some declarations like PORT, DDR, TCCR0 and many more. These are all registers which configures various functions of Atmega and by changing these value we make different functions. All the details about the registers are commented just below them. Now go down and find following infinite while loop there. We can start writing our part of program just before the while loop. And as for most of the applications we want microcontroller to perform the same task forever we put our part of code in the infinite while loop provided by the code wizard !

```
While                                                    (1)
{
// Place your code here

};
}
```

See how friendly this code wizard is, all the work (configuring registers) automatically done and we don't even need to understand and go to the details about registers too !

Now we want to generate the hex file, so first compile the program. Either press F9 or go to **Project ---> Compile**.

It will show compilation errors if any. If program is error free we can proceed to making of hex file. So either press Shift+F9 or go to **Project ----> Make**. A pop up window will come with information about code size and flash usage etc.

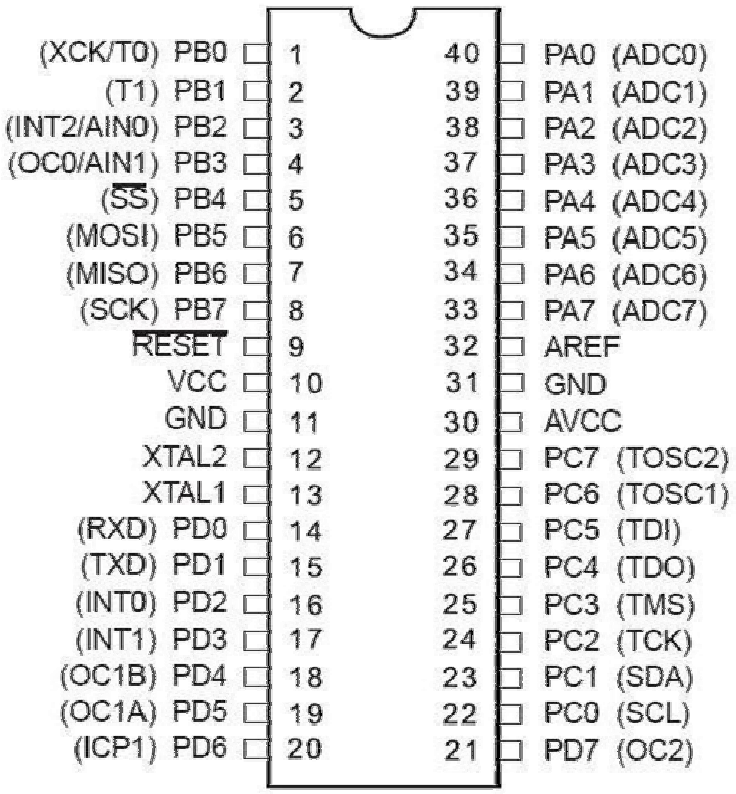
So the machine file is ready now! It is in the same folder where we saved those 3 files.

Chapter 6 Introduction to Atmega 16 Microcontroller

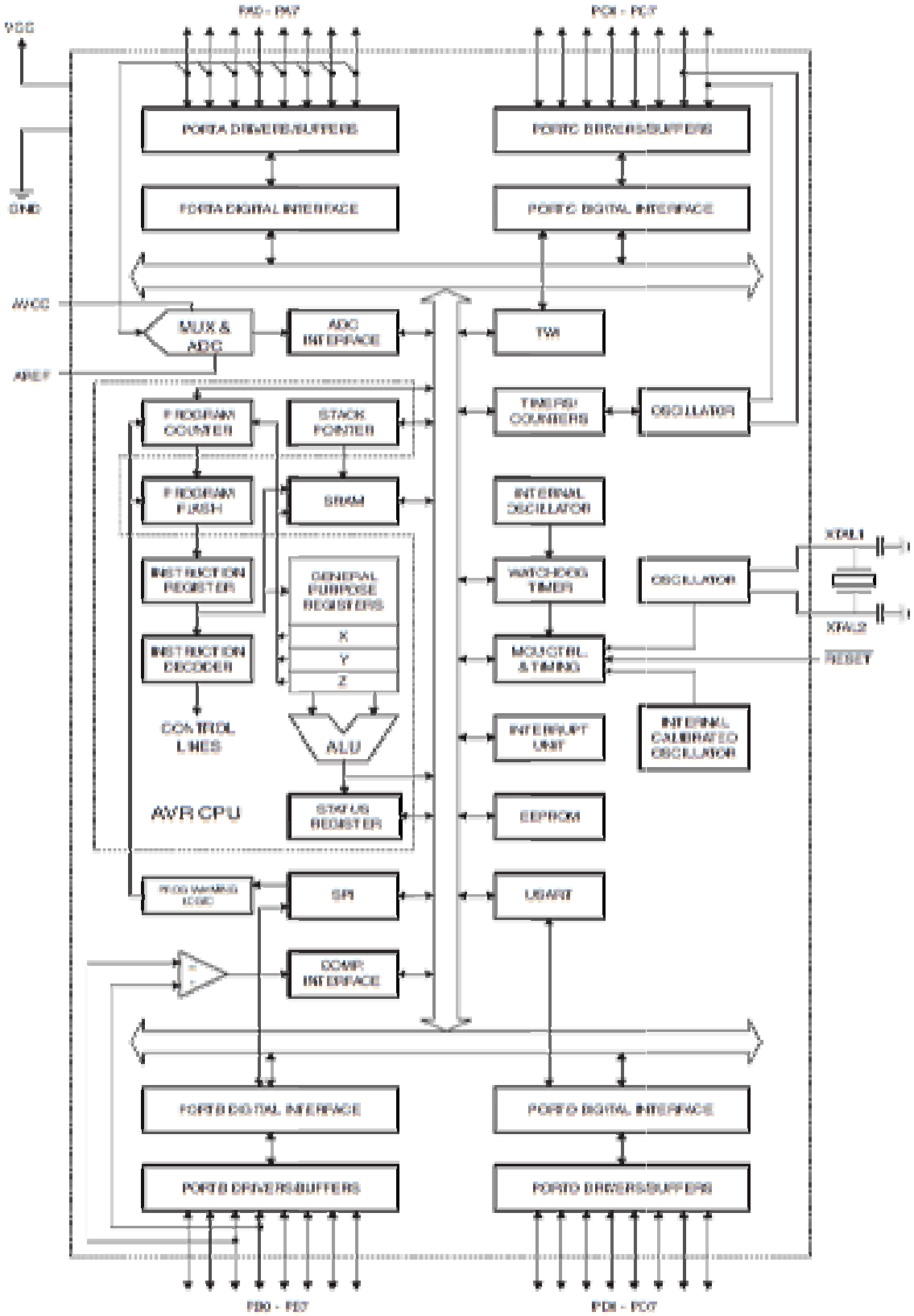
6.1 Features

- Advanced RISC Architecture
- Up to 16 MIPS Throughput at 16 MHz
- 16K Bytes of In-System Self-Programmable Flash
- 512 Bytes EEPROM
- 1K Byte Internal SRAM
- 32 Programmable I/O Lines
- In-System Programming by On-chip Boot Program
- 8-channel, 10-bit ADC
- Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture
- Four PWM Channels
- Programmable Serial USART
- Master/Slave SPI Serial Interface
- Byte-oriented Two-wire Serial Interface
- Programmable Watchdog Timer with Separate On-chip Oscillator
- External and Internal Interrupt Sources

6.2 Pin Configuration



6.3 Block Diagram



6.4 Pin Descriptions

VCC: Digital supply voltage. (+5V)

GND: Ground. (0 V) Note there are 2 ground Pins.

Port A (PA7 - PA0)
Port A serves as the analog inputs to the A/D Converter. Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. When pins PA0 to PA7 are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Port B (PB7 - PB0)
Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). Port B also serves the functions of various special features of the ATmega16 as listed on page 58 of datasheet.

Port C (PC7 - PC0)
Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). Port C also serves the functions of the JTAG interface and other special features of the ATmega16 as listed on page 61 of datasheet. If the JTAG interface is enabled, the pull-up resistors on pins PC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

Port D (PD7 - PD0)
Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). Port D also serves the functions of various special features of the ATmega16 as listed on page 63 of datasheet.

RESET: 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.

XTAL1: External oscillator pin 1

XTAL2: External oscillator pin 2

AVCC: AVCC is the supply voltage pin for Port A and the A/D Converter. 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.

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

6.5 Digital Input Output Port

So let's start with understanding the functioning of AVR. We will first discuss about I/O Ports. Again I remind you that I will be using and writing about **Atmega-16**. Let's first have a look at the Pin configuration of Atmega-16. Image is attached, click to enlarge.

You can see it has 32 I/O (Input/Output) pins grouped as A, B, C & D with 8 pins in each group. This group is called as PORT.

- PA0 - PA7 (PORTA)
- PB0 - PB7 (PORTB)
- PC0 - PC7 (PORTC)
- PD0 - PD7 (PORTD)

Notice that all these pins have some function written in bracket. These are additional function that pin can perform other than I/O. Some of them are.

- ADC (ADC0 - ADC7 on PORTA)
- UART (Rx,Tx on PORTD)
- TIMERS (OC0 - OC2)
- SPI (MISO, MOSI, SCK on PORTB)
- External Interrupts (INT0 - INT2)

6.6 Registers

All the configurations in microcontroller is set through 8 bit (1 byte) locations in RAM (RAM is a bank of memory bytes) of the microcontroller called as **Registers**. All the functions are mapped to its locations in RAM and the value we set at that location that is at that Register configures the functioning of microcontroller. There are total 32 x 8bit registers in Atmega-16. As Register size of this microcontroller is 8 bit, it called as 8 bit microcontroller.

Chapter 7 I/O Ports:

Input Output functions are set by Three Registers for each PORT.

- DDRX ----> Sets whether a pin is Input or Output of PORTX.
- PORTX ---> Sets the Output Value of PORTX.
- PINX -----> Reads the Value of PORTX.

Go to the page 50 in the datasheet or you can also see the I/O Ports tab in the Bookmarks.

7.1 DDRX (Data Direction Register)

First of all we need to set whether we want a pin to act as output or input. DDRX register sets this. Every bit corresponds to one pin of PORTX. Let's have a look on DDRA register.

Bit	7	6	5	4	3	2	1	0
PIN	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0

Now to make a pin act as I/O we set its corresponding bit in its DDR register.

- To make Input set bit 0
- To make Output set bit 1

If I write **DDRA = 0xFF** (0x for Hexadecimal number system) that is setting all the bits of DDRA to be 1, will make all the pins of PORTA as Output.

Similarly by writing **DDRD = 0x00** that is setting all the bits of DDRD to be 0, will make all the pins of PORTD as Input.

Now let's take another example. Consider I want to set the pins of PORTB as shown in table,

PORT-B	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0
Function	Output	Output	Input	Output	Input	Input	Input	Output
DDRB	1	1	0	1	0	0	0	1

For this configuration we have to set DDRB as **11010001** which in hexadecimal is**D1**. So we will write **DDRB=0xD1**

Summary

- DDRX -----> to set PORTX as input/output with a byte.
- DDRX.y ---> to set yth pin of PORTX as input/output with a bit (works only with CVAVR).

7.2 PORTX (PORTX Data Register)

This register sets the value to the corresponding PORT. Now a pin can be Output or Input. So let's discuss both the cases.

- **Output Pin**

If a pin is set to be output, then by setting bit 1 we make output **High** that is +5V and by setting bit 0 we make output **Low** that is 0V.

Let's take an example. Consider I have set DDRA=0xFF, that is all the pins to be Output. Now I want to set Outputs as shown in table,

PORT-A	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0
Value	High(+5V)	High(+5V)	Low(0V)	Low(0V)	Low(0V)	High(+5V)	High(+5V)	Low(0V)
PORTA	1	1	0	0	0	1	1	0

For this configuration we have to set **PORTA** as **11000110** which in hexadecimal is**C6**. So we will write **PORTA=0xC6;**

- **Input Pin**

If a pin is set to be input, then by setting its corresponding bit in PORTX register will make it as follows,Set bit 0 ---> Tri-Stated Set bit 1 ---> Pull Up

Tristated means the input will ***hang*** (no specific value) if no input voltage is specified on that pin. Pull Up means input will go to **+5V** if no input voltage is given on that pin. It is basically connecting PIN to +5V through a 10K Ohm resistance.

Summary

- PORTX ----> to set value of PORTX with a byte.
- PORTX.y --> to set value of y^{th} pin of PORTX with a bit (works only with CVAVR)

7.3 PINX (Data Read Register)

This register is used to read the value of a PORT. If a pin is set as input then corresponding bit on PIN register is,

- 0 for Low Input that is V < 2.5V
- 1 for High Input that is V > 2.5V (Ideally, but actually 0.8 V - 2.8 V is error zone !)

For an example consider I have connected a sensor on PC4 and configured it as an input pin through DDR register. Now I want to read the value of PC4 whether it is Low or High. So I will just check 4th bit of PINC register.

We can only read bits of the PINX register; can never write on that as it is meant for reading the value of PORT.

Summary

- PINX ----> Read complete value of PORTX as a byte.
- PINX.y --> Read y^{th} pin of PORTX as a bit (works only with CVAVR).

7.4 A SMALL NOTE ABOUT "DELAY"

C has inbuilt libraries which contain many pre-built functions. One such function is "Delay", which introduces a time delay at a particular step. To invoke it in your program, you need to add the following line at the beginning of your code:

#include<delay.h>;

Thereafter, it can be used in the program by adding the following line:

delay_ms(X);

Where X is the time delay you wish to introduce at that particular step in milliseconds.

I hope you must have got basic idea about the functioning of I/O Ports. For detailed reading you can always refer to datasheet of Atmega.

Chapter 8 LCD Interfacing

Now we need to interface an LCD to our microcontroller so that we can display messages, outputs, etc. Sometimes using an LCD becomes almost inevitable for debugging and calibrating the sensors (discussed later). We will use the 16x2 LCD, which means it has two rows of 16 characters each. Hence in total we can display 32 characters.

8.1 Overview of LCD Display

LCD displays are widely used in many applications like mobile phones, robotics, DVD players, Measurement instruments etc. Intelligent LCD displays are very capable because they can display complete ASCII character set and even graphics. These displays are easily connected with micro controller and microprocessors. LCD displays are complete embedded system in them, because it include microcontroller, RAM and ROM.

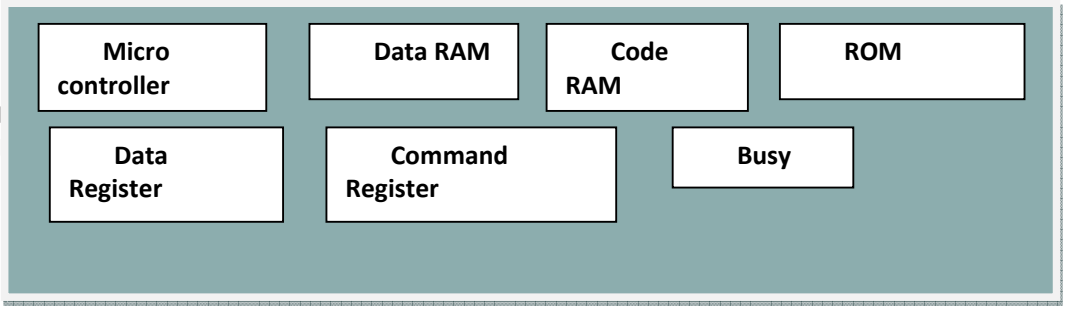
16X2 LCD DISPLAY

LCD Modules can present textual information to user. It's like a cheap "monitor" that you can hook in all of your gadgets.
They come in various types. The most popular one is 16x2 LCD module. It has 2 rows & 16 columns.

The intelligent displays are two types:

a)Text Display
b)Graphics Display
Text display can display all character set and graphics display can show anyGraphics because they are interfaced pixel wise.
In recent year the LCD is finding widespread use replacing LEDs (seven segment LEDs or multisegment LEDs).
This is due to the following reasons:
a)The declining prices of LCDs.
b)The ability to display the numbers, characters and graphics. This is not possible in LEDs,which can display the numbers and few characters.
c)Incorporation of a refreshing controller into the LCD, Thereby reliving the CPU of the task of refreshing the LCD.In contrast,the LED must be refreshed by the CPU (or in some other way) to keep displaying the data.
Theinterfacing of LCD is quite difficult.But we will try to make it simple and let us explain it for you.
We will learn how to interface the text intelligent LCD display. These displays are available in the market of 16 column and one Row and more than one row displays.

- **Block Diagram of LCD Display**



DATA RAM:

This RAM is storing the ASCII values of corresponding characters which will be displayed on the LCD . For each column there is one location in the RAM. When we will store the ASCII value at that location than its corresponding character will be displayed on the screen.

CODE RAM:

This RAM stores the binary pattern according to the character.

ROM:

This ROM stores the binary pattern which is according to the Pixels of LCD and there are patterns of every character.

COMMAND REGISTER:

It stores various commands for proper functioning.

DATA REGISTER:

This register work as buffer for data lines and the internal buses of LCD. The ASCII values of characters will be given to the data register.

BF (BUSY FLAG):

It indicates the internal working of the LCD. It show whether LCD is busy in any operation or not.

If BF=0 (LCD is idle we can proceed for next operation)
If BF=1 (LCD is busy we cannot proceed for next operation and we have to wait unless operation completes).

DESCRIPTION OF PINS IS GIVEN BELOW:

VCC, VSS and VO:

While VCC and VSS provide +5 Volts and ground, respectively VEE is used for controlling LCD contrast

RS (REGISTER SELECT):

The RS pin is used to select Data Register or Command Register.

If RS=0, CR Register is selected, allowing the user to send a command such as clear display,curser at the home etc.
If RS=1, DR Register is selected, allowing the user to send data to be display on the LCD.

R/W (READ/WRITE):

When R/W=0, Write operation..

When R/W=1 Read Operation

EN (ENABLE):

The Enable pin is used by the LCD to latch binary bits available on its data pins. When data is supplied to data pins, a negative edge is applied to this pin So that the LCD latches in the data present at the data pins. This pulse must be a minimum of 450 ns wide.
There should be positive edge at EN pin when read operation is required.

D7-D0:

This is 8-bit data pins. D7-D0 are used to send information to the LCD or read the contents of the LCD's internal registers.

BK-LED (LEDA, LEDK):

These pins are used to give the supply to the back light of the LCD display. So, that content of the LCD display can be viewed in the dark.

8.2 Circuit Connection

There are 16 pins in an LCD; See reverse side of the LCD for the PIN configuration.
The connections have to be made as shown below:

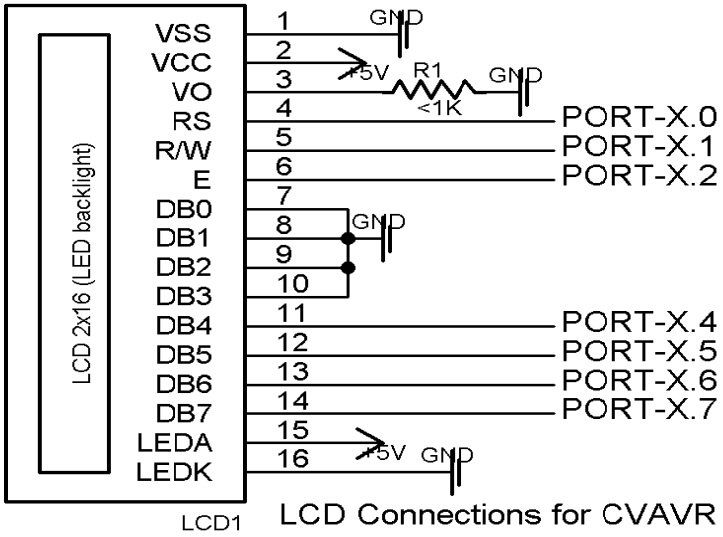


Figure 14: LCD connections

8.3 Setting up in Microcontroller

When we connect an LCD to Atmega16, one full PORT is dedicated to it, denoted by PORT-X in the figure. To enable LCD interfacing in the microcontroller, just click on the LCD tab in the Code Wizard

and select the PORT at which you want to connect the LCD. We will select PORTC. Also select the number of characters per line in your LCD. This is 16 in our case. Code Wizard now shows you the complete list of connections which you will have to make in order to interface the LCD. These are nothing but the same as in the above figure for general PORT-X.

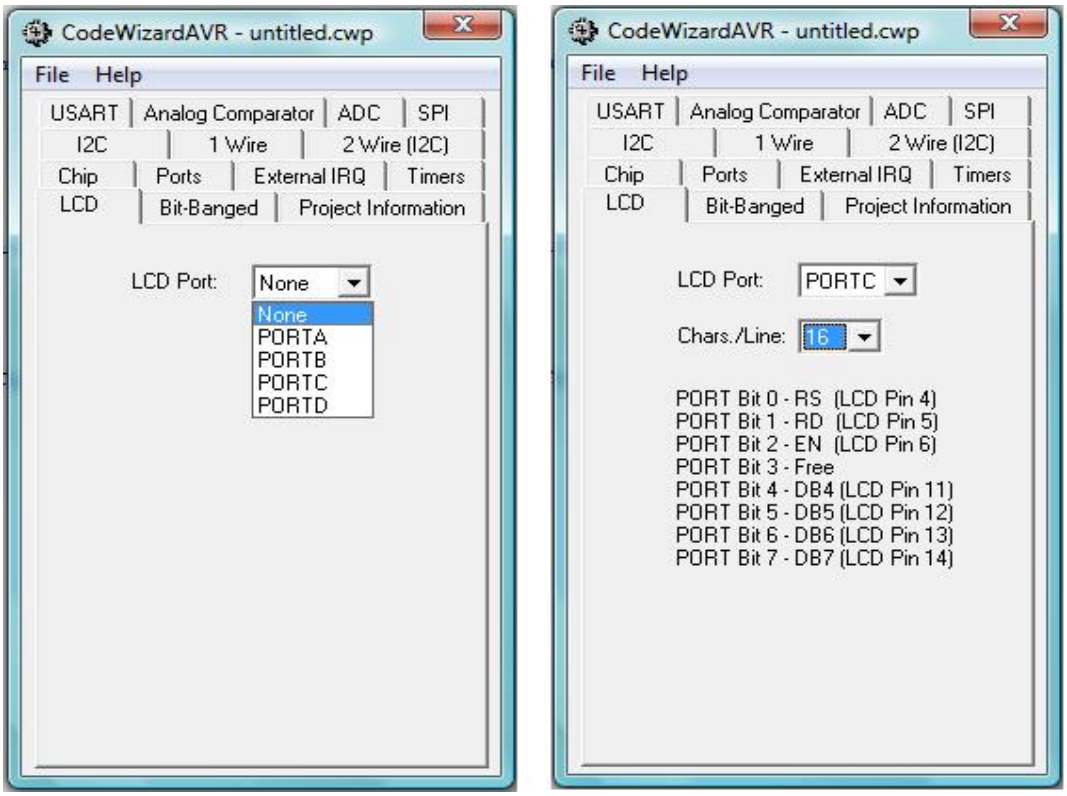


Figure 15: LCD settings on CVAVR wizard window.

As you can see, there are some special connections other than those to uC, Vcc and gnd. These are general LCD settings. Pin 3 (VO) is for the LCD contrast, ground it through a <1kΩ resistance/ potentiometer for optimum contrast. Pin 15 & 16 (LEDA and LEDK) are for LCD backlight, give them permanent +5V and GND respectively as we need to glow it continuously.

8.4 Printing Functions

Now once the connections have been made, we are ready to display something on our screen. Displaying our name would be great to start with. Some of the general LCD functions which you must know are:

8.4.1 lcd_clear()

Clears the lcd. Remember! Call this function before the while(1) loop, otherwise you won't be able to see anything!

8.4.2 lcd_gotoxy(x,y)

Place the cursor at coordinates (x,y) and start writing from there. The first coordinate is (0,0). Hence, x ranges from 0 to 15 and y from 0 to 1 in our LCD. Suppose you want to display something starting from the 5^{th} character in second line, then the function would be

lcd_gotoxy(5,1);

8.4.3 lcd_putchar(char c)

To display a single character.E.g.,
lcd_putchar('H');

8.4.4 lcd_putsf(constant string)

To display a constant string.Eg,

lcd_putsf("IIT Kanpur");

8.4.5 lcd_puts(char arr)

To display a variable string, which is nothing but an array of characters (data type char) in C language . e.g., You have an array char c[10] which keeps on changing. Then to display it, the function would be called as
lcd_puts(c);

Now we have seen that only characters or strings (constant or variable) can be displayed on the LCD. But quite often we have to display values of numeric variables, which is not possible directly. Hence we need to first convert that numeric value to a string and then display it. For e.g., if we have a variable of type integer, say **int**k, and we need to display the value of k (which changes every now and then, 200 now and 250 after a second... and so on). For this, we use the C functions **itoa()** and **ftoa()**, but remember to include the header file **stdlib.h** to use these C functions.

8.4.6 itoa(intval, char arr[])

It stores the value of integer val in the character array arr. E.g., we have already defined int i and char c[20], then

```
itoa(i,c);
lcd_puts(c);
```

Similarly we have

8.4.7 ftoa(float val, char decimal_places, char arr[])

It stores the value of floating variable f in the character array arr with the number of decimal places as specified by second parameter. E.g., we have already defined float f and char c[20], then

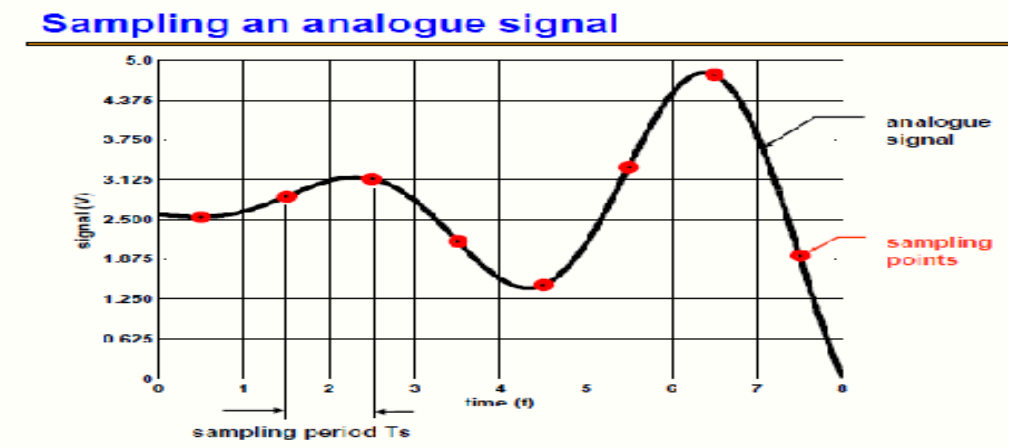
```
ftoa(f,4,c);                    // till 4 decimal places
lcd_puts(c);
```

Now we are ready to display anything we want on our LCD. Just try out something which you would like to see glowing on it!

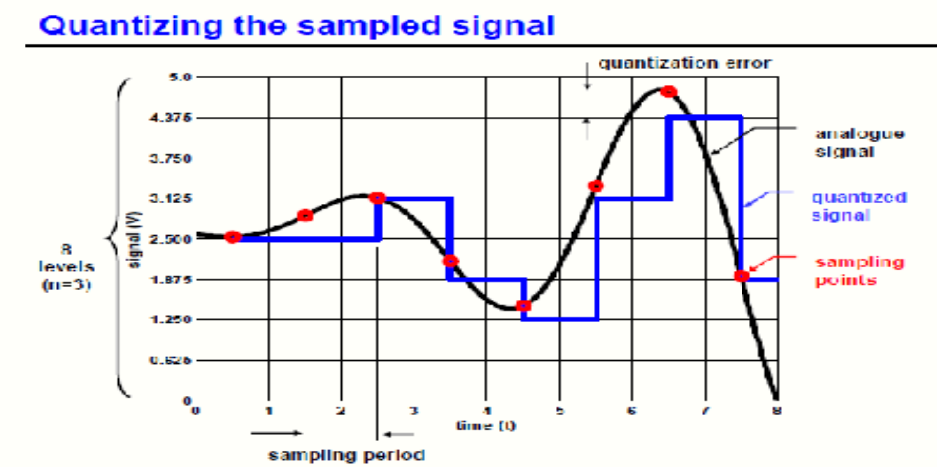
Most of the physical quantities around us are continuous. By continuous, we mean that the quantity can take any value between two extremes. For example, the atmospheric temperature can take any value within a certain range. If an electrical quantity is made to vary directly in proportion to this value then what we have is an Analogue Signal. Now we have brought a physical quantity into the electrical domain. The electrical quantity in most cases is voltage. To bring this quantity into digital domain we have to convert this into digital form. For this an ADC or analogue to digital converter is needed. Most modern MCU including AVRs have an ADC on chip.
An ADC converts an input voltage into a number. An ADC has a resolution. A 8 bit ADC has a range of 0-255. (28=256) The ADC also has a Reference Voltage (ARef). When the input voltage is GND the output is 0 and when the input voltage is equal to ARef the output is 255. So the input range is 0 to ARef and the output range is 0 to 255.

Input Voltage		**Digital Output**
0V	→	**0**
2.5V	→	**127**
5V	→	**255**

You can see that any analogue signal is not perfectl y converted – a factor that affects the output quality is the "sampling rate". The ADC cannot continuousl y read the input signal and change its output – it does so in certain time intervals. The frequency at which it samples the input is called its sampling rate.

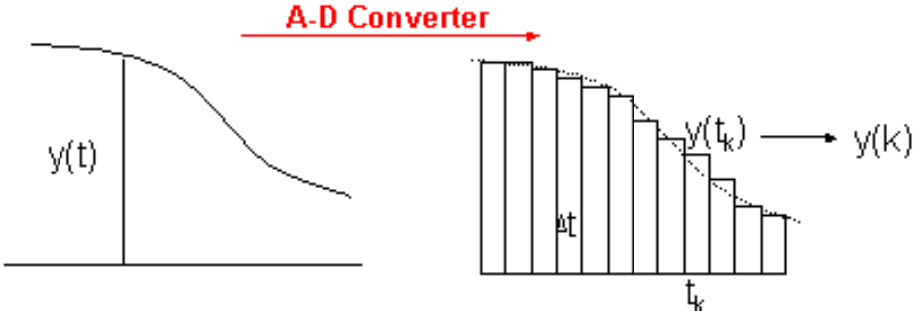


Also, the value of the analogue signal read is not stored perfectl y - for example, a voltage of 2.501 would be read as 2.5V i.e. 127 in digital format. Thus, the signal is "quantized", or, in other terms, there is certain graininess to the digital signal that you obtain. Ho w accurate your digital signal is depends on your resolution – higher resolution will make your digital signal more accurate.



9.1 Theory of operation

What we have seen till now that the input given to uC was digital, i.e., either +5 V (logic 1) or 0V (logic 0). But what if we have an analog input, i.e., value varies over a range, say 0V to +5V? Then we require a tool that converts this analog voltage to discrete values. Analog to Digital Converter (ADC) is such a tool.



ADC is available at PORTA of Atmega16. Thus we have 8 pins available where we can apply analog voltage and get corresponding digital values. The ADC register is a 10 bit register, i.e., the digital value ranges from 0 to 1023. But we can also use only 8 bit out of it (0 to 255) as too much precision is not required.

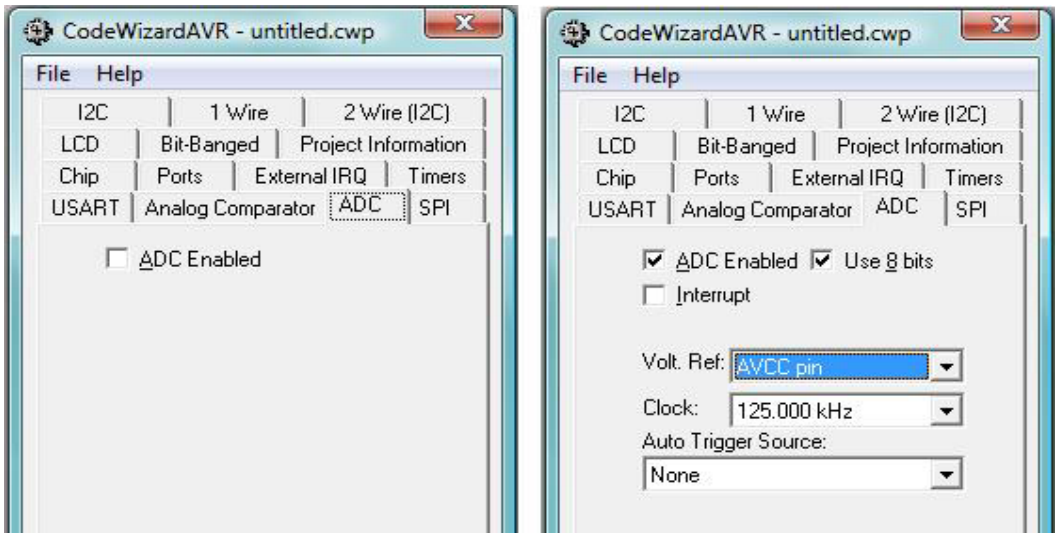
Reference voltage is the voltage to which the ADC assigns the maximum value (255 in case of 8 bit and 1023 for 10 bit). Hence, the ADC of Atmega16 divides the input analog voltage range (0V to Reference Voltage) into 1024 or 256 equal parts, depending upon whether 10 bit or 8 bit ADC is used. For example, if the reference voltage is 5V and we use 10bit ADC, 0V has digital equivalent 0, +5V is digitally 1023 and 2.5V is approximately equal to 512.

$ADC = V_{in} \times 255 / V_{ref}$ (8 bit)

$ADC = V_{in} \times 1023 / V_{ref}$ (10 bit)

9.2 Setting up Microcontroller

To enable ADC in Atmega16, click on the ADC tab in Code Wizard and enable the checkbox. You can also check "use 8 bits" as that is sufficient for our purpose and 10 bit accuracy is not required. If the input voltage ranges from 0 to less than +5V, then apply that voltage at AREF (pin 32) and select the Volt. Ref. as AREF pin. But if it ranges from 0 to +5 V, you can select the Volt. Ref. as AVCC pin itself. Keep the clock at its default value of 125 kHz and select the Auto Trigger Source as Free Running. You can also enable an interrupt function if you require.



9.3 Function for getting ADC

Now when you generate and save the code, all the register values are set automatically along with a function:

unsigned char read_adc(unsigned char adc_input).

This function returns the digital value of analog input at that pin of PORTA whose number is passed as parameter, e.g., if you want to know the digital value of voltage applied at PA3, and then just call the function as,

read_adc(3);

If the ADC is 8 bit, it will return a value from 0 to 255. Most probably you will need to print it on LCD. So, the code would be somewhat like

```
int a; char c[10];          // declare in the section of global variables
a=read_adc(3);
itoa(a,c);
lcd_puts(c);
```

Chapter 10 Timers

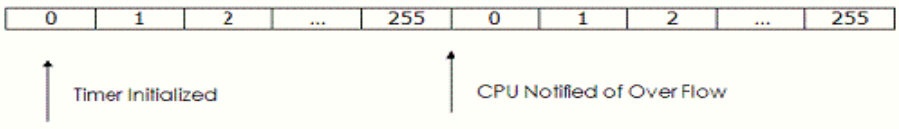
10.1 What is a Timer?

We use timers every day - the simplest one can be found on your wrist. A simple clock will time the seconds, minutes and hours elapsed in a given day - or in the case of a twelve hour clock, since the last half-day. AVR timers do a similar job, measuring a given time interval.
An AVR timer in simplest term is a register. Timers generally have a resolution of 8 or 16 bits. So an 8 bit timer is 8 bits wide, and is capable of holding value within 0-255. But this register has a magical property - its value increases/decreases automatically at a predefined rate (supplied by user). This is the timer clock. And this operation does not need CPU's intervention.

The AVR timers are very useful as they run asynchronous to the main AVR core. This is a fancy way of saying that the timers are separate circuits on the AVR chip which can run independent of the main program, interacting via the control and count registers, and something called timer interrupts.

10.2 How to Use Timer

Since Timer works independently of CPU it can be used to measure time accurately. Timer upon certain conditions takes some action automatically or informs CPU. As we know a timer is an 8 bit register that keeps on increasing its value, so one of the basic conditions is the situation when timer register OVERFLOWS i.e. it has counted up to its maximum value (255 for 8 BIT timers) and rolled back to 0. In this situation timer can issue an interrupt and you must write an Interrupt Service Routine (ISR) to handle the event. There are three different timers available in Atmega16 and all the timers work in almost same way. They are TIMER0, TIMER1 and TIMER2.



10.3 Prescalar

The Prescalar is a mechanism for generating clock for timer by CPU clock. Every CPU has a clock source and the frequency of this source decides the rate at which instructions are executed by the processor. Atmega has clocks of several frequencies such as 1 MHz, 8 MHz, 12 MHz, 16 MHz (max). The Prescalar is used to divide this clock frequency and produce a clock for TIMER. The Prescalarcan be set to produce the following types of clocks:

No Clock(Timer stop)
No prescaling (clock frequency = CPU frequency)
FCPU/8
FCPU/64
FCPU/256
FCPU/1024
External clock, however, it will rarely be used.

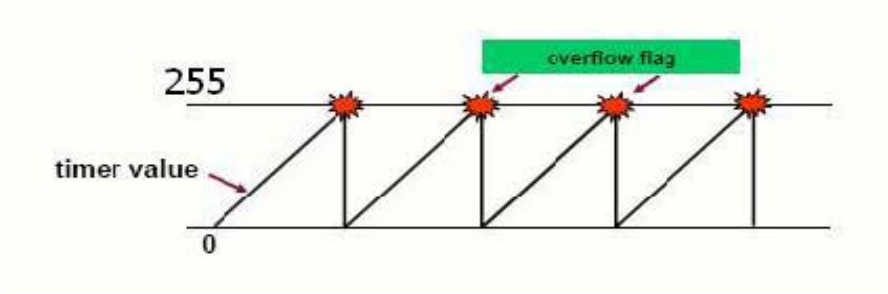
10.4 Timer Mode

Timers are usually used in one of the following modes:

Normal
CTC
Fast PWM
Phase correct PWM

10.5 Normal Mode

A timer running in normal mode will count up to its maximum value. When it reaches this maximum value, it issues an Overflow interrupt and resets the value of the timer to its original value.



In the above case, you can see that the time period is 256 times the time period of the clock. 255 clock cycles are required to attain the maximum value and one clock cycle to clear the timer value.

So, $f_{timer} = f_{clock} / 256$

10.6CTC Mode

Here we shall see how to use a timer in compare mode. In the normal mode, we set the clock of the timer using a prescalar and the let the timer run. When it overflowed, we set up an interrupt to handle the overflow. While simple, this mode has its limitations. We are confined to a very small set of values of frequency for the timer. This limitation is overcome by the compare mode.

Compare mode makes use of a register known as the Output Compare Register which stores a value of our choice. The timer continuously compares its current value with the value on the register and when the two values match, the following events can be configured to happen:

1. A related Output Compare pin can be made set (put to high), cleared (put to low) or toggled automatically. This mode is ideal for generating square waves of different frequency.
2. It can be used to generate PWM signals used to implement a DAC digital to analog converter which can be used to control the speed of DC motors.
3. Simply generate and interrupt and call a handler.

On a compare match, the timer resets itself to 0. This is called CTC – Clear Timer on Compare Match.

In this case, suppose we set our event to toggle the output pin. In that case, the output pin will remain high for one time period of the timer and will remain low for another time period.

So, $t_{out} = 2 * t_{timer}$

From the normal case, we can draw an analogy to find out ttimer.

$t_{timer} = t_{clock} * (OCR + 1)$

So, finally, we have the frequency as,
$f_{out} = f_{clock} / (2 * (OCR + 1))$

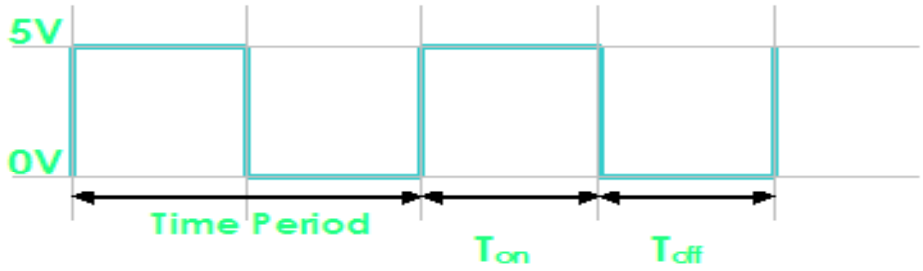
10.7Pulse Width Modulation (PWM) Mode

A digital device like a microcontroller can easily work with inputs and outputs that have only two states - on or off. So you can easily use it to control a LED's state i.e. on or off. In the same way you can use it to turn “on or “off” any electrical device by using proper drivers (transistor, triac, relays etc). But sometimes you need more than just "on" & "off" control over the device. For example, if you want to control the brightness of an LED (or any lamp), or the speed of DC motor, then digital on/off signals will not suffice. This situation is very smartly handled by a technique called **as PWM or Pulse Width Modulation.**

PWM is the technique used to generate analog signals from a digital device like a MCU.

(A) PWM:Pulse Width Modulation

A microcontroller can only generate two levels on its output lines, HIGH=5V and LOW=0V. But what if we want to generate 2.5V or 3.1V or any voltage between 0-5 volt as output? For these requirements, instead of generating a constant DC voltage output we generate a square wave, which has high = 5V and Low = 0V.

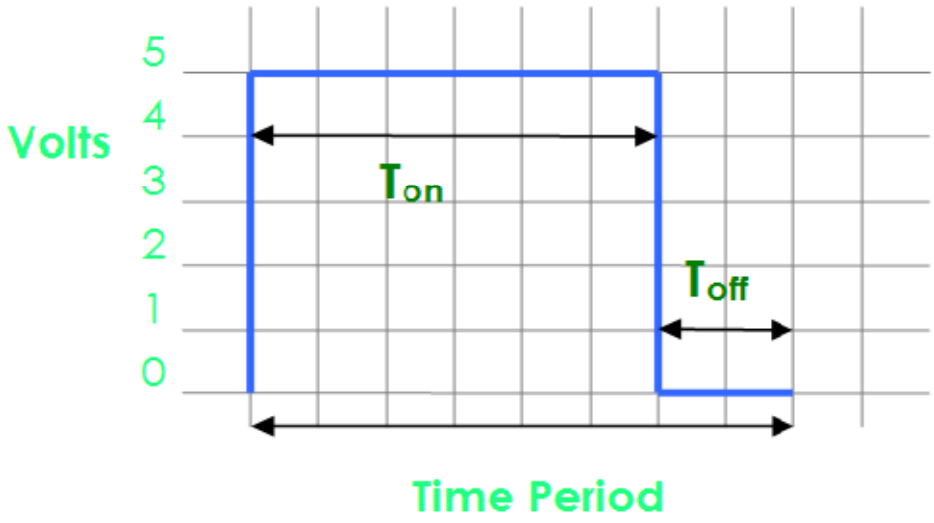


A term called as Duty Cycle is defined as

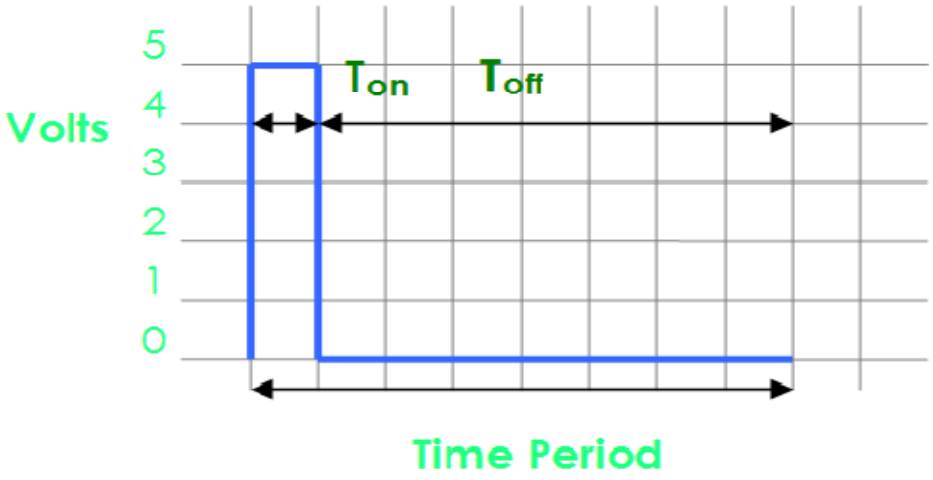
$d = t_{on} / t_{total} * 100\%$

So you can see that the duty cycle in the above case is 50%. If the frequency of such a wave is sufficiently high (say 500 Hz) then the output you get is half of 5V i.e. 2.5 V. Thus if this output is connected to a motor (by means of suitable drivers) it will run at 50% of its full speed at 5V. The PWM technique utilizes this fact to generate any voltage between two extremes (for example between 0-12 volts). The trick is to vary the duty cycle between 0 to 100% and get same percentage of input voltage to output.

Consider the following examples:



Here the duty cycle is 75%. So the equivalent analog voltage output is 3.75V.



Here the duty cycle is 12.5%. So the analog voltage output is 0.625V.

(B) PWM Signal Generation Using Avr Timers

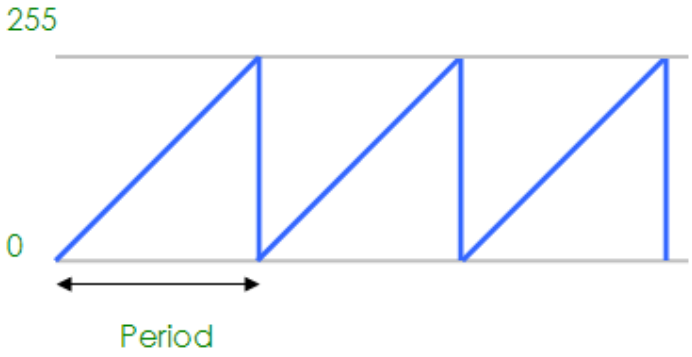
In AVR microcontrollers, PWM signals are generated by timers. There are two methods by which you can generate PWM from timers:

1. Fast PWM

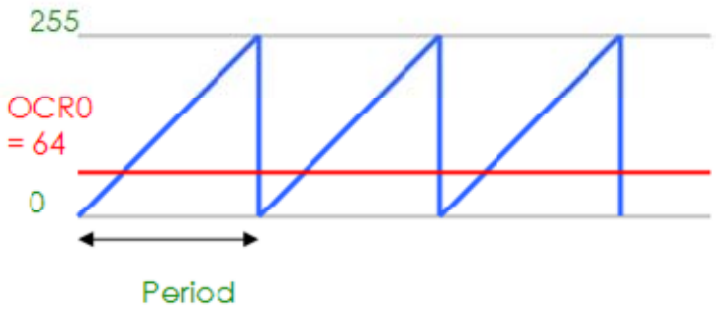
2. Phase Correct PWM

These will be clear as we proceed.

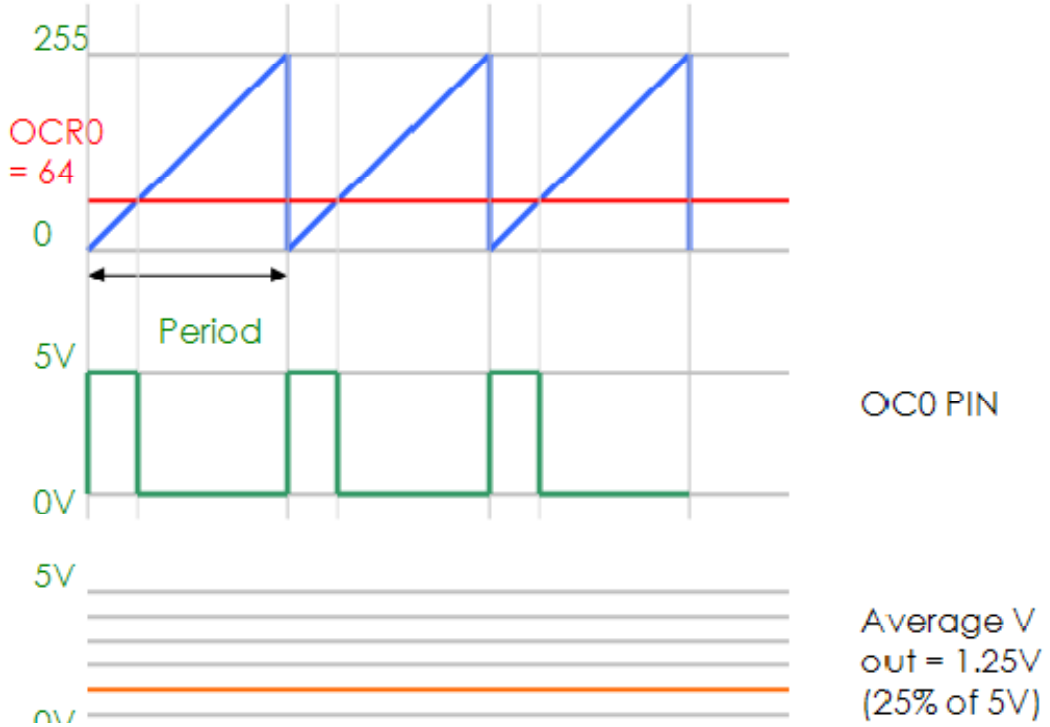
We will use the simplest timer, TIMER0 for PWM generation. So we have an 8 bit counter counting from 0 to 255 and then resetting to 0 and so on. This can be shown on graph as:



The period depends upon the PRESCALAR settings. Now for PWM generation from this count sequence OCR0 (Output Compare Register Zero) is used (Zero because it is for TIMER0 and there are more of these for TIMER1 & TIMER2). We can store any value between 0-255 in OCR0, say we store 64 in OCR0 then it would appear in the graph as follows (the RED line).



When the TIMER0 is configured for fast PWM mode, then, while the timer is counting up, whenever the value of TIMER0 counter matches the value in the OCR0 register, an output PIN is pulled low (0) and when counting sequence begin again from 0 it is SET again (pulled high=VCC). This is shown in the figure 3. This PIN is named OC0 and you can find it in the PIN configuration of ATmega32.



From the figure, you can see that a wave of duty cycle of 64/256 = 25% is produced by setting OCR0 to 64. You can set OCR0 to any value and get a PWM of duty cycle of (OCR0 / 256). When you set it to 0 you get a 0% duty cycle while setting it to 255 will give you a 100% duty cycle output. Thus by varying duty cycle you can get an analog voltage output from the OC0 PIN.

In the inverting mode the value of the OC0 pin is just the reverse of that in the above figure. So whenever the value of the TIMER0 counter is less than OCR0 value then the OC0 pin is LOW else it is HIGH. You can select the inverting or non-inverting mode in the "Output" field in the CodeWizard AVR. The inverting and non-inverting modes will have different duty cycles which are related as

$d_{inv} + d_{non}\text{-}i_{nv} = 100\%$

You can see that

$t_{out} = t_{timer} = 256 * t_{clock}$

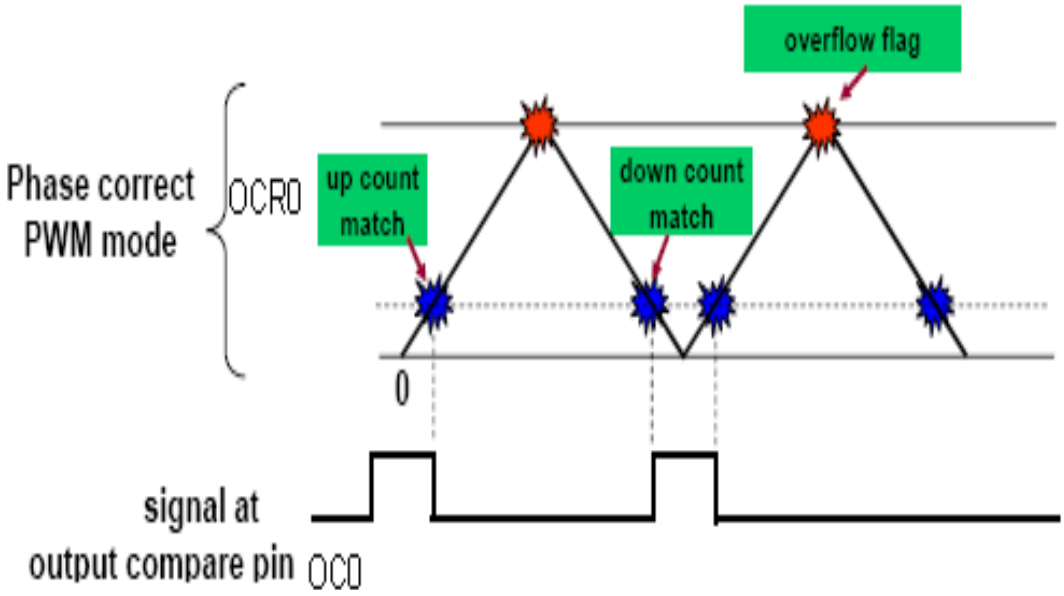
Hence,

$f_{out} = f_{clock} / 256$

(C) Phase Correct PWM Mode

This mode is very similar to the Fast PWM mode except that whenever the value of the timer reaches its maximum value then instead of clearing the value of the timer it simply starts counting down.

The value of the pin toggles only when the value of the OCR0 matches with the TIMER0 counter.



Here,

$t_{out} = t_{timer} = 256*t_{clock}* OCR$

Hence,

$f_{out} = f_{clock} / (2*OCR)$

Overview of Timers in ATmega16

	Timer 0	Timer 1	Timer 2
Overall	- 8-bit counter - 10-bit prescaler	- 16-bit counter - 10-bit prescaler	- 8-bit counter - 10-bit prescaler
Functions	- PWM - Frequency generation Event counter Output compare	- PWM - Frequency generation Event counter Output compare 2 channels - Input capture	- PWM - Frequency generation Event counter Output compare
Operation modes	Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM	Normal mode - Clear timer on compare match - Fast PWM - Phase correct PWM

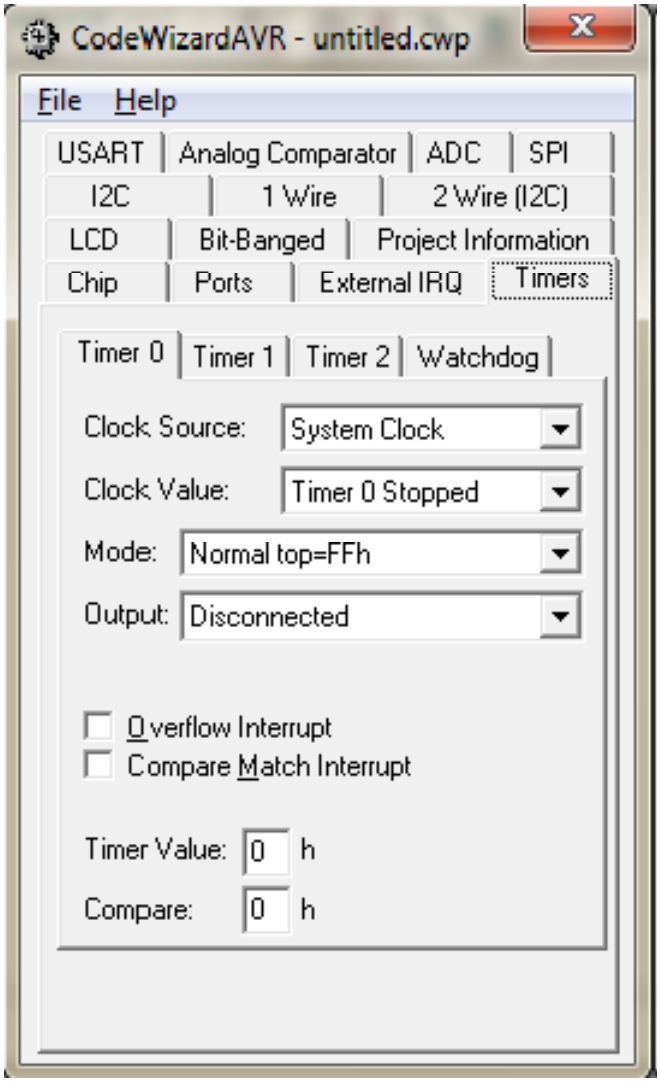
■ Timer 1 has the most capability among the three timers.

Timers can be configured in the CodeWizard AVR window while setting up a new project. To set up a timer, follow these instructions:

- Open CodeVision AVR and click on File -> New.

- The window shown in the above figure will appear. Click on “Yes”.
- On the CodeWizard AVR window, select your chip and frequency.
- Click on the “Timers” tab on the top.

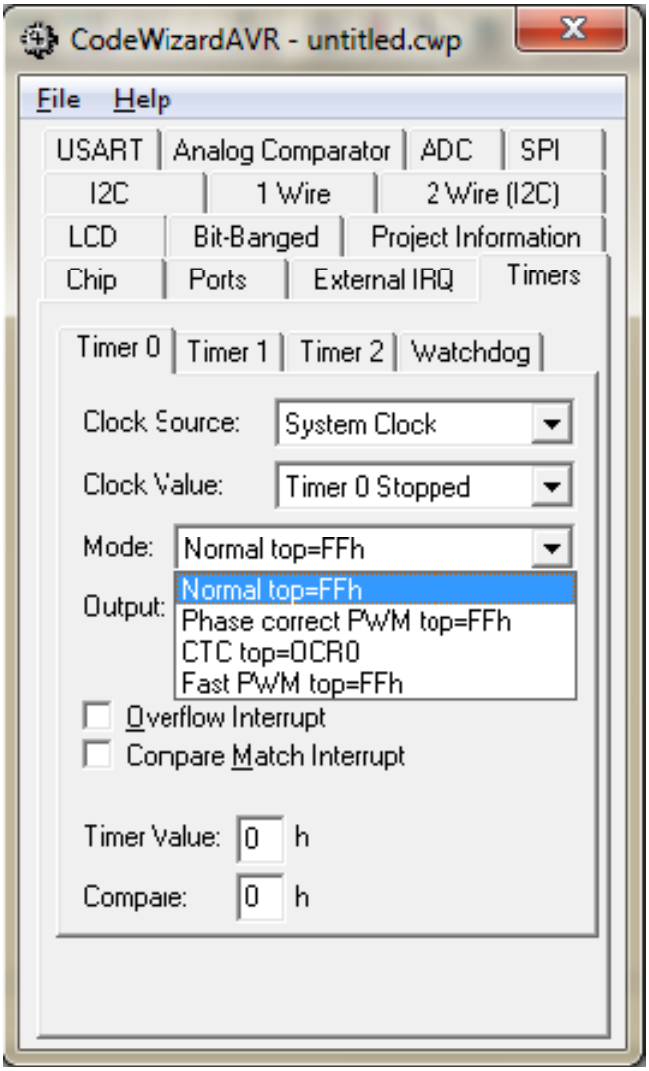
10.8 SETTING UP TIMERS IN CODEVISION AVR

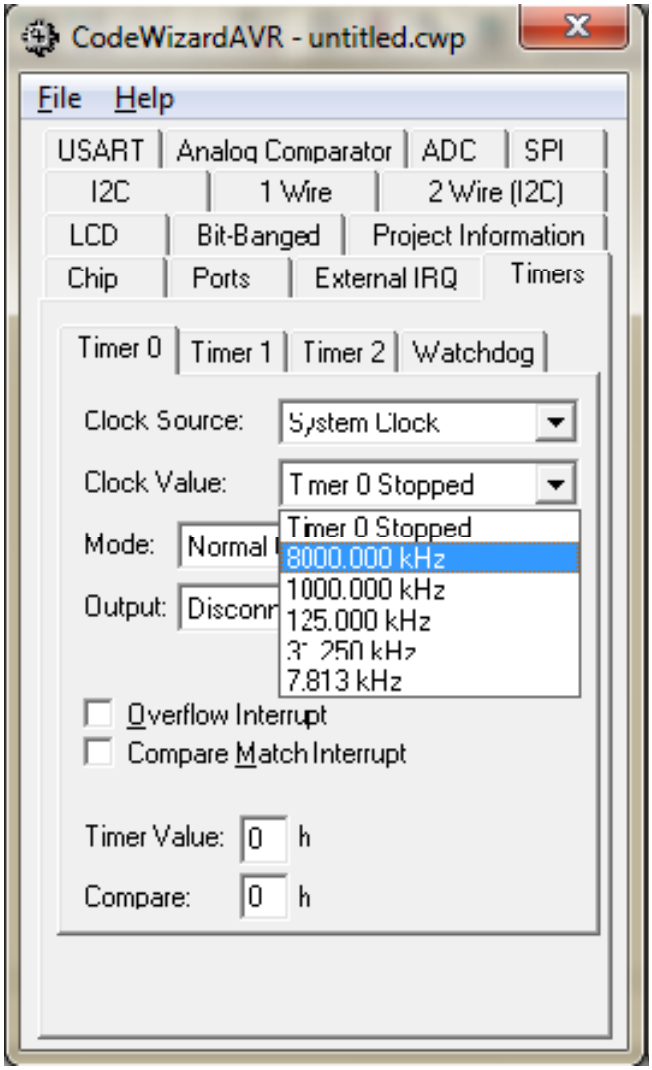


The window will allow you to configure TIMER0, TIMER1, TIMER2. For now, you can ignore the Watchdog timer.

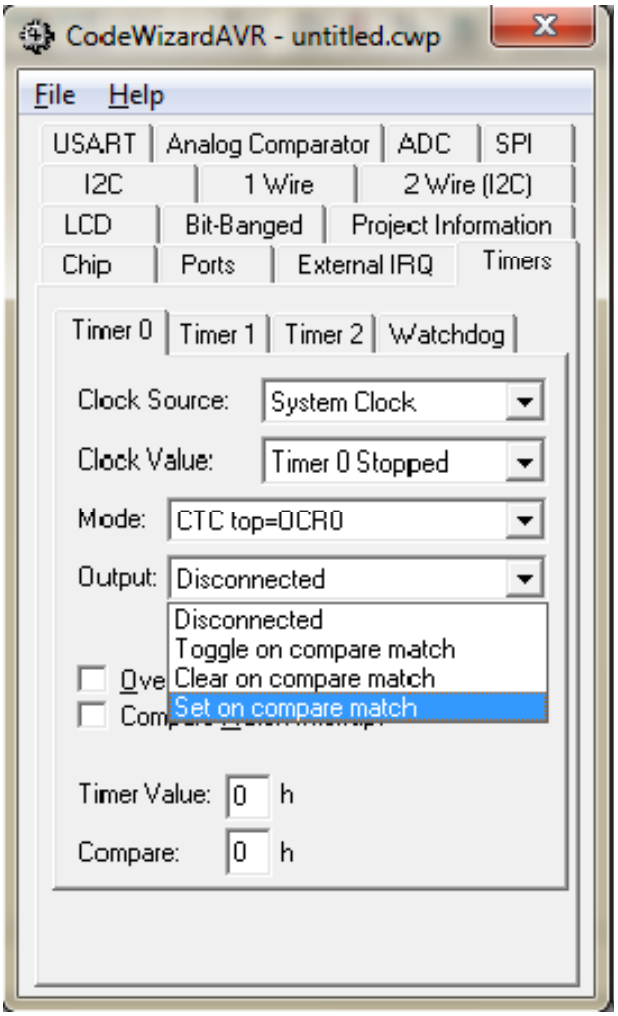
- The “Clock Source” menu will allow you to select the clock source for the current timer – you can either select the system clock or an external clock. For most purposes, you can select “System Clock”.
- The “Clock Value” menu will allow you to set the frequency of the clock of the timer. Select the appropriate value:

- Select the timer mode in the “Mode” menu:





In the “Output” menu, select the appropriate option for your timer. This menu will have different options depending upon your timer mode, eg. Toggle, Set, Reset and Inverting, Non-Inverting, etc.:



- You can select to enable Overflow Interrupt and Compare Match Interrupt by selecting the appropriate checkboxes.
- The “Timer Value” option will allow you to select the starting value of the timer. The default value is 0.
- The “Compare Value” option will allow you to select the value for OCR. This value is required for CTC and PWM modes as well as Compare Match Interrupt. Enter the value in hexadecimal numbers.

Essentially, you’re done configuring your timer. You can now configure the remaining settings of the timer and click on File -> Generate, Save and Exit. After saving the files, you can configure your Interrupt handlers in the code window, if you activated any.

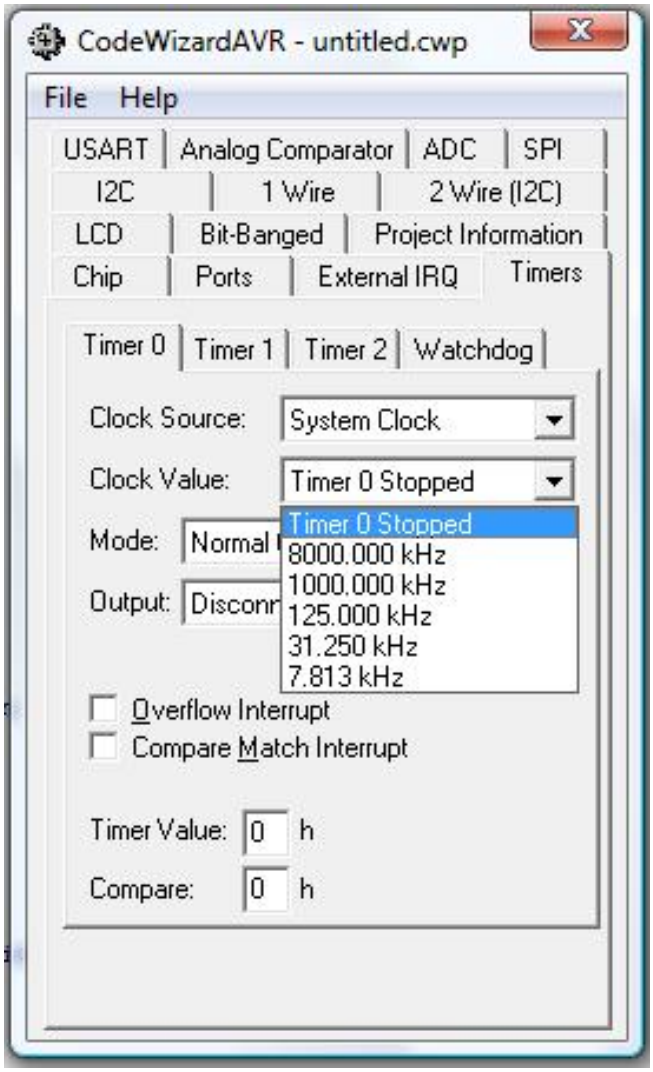
Atmega 16 has following timers,

- Timer 0, 8 bit
    - Timer 1, 16 bit consisting of two 8 bit parts, A and B
    - Timer 2, 8 bit

Now there are two clocks,

1. **System Clock (fs):** This is the clock frequency at which Atmega is running. By default it is 1 MHz which can be changed by setting fuse bits.

2. **Timer Clock (ft):** This is the clock frequency at which timer module is running. Each timer module has different clocks.



Now **ft** can be in ratios of **fs.** That is, ft = fs, fs/8, fs/64 ...

For example, if we keep fs = 8 MHz then available options for ft are: 8 MHz, 1 MHz, 125 KHz ... (look in the image)

There are total 4 settings for Timers,

1. **Clock Source:** Source for timer clock, keep it as system clock. You can also provide external clock. Read datasheet for more information about external clock source.

2. **Clock value:** This is value of **ft.** Drop down for available options of ft. Chose whichever is required

3. **Mode:** There are many modes of timers. We will be discussing following 2 modes,

    a. Fast PWM top = FFh
    b. CTC top=OCRx (x=0, 1A, 2)

4. **Output**: Depending upon the mode we have chosen there are options for output pulse. We will look in detail later.

Basically each Timer has a counter unit with size 8 bit for **Timer 0, 2** and 16 bit for **Timer 1.**I will be talking about Timer 0 and same will follow for other Timers.

Each counter has a register which increments by one on every rising edge of timer clock. After counting to its full capacity, 255 for 8 bit, it again starts from 0. By using this register we can have different modes.

- Timer/Counter (**TCNT0**) and Output Compare Register (**OCR0**) are 8-bit registers.

- **TOP:** The counter reaches the TOP when it becomes equal to the highest value in the count sequence. The TOP value can be assigned to be the fixed value 0xFF (MAX) or the value stored in the OCR0 Register. The assignment is dependent on the mode of operation.

10.9 Fast PWM Mode

PWM = Pulse Width Modulation.

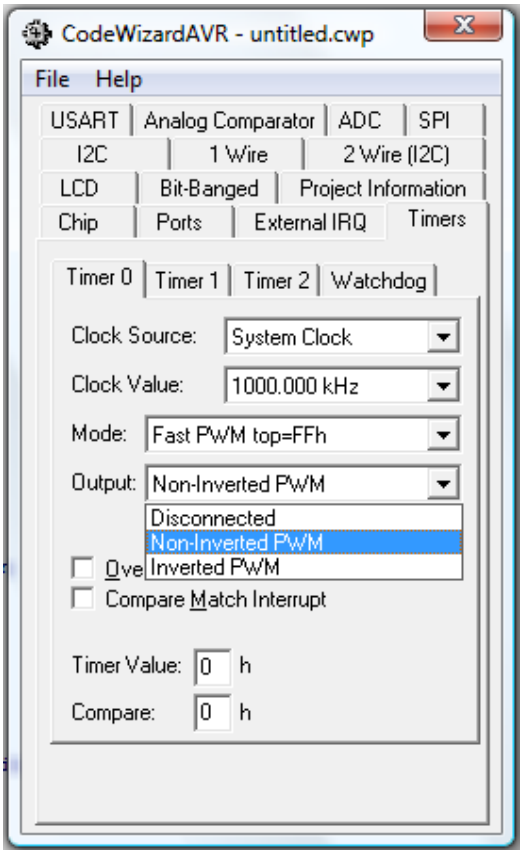
This mode is used to generate pulse with

- Fixed Frequency (F)
- Variable Duty Cycle (D)

F = Ft / 256

D = OCR0 / 255 **(non inverted)**

D = (255-OCR0) / 255 **(inverted)**



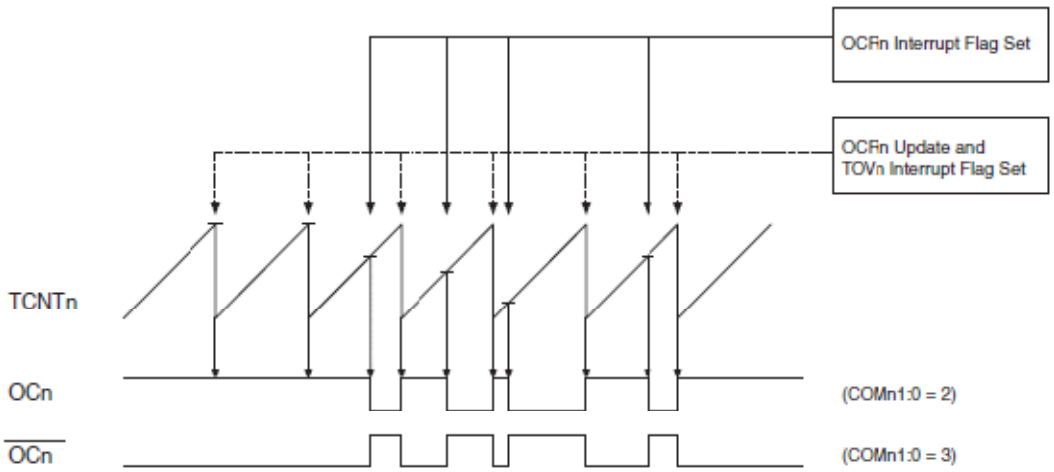
By changing OCR0 value we can change the duty cycle of the output pulse.

As OCR0 is an 8bit register it can vary from 0 o 255.

$$0 \leq OCR0 \leq 255$$

Pins for Output pulse,

Timer 0 : OC0, pin 4
Timer 1A : OC1A, pin 19
Timer 1B : OC1B, pin 18
Timer 2 : OC2, pin 21



10.10 CTC Mode

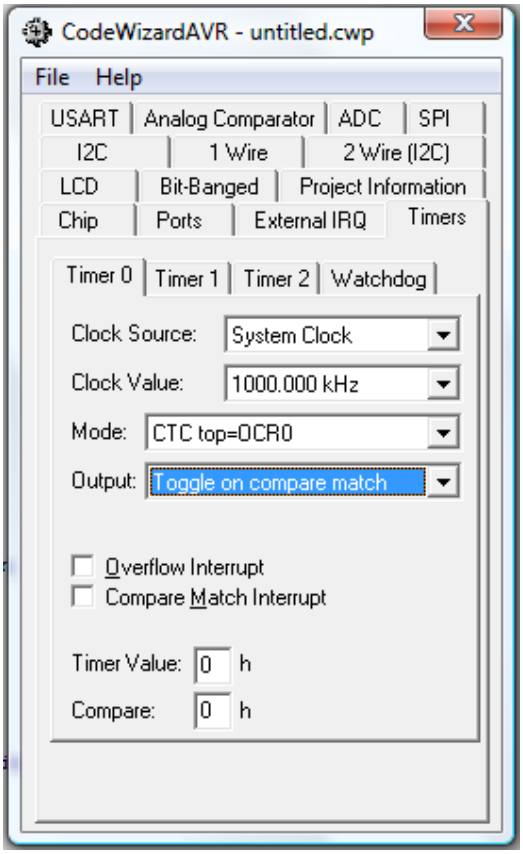
CTC = Clear Timer on Compare Match.

This mode is to generate pulse with,

- Fixed Duty Cycle (D = 0.5)
- Variable Frequency (F)

$$F = \frac{ft}{2\,(OCR0+1)}$$

D= 0.5



By changing value of OCR0 we can change the value of output pulse frequency. As OCR0 is an 8bit register it can vary from 0 to 255.

$$0 \le OCR0 \le 255$$

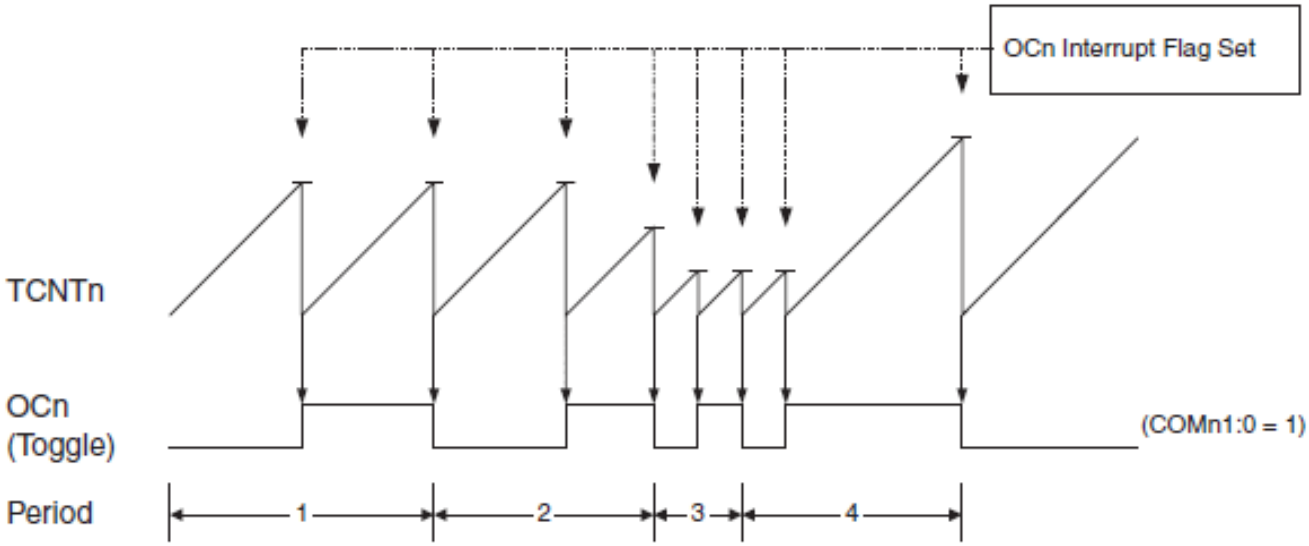


Figure 17: CTC Mode timing diagram

Chapter 11 Communication

11.1 Data Transfer

Knowledge of data transfer is very important for any embedded system developer. In any embed ed system data is moved between several units like between RAM and CPU. There are many methods and technique for data transfers each having its own pros and cons. So different data transfer technique is used in different situations. Some example of data transfer is

- ***Simple parallel transfer***-Used to transfer 8, 16, and 32 ... bits of data in the same time.
- ***Asynchronous Serial Transfer (USART)*** –It is an old but still in use mode of serial communication uses only 2 lines (+1 additional line for GND).
- ***SPI*** - *Serial Peripheral Interface* - It is a standard mode of communication between different ICs.
- ***USB*** - A very advance, high speed and complicated serial Bus used in PCs to connect almost anything to it.

11.2 Classification

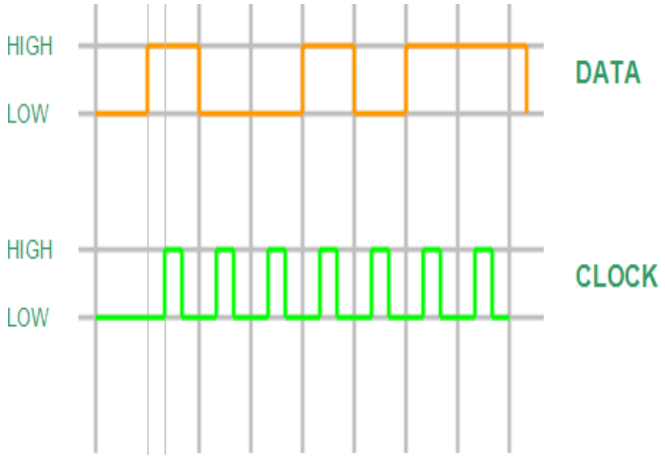
Modes of Data Transfer can be broadly divided into two types:

1. PARALLEL TRANSFER - In this mode a number of bits (say 8,16 or 32) are transferred at a time. Thus they require as many electrical line as the number of bits to be transferred at once. This method is fast but its disadvantage is that it uses more number of lines. So they are basically used when the units involved in data transfer are physically close and almost fixed with each other for a long time. For eg. CPU and RAM, the PCI Cards inside the PC. These are close to each other and packed inside the CPU box and are disconnected from each other less frequently.
2. SERIAL TRANSFER - In this mode only one bit is transfer at once. So to transfer 8 bits, 8 cycles are required. So these require less number of physical lines (like SPI use 3 lines). The advantage is that due to less number ICs using these technologies are small with low PIN count.

Modes of Data Transfer can also be divided into

1. SYNCHRONOUS TRANSMISSION-. In this type the actual data is transferred BIT by BIT on the DATA line. The clock line signals the end of 1 bit and the start of another bit. When the clock line changes its level, that is when it goes HIGH from a LOW level or vice versa the data is transferred. When the CLOCK line goes HIGH it signals that a new bit is available for transfer. The "other" device which is receiving the data reads the data line at the rising edge or the falling edge of the clock depending upon our settings.

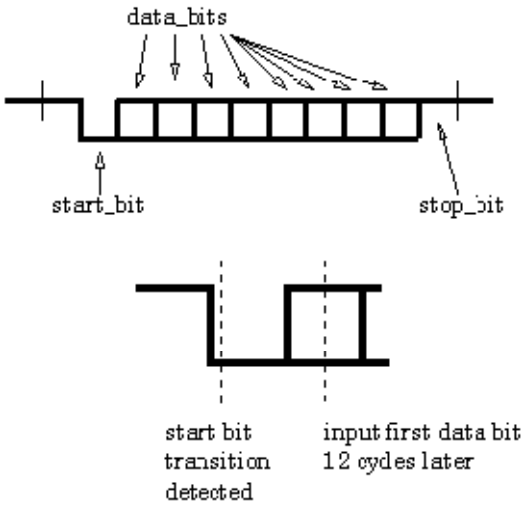
The diagram corresponds to the transfer of the data 10010111. It corresponds to the value of the data at every rising edge of the clock.



2. ASYNCHRONOUS TRANSMISSION - Asynchronous transmission allows data to be transmitted without the sender having to send a clock signal to the receiver. Instead, the sender and receiver must agree on timing parameters in advance and special bits are added to each word which are used to synchronize the sending and receiving units. When a word is given for Asynchronous transmissions, a bit called the "Start Bit" is added to the beginning of each word that is to be transmitted. The Start Bit is used to alert the receiver that a word of data is about to be sent, and to force the clock in the receiver into synchronization with the clock in the transmitter. One solution is to have both devices share the same clock source.

11.3 Baud Rate

Baud Rate is a measurement of transmission speed in asynchronous communication. The devices that allow communication must all agree on a single speed of information - 'bits per second'.



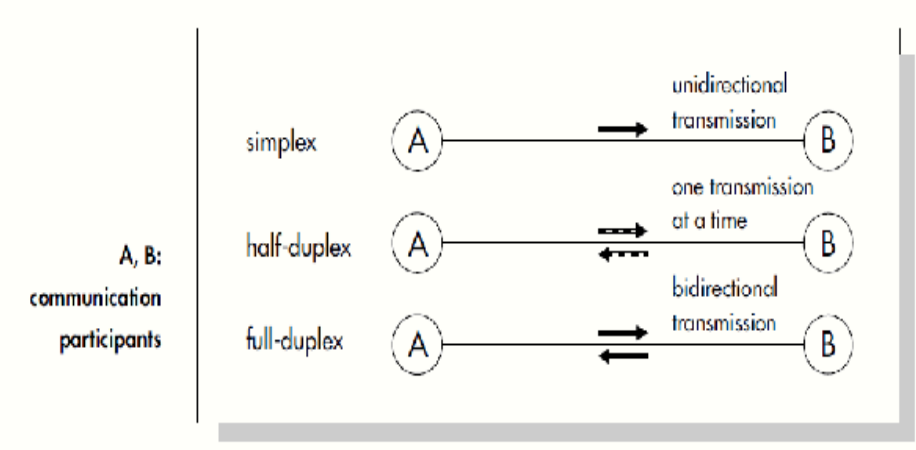
When the entire data word has been sent, the transmitter may add a Parity Bit that the transmitter generates. The Parity Bit may be used by the receiver to perform simple error checking. Then at least one Stop Bit is sent by the transmitter.

When the receiver has received all of the bits in the data word, it may check for the Parity Bits (both sender and receiver must agree on whether a Parity Bit is to be used), and then the receiver looks for a Stop Bit.

In short, asynchronous data is 'self synchronizing'.

Transmission	*Advantages*	*Disadvantages*
Asynchronous	Simple & Inexpensive	High Overhead
Synchronous	Efficient	Complex and Expensive

11.4 Different Communication Techniques



Chapter 12 SPI: Serial Peripheral Interface

12.1 Theory of Operation

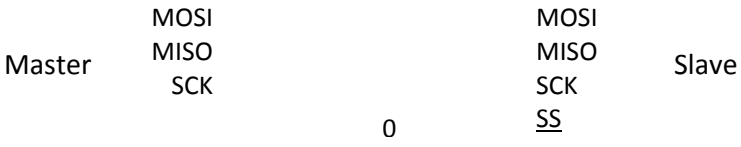
The Serial Peripheral Interface Bus or SPI bus is a synchronous serial data link used to communicate between two or more microcontroller and devices supporting SPI mode data transfer. Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select (chip select) lines.

This communication protocol consists of folliwng lines or pins,

1. **MOSI** : **M**aster **O**ut **S**lave **I**n (Tx for Master and Rx for Slave)
2. **MISO** : **M**aster **I**n **S**lave **O**ut (Rx for Master Tx for Slave)
3. **SCK** : **S**erial **C**lo**c**k (Clock line)
4. **SS**: **S**lave **S**elect (To select Slave chip) (if given 0 device acts as slave)

Master: This device provides the serial clock to the other device for data transfer. As a clock is used for the data transfer, this protocol is Synchronous in nature. **SS** for Master will be disconnected.

Slave: This device accepts the clock from master device. **SS**for this has to be made 0 externally.



Pin connections for SPI protocol

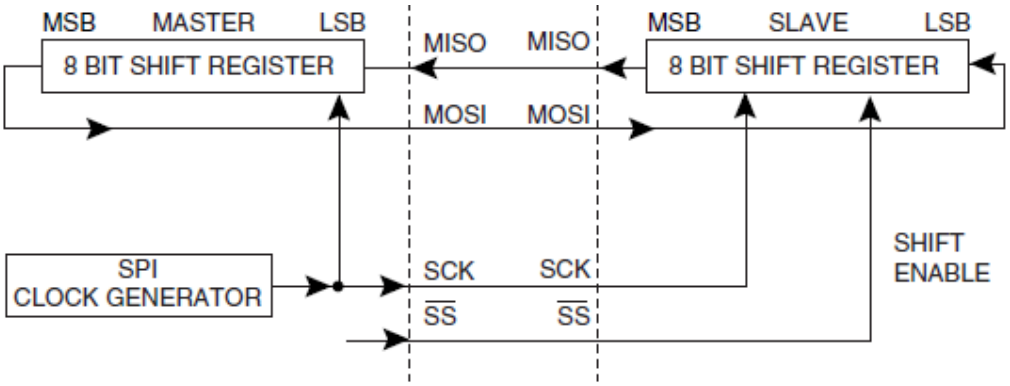
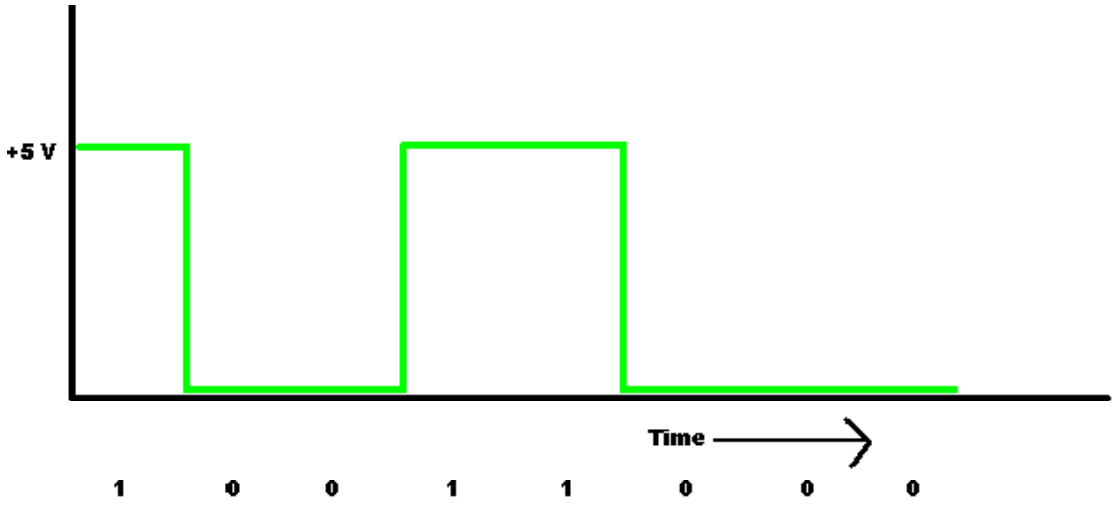


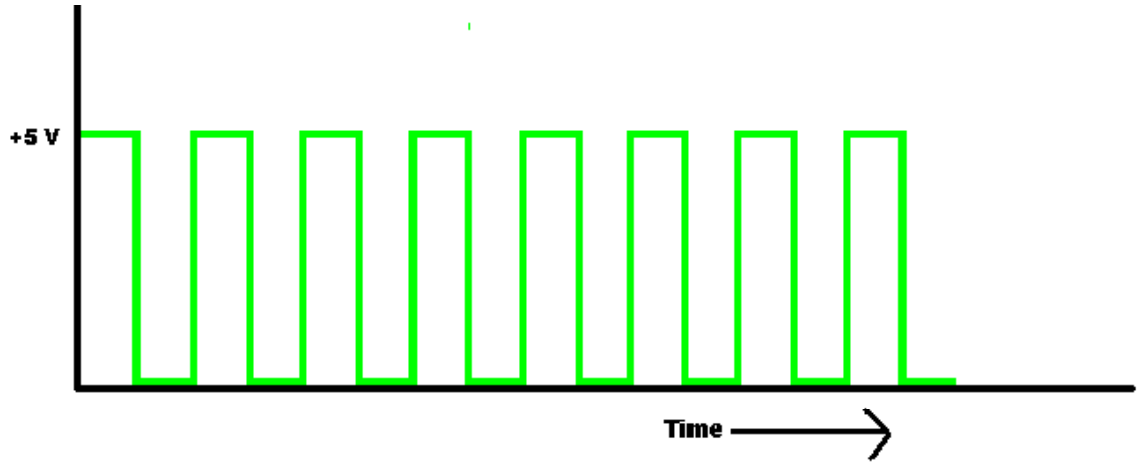
Figure 18: SPI Data Communication

In SPI, data is transmitted serially, i.e. bit by bit as opposed to parallel communication where all the data is sent multiple bits at a time. We will study synchronous SPI, where there is a clock generated and the data is transferred at the rate of the clock pulse. What is clock pulse?

Clock pulse is basically a sequence of alternating 0s and 1s that is used to indicate that one Bit of data has been sent. For example, if you were to send the data 10011000, the signal you sent would look like:



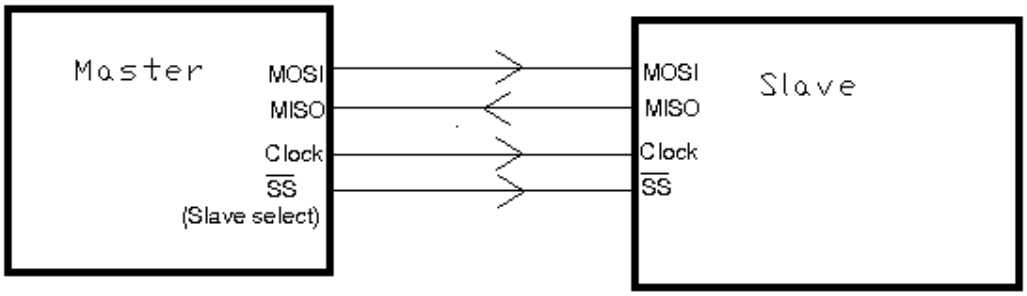
Now as it might be clear there should be some way to tell this signal from 1010 of 100010 of some other. For this, we can do either of two things, set a standard that at a particular rate, the data will be transferred and we keep checking for voltage at time moments T, 2T, 3T so on, or we also send a clock pulse at the same time. Like we decide that whenever we send a new signal, we will make the clock 1. Then the clock signal would look like:



Now if the receiving machine reads the incoming data at negative edges, it will always read what the sender actually meant. This the concept of synchronous data transfers. The sender and the receiver are synchronized by a clock pulse.

Now, in SPI set up when we have to set up communication between two systems, first we have make one system as master and the other as slave. The difference between master and the slave is that the clock pulse is generated by the master and both master and the slave agree to work on the clock frequency that is set up by the master.

The basic connections in any SPI set up are as follows:



MOSI is Master Out Slave In, so it is the channel where the master sends the data to the slave and the slave receives it.

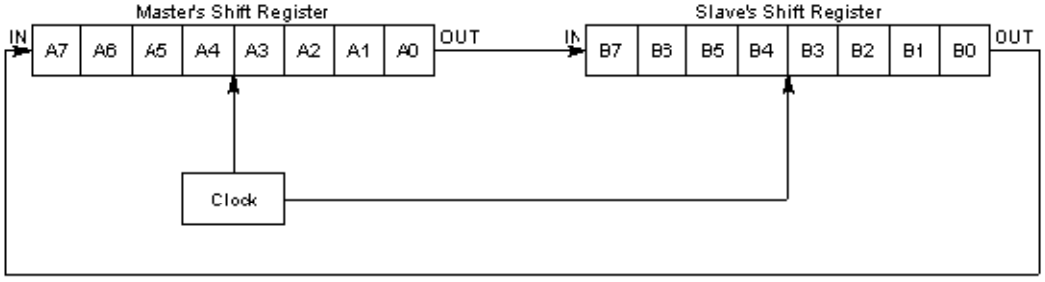
MISO is Master In Slave Out, so it is the channel where the slave sends the data and the master receives it.

Clock is the clock that is send by the master.

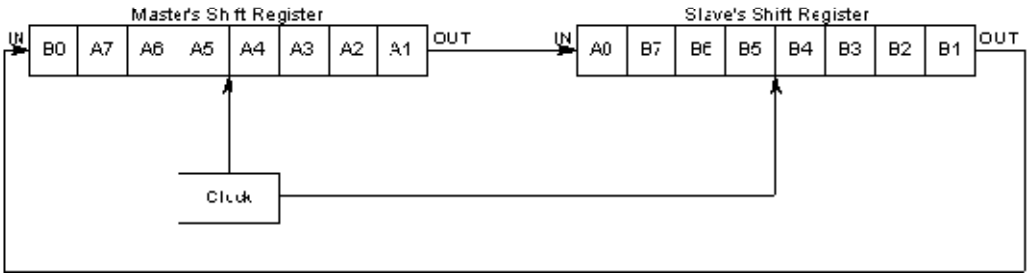
SS is slave select when the master wants to send data to a particular slave it makes its SS pin low, sends the data and then again makes it high. It is especially useful when multiple slaves are connected to the master, but the basic purpose is to select the slave and transmitting data to it.

In SPI, the data the data is exchanged between the transmitter and the receiver. It happens as follows:

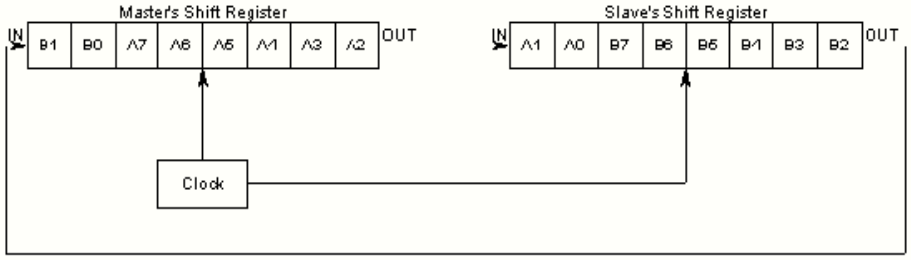
First the master and the slave keep the data bits to transfer in their respective registers. Suppose master wants to send bm1,bm2,bm3…..bm8 and the slave wants to send bs1, b s2, b s3 …..b s8. What happens is as follows:



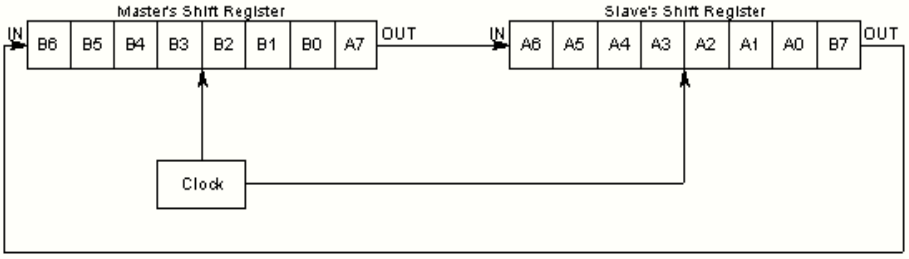
Master generates the first clock pulse:



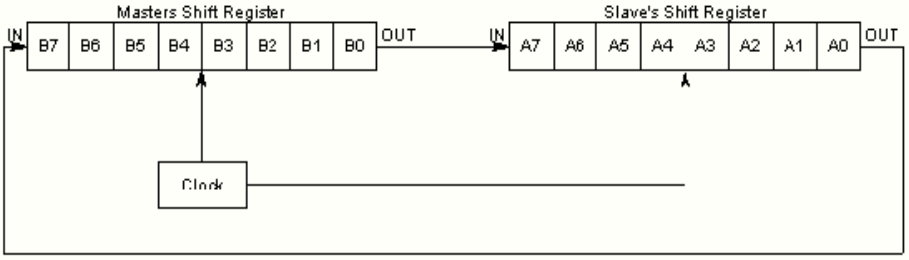
Master generates the second clock pulse:

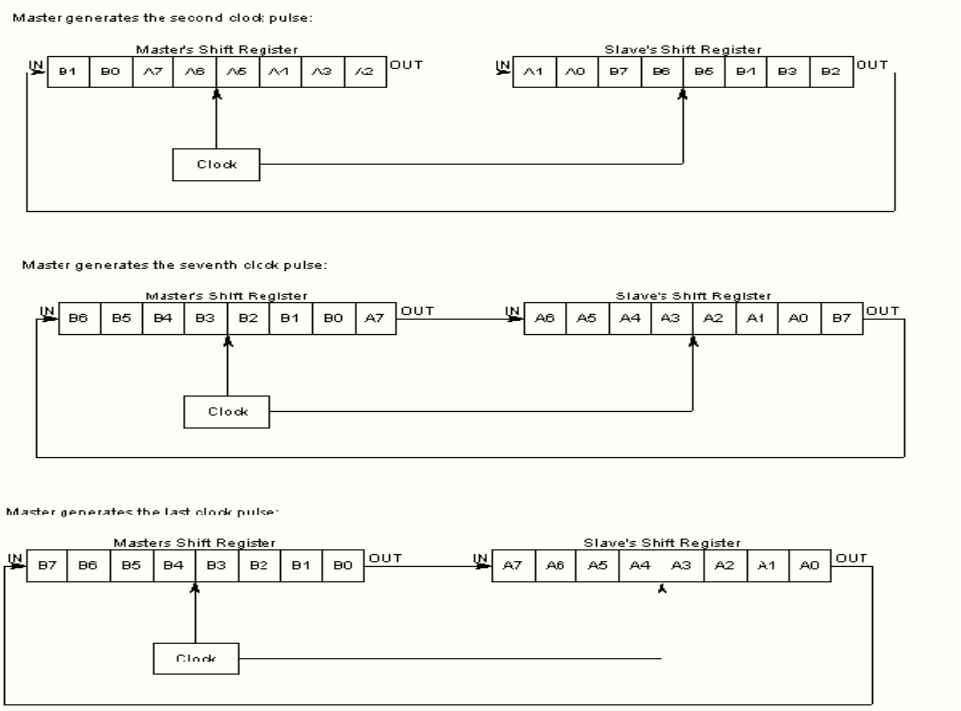


Master generates the seventh clock pulse:



Master generates the last clock pulse:

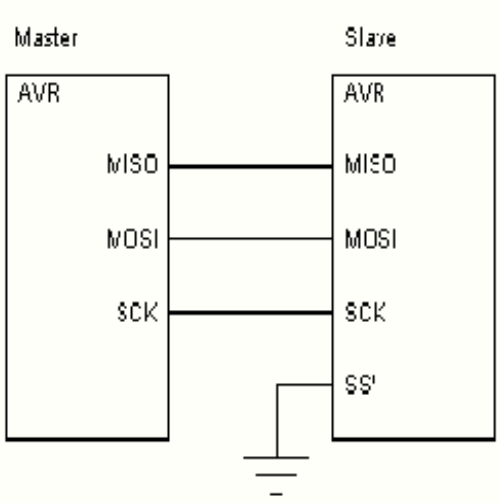




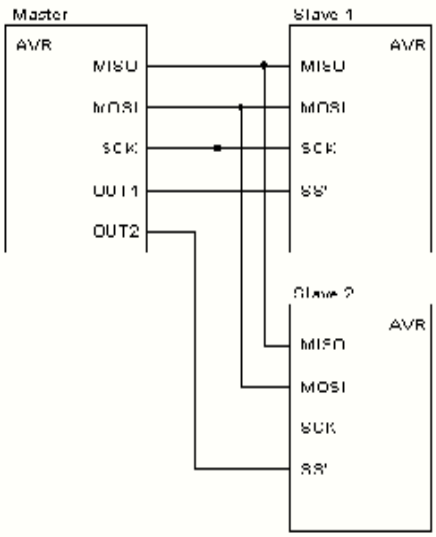
The data is transferred one bit at a time between the master and the slave, and at the end of 8 cycles the data is completely exchanged.

The following figures show a typical setup used with SPI:

One master and one slave



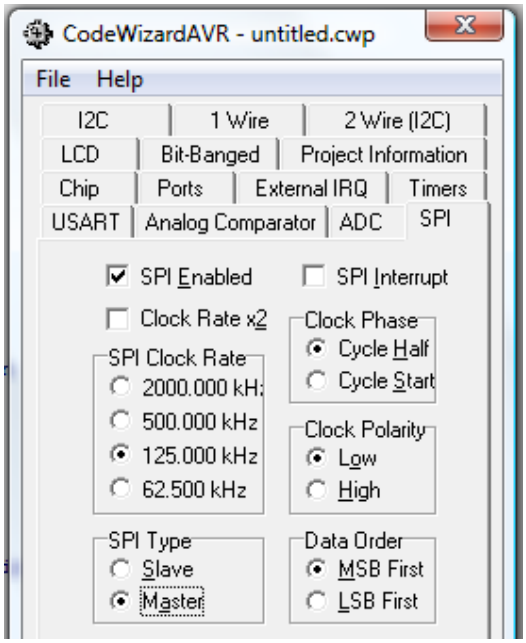
One master and two slaves



HOW TO USE CODE WIZARD TO SET UP SPI CONNECTION

12.2 Setting up SPI in Microcontroller

12.2.1 Master Microcontroller

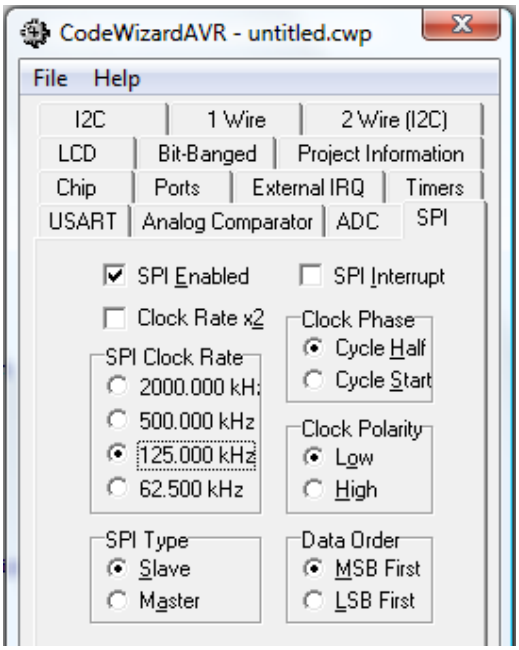


Open Code Wizard and go to **SPI** tab. **Enable SPI** and choose Master in SPI Type.

Select clock rate depending upon your data transfer speed requirement.

Keep other settings as default.

12.2.2 Slave Microcontroller



Open Code Wizard and go to **SPI** tab. **Enable SPI** and choose Slave in SPI Type.

Select clock rate depending upon your data transfer speed requirement.

Keep other settings as kept in the master microcontroller.

12.3 Data Functions

You can transmit or receive 1 byte of data at a time.

12.3.1 Transmit Data

spi(1 byte data); Example: spi('A');

12.3.2 Receive Data

c = spi(0);// c is 1 byte variable Example: char c = spi(0);

12.4 Connecting MCU To Another MCU

Just connect the MOSI, MISO, SCK, and SS to SS. Use the following function to send data: spi(character)

When you use this function in the master, it writes the character to the register and sends the data to slave. In case of the slave, the data is written to the register and the cpu waits for the master to send data when it also transmits the data written into the register.

SAMPLE PROGRAMME:

MASTER:

```
char a=spi(0xFF);
lcd_putchar(a);//displays 1 on the lcd
```

SLAVE:

```
char b=spi('1');
```

Chapter 13 USART Communication

UART (Universal Asynchronous Receiver Transmitter) is a way of communication between the microcontroller and the computer system or another microcontroller. There are always two parts to any mode of communication-a Receiver and a Transmitter. Hence, our Atmega can receive data as well as send data to other microcontroller, computer or any other device.

13.1 USART

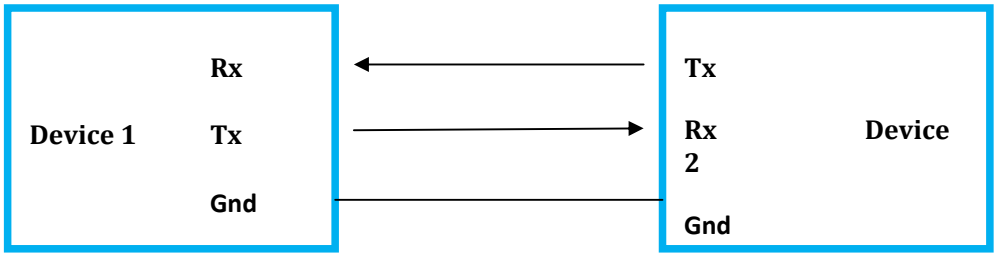
Like many microcontrollers, AVR also has a dedicated hardware for serial communication. This part is called the USART - Universal Synchronous Asynchronous Receiver Transmitter. This special hardware makes your life as a programmer easier. You just have to supply the data you need to transmit and it will do the rest. The advantage of hardware USART is that you just need to write the data to one of the registers of USART and your done, you are free to do other things while USART is transmitting the byte.
Also the USART automatically senses the start of transmission of RX line and then inputs the whole byte and when it has the byte it informs you (CPU) to read that data from one of its registers.

The USART of AVR is very versatile and can be setup for various different modes as required by your application. In this tutorial we will show you how to configure the USART in a most common configuration and simply send and receive data.

13.2 Hardware Aspect of USART

USART consists of only three connections – Rx, Tx and GND. Rx means Receive and Tx means Transmit. The GND connection is for a common reference level.

Here's a simple diagram explaining the connections:



Notice how Tx is connected to Rx, and Rx is connected to Tx.

13.3 Baud Rate

Baud is a measurement of transmission speed in asynchronous communication. The computer, any adaptors, and the UART must all agree on a single speed of information - 'bits per second'.

13.4 Data Transmission

In Asynchronous mode, data is transmitted in frames. Each frame has a start bit, data bits, optional parity bit, and stop bits.

A start bit signals the beginning of data transmission. Data bits are the actual data to be transmitted. Stop bits signal the end of transmission.

An optional parity bit can be transmitted before the stop bits. This bit represents the number of "logical highs" in the transmission. If there are odd numbers of logical highs in the transmission,then the parity bit has a logical high value. If there are even numbers, then the parity bit takes the value logical low. Parity bits are generally used in error detection.

13.5 UART: Theory of Operation

Figure 16 illustrates a basic UART data packet. While no data is being transmitted, logic 1 must be placed in the Tx line. A data packet is composed of 1 start bit, which is always a logic 0, followed by a programmable number of data bits (typically between 6 to 8), an optional parity bit, and a programmable number of stop bits (typically 1). The stop bit must always be logic 1.

Most UART uses 8bits for data, no parity and 1 stop bit. Thus, it takes 10 bits to transmit a byte of data.

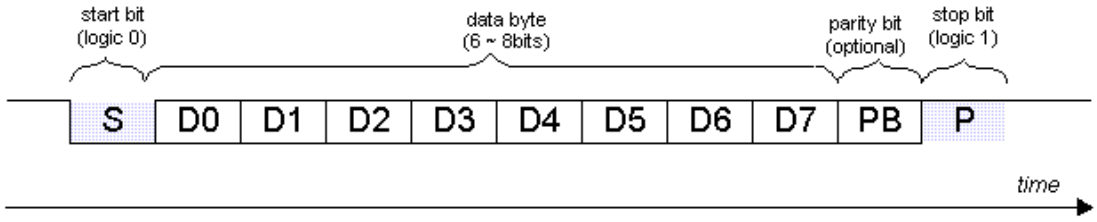


Figure 19: Basic UART packet format: 1 start bit, 8 data bits, 1 parity bit, 1 stop bit.

BAUD Rate: This parameter specifies the desired baud rate (bits per second) of the UART. Most typical standard baud rates are: 300, 1200, 2400, 9600, 19200, etc. However, any baud rate can be used. This parameter affects both the receiver and the transmitter. The default is 2400 (bauds).

In the UART protocol, the transmitter and the receiver do not share a clock signal. That is, a clock signal does not emanate from one UART transmitter to the other UART receiver. Due to this reason the protocol is said to be **asynchronous**.

Since no common clock is shared, a known data transfer rate (baud rate) must be agreed upon prior to data transmission. That is, the receiving UART needs to know the transmitting UART's baud rate (and conversely the transmitter needs to know the receiver's baud rate, if any). In almost all cases the receiving and transmitting baud rates are the same. The transmitter shifts out the data starting with the LSB first.

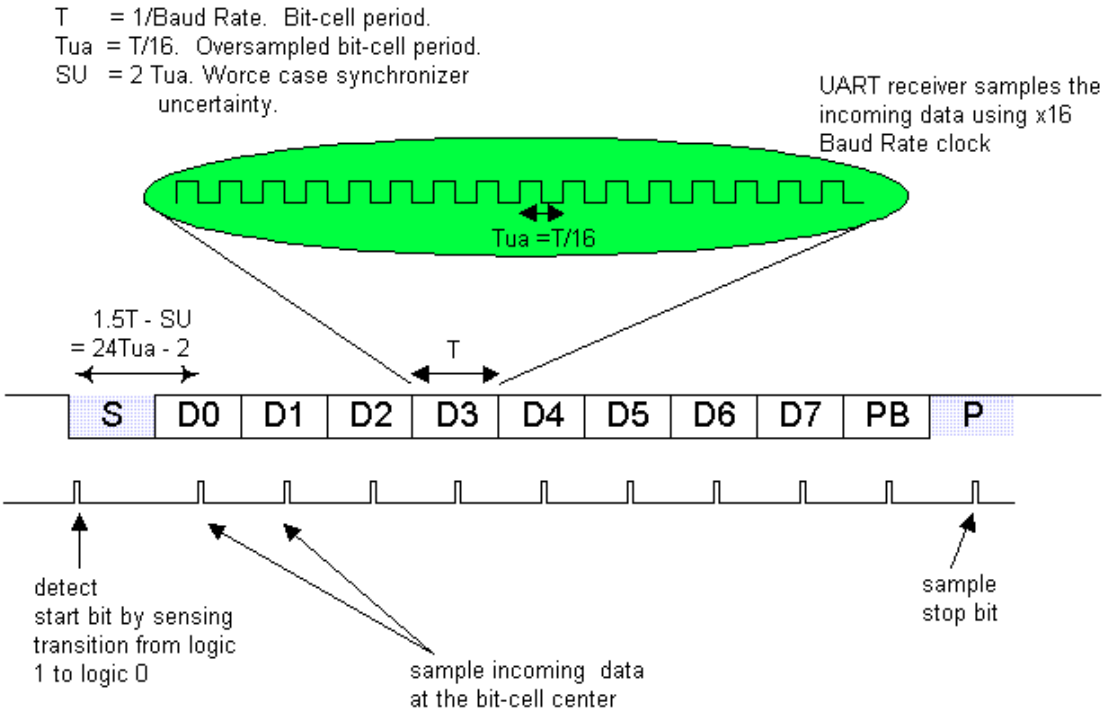
Once the baud rate has been established (prior to initial communication), both the transmitter and the receiver's internal clock is set to the same frequency (though not the same phase). The receiver "synchronizes" its internal clock to that of the transmitter's at the beginning of every data packet received. This allows the receiver to sample the data bit at the bit-cell center.

A key concept in UART design is that UART's internal clock runs at much faster rate than the baud rate. For example, the popular 16450 UART controller runs its internal clock at 16 times the baud rate. This allows the UART receiver to sample the incoming data with granularity of 1/16 the baud-rate period. This "oversampling" is critical since the receiver adds about 2 clock-ticks in the input data synchronizer uncertainty. The incoming data is not sampled directly by the receiver, but goes through

a synchronizer which translates the clock domain from the transmitter's to that of the receiver. Additionally, the greater the granularity, the receiver has greater immunity with the baud rate error.

The receiver detects the start bit by detecting the transition from logic 1 to logic 0 (note that while the data line is idle, the logic level is high). In the case of 16450 UART, once the start-bit is detected, the next data bit's "center" can be assured to be 24 ticks minus 2 (worse case synchronizer uncertainty) later. From then on, every next data bit center is 16 clock ticks later. Figure 2 illustrates this point.

Once the start bit is detected, the subsequent data bits are assembled in a de-serializer. Error condition maybe generated if the parity/stop bits are incorrect or missing.



13.6 Serial Port of Computer

We will be using Serial Port for communication between the uC and the computer. A serial port has 9 pins as shown. If you have a laptop, then most probably there won't be a serial port. Then you can use a USB to serial Converter.

If you have to transmit one byte of data, the serial port will transmit 8 bits as one bit at a time. The advantage is that a serial port needs only one wire to transmit the 8 bits.

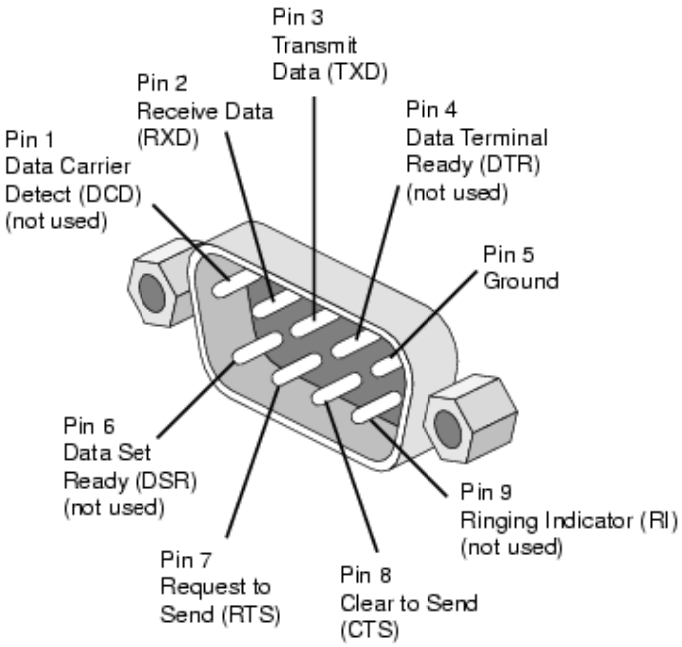
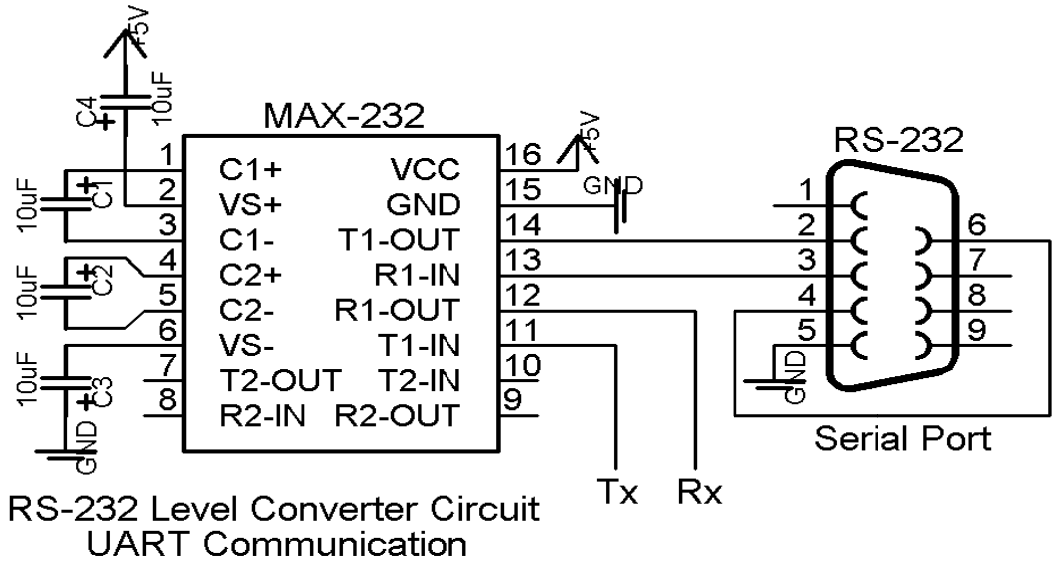


Figure 20: A Serial Port

Pin 3 is the Transmit (TX) pin, pin 2 is the Receive (RX) pin and pin 5 is Ground pin. Other pins are used for controlling data communication in case of a modem. For the purpose of data transmission, only the pins 3 and 5 are required.

The standard used for serial communication is **RS-232** (Recommended Standard 232). The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels. Valid signals are plus or minus 3 to 15 volts. The range near zero volts is not a valid RS-232 level; logic one is defined as a negative voltage, the signal condition is called marking, and has the functional significance of OFF. Logic zero is positive; the signal condition is spacing, and has the function ON.

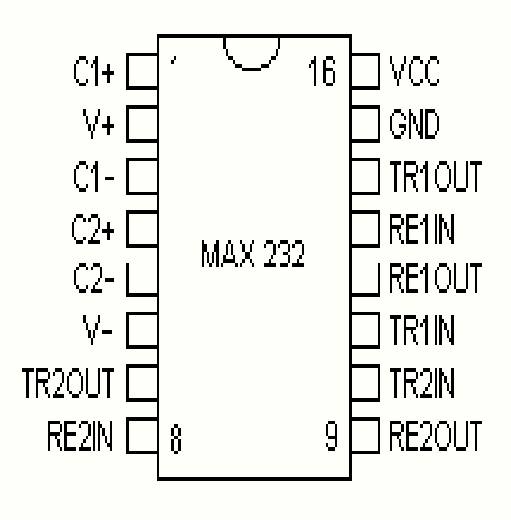
Now we know that this is not the voltage level at which our microcontroller works. Hence, we need a device which can convert this voltage level to that of CMOS, i.e., logic 1 = +5V and logic 0 = 0V. This task is carried out by an IC MAX 232, which is always used with four 10uF capacitors. The circuit is shown:



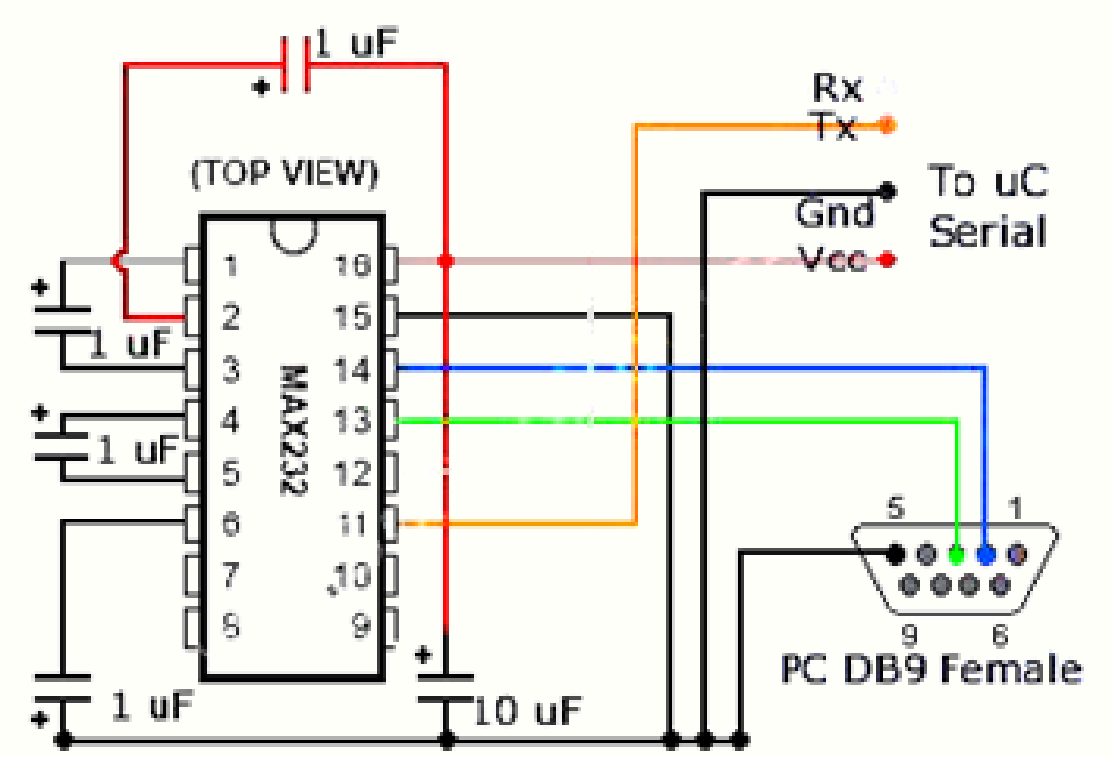
The Rx and Tx shown in above figure (pins 11 and 12 of MAX232) are the Rx and Tx of Atmega 16 (PD0 and PD1 respectively).

USART connection with a computer is accomplished through a protocol called RS232. RS232 is an asynchronous serial communication protocol widely used in computers and digital systems. A simple example is the serial port used in old computers.

One thing to note about this protocol is that, while in an MCU circuit, HIGH = 5V and LOW = 0, for RS232 the values are +12V and -12V respectively. For this conversion, an IC called MAX232 is used:

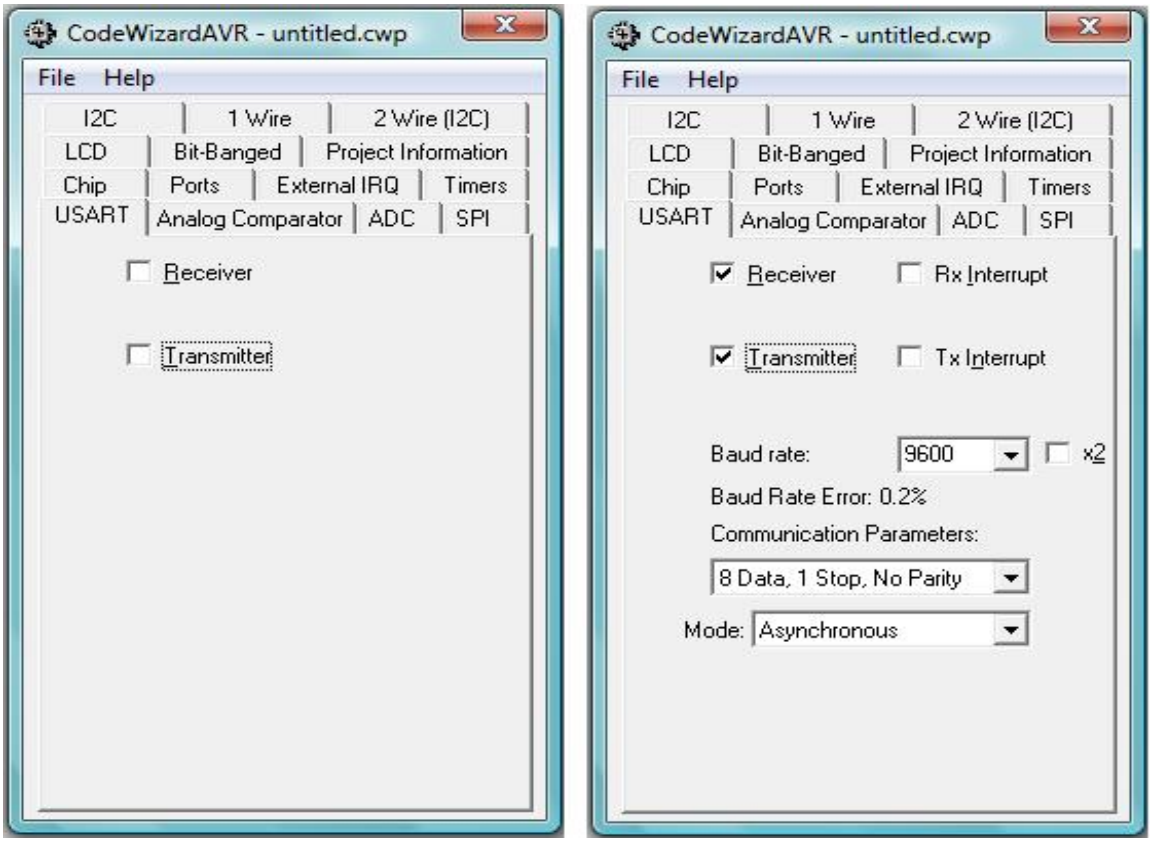


The schematics required are:



13.7 Setting up UART in microcontroller

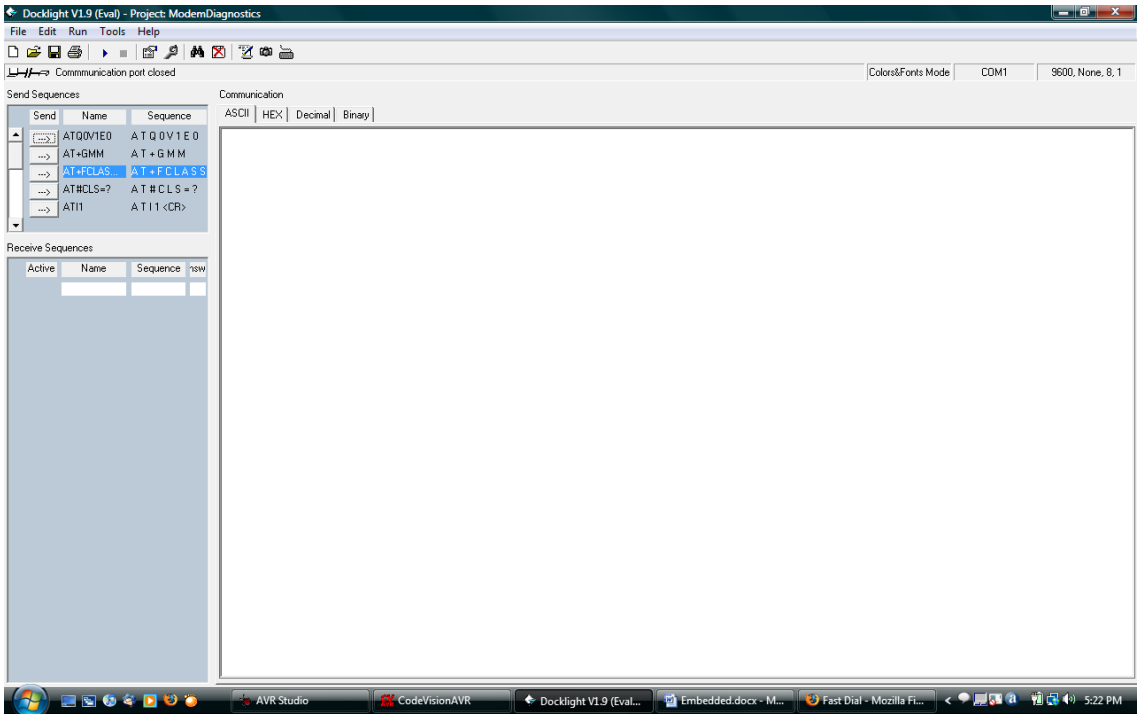
Once our electronic circuit is complete, we can start with coding. To enable UART mode in Atmega16, click on the USART tab in Code Wizard. Now depending upon your requirement, you can either check receiver, transmitter or both. You get a list of options to select from for the baud rate. **Baud Rate** is the unit of data transfer, defined as bits per second. We will select 9600 as it is fair enough for our purpose, and default setting in most of the applications (like Matlab, Docklight, etc.). Keep the communication parameters as default, i.e., 8 data, 1 stop and no parity and mode asynchronous. You also have option of enabling Rx and Tx Interrupt functions, but we won't do at this point.



Once you generate and save the code, all the register values are set by CVAVR and you only need to know some of the C functions to transfer data.

13.8 DockLight

To communicate with the computer, you need a terminal where you can send data through keyboard and the received data can be displayed on the screen. There are many softwares which provide such terminal, but we will be using Docklight. Its evaluation version is free for download on internet, which is sufficient for our purpose.



To start with, check the Terminal Settings in Docklight. Go to **Tools -> Project Settings**. Select the Send/Receive communication channel, i.e., the name by which the serial port is known in your computer (like COM1…). In the COM port settings, select the **samevalues** as you had set while coding Atmega 16. So, we will select Baud Rate 9600, Data Bits 8, Stop Bits 1, Parity Bits none. You can select 'none' in Parity Error Character. Click OK.

We are now ready to send/receive data, so, select **Run -> Start Communication**, or, press F5. If your uC is acting as a transmitter, then the characters it sends will appear in the Communication window of Docklight. E.g.,

```
putchar('K');
delay_ms(500);              // Sends character K after every 500ms
```

Hence what you get on the screen is a KKKKKKKKKKKKKKKKKKKKKKKKK…….. one K increasing every 500ms. To stop receiving characters, select **Run -> Stop Communication**, or press F6.

If the receiver option is also enabled in Atmega16, then whatever you type from keyboard will be received by it. You can either display these received characters on an LCD, control motors depending on what characters you send, etc. e.g.,

```
c = getchar() ;          // receive character
lcd_putchar(c);          // display it on LCD
if (c=='A')
    PORTA=255;           // if that character is A, make all pins of PORTA high.
```

When you are done with sending data, select **Run->Stop Communication**, or press F6.

- Rx Interrupt and Tx Interrupt are special interrupts that are called every time data is received / transmitted. Selecting either option will enable more options - leave them to their default values.

- Baud rate will let you select the transmission rate in bps. The line below the option will tell you the error rate – do not set the baud rate so high that the error is large and the line becomes red.

- Communication Parameters will let you set other parameters – leave them to their defaults for most cases.

IMPORTANT: These settings should be the same for both the devices which are communicating with each other.

13.9 Implementing USART in Your Code

Implementing USART is easy – all you need are two functions:

1. putchar(char);

   This function will allow you to transmit data through the USART interface. The argument is a character; you can transmit the ASCII code of the character in hexadecimal form. The transmitted data is stored in a special register in the device which is receiving this data.

2. getchar();

   This function reads data from the special register reserved for USART communication. This function will stall the program while waiting for the data to be transmitted and stored in the register if it does not already exist.

13.9.1 putchar()

To send one character to the buffer, which will be received by the device (uC or computer)? E.g.,

putchar('A');*//sends character 'A'to the buffer*
putchar(c); *// sends a variable character c*

13.9.2 getchar()

To receive one character from the buffer, which might have been sent by the other uC or the computer. E.g., if we have already defined a variable char c, then

c = getchar(); *// receives the character from buffer and save it in variable c*

13.9.3 putsf()

To send a constant string.Eg,

putsf("My Name is XYZ");

SAMPLE PROGRAM

Input MCU

```
// a is a char variable
a = inputFromUser();
putchar(a);
```

LCD MCU

```
a = getchar();
// Program will wait for data
// Data transmitted, now print
printChar(a);
```

What is an interrupt?

A special event that requires the CPU to stop normal program executionand perform some service related to the event.

Examples of interrupts include I/O completion, timer time-out, illegalopcodes, arithmetic overflow, divide-by-0, etc.Functions of InterruptsCoordinating I/O activities and preventing CPU from being tied upProviding a graceful way to exit from errorsReminding the CPU to perform routine tasks

Interrupt maskabilityInterrupts that can be ignored by the CPU are called maskable interrupts.A maskable interrupt must be enabled before it can interrupt the CPU.An interrupt is enabled by setting an enable flag.Interrupts that can't be ignored by the CPU are called nonmaskableinterrupts.

Interrupt priority

Allow multiple pending interrupt requestsResolve the order of service for multiple pending interrupts

Interrupt serviceCPU executes a program called the interrupt service routine.A complete interrupt service cycle includesSaving the program counter value in the stackSaving the CPU status (including the CPU status register and some other registers)in the stackIdentifying the cause of interruptResolving the starting address of the corresponding interrupt service routineExecuting the interrupt service routineRestoring the CPU status and the program counter from the stack

Restarting the interrupted programInterrupt vectorStarting address of the interrupt service routine /Interrupt vector tableA table where all interrupt vectors are storedMethods of determining interrupt vectorsPredefined locations (Microchip PIC18, 8051 variants)Fetching the vector from a predefined memory location (HCS12)Executing an interrupt acknowledge cycle to fetch a vector number in orderto locate the interrupt vector (68000 and x86 families)

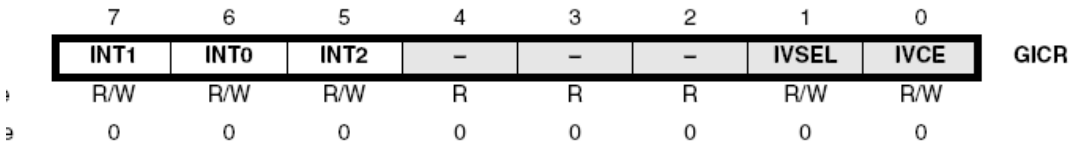
Steps of interrupt programming

Step 1. Initializing the interrupt vector table

Step 2. Writing the interrupt service routine

Step 3. Enabling the interrupt

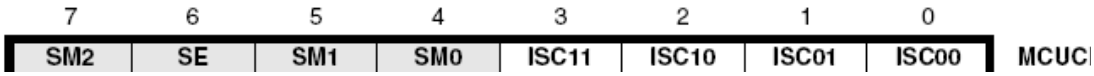
Interrupts Enabling



7	6	5	4	3	2	1	0	
INT1	INT0	INT2	–	–	–	IVSEL	IVCE	GICR
R/W	R/W	R/W	R	R	R	R/W	R/W	
0	0	0	0	0	0	0	0	

INT0,INT1,INT2 are set for enabling the corresponding interrupts

Interrupt Levels



7	6	5	4	3	2	1	0	
SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCI

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

Interrupt Flag

This register contains the interrupt flags for the interrupts 0, 1, 2You can poll this to check for an interrupt.Writing a 1 to bit position acknowledge the interrupt

7	6	5	4	3	2	1	0	
INTF1	INTF0	INTF2	–	–	–	–	–	GIFR
R/W	R/W	R/W	R	R	R	R	R	
0	0	0	0	0	0	0	0	

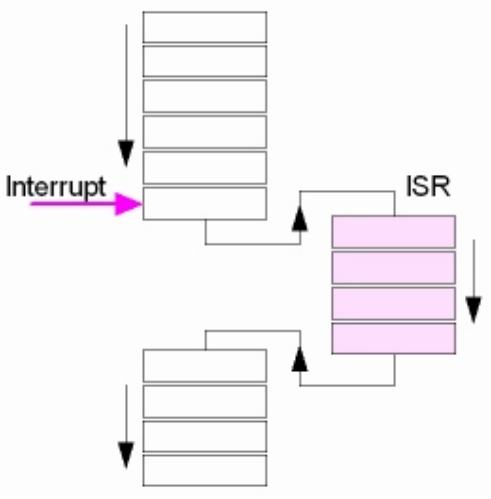
An interrupt is a signal that stops the current program forcing it to execute another program immediately. The interrupt does this without waiting for the current program to finish. It is unconditional and immediate which is why it is called an interrupt.

The concept of an interrupt in reference to microcontrollers is similar to our daily life concept of interrupts: suppose you are reading this tutorial and suddenly your mobile rings - what you do is you stop reading for a while and attend the phone call, and then resume reading from where you left it.

The whole point of an interrupt is that the main program can perform a task without worrying about an external event.

Example

For example if you want to read a push button connected to one pin of an input port you could read it in one of two ways either by polling it or by using interrupts.



14.1 Polling

Polling is simply reading the button input regularly. Usually you need to do some other tasks as well e.g. read an RS232 input so there will be a delay between reads of the button. As long as the delays are small compared to the speed of the input change then no button presses will be missed.

If however you had to do some long calculation then a keyboard press could be missed while the processor is busy. With polling you have to be more aware of the processor activity so that you allow enough time for each essential activity.

14.2 Hardware interrupt

By using a hardware interrupt driven button reader the calculation could proceed with all button presses captured. With the interrupt running in the background you would not have to alter the calculation function to give up processing time for button reading.

The interrupt routine obviously takes processor time – but you do not have to worry about it while constructing the calculation function.

You do have to keep the interrupt routine small compared to the processing time of the other functions - it's no good putting tons of operations into the ISR. Interrupt routines should be kept short and sweet so that the main part of the program executes correctly e.g. for lots of interrupts you need the routine to finish quickly ready for the next one.

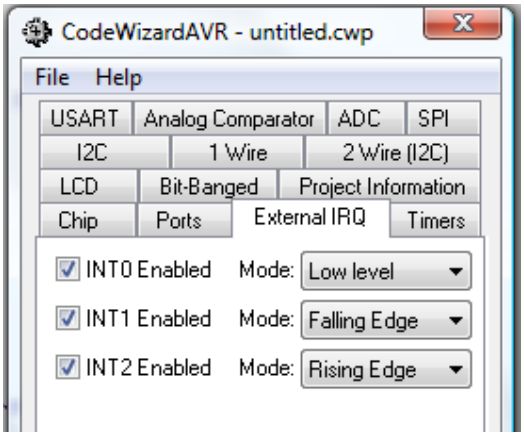
14.3 Hardware Interrupt or polling?

The benefit of the hardware interrupt is that processor time is used efficiently and not wasted polling input ports. The benefit of polling is that it is easy to do.

Another benefit of using interrupts is that in some processors you can use a wake-from-sleep interrupt. This lets the processor go into a low power mode, where only the interrupt hardware is active, which is useful if the system is running on batteries.
Hardware interrupt Common terms

Terms you might hear associated with hardware interrupts are ISR, interrupt mask, non maskable interrupt, an asynchronous event, and interrupt vector and context switching.

14.4 Setting up Hardware Interrupt in Microcontroller



There are 3 external interrupts in Atmega 16. They are

- INT0 : PD2, Pin 16
- INT1 : PD3, Pin 17
- INT2 : PB2, Pin 3

There are 3 modes of external Interrupts,

1. **Low Level:** In this mode interrupt occurs whenever it detects a '0' logic at INT pin. To use this, you should put an external pull up resistance to avoid interrupt every time.

2. **Falling Edge:**In this mode interrupt occurs whenever it detects a falling edge that is '1' to '0' logic change at INT pin.

3. **Rising Edge:**In this mode interrupt occurs whenever it detects a falling edge that is '0' to '1' logic change at INT pin.

14.5 Functions of Interrupt Service Routine

After generating the code, you can see the following function,

```
// External Interrupt 0 service routine
interrupt [EXT_INT0] void ext_int0_isr(void)
{
// Place your code here

}
```

You can place your code in the function, and the code will be executed whenever interrupt occurs.

14.6 Timer Interrupt

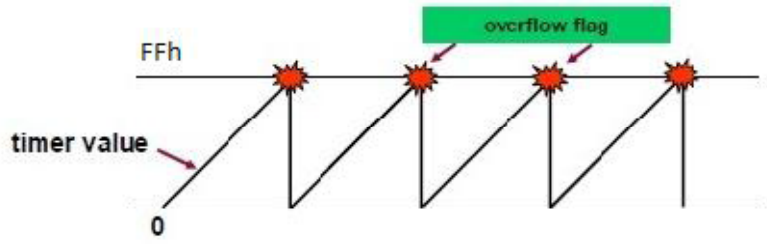
Prior to learning about timers you need to know the concept of interrupt because timers mainly interact with CPU through interrupts. The concept of an interrupt in reference to microcontrollers is similar to our daily life concept of interrupts: suppose you are reading this tutorial and suddenly your mobile rings - what you do is you stop reading for a while and attend the phone call, and then resume reading from where you left it. This is exactly what an interrupt does in MCU. While the MCU is executing a program, if there is something that needs immediate attention, an interrupt is generated by that task and the execution of the current program is left at that time and interrupt is handled. After that, the execution of program continues as usual from the point where it was stopped. The timers run parallel and independent of the CPU at a specific frequency, and interact with the CPU by issuing interrupts.

There are two types of interrupts:

- Overflow interrupt
- Compare match Interrupt

14.7 Overflow Interrupt

Overflow interrupt is triggered whenever the timer register overflows, i.e. reaches its maximum value (in this case, 255, or, in hexadecimal, FFh).



In order to use overflow interrupt, you should first decide what your clock frequency will be (To learn how to do that, see “How to define clock frequency for a timer?” below). Then check the checkbox written “overflow interrupt” that appears in the Timers tab in the CodeWizard AVR (Note: The “Timer value” field can be used to set the initial value of the timer. By default, it is set to 0). Now

when you generate your file, you will find that a function appears in the code:

```
.
.
.
.
#include <mega16.h>
```

```
// Timer 0 overflow interrupt service routine
interrupt [TIM0_OVF] void timer0_ovf_isr(void)
{
// Place your code here
}
// Declare your global variables here
.
.
.
.
```
Now you can place a code here that is executed every time an overflow interrupt is generated.

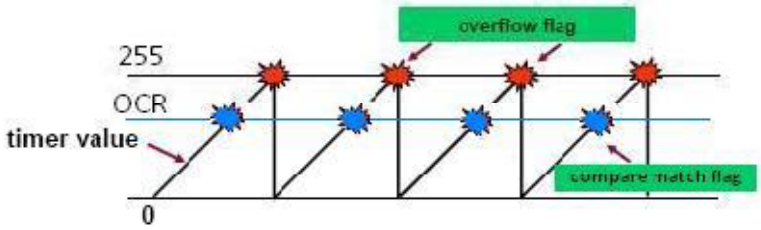
This interrupt can be used to measure time intervals larger than 1 cycle. For example, let us suppose that an LED is connected to say pin 0 of Port A and you want to blink it at 0.5 Hz using overflow interrupts. Since the system frequency is 8 Mhz, you can set an appropriate clock speed say, FCPU/1024 (see "Prescalar" below) and then do it as follows:

```
int count = 0;

interrupt [TIM0_OVF] void timer0_ovf_isr(void)

{
//Increment our variable
count++;
if(count==61)
{
PORTA.0=~PORTA.0; //Invert the Value of PORTA
count=0;
}
}
```
This function will increment the "count" variable every time the overflow interrupt is called and when the appropriate time has passed, will toggle the value of PORTA.0.

14.8 Compare Match Interrupt

A Compare Match Interrupt is issued by a timer whenever the value of the timer becomes equal to a certain predefined value. This predefined value is stored in a register known as the Output Compare Register.

Compare match Interrupts are required by the CTC, Fast-PWM, and Phase correct PWM modes of a timer (see below).



In order to use compare match interrupt, check the checkbox written “compare match interrupt” that appears in the Timers tab in the CodeVision Wizard. Set the value of the OCR in the Compare value in hexadecimal numbers. Now when you generate your file, you will find that a function appears in the code:

```
.
.
.
.
#include <mega16.h>
// Timer 0 output compare interrupt service routine
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
// Place your code here
}
// Declare your global variables here
.
.
.
.
```

Now you can place a code that is executed every time a compare match interrupt is generated, as shown:

```
interrupt [TIM0_COMP] void timer0_comp_isr(void)
{
//Enter your handler code here
}
```

- Understanding internal EEPROM:
- Why EEPROM?
- Provides non-volatile memory storage space
- AVR has internal EEPROM of 512 bytes
- Endurance of atleast 100,000 write/erase cycles
- Organized as a separate data space in which single bytes can be erased/written
- Fuse bit configuration changes for EEPROM:
- HFUSE (previous value=0xC9)
- Revised HFUSE value=0xC1
- Setting EESAVE to 0 (programmed) will reserve the EEPROM contents through the
- chip erase

➢ Understanding internal EEPROM (cont'd...):

- Configuring internal EEPROM:
  - EEPROM Address Register: EEAR

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	EEAR11	EEAR10	EEAR9	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	X	X	X	X	

  - This EEPROM locations to be addresses starting from 0x0000 to 0x0511
  - EEPROM Data Register: EEDR

Bt	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

  - EEPROM Control Register: EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

Bits	Description

EERIE	EEPROM Ready Interrupt Enable
EEMWE	EEPROM Master Write Enable
EEWE	EEPROM Write Enable
EERE	EEPROM Read Enable

- NOTE: EEWE should be set to logical one within four clock cycles after setting EEMWE to logical one (security feature).

```
void EEPROM_write(unsigned intuiAddress, unsigned char ucData)
{
/* Wait for completion of previous write */
while(EECR & (1<<EEWE));
/* Set up address and data registers */
EEAR = uiAddress;
EEDR = ucData;
/* Write logical one to EEMWE */
EECR |= (1<<EEMWE);
                        /* Start eeprom write by setting EEWE */
                        EECR |= (1<<EEWE);
}
unsigned char EEPROM_read(unsigned intuiAddress)
{
/* Wait for completion of previous write */
while(EECR & (1<<EEWE));
/* Set up address register */
EEAR = uiAddress;
/* Start eeprom read by writing EERE */
EECR |= (1<<EERE);
/* Return data from data register */
return EEDR;
}
```

Section C

Robotics

Chapter 16 Introduction to Robots

Robotics is the science of designingand building robots suitable for real-life applications in automated manufacturing and other non-manufacturing environments. Robots are the means of performing multifarious activities for man's welfare in most planned and integrated manner, maintaining their own flexibility to do any work, effecting enhanced productivity, guaranteeing quality, assuring reliability and assuring safety to workers. The word robot came from the Czechoslovakin word Robota which means a worker or a slave doing heavy work.

Over the course of human history the emergence of certain new technologies have globally transformed life as we know it. Disruptive technologies like fire, the printing press, oil and television have dramatically changed both the planet we live on and mankind itself, most often in extraordinary and unpredictable ways. In pre-history these disruptions took place in less than a generation.

We are currently at the edge of one such event .In ten years robotic systems will fly our planes, grow our food, explore space, discover life saving drugs, fight our wars, sweep our homes and deliver our babies.

In the process, this robotics driven disruptive event will create a new 200 billion dollar global industry and change life as you now know it, forever. Just as the present generation cannot imagine world without electricity, the future generation will never know a world without robots.

The Three Laws of Robotics are:

- A robot may not injure a human being, or, through inaction, allow a human being to harm them.
- A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.
- A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

- **DEFINITION OF ROBOTICS**

American Heritage Dictionary: robot (rbt,-bt) n.

- A mechanical device that sometimes resembles a human being and is capable of performing a variety of often complex human tasks on command or by being programmed in advance.
- A machine or device that operates automatically or by remote control.
- A person who works mechanically without original thought, especially one who responds automatically to the commands of others.

- **Webster**

- a : a machine that looks like a human being and performs various complex acts (as walking or talking) of a human being; a similar but fictional machine whose lack of capacity for human emotions is often emphasized b : an efficient insensitive person who functions automatically.

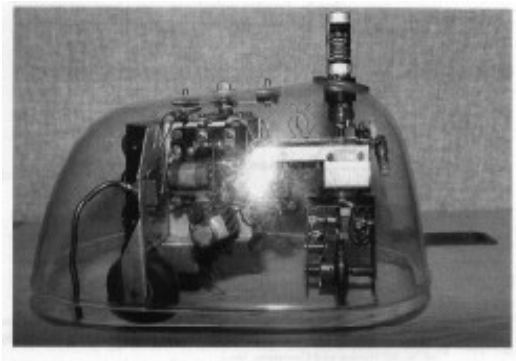
- A device that automatically performs complicated often repetitive tasks.

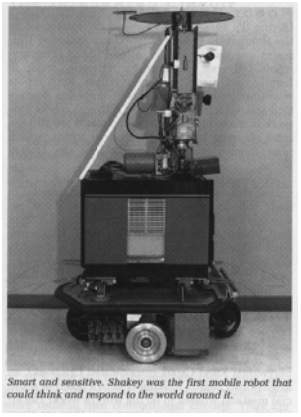
- A mechanism guided by automatic controls.

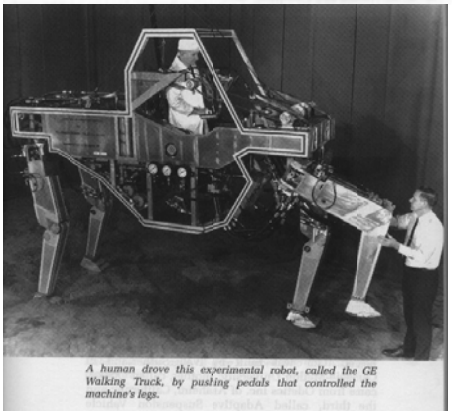
- **History**

One of the first robots was the clepsydra or water clock, which was made in 250 B.C. It was created by Ctesibius of Alexandria, a Greek physicist and inventor. The earliest remote control vehicles were built by Nikola Tesla in the 1890's. Tesla is best known as the inventor of AC electric power, radio (before Marconi), induction motors, Tesla coils, and other electrical devices. Other early robots (1940's - 50's) were Grey Walter's "Elsie the tortoise" ("Machinaspeculatrix") and the Johns Hopkins "beast." "Shakey" was a small unstable box on wheels that used memory and logical reasoning to solve problems and navigate in its environment. It was developed by the Stanford Research Institute (SRI) in Palo Alto, California in the 1960s. The General Electric Walking Truck was a large (3,000 pounds) four legged robot that could walk up to four miles an hour. The walking truck was the first legged vehicle with a computer-brain, developed by Ralph Moser at General Electric Corp. in the 1960s. The first modern industrial robots were probably the "Unimates", created by George Devol and Joe Engleberger in the 1950's and 60's. Engleberger started the first robotics company, called "Unimation", and has been called the "father of robotics."

Isaac Asimov and Joe Engleberger (image from Robotics Society of America web site)



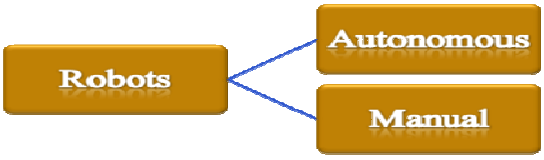




Left: Grey Walter's tortoise, restored recently by Owen Holland and fully operational (from Arkin, 1998). **Center:**Shakey the robot. Shakey was the first mobile robot that could think independently and interact with its surroundings (from Wickelgren, 1996). **Right:** General Electric Walking Truck. A human controlled the stepping of this robot by pushing pedals with his feet. The complicated coordination of movements within a leg and between different legs during stepping was controlled by a computer (from Wickelgren, 1996).

16.1 What Is A Robot?

- *"A re-programmable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks."*
- The word "ROBOT" is derived from the word ROBOTA which means 'compulsory labor'.
- A Robot is an electronic device which can be controlled digitally and manually,basically it reduces the human efffort in every aspect.
- It is a field of Engineering that covers the mimicking of human behavior.
- Robotics includes the knowledge of Mechanical, Electronics, Electrical & Computer Science Engineering.



A robot must have the following *essential* characteristics:

- **Mobility:**It possesses some form of mobility.
- **Programmability:**implying computational or symbol- manipulative capabilities that a designer can combine as desired (a robot is a computer). It can be programmed to accomplish a large variety of tasks. After being programmed, it operates automatically.
- **Sensors:**on or around the device that are able to sense the environment and give useful feedback to the device
- **Mechanical capability:** enabling it to act on its environment rather than merely function as a data processing or computational device (a robot is a machine);
- **Flexibility:** it can operate using a range of programs and manipulates and transport materials in a variety of ways.

Isaac Asimov's Three Laws of Robotics:
The term *robotics* was coined in the 1940s by science fiction writer Isaac Asimov.

- **Law Zero** A robot may not injure humanity, or, through inaction, allow humanity to come to harm.
- **First Law** A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
- **Second Law** A robot must obey orders given it by human beings, except where such orders would conflict with the First Law.
- **Third Law** A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

Introduction to Autonomous Robots:
Any Autonomous Robot consists of following essential parts.

1. **Robot Chassis and actuators**

   Includes wheeled or any type of chassis with all the necessary actuators fitted on the chassis to achieve desired goal. We mostly use DC geared motors as actuators.

2. **Electronics**

   Electronics includes Sensors, motion control circuits, power management system etc.

3. **Power Source**

   Usually battery pack consisting of Lead acid, Nickel cadmium, Nickel metal hydride or Lithium batteries is used.

4. **Intelligence**

This is the most important part of the autonomous robots. Usually intelligence is achieved by using Microcontroller.

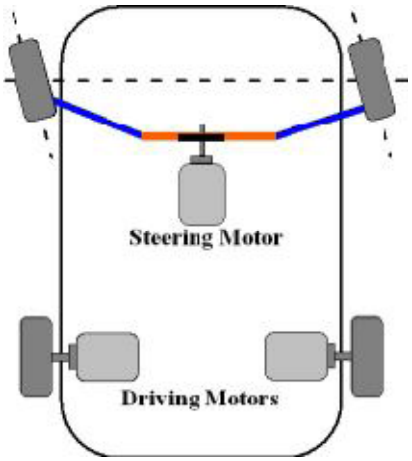
First step in making an autonomous robot is to chalk out what tasks we are expecting the robot to perform. After gauging these we get a vague idea about the design and appearance of the robot.

16.2 Robot Chassis Designing

Selecting the Drive Mechanism for the wheeled motion:

16.2.1 Robot with steering wheel:

- Power for motion is provided by back wheels and turning is achieved using front wheels.
- This scheme is similar to that of cars.



Advantages:

1. When path to be followed is straight in nature with curved turns this configuration gives fastest speed and graceful path following.
2. Don't need to modify left or right wheels velocity to follow the path. This is very advantageous when we want precision velocity control. In this case back wheels take care of velocity control and front wheels take care of direction control.

Disadvantages:

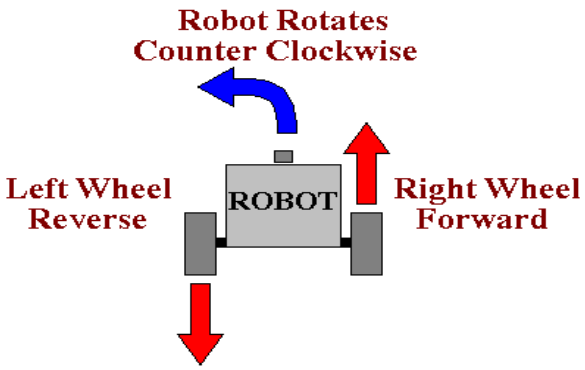
1. It will not able to take very sharp turns. Hence it is difficult to move robot on the grid of lines.
2. Somewhat difficult and expensive to make.
3. Front wheels will need position feedback to control turning control.

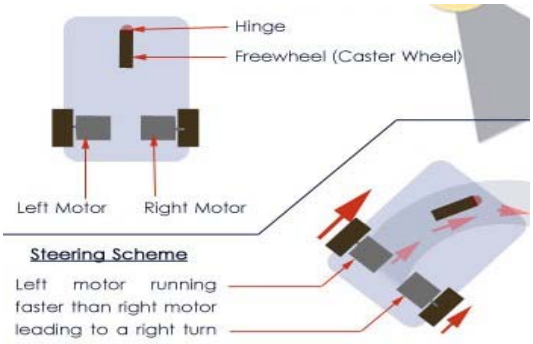
16.2.2 Robot with differential drive:

- A method of controlling a robot where the left and right wheels are powered independently.
- The Three Wheel Differential drive uses two motors and a caster or an omni-directional wheel easiest to design and program.


Figure 21: Ball bearing caster, wheel based swivel caster and omni directional wheel

- The radius and centre of rotation can be varied by the varying the relative speed of rotation between the two motors.





- Rotating the wheels in different directions provides a sharp turn.
- For a smooth turn, rotate the wheels in the same direction but with different speeds. Greater the difference in speeds, smaller the radius of rotation.

Advantages:

1. Zero turning radius achievable.
2. Easy to move when path to be followed is contoured and zigzag in nature. E.g., navigating along the maze of lines.

Disadvantages:

1. If we want to move along curved path we have to control left and right motor's velocity independently. Hence precision velocity control becomes difficult as actual velocity of the robot will be average of the both wheels.

Chapter 17 Motors and Motor Drivers

17.1 Introduction to Motors

Device used to convert electrical energy into mechanical energy is called a motor. A motor is very useful in robotics and embedded systems to give movement and designing control systems.
Some Motors used in embedded systems and robotics are described as below:

i. Stepper motors
ii. Servo motors
iii. Geared DC Motors

Stepper motors

A motor which divides its full rotation into a large number of steps is called a stepper motor. These steps help in precise, step by stepmovement of the rotor giving precise control over movement.

The shaft of stepper motor has permanent magnets attached to it. The rotor is surrounded by series of coils which can be turned on and off and the magnetic field change causes the rotor to move.As the movement is controlled by turning coils on or off, it is easier to program the movements of stepper motor using microcontrollers.

These motors are also called open loop systems and do not have any feedback mechanism. These have very simple design and are often less reliable.Such motors are used in variety of devices like linear actuators, printers etc.

Servo motors

A servo motor is defined as the motor that allows more precise control of position, velocity, or torque using feedback loops. These feedback loops help in stability analysis and give better control over movement. These are also called closed loop systems.

These motors cannot rotate continually and hence cannot be used for driving wheels. These motors usually have a 90-180 degree movement. Servo motors are normally used in machine tools and automation robots.

Programming these motors is more complex. Also the design is more complex. These are more reliable than Stepper motors.

Geared DC motors

Gear motors are motors with an integrated gearbox. The function of gearbox is to increase torque generating capacity of the motor reducing its output speed. Hence, the need for speed reducing arrangement is eliminated.

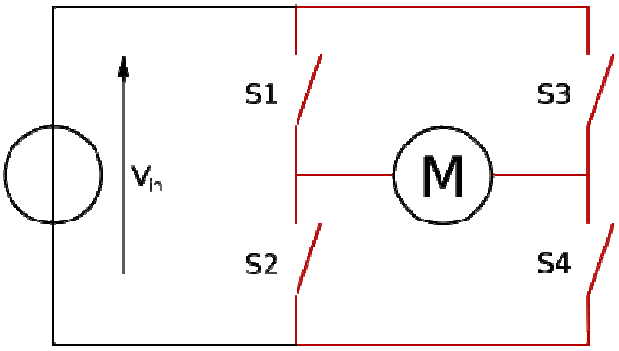
Geared motors have a variety of applications. Some of its uses are as in wheelchairs, stair wheels etc.

17.2 H- Bridge:

- It is an electronic circuit which enables a voltage to be applied across a load in either direction.
- It allows a circuit full control over a standard electric DC motor. That is, with an H-bridge, a microcontroller, logic chip, or remote control can electronically command the motor to go forward, reverse, brake, and coast.
- H-bridges are available as integrated circuits, or can be built from discrete components.

- A "double pole double throw" relay can generally achieve the same electrical functionality as an H-bridge, but an H-bridge would be preferable where a smaller physical size is needed, high speed switching, low driving voltage, or where the wearing out of mechanical parts is undesirable.

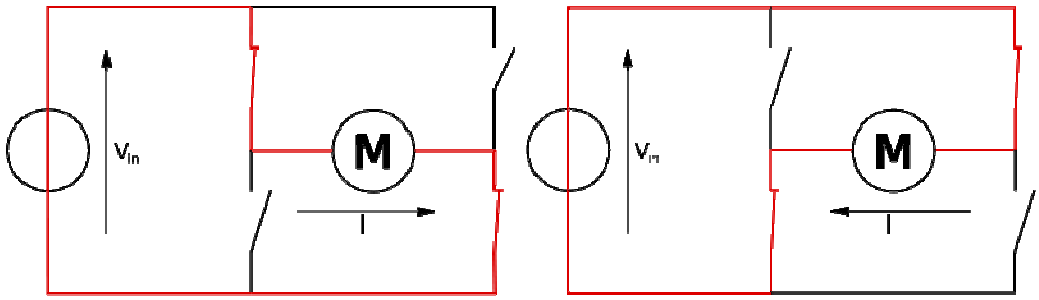
- The term "H-bridge" is derived from the typical graphical representation of such a circuit, which is built with four switches, either solid-state (eg, L293/ L298) or mechanical (eg, relays).



S1	S2	S3	S4	Result
1	0	0	1	Motor rotates in one direction
0	1	1	0	Motor rotates in opposite direction
0	0	0	0	Motor free runs (coasts)
0	1	0	1	Motor brakes
1	0	1	0	Motor brakes

Structure of an H-bridge (highlighted in red)

- To power the motor, you turn on two switches that are diagonally opposed.



The two basic states of an H-bridge.

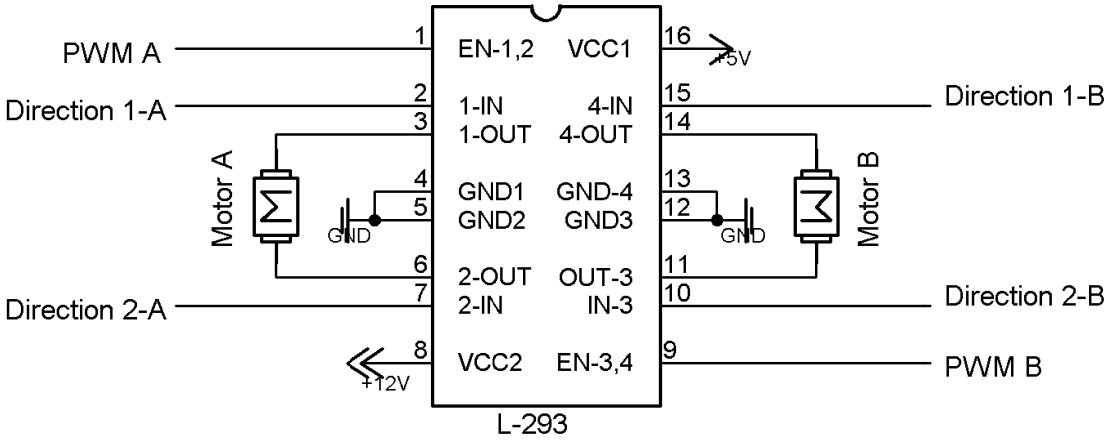
17.3 Motor Driver ICs: L293/L293D and L298

- The current provided by the MCU is of the order of 5mA and that required by a motor is ~500mA. Hence, motor can't be controlled directly by MCU and we need an interface between the MCU and the motor.

- A Motor Driver IC like L293D or L298 is used for this purpose which has two H-bridge drivers. Hence, each IC can drive two motors.

- Note that a motor driver does not amplify the current; it only acts as a switch (An H bridge is nothing but 4 switches).

Figure 23: L293D **Figure 22: L298**



- Drivers are enabled in pairs, with drivers 1 and 2 beingenabled by the Enable pin. When an enable input is high (logic 1 or +5V), the associated drivers areenabled and their outputs are active and in phase with their inputs.

- When the enable pin is low, the output is neither high nor low (disconnected), irrespective of the input.

- Direction of the motor is controlled by asserting one of the inputs to motor to be high (logic 1)and the other to be low (logic 0).

- To move the motor in opposite direction just interchange the logic applied to the two inputs of the motors.

- Asserting both inputs to logic high or logic low will stop the motor.

- Resistance of our motors is about 26 ohms i.e. its short circuit current will be around. 0.46Amp which isbelow the maximum current limit.

- It is always better to use high capacitance (~1000µF) in the output line of a motor driver which acts as a small battery at times of current surges and hence improves battery life.

- **Difference between L293 and L293D:** Output current per channel = 1A for L293 and 600mA for L293D.

17.3.1 Difference between L293 and L298:

- L293 is quadruple half-H driver while L298 is dual full-H driver, i.e, in L293 all four input-output lines are independent while in L298, a half H driver cannot be used independently, only full H driver has to be used.
- Output current per channel = 1A for L293 and 2A for L298. Hence, heat sink is provided in L298.
- Protective Diodes against back EMF are provided internally in L293D but must be provided externally in L298.

17.3.2 Speed Control:

- To control motor speed we can use pulse width modulation (PWM), applied to the enable pins of L293 driver.
- PWM is the scheme in which the duty cycle of a square wave output from the microcontroller is varied to provide a varying average DC output.
- What actually happens by applying a PWM pulse is that the motor is switched ON and OFF at a given frequency. In this way, the motor reacts to the time average of the power supply.

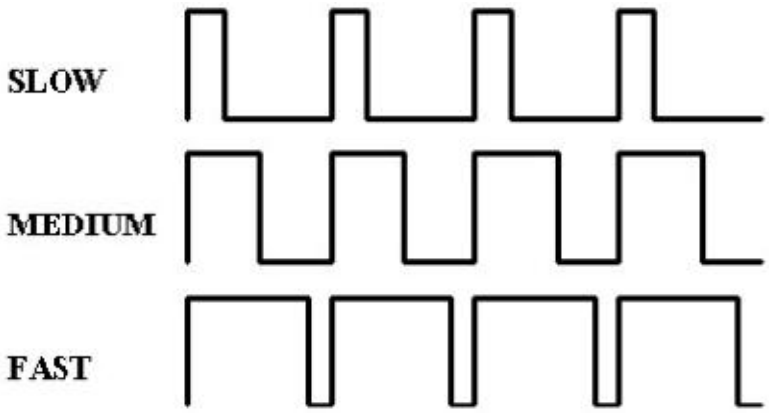


Figure 24: Velocity control of motor using PWM

Chapter 18 Sensors

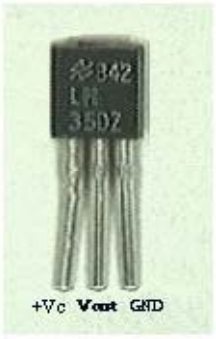
A **sensor** is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument.

Temperature sensor:

Commonly available temperature sensors are LM35,DS1621,thermistor.Thermistor gives resistance proportional to the temperature. But accuracy is not good in thermistor. DS1621 gives digital output in I2C format, so you require a microcontroller to interface to see the temperature. Thermistor require accurate resistance in series to get good reading with accuracy The resistance of thermistors

are 100ohm,1Kohm(the one i have heard). But thermistor creates some headaches although it costs Rs.5. LM35 have 3 terminal Vcc, ground, Vout. So it is easy and gives analog output and it costs Rs.50

LM35:



The most commonly available LM35 is LM35DZ measures temperaturefrom 0 to 100 degree. Normally sensors become inaccurate with age. ButLM35 will not have this problem. We get LM35 in TO-92 package, just likesmall transistors like BC547. The main problem you will be creating isinterchanging leads. Vs range from 4V to 30V. Output is 10mV* degree inCelsius. That is if the temperature is 29 degree then V_{out}=.29V. Let's tryfor a circuit which will glow an LED if the output voltage is greater thansome voltage. Use comparator with a reference voltage at one end andLM35 output at other end, so that comparator become high when thetemperature is above reference. More Links:

http://www.facstaff.bucknell.edu/mastascu/elessonshtml/Sensors/TempLM35.html

http://www.cjseymour.plus.com/elec/tempsens/tempsens.htm

http://www.national.com/pf/LM/LM35.html

LIGHT SENSORS

Light sensors are used to measure the intensity of light. Mostly availablesensors are Cadmium Sulphide LDR sensor, IR senor like photodiode,photo transistor, TSOP1738. For beginners LDR is easy to handle. Soas a beginner better start with LED+LDR combination or IR LED+photodiode . LDR is economical than other sensors and easy to handle.

LIGHT DEPENDENT RESISTOR (LDR):

LDR is basically a resistor whose resistance varies withintensity of light. More intensity less its resistance (i.e, in black it offershigh resistance and in white it offers less resistance). This is the basicsensor which beginners should start with, which is having cost less thanRs.6.Figure below show some of the pictures of LDR which i obtained fromsome site.

Fig.1

Fig.1 shows LDR's with greater surface area; Fig.2 shows the LDR's which are available about Rs.5, which is commonly used. Greater surface area,better the sensing will be. The sensing material is made of Cadmium Sulphide.
Resistance: 400ohm to 400Kohm

Normal resistance variation: 1Kohm to 10Kohm (in the robots which I used for line following for identifying black and white strips)

Sensitivity: about 3msec(Sensitivity is defined as the time taken for output to change when input changes, i got this reading by verifying with ADC interfaced with parallel port, sensitivity of LDR's is in mill seconds. This is the best sensitivity obtained to me).

Voltage ratings: I used it on 3V, 5V and 12V

Practical application in Line follower Robots:LED's are used with LDR which will act as a source of light for LDR because we are placing the LDR below the robot where light is not present. If we want to identify Black and White strips we add a light source with LDR and the white strip reflects light while black won't reflectlight.

Above figure shows how LED is placed with LDR. Here LDR is covered because we want light reflections from ground only,not from sides of LED.Also cover the LED so that the light will move pointed, so that reflection will directly go to LDR. When you attach LED and LDR to the body of the robot, use tape to paste the sensors. Remember if you robot body is of aluminum, then some short circuit or current flow can occurs through the body. So apply tape perfectly so that no short circuit problems occur. Remember that LDR is a resistor and have no polarity while all other sensors have.

PROBLEMS: LDR is mainly used with visible light. So the problem of external light will affect the LDR. The affect of visible light is more in LDR then comes Photo diode, then TSOP1738.
http://www.technologystudent.com/elec1/ldr1.htm
http://www.kpsec.freeuk.com/components/other.htm
http://www.mstracey.btinternet.co.uk/technical/Theory/theorysensors.htm
http://www.tpub.com/neets/book7/26g.htm

PHOTO DIODE
Here is some link for photo diode:
http://www.radioelectronics.
com/info/data/semicond/photo_diode/photo_diode.php
http://en.wikipedia.org/wiki/Photodiode
http://www.lasermate.com/PR.htm
http://electron9.phys.utk.edu/optics421/modules/m4/photodiode.htm

Photo diode works in reverse biase region. A photo diode leads can be identified by seeing the length of the leads. Short lead is the cathodeconnected to greater voltage. The current flowing through the photo diode changes with intensity of the light. You can use it for edge detection. I tried to do edge detection of a table, i got range about 7cm. IR LED is used for producing light. When you are using IR LED be sure that it is working properly by measuring the voltage across the IR LED, should be greater than 2V. When connecting IRLED the voltage of the circuit drops, so be careful that voltage to other circuits won't fall below the level.

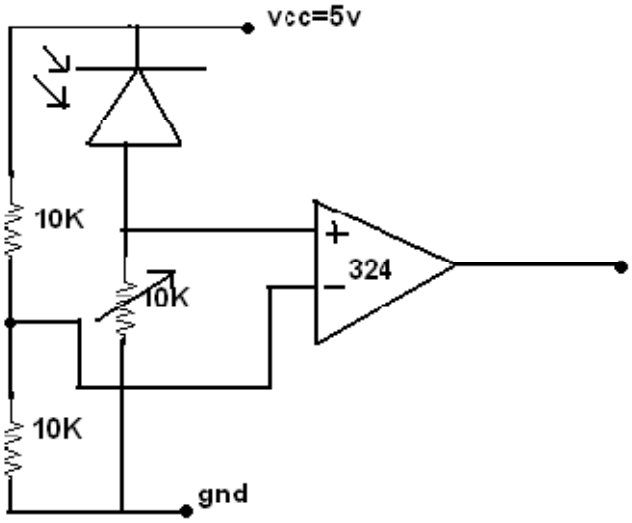
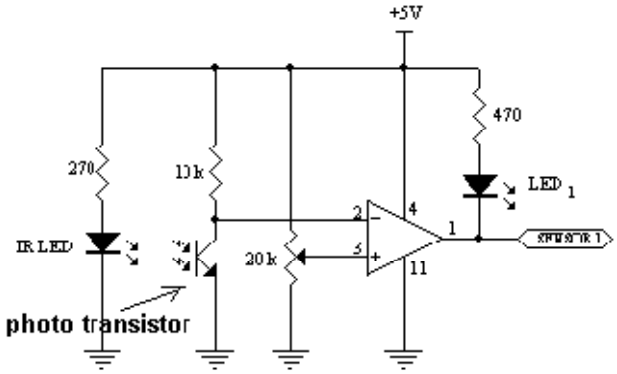


Photo diode and IR led looks same. The only difference is in its color IR LED is some dark in color. If you still can't identify. See this post
http://nod.phpwebhosting.com/~robotics/modules.php?name=Forums& file=viewtopic&t=435

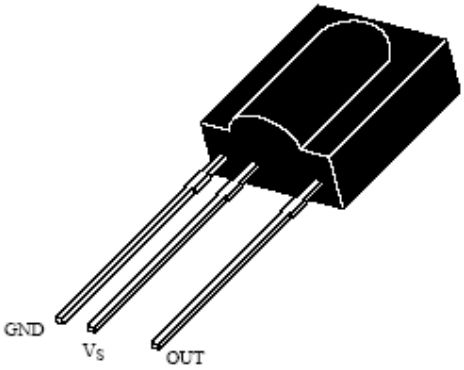
PHOTO TRANSISTOR

I haven't used photo transistor. But a photo transistor is one in which base is like the receiver of light. When light falls there will be a short circuit between collector and emitter. This can be used in optical communications. I heard that you can make a photo transistor by cutting the upper portion of transistor BC107 and leaving the base. You can use either IR or laser (cheap one available). But in case of transmission we require line of sight propagation. Here is a circuit for detection of IR using photo transistor.
http://www.kmitl.ac.th/~kswichit/LFrobot/LFrobot.htm



Here when light is not there then the resistance of transistor will be high, so the V-(pin2)>V+(pin3) making output of comparator LOW. That is when no reflection from ground or any obstacle on the IR. When light is there then the resistance will be very less and V+>V-. So output of comparator is HIGH. Suppose if you are using it for line detection, then there is reflection of IR from the white surface, but IR radiations are absorbed by black surface, so no or less reflection from the surface in black strip. Remember to check the voltage across IR to see whether IR LED is working or not and it should be greater than 2V. When black strip comes, output of comparator become 0V and the LED glows(visible light LED).

TSOP1738



Supply Voltage (Pin 2) VS –0.3...6.0 V
Supply Current (Pin 2) IS 5 mA
Output Voltage (Pin 3) VO –0.3...6.0 V
Output Current (Pin 3) IO 5 mA
Continuous data transmission possible
(up to 2400 bps)
Suitable burst length .10 cycles/burst
cost-Rs.15

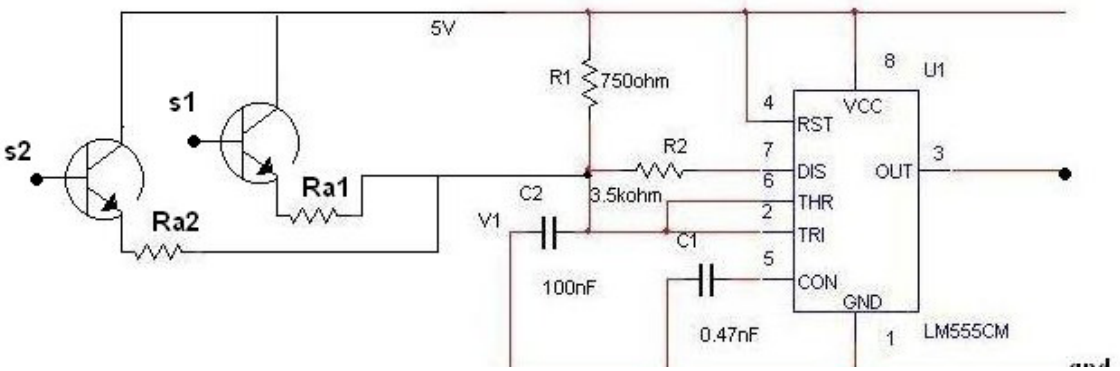
DISTANCE MEASUREMENTS

For a small distance measurement we can use a photo diode or photo transistor, but only distance up to 5-7cm. You just connect the output to ADC or any comparator to measurement. Suppose ifwe use one LM324 for distance measurement, you can measure 1cm ,2cm,3cm,4cm. You just connect a 330 ohm in series with IR LED. At the other end use a photo diode in reverse region.
http://www.multyremotes.com/IRSw.htm
http://www.roboticsindia.net/modules.php?name=Forums&file=viewtopic&t=410
http://www.windowschallenge.com/Final%20Reports%5CFINAL%20RE
PORT-FINAL.doc
http://www.robotroom.com/Infrared555.html
http://www.students.uwosh.edu/~piehld88/laser.htm
http://www.wanyrobotics.com/distance.html #
If you want a good distance then you should use 38Khz modulated IR with TSOP1738 detector. Use IC555 to generate 38Khz square wave. Better tutorials available in roboticsindia, see

http://www.roboticsindia.net/modules.php?name=News&file=article&sid=32&mode=&order=0&thold=0

http://www.roboticsindia.net/modules.php?name=News&file=article&sid=35&mode=&order=0&thold=0

You can get range about 1 Meter. If you want to measure various distance then you should vary Ra of the IC555. Suppose if you want to measure distance from a fixed point, then you have to vary the frequency of IC555.You can do it fixing Rb>Ra and vary Ra so that frequency will vary slightlyfrom some 36Khz to 40Khz and find corresponding reading. You can do itby using the following technique



Suppose if you want to measure distance from afixed point. This is done by varying the frequency. When S1=1 thenRaeq=R1//Ra1= R1Ra1/(Ra1+R1). So this will produce a differentfrequency some between 36 to 40Khz. When S2=1 Raeq=R1//Ra2. Andwhen S1=S2=1 then Raeq=R1//Ra1//Ra2. By varying S1,S2 you canmeasure the distance from it. But this mostly require the need of somecircuitary. Better go for a microcontroller. Adjust Ra1,Ra2 so that desiredfrequencies are obtained.

About the range of IR sensor
http://www.triindia.co.in/forums/viewtopic.php?t=12
http://www.triindia.co.in/forums/viewtopic.php?t=4
http://users.triera.net/zupanbra/senzor.html

COLOR SENSING:

I haven't heard of availability of color sensors. But we can make it fromscratch. Suppose if you want to sense the color of ball. First thing you haveto bring robot near the ball. The distance of the ball from the robot shouldbe fixed. Second thing the effects of external light. First make the robot afixed distance from the ball. This is made by using a IR LED+photo diodecombination. Bring robot close so that you will get a good response. Forsensing color you use LED+LDR combination. But the problem withexternal light will be higher in this case. So you should provide somemechanical mechanism to hide external light. Photo diode or modulated38Khz+TSOP is used for distance measurement. But photo diode is enoughto get distance about 5cm or near. Use comparator output to the output ofphoto diode for distance measurement. Use LDR to sense the color. But theaccuracy is a real problem.

RANGE FINDING:
http://www.roboticsindia.net/modules.php?name=Forums&file=viewtop
ic&t=148
SONAR
http://www.leang.com/robotics/info/articles/minison/minison.html

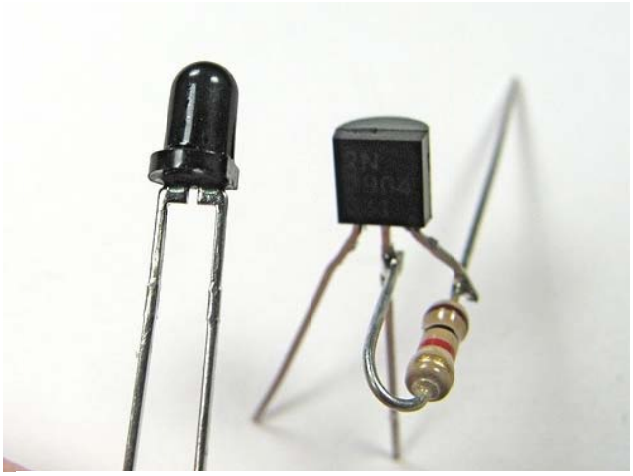
ANGLE MEASUREMENT

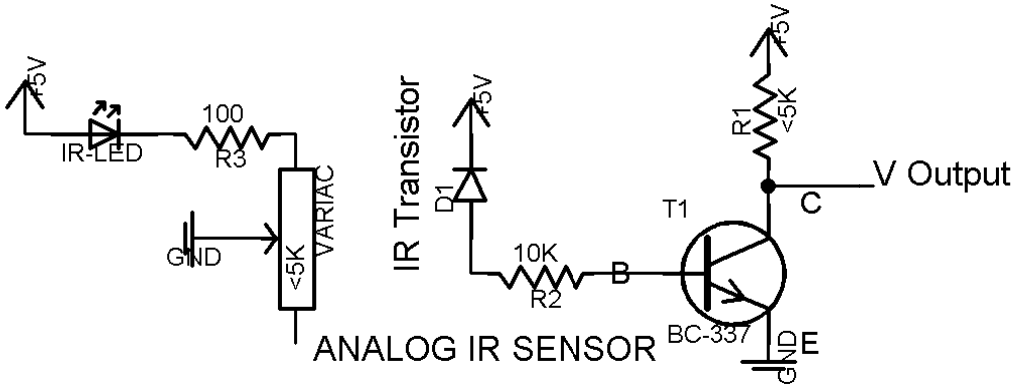
Suppose if your robot is going over an inclined plane and we want tomeasure the inclination of the plane then we should go for anglemeasurement. This is done simply using the principle of a pendulum. Ifyou goes up an inclined plane a pendulum kept perpendicular to thegravity will change its angle with normal. Attach a somewhat heavy ball tothe shaft of the variable resistor (potentiometer). When robot goes up theplane, the inclination of the pendulum changes and the shaft of thevariable resistor will vary. You connect the output of the variable resistorto ADC to get reading. Instead of pendulum and variable resistor keep asource of light perpendicular to the ground

and keep some LDR's in therobot so that the light source moves and the reading of a LDR's willchange. Keep some 5 LDR's for good accuracy. The robot will go up anddown an inclined plane. So greater the number of LDR's greater will beaccuracy. Suppose if you use one LDR and connecting LDR output toADC and measuring angle is not good because it won't be able to detectwhetherrobot is going up or down in an inclined plane(the readings willremain same for both side. Suppose if we take +20 degree then the LDRreading will be same for -20degree). So it is better to use some LDR's orpotentiometer shaft attached with a heavy ball and output ofpotentiometer to ADC.

Now we will discuss about IR sensors.

18.1 Analog Sensor





The IR analog sensor consists of:

Transmitter: An Infra Red emitting diode

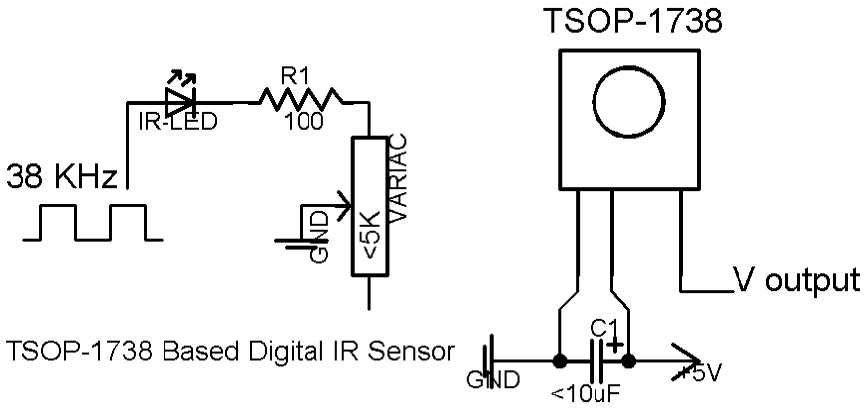
Receiver: A Phototransistor (also referred as photodiode)

It is better to keep R2 as a variac to vary the sensitivity.

The output varies from 0V to 5V depending upon the amount of IR it receives, hence the name analog.

The output can be taken to a microcontroller either to its ADC (Analog to Digital Converter) or LM 339 can be used as a comparator.

18.2 Digital IR Sensor - TSOP Sensor

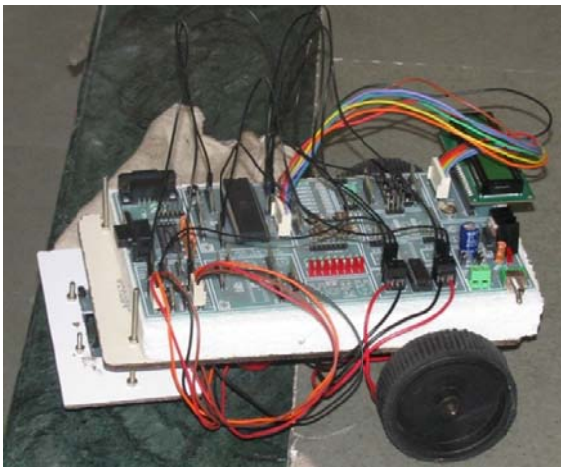


TSOP-1738 Based Digital IR Sensor

- TSOP 1738 Sensor is a digital IR Sensor; It is logic 1 (+5V) when IR below a threshold is falling on it and logic 0 (0V) when it receives IR above threshold.
- It does not respond to any stray IR, it only responds to IR falling on it at a pulse rate of 38KHz. Hence we have a major advantage of high immunity against ambient light.
- No comparator is required and the range of the sensor can be varied by varying the intensity of the IR emitting diode (the variac in figure).

19.1 Line following Robot

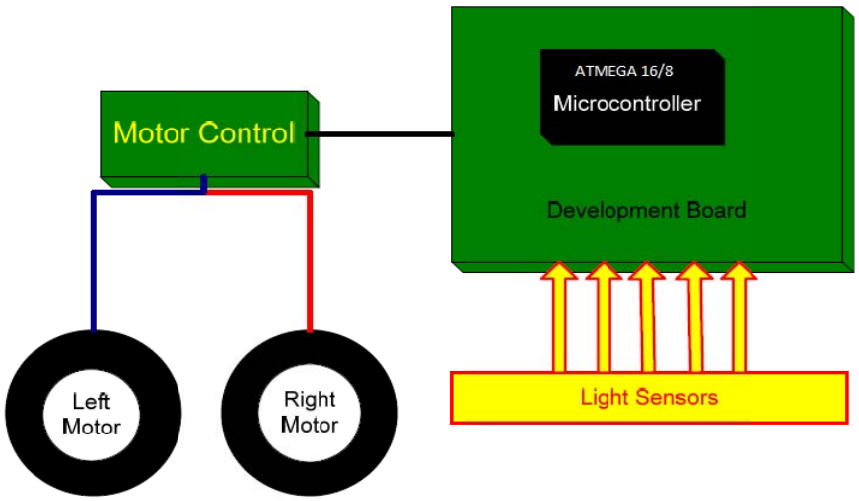
A Line following robot is a machine which can follow a given path. A Path can be in the form of a black line on a white surface or reverse of it. The Path can also be in the form of magnetic lines which is out of scope of this course. Such robot is a closed loop system having a feedback mechanism to correct wrong moves. Line following robots can be driven using analog comparator based circuit like opamp or by microcontroller both using feedback from analog or digital sensors.



Detection of line is usually done by emitting light on the surface. If light gets reflected back, then it is white surface else the surface is black. Emission of light is usually done with the help of IR LEDs (Light Emitting Diodes). An array of such LEDs is used. The emitted light is then received using phototransistors, LDR (Light Dependent Register), TSOP. Based on the value of light received, the received energy is converted to electrical energy.

If the robot gets off the track, the speed of wheel motors is proportionally varied to make it back on the track.

Line following property of a robot is very useful in designing automated cars. Also, industrial robots may use such mechanism in order to be on proper path.



Block Diagram for Line Following Robot

Chapter 20 Definitions of Embedded system

1:- An **Embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing constraints.

or

An **Embedded system** is a Software program on a H/W chip designed for a specific purpose and can also contain some mechanical moving parts.

or

An **Embedded system** is a special-purpose computer system designed to perform one or a few dedicated functions, often with real-time computing.

or

An **Embedded system** is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device.

or

A specialized computer system that is part of a larger system or machine. Typically, an Embedded system is housed on a single microprocessor board with the programs stored in ROM.

2:-Types of Embedded Sys:-

Real time Embedded Systems

Non-Realtime Embedded Systems

3:- Non real-time embedded system is one in which there is no deadline, even if fast response or high performance is desired or preferred.

4:- Real time Embedded Sys

Realtime Embedded Systems are again sub- classified as

Soft real-time embedded systems

Hard real-time embedded systems based on the consequence of missing the deadline time.

Hard real-timeembedded system, are one in which the completion of an operation after its deadline is considered useless - ultimately, this may lead to a critical failure of the complete system(Missile defence system, Anti-collision system).

Soft real-time system on the other hand will tolerate such lateness, and may respond with decreased service quality (e.g., dropping frames while displaying a video) otherwise called as graceful degradation.

5:- CPU specific buzz words:-A microprocessor incorporates most or all of the functions of a central processing unit (CPU) on a single integrated circuit (IC).

6:- What's Microcontroller ?:-A microcontroller (also MCU or µC) is a small computer on a single integrated circuit consisting of a relatively simple CPU combined with support functions such as a crystal oscillator, timers, watchdog, serial and analog I/O etc.

7:- What's ARM?:- The ARM architecture (previously, the Advanced RISC Machine, and prior to that Acorn RISC Machine) is the most widely used 32-bit processor architecture in the world.

8:- P I C ?:- It is a family of Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1640 originally developed by General Instrument's Microelectronics Division. The name PIC initially referred to "Peripheral Interface Controller".

9:- What's AVR? :- The AVR(Advanced virtual risc) is a Modified Harvard architecture 8-bit RISC single chip microcontroller (µC) which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage.

10:- THUMB :- To improve compiled code-density, processors from the ARM7TDMI on have featured the Thumb mode. When in this mode, the processor executes 16-bit instructions. Most of these 16-bit-wide Thumb instructions are directly mapped to normal ARM instructions.

11:- What's an Interrupt ?:- It is an asynchronous signal from hardware indicating the need for attention or a synchronous event in software indicating the need for a change in execution. A *hardware interrupt* causes the processor to save its state of execution via a context switch, and begin execution of an interrupt handler.

12:- What's a Cross Compiler ? :- A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. Cross compiler tools are generally found in use to generate executables for embedded system or multiple platforms.

or

It is a tool that one must use for a platform where it is inconvenient or impossible to compile on that platform, like microcontrollers that run with a minimal amount of memory for their own purpose.

14:- What's a Linker ? :- A linker is a program that takes one or more objects files generated by a compiler and combines them into a single executable program.

15:- What's a Loader ?:- It is the part of an operating system that is responsible for loading programs from executables (i.e., executable files) into memory, preparing them for execution and then executing them.

16:- System software:- System software is closely related to, but distinct from Operating System software. It is any computer software that provides the infrastructure over which programs can operate, i.e it manages and controls computer hardware so that application software can perform.

Examples:- Other than OS, Anti-virus software, Communication software and printer drivers.

17:- Simulators:- A application that creates an environment that is as close as possible to reality. In flight simulators, engineers create a cockpit environment identical to the one in a real airplane. In a flight simulator a pilot will see, hear and feel like he or she is in a real aircraft.

18:- Emulator:- It is a program/environment which duplicates (provides an emulation of) the functions of one system using a different system, so that the second system behaves like (and appears to be) the first system. This focus on exact reproduction of external behavior is in contrast to some other forms of computer simulation, which can concern an abstract model of the system being simulated.

19:- What's Bottleneck ?:- A limit of throughput between computer processor and memory.

20:- Throughput:- The maximum number of instructions a processor executes per unit time is called its throughput. The same is the case for microcontrollers also. The higher the throughput(which again depends on various factors) the better is the processor/microcontroller.

21:- Instruction set capability:- This refers to the capability the instruction set of that particular microprocessor/ microcontrollers have such as with code density and DSP related instructions or some instructions which are complexly implemented.

22:- Firmware :- It is a term sometimes used to denote the fixed, usually rather small, programs that internally control various electronic devices.

23:- Device driver:- It is a computer program allowing higher-level computer programs to interact with a hardware device. A device driver simplifies programming by acting as an abstraction layer between a hardware device and the applications or operating systems that use it.

24:- Board Support package:- BSP is implementation specific support code for a given board that conforms to a given operating system. It is commonly built with a bootloader that contains the minimal device support to load the OS and device drivers for all the devices on the board.

25:- Boot Sequence :- A boot sequence is the initial set of operations that the computer performs when it is switched on. The boot loader typically loads the main operating system for the computer

26:- Watchdog Timer :- A watchdog timer (or computer operating properly timer) is a computer hardware timing device that triggers a system reset if the main program, due to some fault condition, such as a hang.A special timer that is used specifically for the purpose of resetting the system if it overflows. A piece of program resets the watchdog timer every so often before the watchdog timer expires. However, if the program fails to reset the timer, indicating that the program has crashed or entered into some infinite loop(indicating unwanted and unexpected program or system behavior), the watchdog timer overflows and this generates a processor reset signal.

27:- Pre-emption : Preemption:- It is the act of temporarily interrupting a task being carried out by a computer system, without requiring its cooperation, and with the intention of resuming the task at a later time. Such a change is known as a context switch.

28:- Multi-tasking :- Multitasking is a method by which multiple tasks, also known as processes, share common processing resources such as a CPU.

29:- Multi-Processing :- Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them.

30:- Multithreading:-The ability of an operating system to execute different parts of a program, called *threads,* simultaneously. The programmer must carefully design the program in such a way that all the threads can run at the same time without interfering with each other.

31:- POSIX :- Portable Operating System Interface, is the collective name of a family of related standards specified by the IEEE to define the application programming interface (API), along with shell and utilities interfaces for software compatible with variants of the Unix operating system.

32:- MISRA:- Motor Industry Software Reliability Association. This association sets standards and guidelines for usage of C/C++ coding in safety critical systems such as automotive domain in particular. 127 rules in coding C has to be followed which makes C code so called Safe code.

33:- Fuzzy Logic :- Fuzzy logic is a form of multi-valued logic to deal with reasoning that is approximate rather than precise.

Chapter 21 GLOSSARY

- **ADC:-** Acronym for analog to digital converter.An electronic device or system that encodes analog voltage/current to a multilevel digital number.

- **ADDRESS BUS:-** A group of signals in a microprocessor system that indicates the address of the memory location from where the data is to be read or written to.

- **ASCII:-** American standards committee for information interchange.A 7-bit coding scheme for alphabets,numerals,punctuation,as well as control characters.

- **ASIC:-** Acronym for Application Specific Integrated Circuit.

- **ASSEMBLER:-** A software program that takes a text file called the source file, and converts it into another file with the machine op-codes(simply called the machine code or object code).

- **BCD:-** Acronym for Binary Coded Decimal.

- **BOD:-** Brownout Detector.An electronic device that detects a drop in supply voltage below a threshold and generates a signal to reset the processor till the supply voltage is restored to acceptable level.

- **BIT:-** Binary digit.

- **Bps:-** Acronym for Bits per second.

- **Byte:-** A number with a maximum of eight bits. Thus the byte-wide number is between 0 and 255(decimal).

- **CISC:-** Acronym for complex instruction set computer. This a type of processor architecture that is characterized by a variable instruction length, usually small numbers of register and multiple address modes.

- **COMPILER:-** A software program that converts a high-level language source code into the machine language that the processor can execute.

- **COUNTER:-** A register that is incremented for each occurrence of an event on an input pin of the counter.The event is indicated by a pulse.For each pulse,the counter is incremented

- **CRITICAL SECTION:-** A chunk of code that must be executed without any interruption for proper operation.

- **CPLD:-** Acronym for complex programmable logic device.A programmable logic device rich in gates and interconnection circuitry.

- **CPU:-** Acronym for central processing unit. The CPU is the computational and control unit of a computer.

- **DAC:-** Acronym for Digital to Analog converter. A device or a process that converts a digital number to a corresponding analog voltage or current.

- **Data Bus:-** A set of signals that carries the datainformation between the processor and the memory and/or I/O devices.

- **DEBUG:-** To correct mistakes in a program.

- **DEBUGGER:-** A software program that assists in debugging a piece of code.vided the softwarewit

- **Development Host:-** A computer system that hosts development software like an assembler,compiler,programmer,debugger,etc.

- **Dongle:-** A mechanism to ensure only authorized users can use a particular software with the help of an electronic lock provided with the software to deter software piracy.The dongle needs to be connected to the PC parallel port,serial port,or the USB to be able to use the software.

- **DUPLEX:-** Term used in communication system that means that both the transmitter and the receiver can send and receive signals at the same time.

- **Embedded controller:-** A piece of hardware including some processor and software that controls a device.

- **Emulator:-** A device that mimics the behavior and functions of another device.A processor emulator.

- **EMI:-** Acronym for electro-magnetic interference. This is a phenomenon by which a device or a system can generate an electromagnetic field in the radio frequency spectrum with the potential to disrupt operation of other electronic components or system in the vicinity.

- **EEPROM:-** Acronym for Electrically Erasable and Programmable Read Only Memory.

- **EPROM:-** Acronym for Electrically Programmable Read Only Memory.

- **FINITE STATE MACHINE:-** A device that stores the status of something at a given time, with some inputs that can change the state and and/or outputs.

- **FLAG:-** A bit used by a program to remember something or to convey binary information to another piece of program.

- **FLASH MEMORY:-** A non volatile memory that can be erased and reprogrammed in units of memory called blocks. The name flash memory means the memory cells can be erased in an electron tunneling process in a flash by removing an electronic charge from a floating gate associated with each memory cell.

- **FPGA:-** Acronym for Field Programmable Gate Array.A large and dense programmable logic device.

- **FSM:-** Acronym for finite state machine.

- **Full Duplex :-** Same as Duplex.

- **Glitch :-** An unwanted transient signal transition from the current level to the other level and back to the original level.

- **HALF Duplex:-** Term used in communication systems that means that either the transmitter can send or the receiver can receive signals at a given time.

- **Handshake signal:-** control and feedback signals used between two(or more) devices to facilitate exchange of data.

- **Hexadecimal:-** A base -16 number system. The numbers go from 0 to 9, A,B,C,D,E and F.

- **Host:-** A computer system acting as a master that provides services to other connected devices or systems.

- **IC :-** Acronym for integrated circuit. A semiconductor chip with many transistor and resistors connected to make an electronic circuit.

- **ICE:-** Acronym for In-circuit emulator. A development tool for developing microprocessor based system and devices. The ICE mimics the operation of the target processor during the development process.

- **IIC or I2C:-** Acronym for Inter–IC communication. A communication bus with only two signals.

- **Infrared:-** Part of the light spectrum that is just above that of visible light in the red end of the spectrum.

- **IRDA :-** Acronym for Infrared Data association.IrDA is an industry-sponsored organization to design standards for the hardware and software used in infrared communication links.

- **Instruction:-** Lowest level command given to the processor by a program.
- **Interrupt:-** An asynchronous signal generated by a peripheral device to the processor that, when asserted, indicates to the processor to take notice and execute a special piece of program called an ISR.

- **I/O:-** Input Output .peripheral devices of a processor to interact with the physical environment around it.

- **I/O map:-** A table containing the addresses, within the I/O space, of the input and output devices of a computer system.

- **I/O space:-** A type of addressing region that allows a processor to connect I/O devices.

- **Instruction pointer:-** A special register in a processor that points to an address in the program memory from where the current instruction is being executed by the processor.

- **ISA:-** Acronym for industry standard architecture .ISA is a bus architecture used in IBM personal computers. It allows connectivity between processor and the associated peripheral circuits and devices.

- **ISR:-** Acronym for interrupt subroutine. A program that is executed by the processor when an interrupt occurs in a computer system.

- **Interrupt vector:-** Address of an ISR.

- **Latency:-** Usually associated with the interrupts in a computer system and refers to the time it takes to takes to respond to an interrupt signal.

- **LED:-** Acronym for light emitting Diode. A semiconductor device that emits light when a voltage of appropriate polarity and value is applied to it.

- **Load-store architecture:-** A processor architecture in which the memory is accessed using only the load and store commands. No other operations are allowed on the memory contents directly.

- **Logic analyzer:-** an instrument to observe digital signals as a function of time. The logic analyzer has a flurescent or LCD display that signals.

- **Logic gate:-** A digital circuit that has one or more inputs and an output.It performs a logical operation (such as AND,OR,XOR,XNOR,NAND,NOR,NOT) on the inputs and produces a result on the
- Output.

- **Microcomputer:-** A microprocessor and associated support circuitry, peripheral I/O components,and memory (program as well as data) put together to form a small computer specifically for data acquisition and control applications.

- **Microcontroller:-** A microcomputer on a single chip.

- **Microprocessor:-** A central Processor unit(CPU) on a single chip.

- **MNEMONIC:-** An abbreviation, an aid for remembering the code of a processor.

- **NVRAM:-** Acronym for Non volatile Random access memory. RAM with battery backup for retaining the contents of the RAM when the main power is put off.

- **Oscilloscope:-** An instrument to observe electrical signals as a function of time.It has a fluorescent or LCD display for observing the signals.

- **Object code:-** A program that a processor can execute directly.

- **Pipeline:-** Refers to an intrernal implementation of a processor in which the instructions are continuously being fetched by a section of the processor and placed in a queue for execution section section of the processor, which in turn places the results in an output queue to be stored back to the designated destination. A nonpipelined processor, on the other hand, fetches an instruction, executes it, and stores the results before fetching the next instruction. Pipelining improves overall execution speed because of overlapping of the various stages of program execution.

- **PLD:-** Acronym for Programmable Logic Device. A digital circuit whose functionality can be changed as per the logic required. The PLD has a combination of AND,OR and NOT gates connected through a network. By choosing the right gates and the right interconnects, the PLD can be made to implement any logic function.

- **Program Counter:-** Same as Instruction Pointer.

- **PWM:-** Acronym for Pulse Width Modulation. In PWM, the pulse width of the frequency is changed while keeping the frequency constant. This changes the DC value of the signal. For low width pulse, the DC Value is smaller than a pulse of higher width.

- **RAM:-** Acronym for Random Access Memory. A memory device any part of which can be accessed directly. In the early days of computing, this was contrasted with tape memory, which was sequential access memory. Now RAM usually means some sort of volatile, read/write memory.

- **RISC:-** Acronym for Reduced Instruction Set Computer. A type of computer architecture with a small number of minimum instructions, characterized by a very regular instruction structure of fixed length, a load-store approach to memory access, and a large number of register. Contrast it with CISC.

- **RESET:-** To restart. A signal in a processor that initializes the internal register and control circuit to a default value and starts executing the program from the first memory location.
- **RESET VECTOR:-** Address of reset code.
- **RESET ADDRESS:-** The address that the processor first accesses for the first instruction, after the reset signal is applied.
- **RESET POINTER:-** Address of the reset code.
- **RESET CODE:-** The program that the user writes as part of the system initialization.

- **Rs-232:-** A protocol for serial asynchronous transfer of data.

- **Simplex:-** Used in communication. Simplex refers to communication in only one direction and not in the reverse direction.

- **Simulator:-** A software for monitoring the execution of a program. Simulator allows the user to execute the program instruction and inspect register, memory and I/O port contents. The simulator allows section of program to run at full speed by placing break points.

- **SPI:-** Acronym for serial peripheral Interconnect. A four-wire serial synchronous serial communication protocol between two devices or ICs.

- **STACK:-** a read/write storage space used for storing the return address of a calling program. The stack has a **last in first out** structure. The value written last is read first.

- **Stack Pointer:-** An address register that points to the current top location in the stack.

- **Startup code:-** A section of a program that is executed for system initialization at the very beginning.

- **Timer:-** A counter that is incremented by a clock signal.

- **Target device:-** Refers to the processor in the target system that is under development.

- **UART:-** Acronym for universal asynchronous Receiver Transmitter.A serial communication device or IC that converts a byte of data into serial bits and transmits it out at a certain rate.Similarly, It recevices incoming serial bits and assembles these bits into a byte for the host.

Chapter 22 References

[1] Atmega 16 Datasheet

[2] www.wikipedia.org

[3] http://www.cmosexod.com/micro_uart.htm

[4] http://www.best-microcontroller-projects.com/hardware-interrupt.html

[5] http://www.avrtutor.com/tutorial/interrupt/interrupts.php

Experiment Book

Table of Contents

BREADBOARD

Breadboard is used to make circuits. But mostly after testing your circuits on breadboard you will be making PCB. But I never made PCB for any of my circuits. Normally everyone says that if you connect on breadboard then wires may get loose and circuit will get disturbed due to shock.But no such problem occurred to me, all you have to do is to do a good wiring.Then you can gain time for making PCB's.

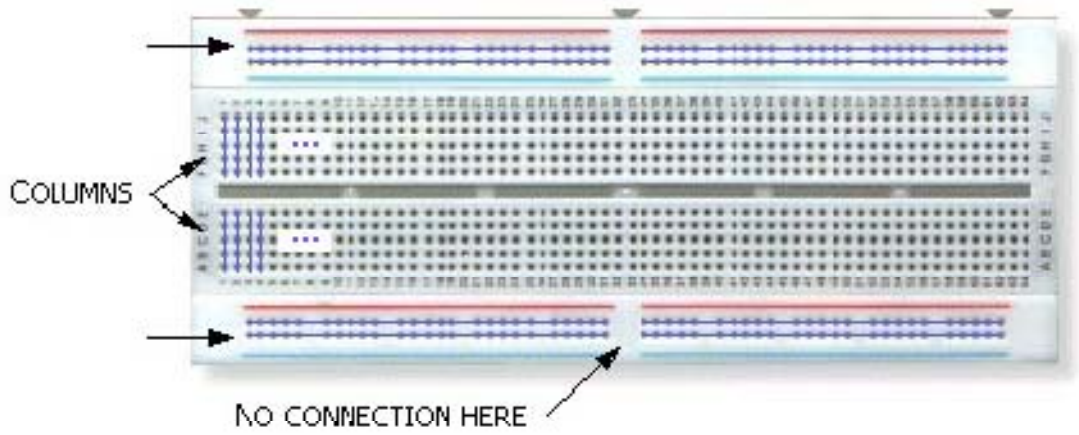
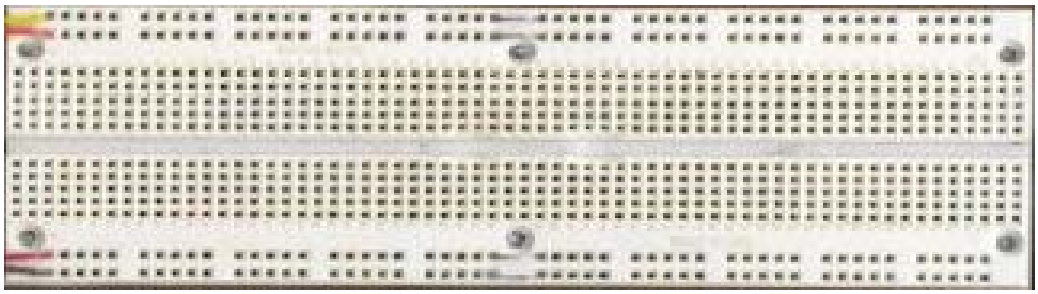
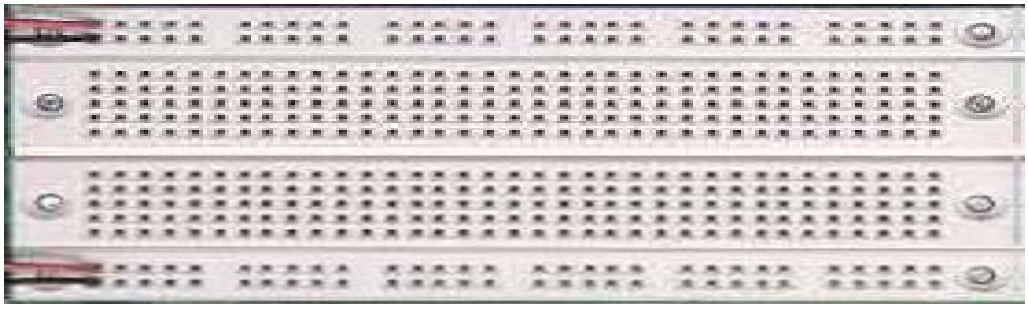


Figure 1: How the holes in a breadboard are connected electrically

Above diagram shows how breadboard connections should be made.So all you require is to do a good wiring. First I will tell about which breadboard should you use. The breadboard is different mainly according to the size of their holes. The breadboard in Figure.1 has the smallest hole size.

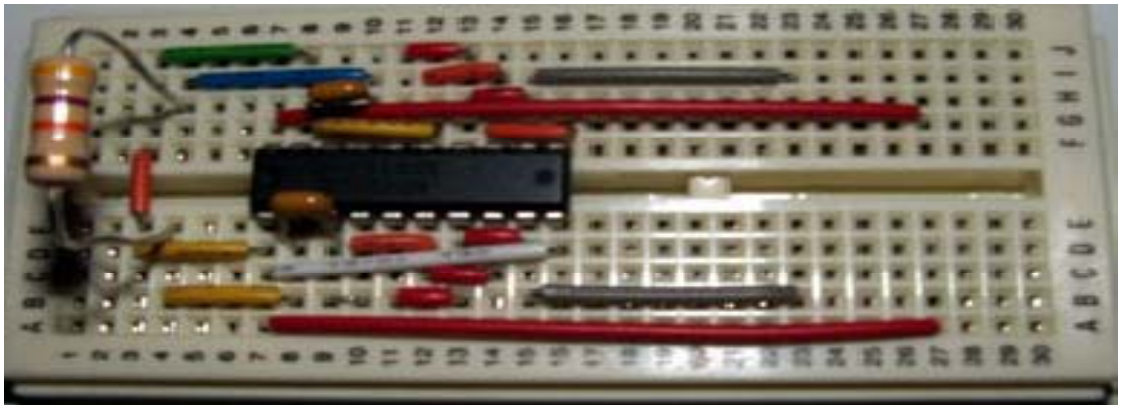


This breadboard has a medium sized holes.I mostly prefer this one. But I have not seen this breadboard nowadays. The one which is available nowadays is given below. Breadboards costs from Rs.80-120(depends on place where you are in India). The main problem with small holes is that, it will be tough to insert IC's like 7805 ,power transistors so on.Even there is problem with size of wires also.

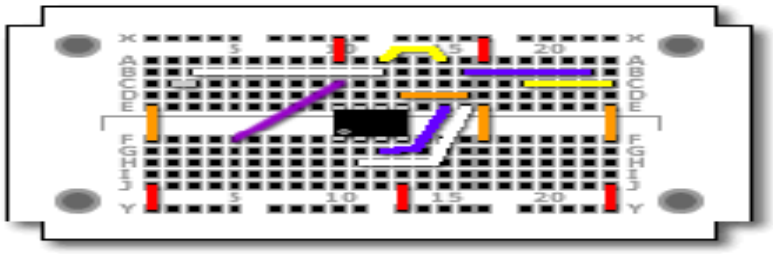


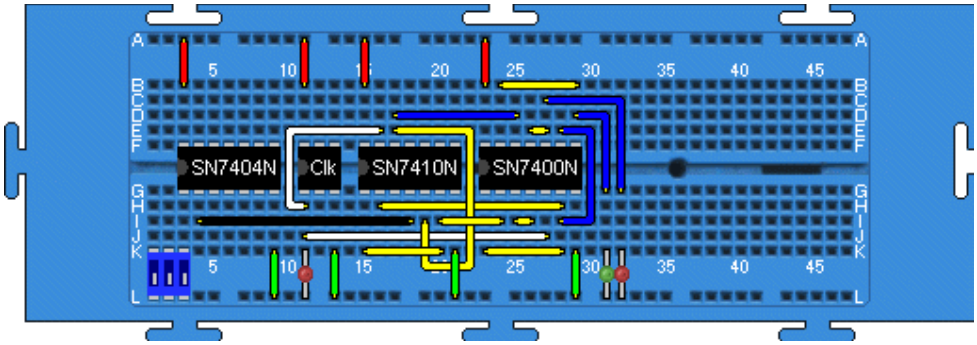
WIRING

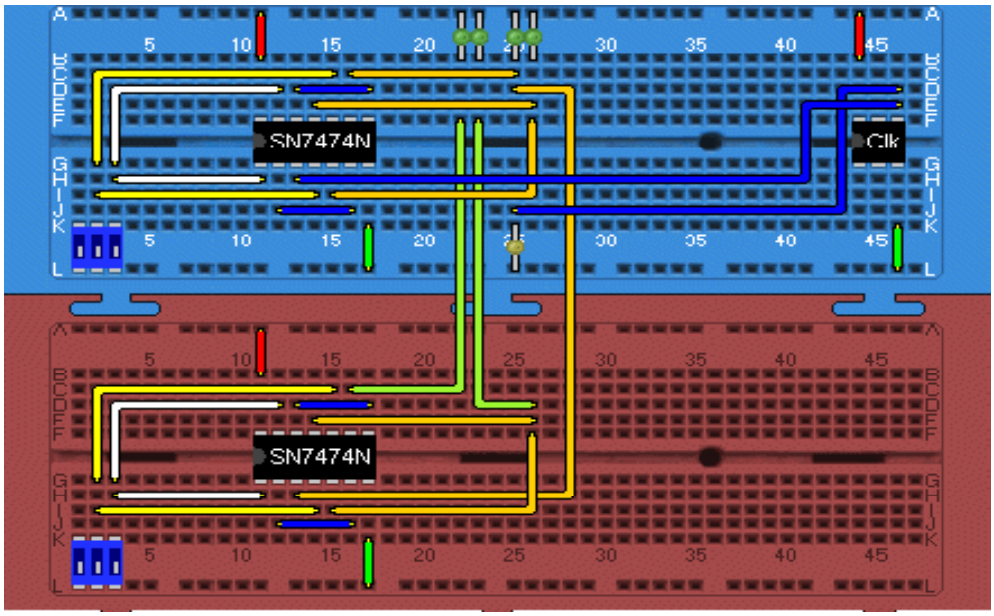
Following figures show good wiring practices you should follow so that your circuit won't be disturbed by any shocks.

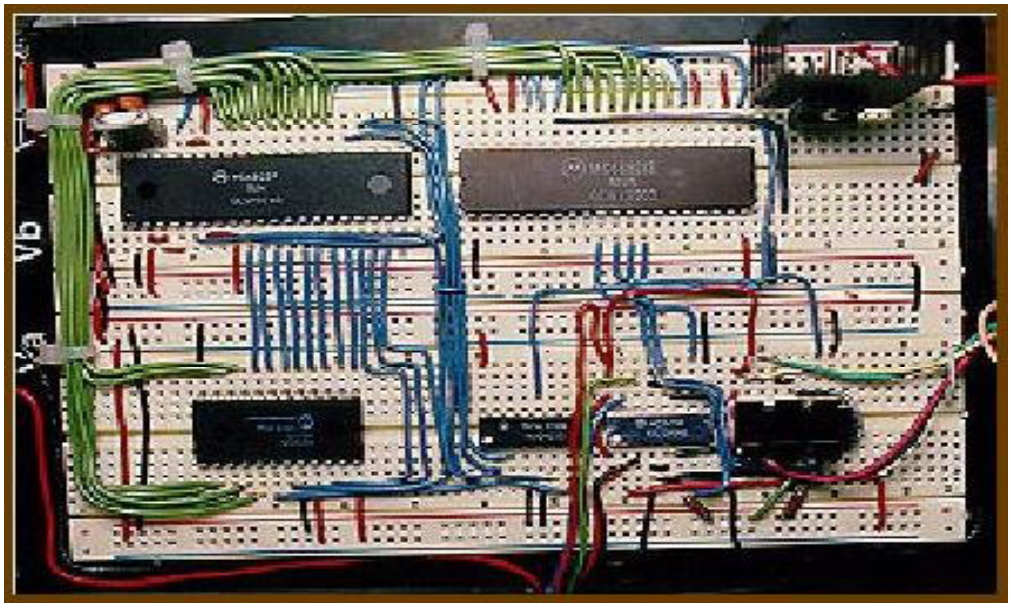


In this you can see that the length of the wires used are of exact length between two points. If you do this type of wirings then no problems occur. But in this you can see that resistor is not properly inserted, for this you should cut the leads of the resistor so that its body is just touching(or touching the breadboard).

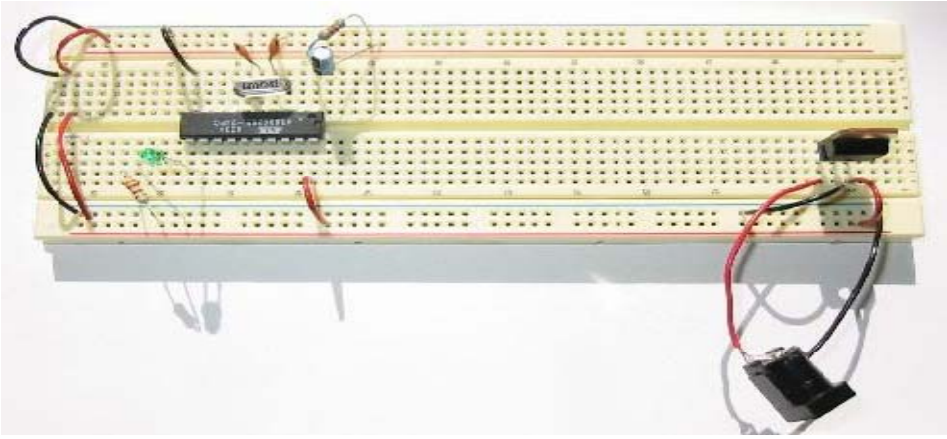








Above figures shows how to make good connections. In last one you can see how they made connections so that no problems will occur. Below you can see what connections you should not have to do.



PROBLEMS OCCURING WITH BREADBOARD

1. As I said above some breadboards will be difficult to insert IC's like 7805,LM317 etc.. due to the small sized holes.

2. I bought a new breadboard for Rs.80 in which some parts of the breadboard is not working. So you should be careful about it.

3. Some part of the breadboard may suddenly create problem. This problem will mostly eat your time. I have connected full circuit for my project and it was working properly two months ago. The ckt consist of a 7805 which convert adapter DC to 5V. Now when i switch ON power supply, the circuit is not working. On examination i found that the output voltage of 7805 is 1.1V even though input voltage to 7805 is >7.5. I was surprised to see this because i had done same ckt a month ago and no changes made in ckt. Then i used another breadboard specially for 7805 to make connections, surprisingly it is working fine giving 4.8-5V.Then i connected the output from that breadboard to my original board where ckt was connected. It is working fine. Then I again tried in same position it is not working giving output of adapter 3-4V and output of 7805 1.1V. I doubted if the voltage regulator input is low to regulate, so I increased the adapter to 9V from 7.5(my adapter has varying voltage starting from 1.5V,3V..). Then the light of the adapter went off. Then I again increased voltage to 12V. Then also light is off. Now I decided for a new position of breadboard, where it worked properly. Then I increased voltage up to 13.5V.For all these input voltages to 7805 the input of 7805 still remained 3.5-3.7V and output .7-.8V. Finally I got a position of the breadboard where it worked fine. The problem was of the position in the breadboard which i used.

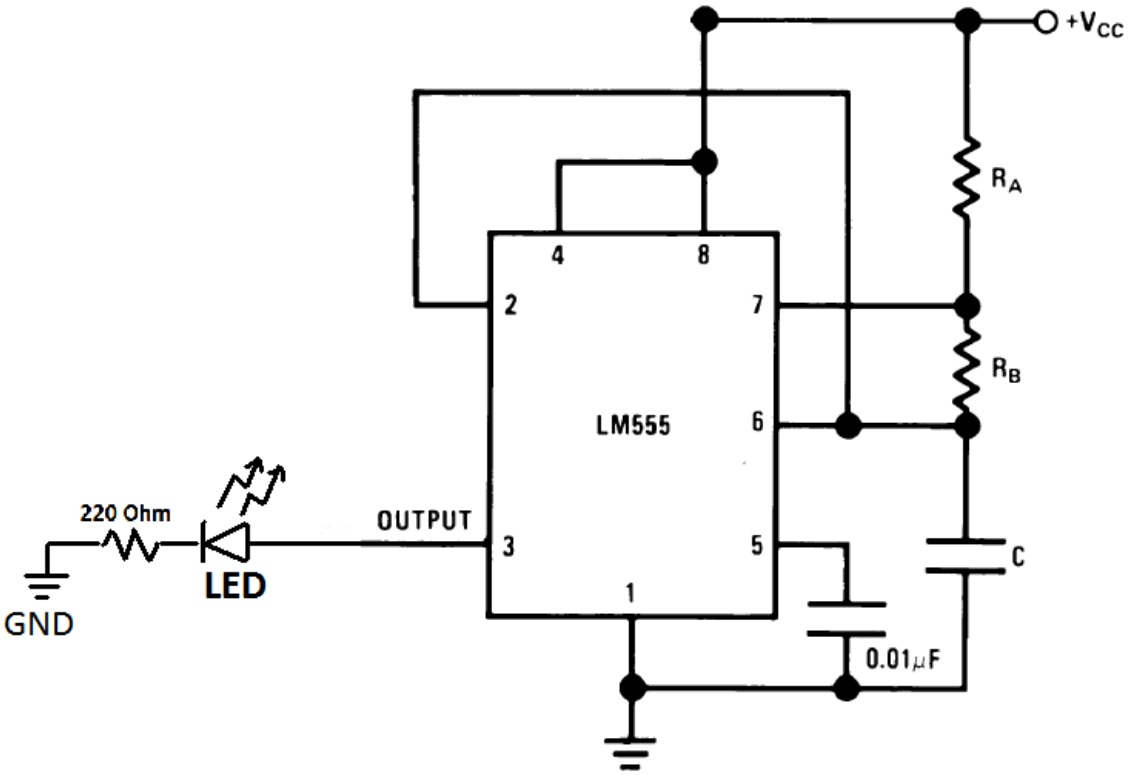
4. One similar problem occurred when I connected LM324 with its input avariable resistor. I rotated the knob of the potentiometer(it has one end on Vcc=10v, other end on ground and middle end is connected to the LM324), the voltage output of the potentiometer is suddenly increasing from .5 to 8.8V suddenly with a small rotation of the knob. I tried to rotate shaft by connecting the middle end to another portion of the breadboard, there it worked fine. I used the same portion of the breadboard where i connected first and tried the same after removing LM324, then also it is working fine. Then i again connected LM324 in same position, still the old problem came. Then I changed the full circuit to another position of the breadboard, it worked fine.

Experiment 1: Monostable and Astable Multivibrator using the 555 IC

Problem Statement

Using 555 IC design an astable multivibrator with Time period = 1sec and duty cycle = 0.75.

Circuit Diagram



Calculation

Take C=100uF

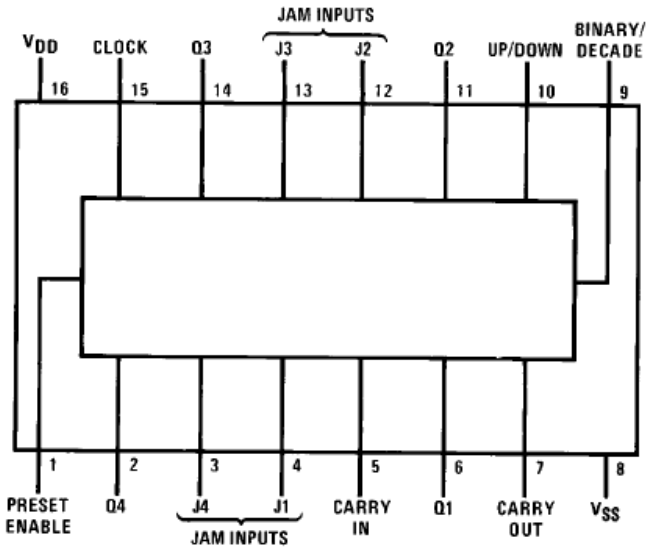
Calculate R_A and R_B for T=1sec and duty cycle=0.75.

Hint: $T = C(R_A + 2R_B)ln2$ and $DS = \frac{R_A+R_B}{R_A+2R_B}$

Experiment 2: Counter on Seven Segment Display

Problem Statement

Part A: Make a Counter using IC 4029 which counts from 0-9 and check the result using LED's.



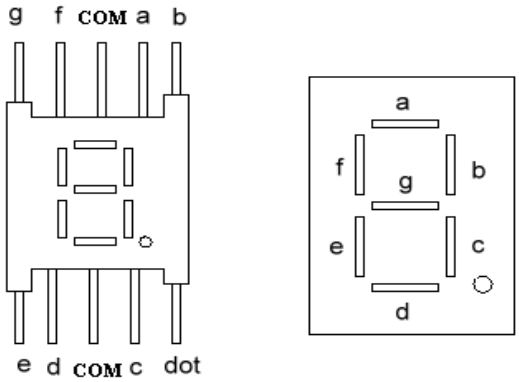
4029 pin configuration

Note: Logic High→+5V Logic Low→0V

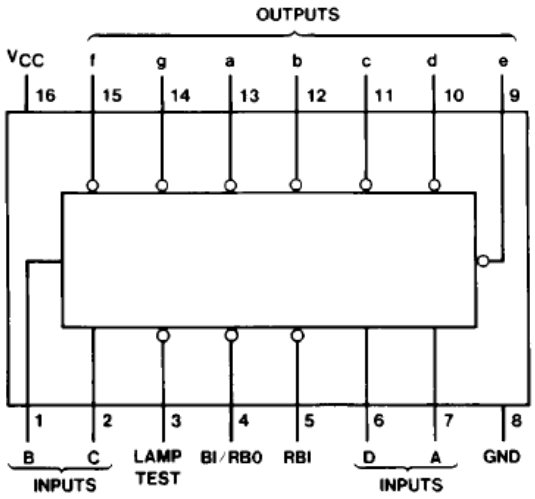
Steps to be performed:

1. Set pin number 1,5,8 and 9 to logic 'low'.
2. Set pin number 10 and 16 to logic 'high'.
3. The output in astable multivibrator from IC555 serves as the clock for the counter circuit. Connect the output of the IC555 to pin 15 of 4029.
4. Connect pins 2,6,11 and 14 to the ground through a 220ohm resistor and an LED. These serve as output pins of the counter. The sequence from Most Significant Bit(MSB) to Least Significant Bit(LSB) is pin 2,14,11,6 i.e. Q4,Q3,Q2,Q1.

Part B: To view the output of the 4029 counter on a Seven Segment Display using the IC7447.



Seven-Segment Display



IC7447 pin configuration

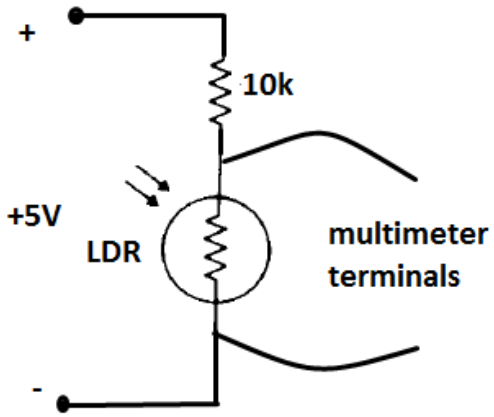
Steps to be performed:

1. The pin configuration of the seven segment display and IC7447 is shown in the figure. Connect the pins 9 through 15 of IC7447 to the corresponding letter pins of the seven segment display.
2. Connect the output of the 4029 to the input pins 1,2,6 and 7 of the 7447IC with A being the LSB and D being the MSB. So the input to the IC is DCBA.
3. Make the pins 3,4 and 5 of the IC7447 logic 'high'.
4. Connect +5V to pin16 and ground (0V) to pin8 of the IC7447.
5. Also ground both the 'COM' pins of the seven segment display.

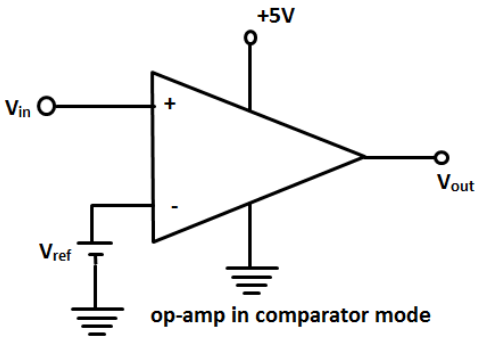
Experiment 3: Introduction to LDR and Operational Amplifier

Problem Statement

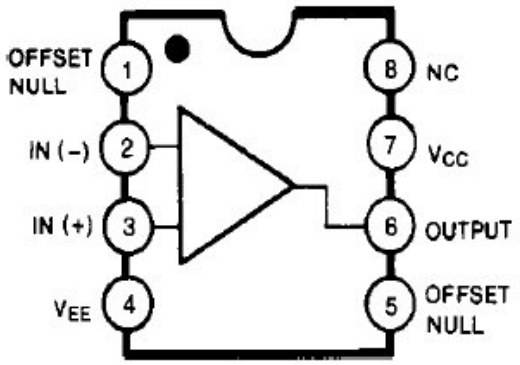
Check the change in potential across an LDR using a multimeter when light falls on it. Digitalize the analog output using an Op-amp in Comparator mode.



- Assemble the above circuit and measure the voltage across the LDR with a digital multimeter with and without any light incident on it.



op-amp in comparator mode



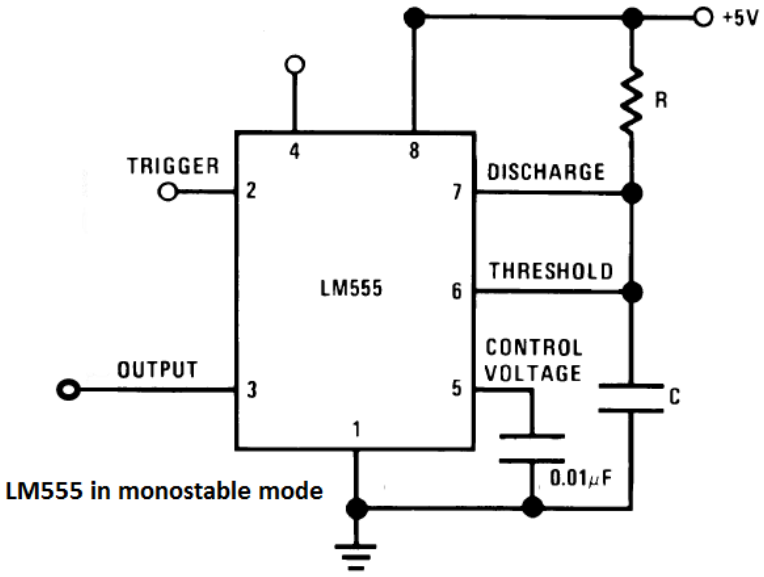
uA741 op-amp pin configuration

- Assemble the above shown Op-amp comparator circuit and set V_{in} as the voltage across the LDR in the previous circuit.
- Set V_{ref} as the voltage at which the comparator output is 'high' with the light incident on LDR and 'low' with no light incident on it.

Experiment 4: LDR based counter

Problem Statement

Design a LDR based counter circuit to increment by one whenever light falls on the LDR. IC 555 is used in monostable mode so that counter it is triggered only when there is a falling edge at its Trigger pin.



LM555 in monostable mode

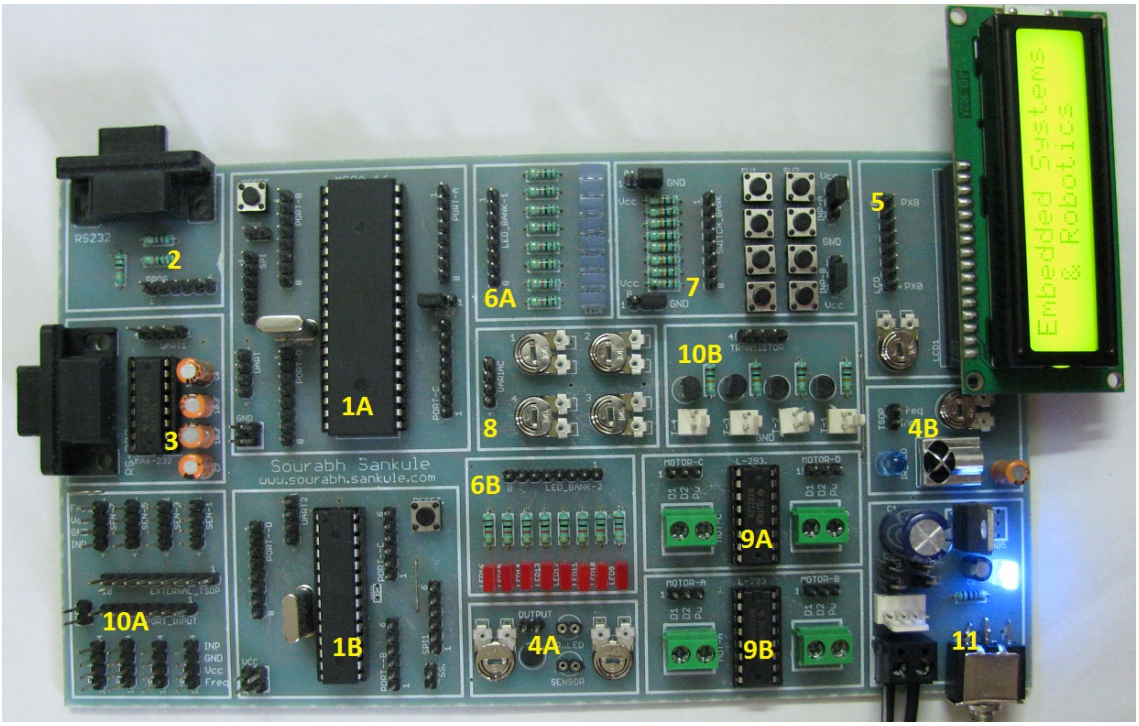
- Assemble the above circuit, take R=9.1k and C=100uF. (T=RCln3)
- Give the output of the Op-amp in Experiment 3 to pin2(Trigger) of LM555. This serves as a method to trigger monostable multivibrator whenever a barrier is placed in front of the LDR.
- Assemble the counter and the seven segment display circuit as in experiment 2 and connect the output pin3 of the multivibrator circuit to pin15(Clock) of IC4029.
- The resulting system should behave as a counter which counts the number of times light has been blocked in front of the LDR.

Overview of AVR based Trainer Board

INTRODUCTION

The Embedded Development Board provided to you has been designed keeping in mind the needs of engineering students and enthusiasts so that they can match up to the present industry standards on latest developments in the rising field of Embedded Systems Design/Development and is better than any existing boards available in the market.

The board has been intensively designed to cover all existing features of AVR microcontrollers and is ready-to-use with its supporting equipment mounted in the periphery.



FEATURES

AT mega 16 & 8 microcontrollers with facility of

- On-board serial programmer.
- Option for In System Programming (ISP) based on externally provided STK500 platform.
- On-board Digital and Analog IR sensors.
- RS232 serial interface for establishing connections between the mcu and computer system.
- On-board LED arrays, keypad and external connections for interfacing a number of devices.
- On-board motor driver circuits etc.

HOW TO USE THE MANUAL

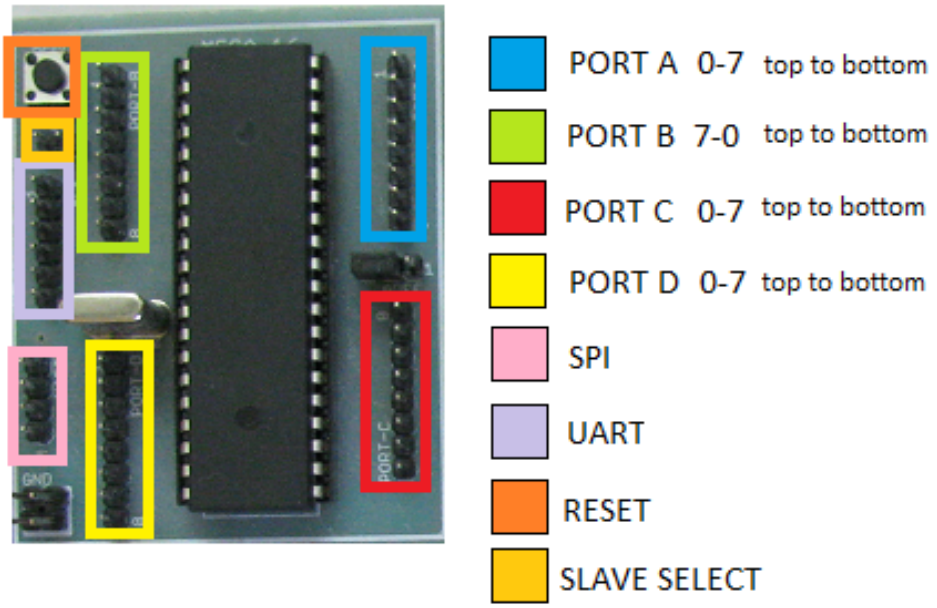
The board has been divided in modules and each module has been thus described with proper part-details, circuit diagram and DIY connection diagram.

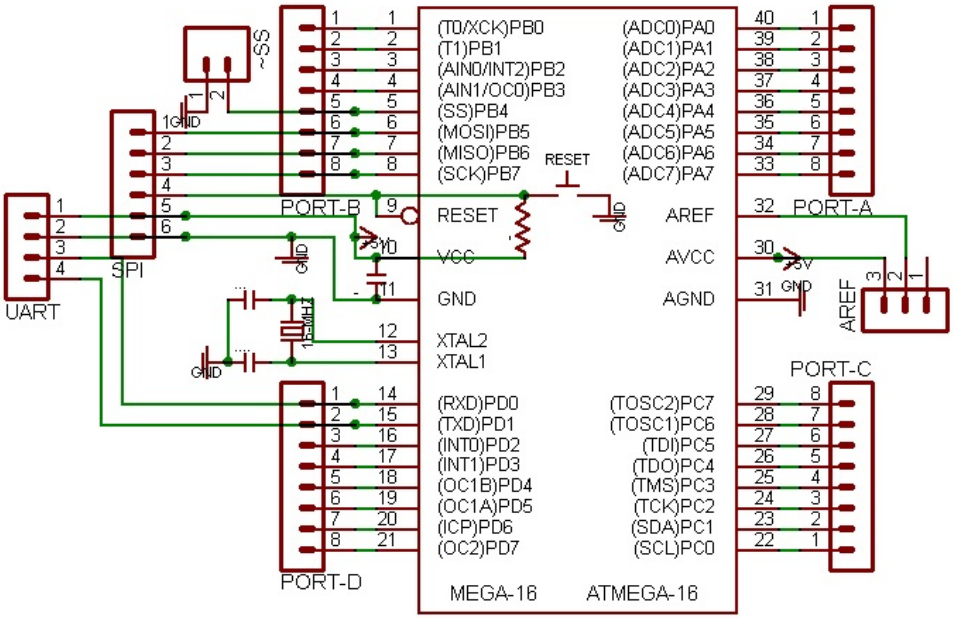
MODULE LIST & DISCRIPTION

1A. Microcontroller Module: Primary

Features an ATmega16 microcontroller with

- Connectors for Ports A,B,C & D
- Aref (reference voltage)
- SPI
- 16MHz external crystal oscillator UART Slave select
- Reset switch
- Ext. GND



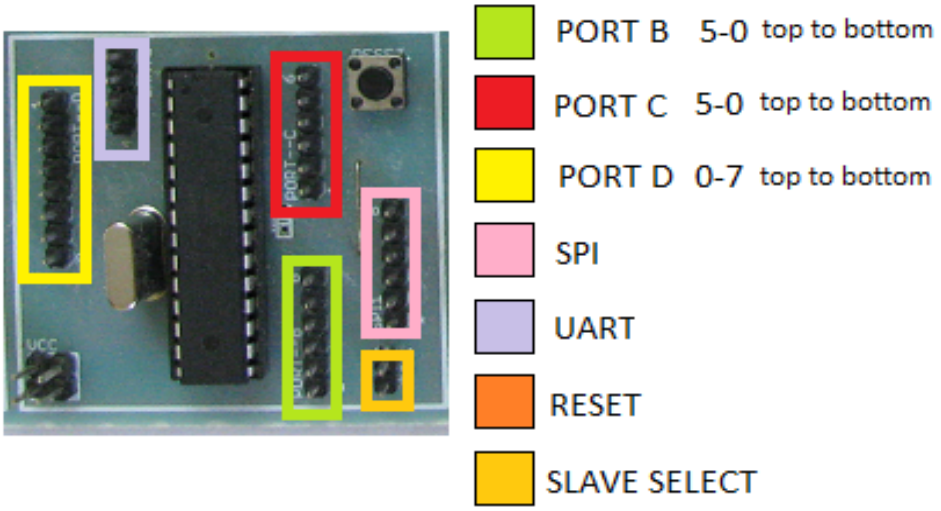


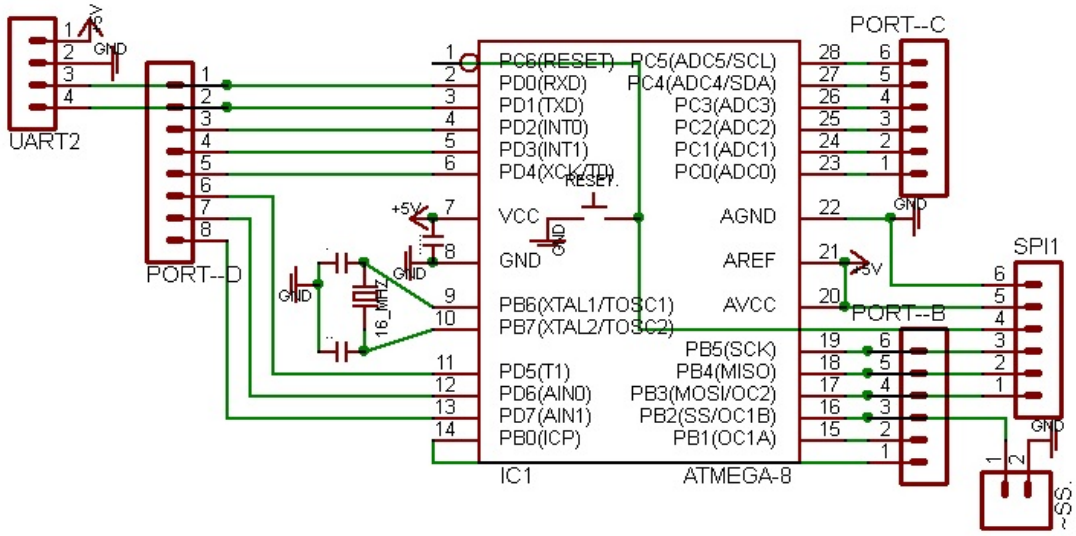
Schematic for mod 1A

1B. Microcontroller Module: Secondary

Features an Atmega 8 microcontroller with

- Connectors for Ports B,C & D
- SPI (pin 6,7,8,9,10,11)
- 16MHz external crystal oscillator
- UART
- Slave select
- Reset switch
- Ext. GND

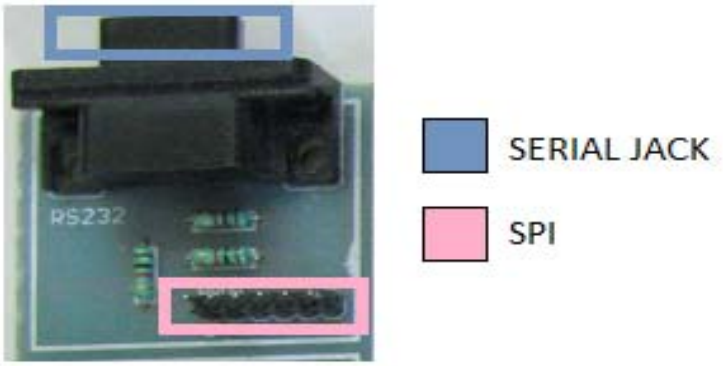


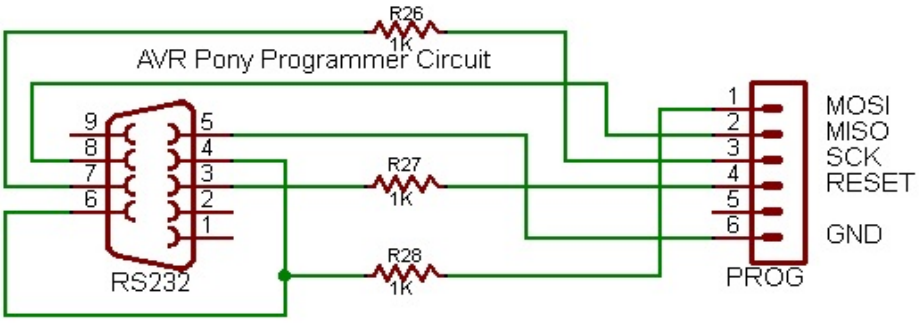


Schematic for module 1B

2. Serial Programmer Module

- RS232 Serial Port
- Connector for SPI



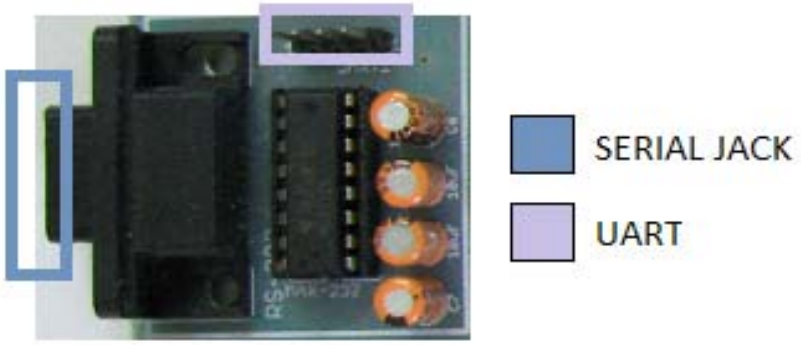


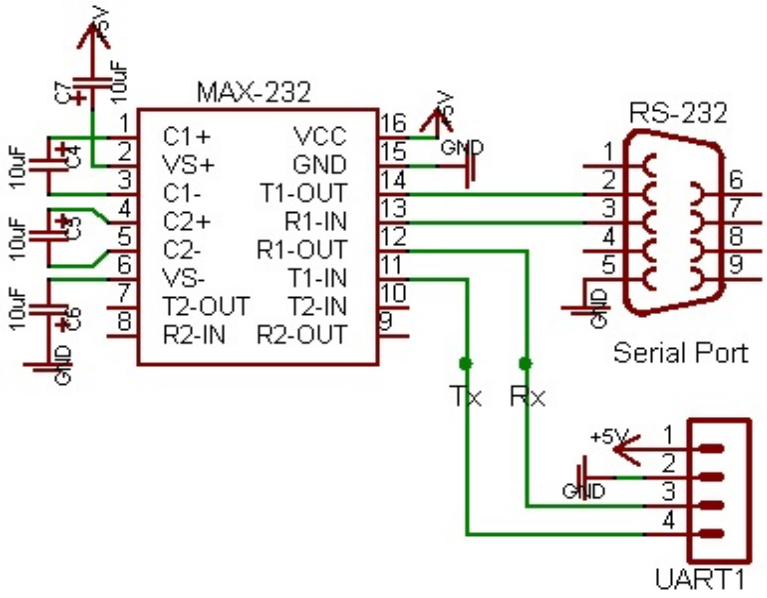
Schematic for module 2

3. Serial Interface Module

- RS232 Serial port

- Connector for UART
- MAX232 IC & circuitry

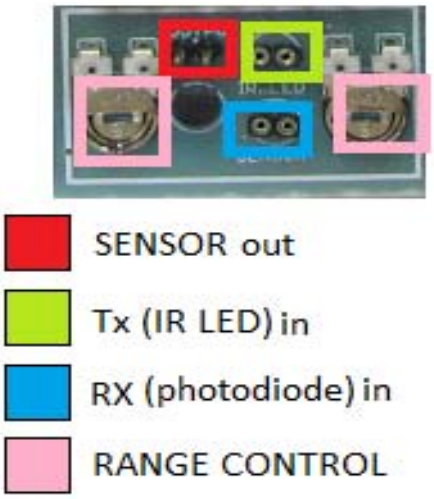




Schematic for module 3

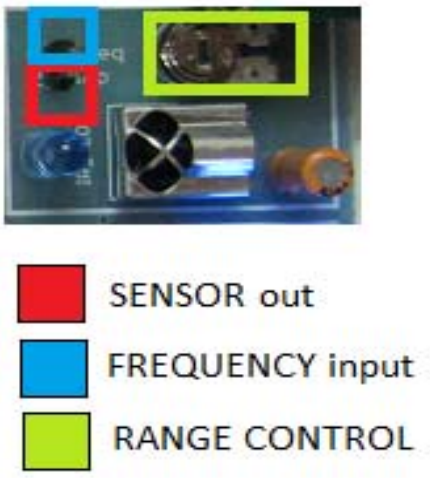
4A. Sensor Module: Analog

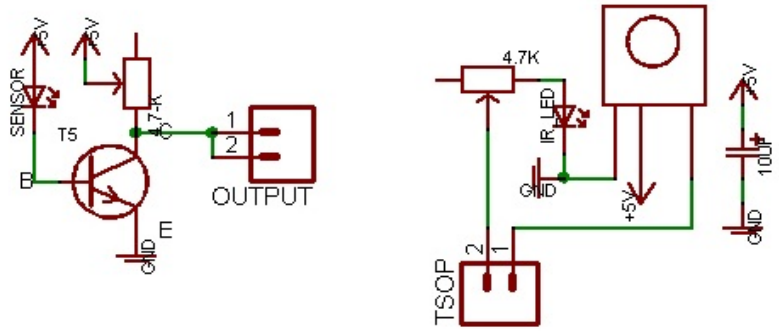
- Tx Rx
- Associated circuitry
- Connector for Output



4B. Sensor Module: Digital

- IR Tx LED
- TSOP 1738 IR Rx
- Associated circuitry
- Connector for output

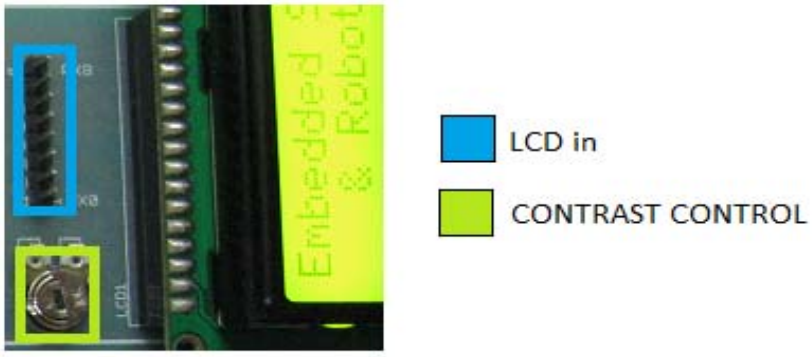


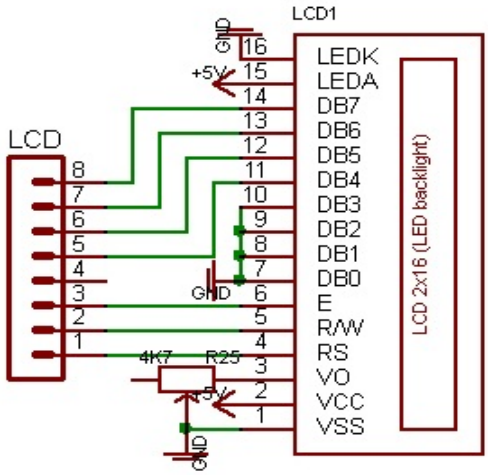


Schematic for modules 4A & 4B

5. LCD interface Module

- 16 pin LCD connector
- PORT connector
- Contrast control

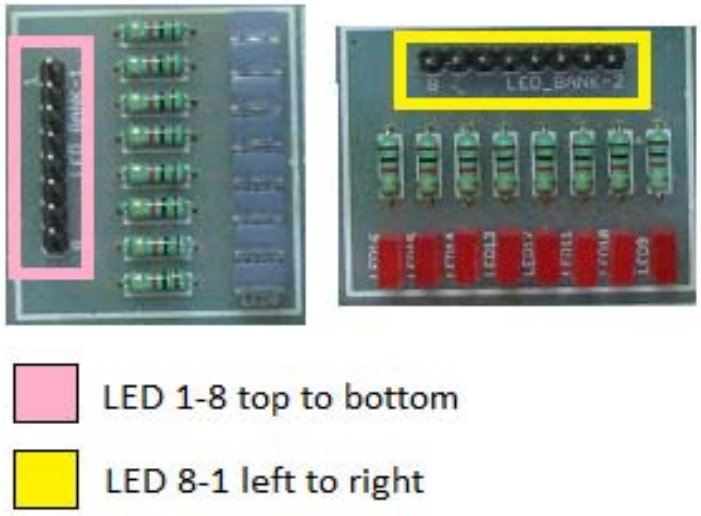


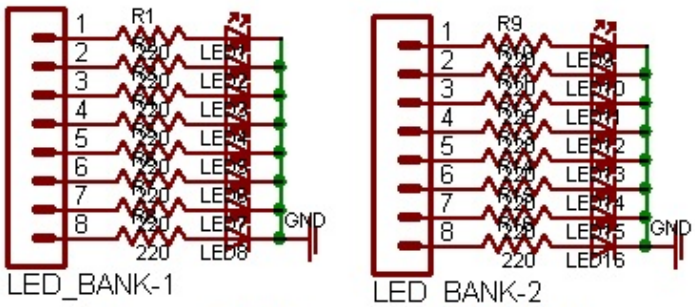


Schematic for module 5

6. LED array modules

- Features 8 LED array
- PORT connector
- Associated circuitry

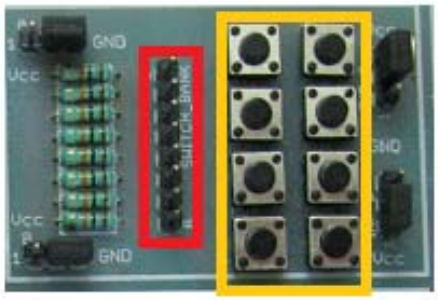




Schematic for mod 6A & 6B

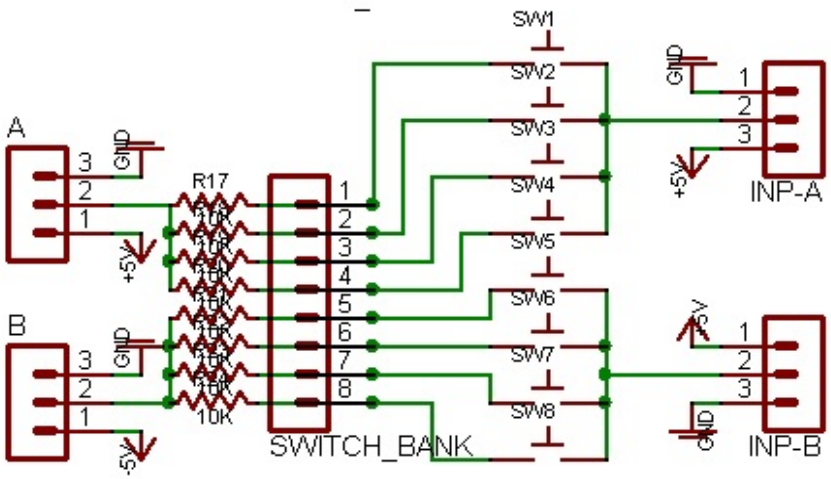
7. Switchpad Module

- 8 Switch-pad
- PORT connector
- External VCC/GND



SWITCHES left to right from top 1 2,3 4,5 6,7 8

SWITCH BANK out 1-8 top to bottom



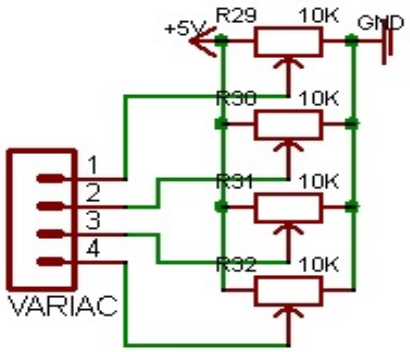
Schematic for module 7

8. Variac Module

- 4 Variacs for ADC and misc. applications
- Output Pins for interfacing

VARIAC out 1-4 top to bottom

VARIAC 1 2, 3 4



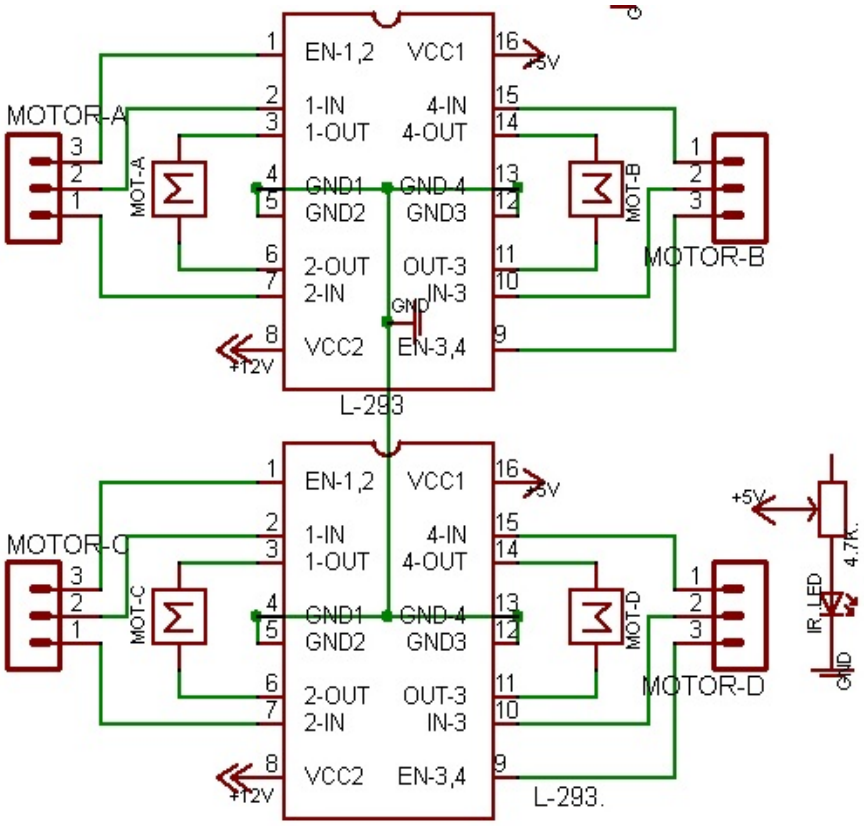
Schematic for module 8

9. Motor Driver Modules

- IC l293D
- Output pins for motors
- Connectors for speed & direction control

MOTOR DRIVER out: direction , speed

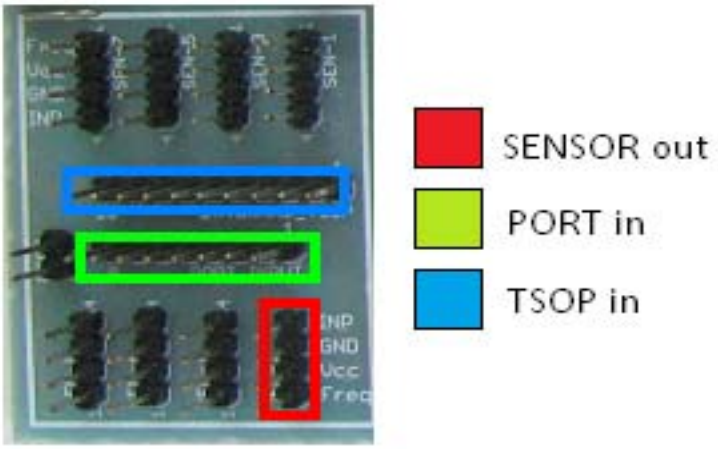
MOTOR out

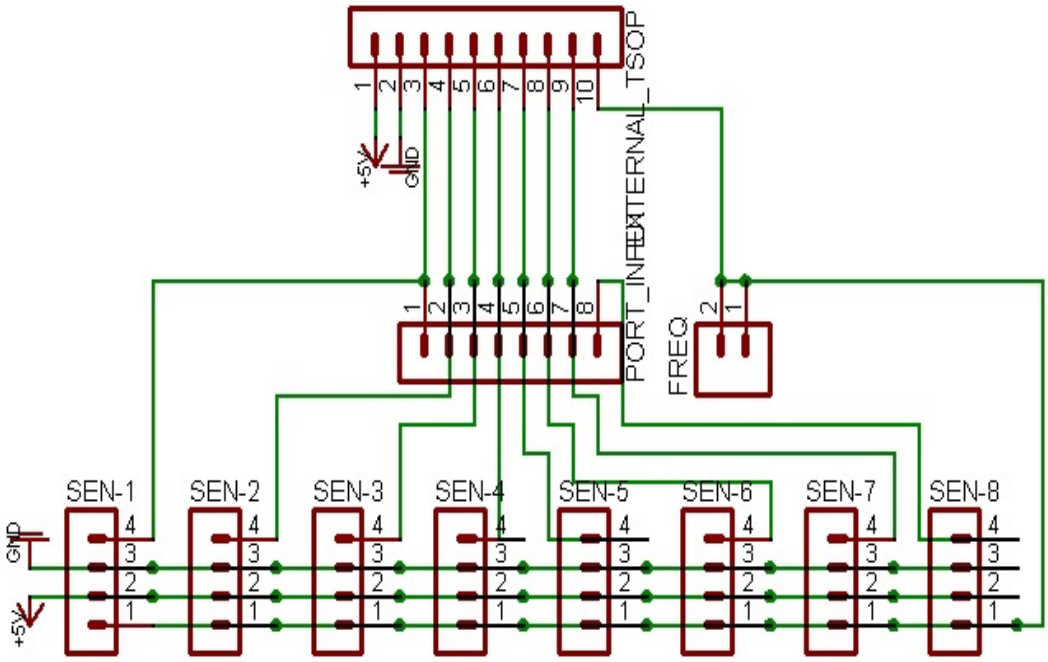


Schematic for module 9A & 9B

10A. External Connection module for digital IR sensors

- Supports as many as 8 TSOP based IR sensors
- PORT connectors for interfacing with atmega

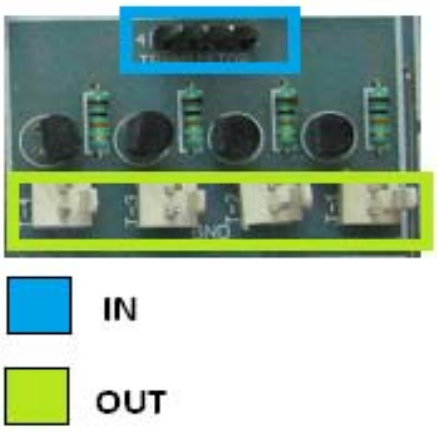


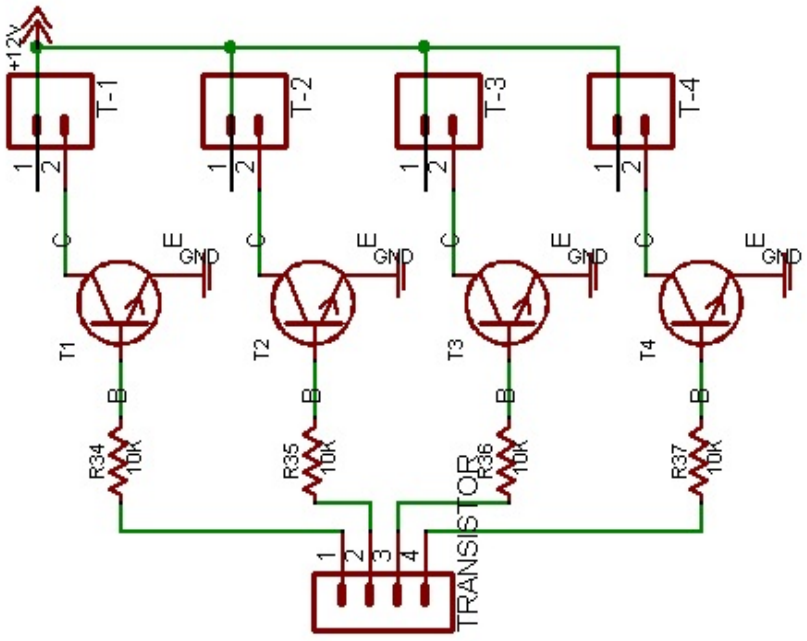


Schematic for module 10A

10B. External Connection module for Buzzers, relays etc

- Supports variety of devices for interfacing purposes**.**
- For example variety of relays, buzzers.
- NPN transistors are used as simple switches.

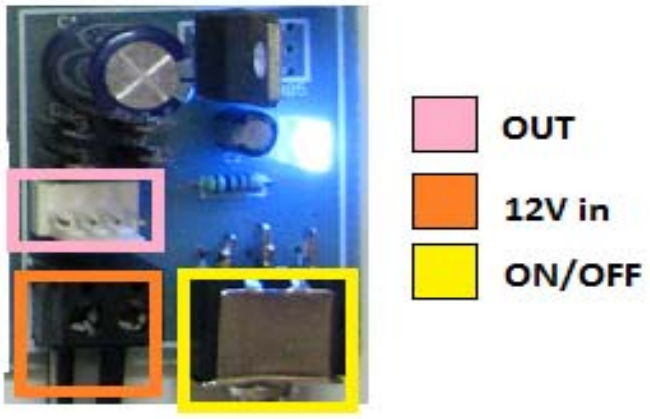


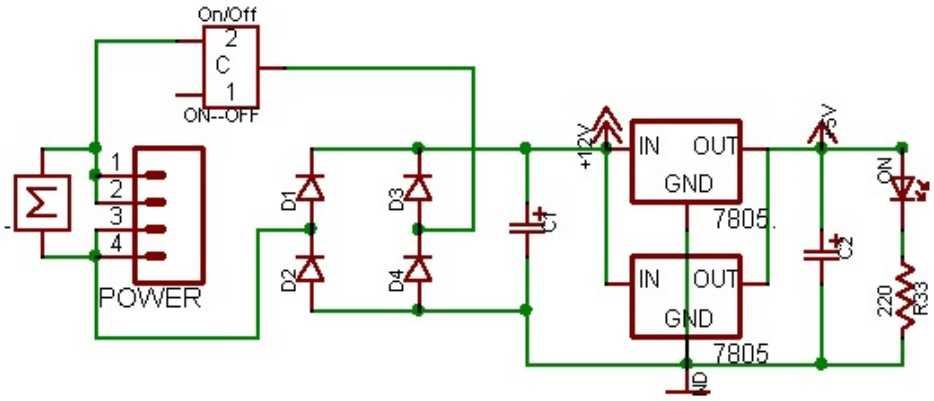


Schematic for module 10B

11. Power supply module

- IC 7805
- Associated circuitry
- Output for 5V supply out



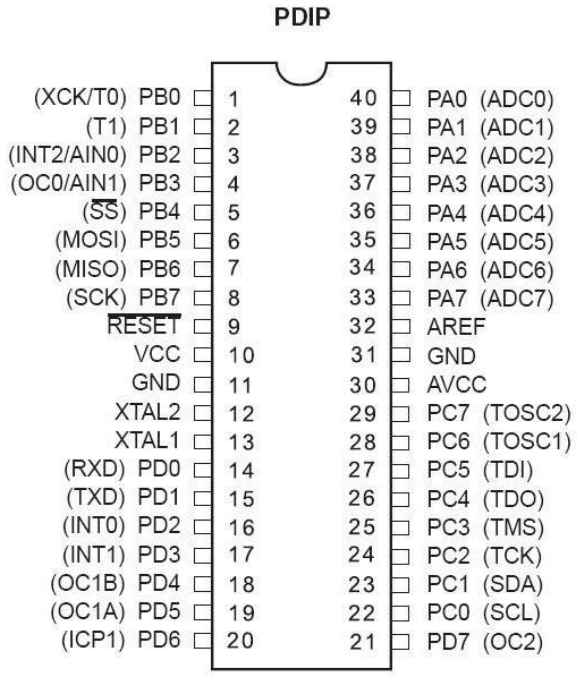


Schematic for module 11

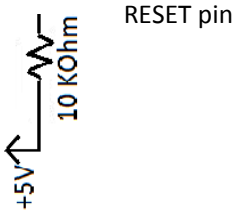
EXPERIMENTS IN EMBEDDED SYSTEMS

Experiment 1: Digital Input/Output in a Microcontroller

Familiarizing with Atmega 16



- Above shown is the pin configuration of Atmega 16 IC.
- It is a 40 pin IC with pin numbers starting from the left side of the U-shaped notch.
- **Pin numbers 11 and 31 should be connected to ground**.
- **Pin number 10 is connected to +5 volts**.
- **Pin numbers 32(AREF) and 30(AVCC) are shorted and connected to +5 volts.**
- **Pin number 9(RESET) should be connected to +5 volts through a pull-up resistor as shown below.**

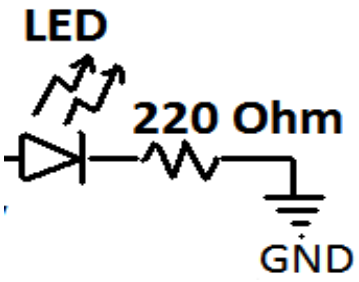


Part A: To glow a visible light LED using Atmega 16

Problem Statement

1. Connect a LED to pin number 40 i.e. PA0 of Atmega.

2. Connect the positive end (longer leg) of the LED to the pin and the negative end (shorter leg) to ground via a 220 Ohm resistor (as shown).



3. Program the IC to switch on and off the LED after certain time intervals.
4. Consider module 1A/1B for mcu and module 6 for LED array connections.

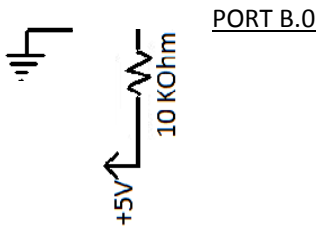
Circuit Overview

Atmega-16 LED

Part B: Using a reset switch to operate an LED

Problem Statement

1. Make the LED connections as in previous experiment.
2. Connect a reset switch to pin number 1 i.e. PB0 of Atmega.
3. Connect one leg of the switch to ground and the other leg to the pin pulled up by a 10K Ohm resistor (as shown).



4. Make PB0 as input pin and program the IC to control the LED with the switch.
5. Consider module 1A for mcu, module 6 for LED array and module 7 for switch pad connections.
6. Now integrate as many switches and LEDs as you want and glow them using switches by making changes in the program code.

Circuit Overview

Atmega-16

LED

SWITCH

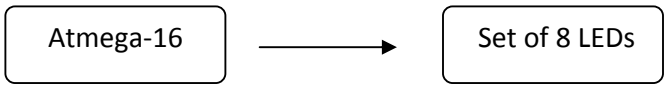
Part C: Making different LED glow patterns

Problem Statement

1. Connect 8 LEDs to the entire PORT C of Atmega i.e. pin number 33 to 40.
2. Each LED is to be connected in the manner shown before.
3. Program the IC to display different LED patterns using the PORT C I/O pins.
4. Some of the patterns for practice are :
    - LEDs glow in a line with certain delay in between.
    - Alternative glowing LEDs.
    - Converging pattern.
    - Diverging pattern.

5. Consider module 1A for mcu, module 6 for LED array connections.

Circuit Overview

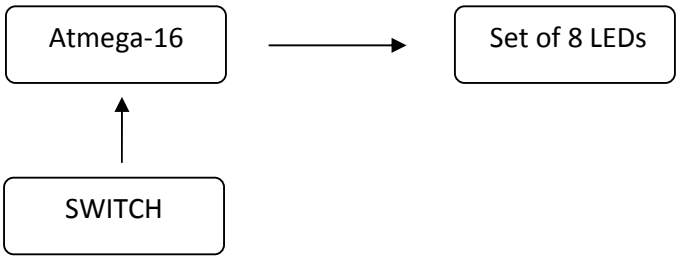


Part D: To change pattern of glowing LEDs using switch

Problem Statement

1. Make the LED connections as before.
2. Make the switch connections to PB0 as before.
3. Program the IC to display different LED patterns on the press and release of the switch.
4. Consider module 1A/1B for mcu, module 6 for LED array and module 7 for switch pad connections.

Circuit Overview



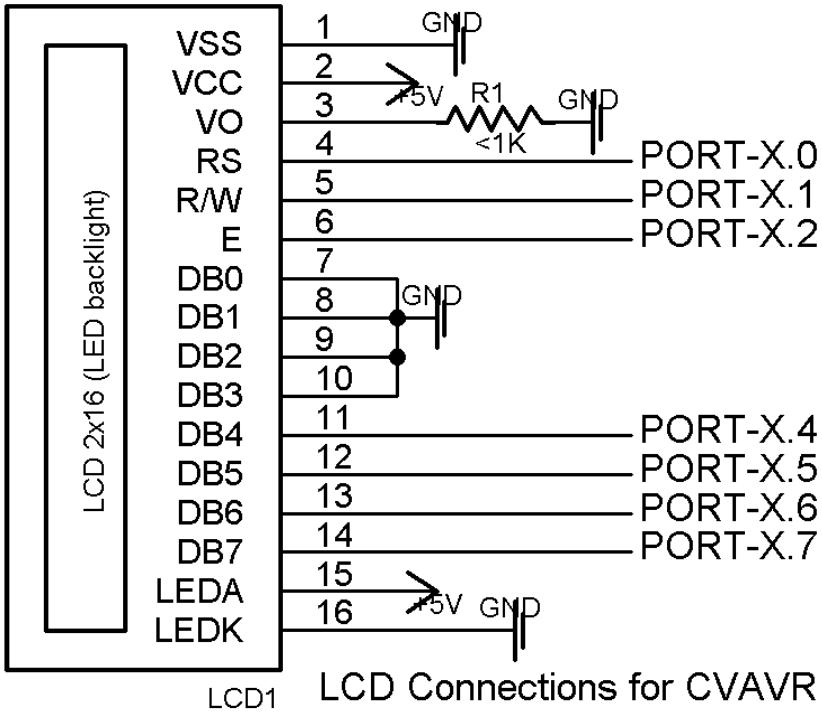
Experiment 2: To interface a 16x2 LCD with Atmega and implement the LCD functions

LCD connections

- Connect a 16x2 LCD with PORT C of Atmega as per the connections shown in module 5.

Actual connections for 16x2 LCD already done on Board

Note: - Port X=C



Part A: To print a constant string on LCD

Problem Statement

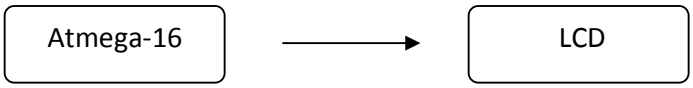
1. Initialize the LCD at PORT C in the CVAVR graphical interface.
2. Program the Atmega to do the following :

    - STATIC PRINTING- Print your name at different locations of the LCD.

Some of the useful LCD functions:

- lcd_clear () – Clears the LCD.
- lcd_gotoxy(x, y) – Takes the cursor to the coordinate (x, y).
- lcd_putchar(c) – Prints the character c on the LCD.
- lcd_puts (str) – Prints the string str on the LCD.
- lcd_putsf ("ABC") – Prints 'ABC' on the LCD. Function used to print constant strings on LCD.

Circuit Overview



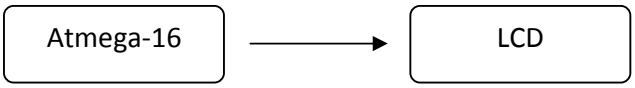
Part B: To print strings in loop on LCD

Problem Statement

1. Initialize the LCD at PORT C in the CVAVR graphical interface.
2. Program the Atmega to do the following :

- DYNAMIC PRINTING- Print your name and make it move (left/right) on LCD.

Circuit Overview



Part C: Display of integer data on LCD

Problem Statement

1. Make the LCD connections as in previous experiment.
2. Program the IC to print the following on the LCD:
    - Your date of birth

Printing an integer on LCD

- Create a character buffer with size atleast 1 more than the number of digits in the integer.
- itoa (int, char[n]) – This function converts the integer to a string.
- Use lcd_puts () to print the string on the LCD.

Example – To print '99' on the LCD

```
int a=99;

char buff[3];

itoa(a,buff);

lcd_gotoxy(0,0);

lcd_puts (buff);    //prints 99 at location (0, 0)
```

Part D: Display of floating point data on LCD

Problem Statement

1. Make the LCD connections as in previous experiment.
2. Program the IC to print the following on the LCD:
    - Your class XII board percentage

Printing a floating point number on LCD

- Create a character buffer with size atleast 1 more than the number of digits in the float number. (decimal point counted as a digit)
- ftoa (float,int,char[n]) – This function converts the integer to a string.
- The second argument (int) of the function decides the number of places after decimal in the string created.
- Use lcd_puts () to print the string on the LCD.

Example – To print '99.99' on the LCD

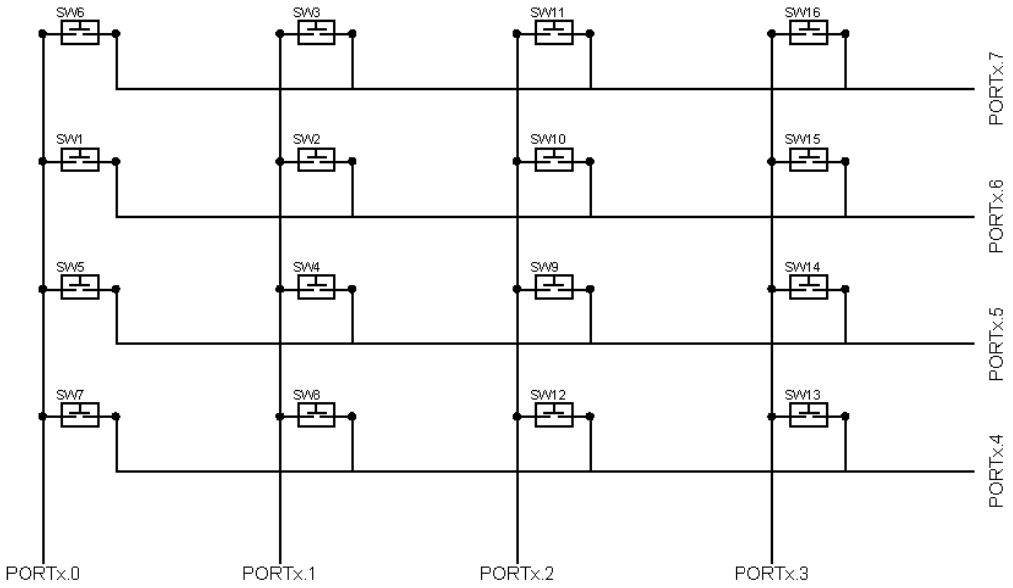
```
float a=99;

char buff[6];

ftoa(a,2,buff);

lcd_gotoxy(0,0);

lcd_puts (buff);     //prints 99.99 at location (0, 0)
```

Experiment 3: Interfacing a 16 key keypad

Problem Statement

1. Interface the given 16 key keypad to the 8 pins of PORT A of Atmega.
2. Make connection to LCD with PORT C as done before.
3. Press the keys of keypad one by one and obtain corresponding display on LCD.
4. Consider module 1A/1B for mcu and 16 matrix keypad connections.

Designing of Keypad



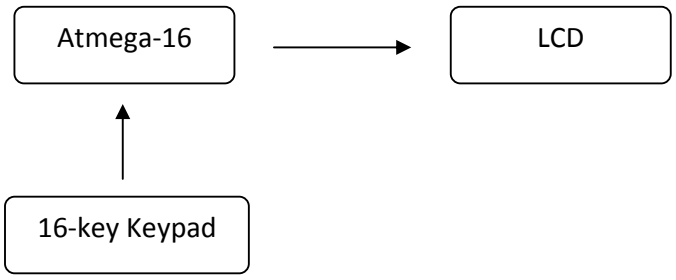
Schematic

- The 16 reset switches are connected in a square matrix fashion with 8 output wires connected to an 8 pin FRC female connector.

Logic behind the code used to interface

- Firstly the first four pins of the port (vertical lines) are made as output pins with 0 output and the remaining four (horizontal lines) are made input pins with an internal pull-up.
- If any switch is pressed then a 0 appears on one of the horizontal line when it is read.
- In the second part the first four pins are made input pins with internal pull-up and remaining four are made output pins with 0 output.
- If any switch is pressed then a 0 appears on one of the vertical line when it is read.
- This way the exact switch pressed is determined.

Circuit Overview



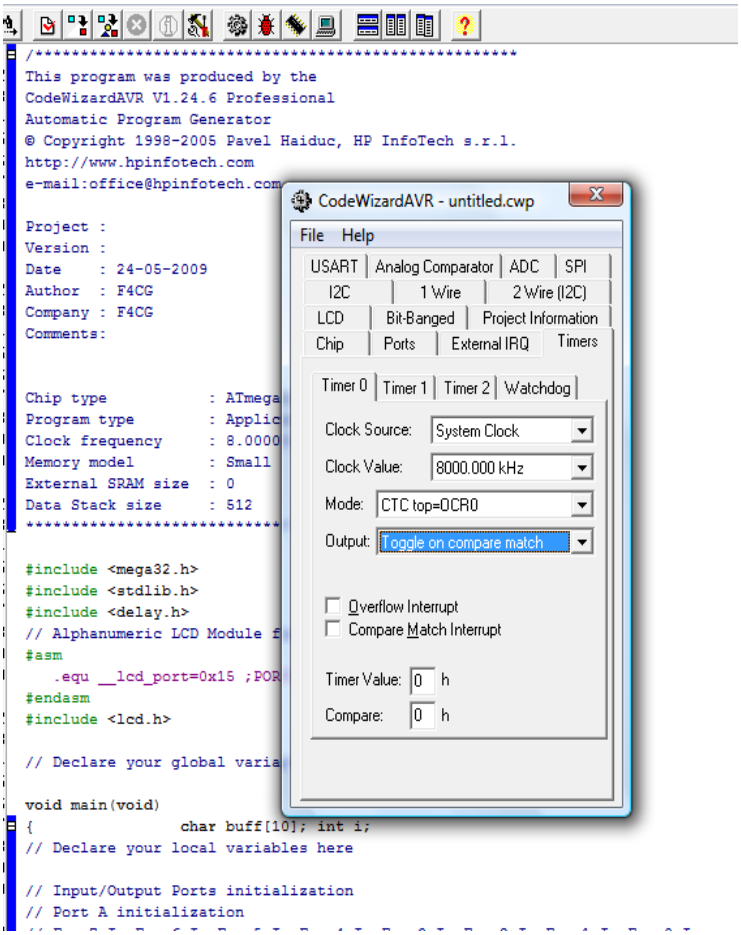
Experiment 4: Working with Timers - CTC mode

Timer in CTC mode

- In CTC mode duty cycle is fixed to 50%, but frequency of pulse can be selected.
- In the graphical interface of CVAVR use Timer 0 and Timer 2 for CTC mode.
  Do following settings:
  - Clock source – System clock
  - Clock value - Select any from the list as per frequency required. Call it f_t
  - Mode- CTCtop=OCRx (x= 0 or 2)
  - Output- Toggle on compare match
- In the program write the value of OCRx register.This determines the output frequency according to the formula:

$$f = f_t / (2 \times (OCRx+1))$$

- The output pulse is obtained at PB3 (for Timer 0) and PD7 (for Timer 2).
- Consider module 1A for mcu pin connections.



Part A

Problem Statement

1. Program the Atmega to generate a clock of the following two frequencies using CTC mode:
    - 38 kHz
    - 120 kHz
2. Measure the frequency and the voltage using a digital multimeter and compare with theoretical predictions

Theoretical value Frequency	Experimental value Frequency	Theoretical value Voltage	Experimental value Voltage

38 kHz		2.5 V	
120kHz		2.5 V	

Part B

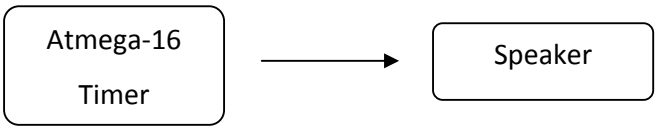
Problem Statement

- Generate a clock pulse of frequency in the audible range (20 Hz to 20 kHz) and apply it on a speaker to make a buzzer.

Connecting a speaker

- Connect one wire of the speaker to ground and the other wire to the output pin of the timer.
- Consider module 1A for mcu, module 10B for External Buzzers connections.

Circuit Overview



Experiment 5: Working with Timers – Fast PWM mode

Timer in Fast PWM mode

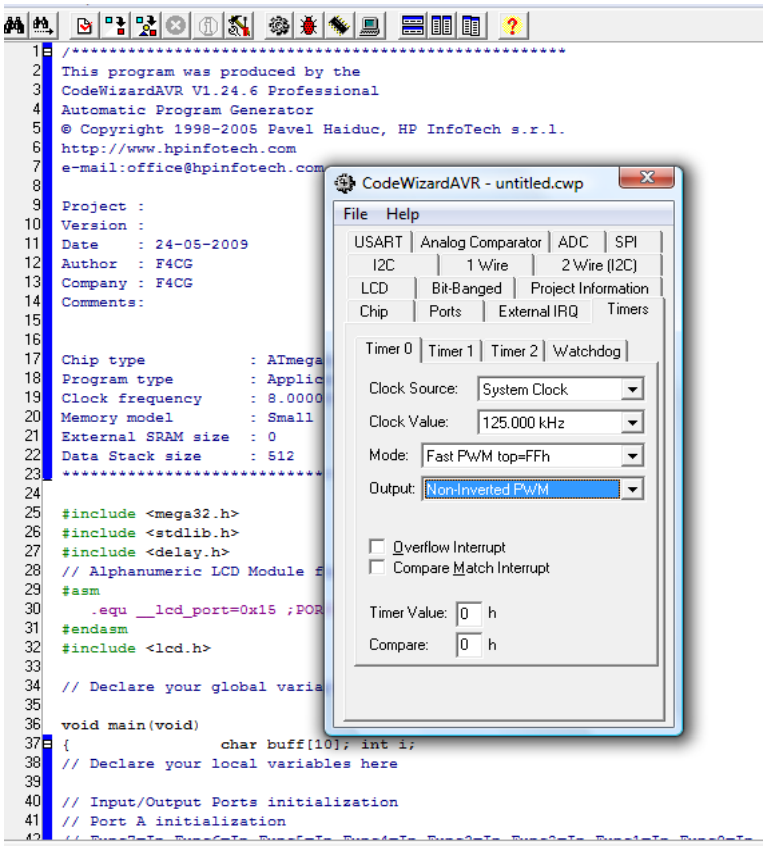
- In Fast PWM mode frequency is fixed, but duty cycle of pulse can be selected.
- In the graphical interface of CVAVR use Timer 0, Timer 1and Timer 2 for Fast PWM mode. Do following settings:
  - Clock source – System clock
  - Clock value - Select any from the list as per frequency required. Call it f_t
  - **The frequency of the pulse would then be fixed to f_t /256 .**
  - Mode- Fast PWM top=FFh
  - Output- Non-Inverted PWM (Inverted PWM is just the negation of this).
- In the program write the value of OCRx register.This determines the output duty cycle according to the formula:

$$\textbf{Duty Cycle} = (\textbf{OCRx} / 256) \times 100\ \%$$

- The output voltage measured by multimeter is given by:

$$\textbf{Voltage} = (\textbf{OCRx} / 256) \times V_{cc}$$

- The output pulse is obtained at PB3 (for Timer 0), PD4 (for Timer 1B), PD5 (for Timer 1A) and PD7 (for Timer 2).
- Consider module 1A for mcu connections.



Problem Statement

1. To generate pulses with duty cycle by using timer in PWM mode:
   - 25%
   - 50%
   - 75%

2. Measure the frequency and the average DC voltage using a digital multimeter and compare with theoretical predictions.

Duty Cycle	Theoretical value Frequency	Experimental value Frequency	Theoretical value Voltage	Experimental value Voltage
25%				
50%				
75%				

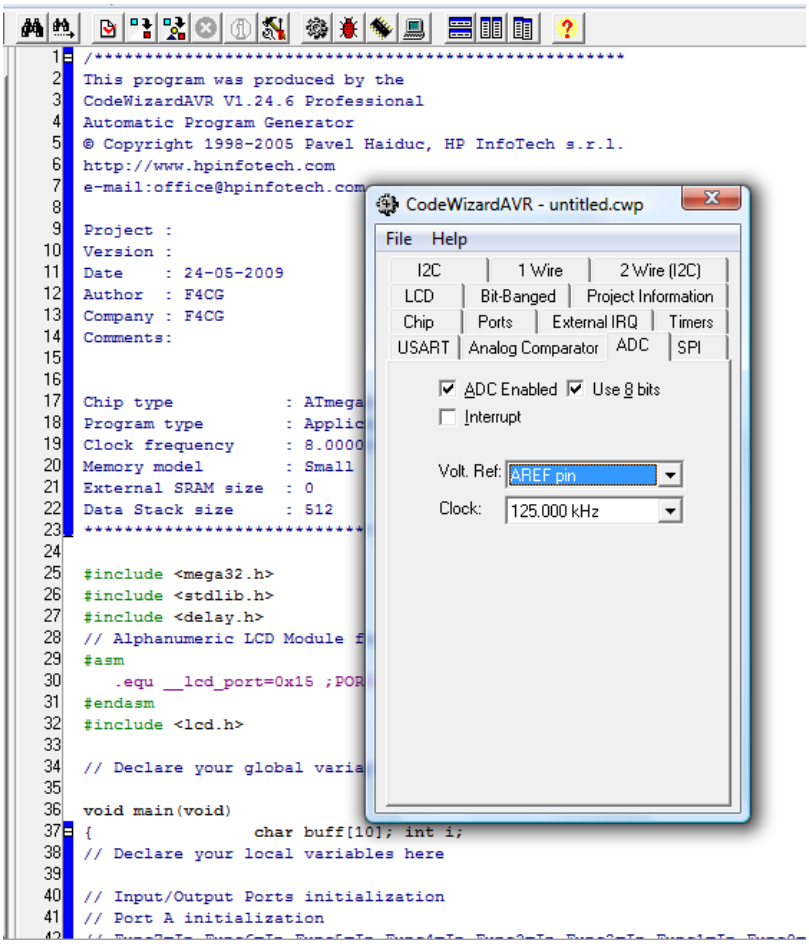
Experiment 6: Working with ADC (Analog to Digital Convertor)

Analog to Digital Convertor

- Atmega offers in-built ADC facility at PORT A i.e. pin number 33 to 40.
- Voltage applied to the ADC pin can vary from 0 volts to the voltage applied at AREF pin i.e. pin 32 of Atmega.
- The digital conversion can be 8-bit or 10-bit at the discretion of the user.
- Function used to read the ADC value is **read_adc(x)** (x is the pin number ranging from 0 to 7).It returns an integer corresponding to the voltage at the pin.
- Consider module 1A for mcu ADC connections.

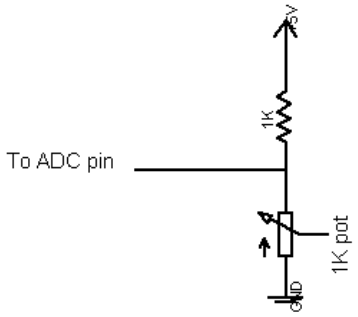
Initializing ADC

- In the CVAVR graphical interface, click on ADC and enable the facility.
- By default 10 bit conversion is done. Select 'Use 8 bits' option for 8 bit conversion.
- Voltage reference can be made AREF or AVCC since both pins are shorted.
- Clock represents the ADC sampling frequency. Select it as 125,000 kHz.



Problem Statement

- Make a voltage divider circuit using a 1k Ohm resistor and a 1k Ohm potentiometer (as shown).



- Connect it to one of the ADC pins of Atmega 16 and read the ADC value.
- Connect a 16x2 LCD at PORT C of Atmega and display the following on LCD:
    - Corresponding digital voltage 8 bit/10 bit (do both cases)
    - Corresponding voltage values (in Volts)

Consider module 1A for mcu ADC, module 8 for VARIAC and module 5 for LCD connections.

Circuit Overview

ADC of
Atmega-16

LCD

Potentiometer
Circuit

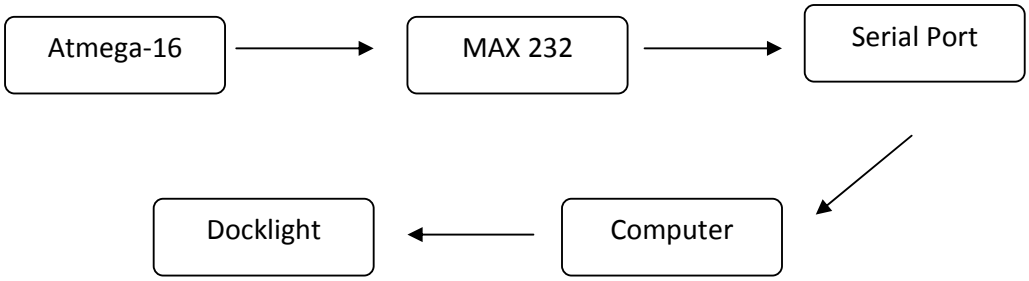
Experiment 7: Data communication by UART

Part A: To send characters from microcontroller and displaying on computer screen.

Problem Statement

1. Initialize UART of Atmega-16 with following parameters:
    - Communication Parameters: 8 Data, 1 Stop, No Parity
    - USART Receiver: On
    - USART Transmitter: On
    - USART Mode: Asynchronous
    - USART Baud rate: 9600
2. Write a program to continuously send a character‘t’ though UART.
3. Confirm the transfer of data on computer using software Docklight. Keep the same communication parameters.
4. Consider module 1A for mcu UART connections, module 3 for Serial Interface Module connections.

Circuit Overview



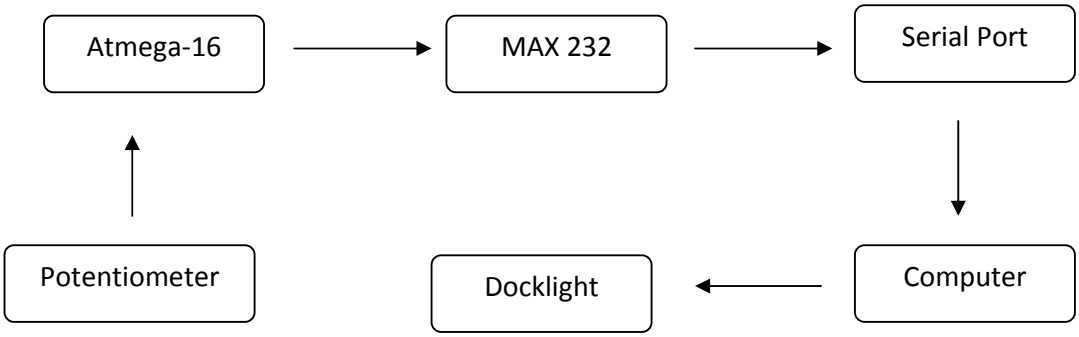
Part B: To send ADC values from uC to computer.

Problem Statement

1. Design a voltage divider circuit using potentiometer to generate analog voltages.
2. Connect the analog output to PA0

3. Initialize the UART in previous part's mode.
4. Write a program to convert the analog voltage to 8 bit digital word and transfer continuously to computer using UART. Give a delay of > 20ms between two readings to avoid crashing of computer.
5. Monitor the values in integer mode on computer using software Docklight.
6. Vary the potentiometer and analyze the readings.

Circuit Overview



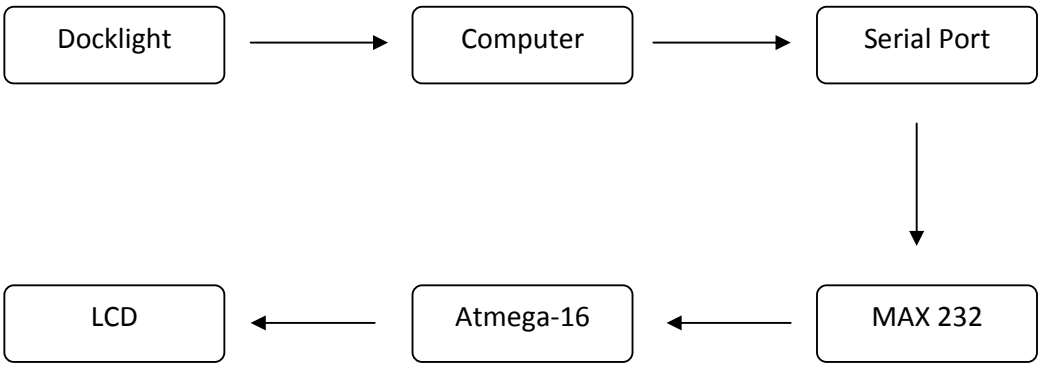
Part C: To send characters from computer keyboard to uC and print on LCD.

Problem Statement

1. Initialize the UART as before.
2. Using Docklight send data from the computer to the uC via UART protocol.
3. Make LCD connections to PORT C of Atmega.

4. Write a program to display the received data on the LCD.
5. Consider module 1A for mcu UART connections, module 3 for Serial Interface Module connections and module 5 for LCD connections.

Circuit Overview



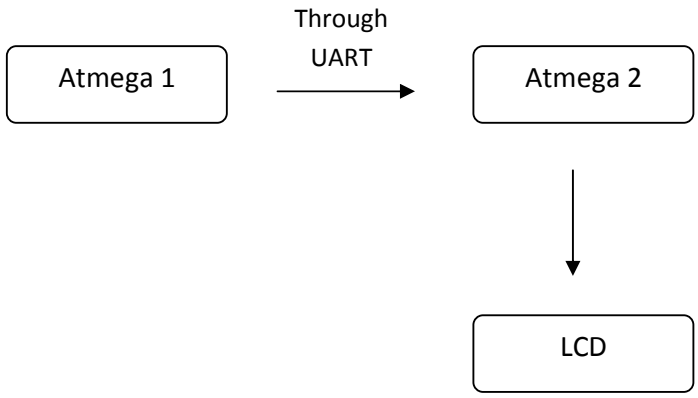
Part D: To send characters from one uC to another and displaying it on LCD.

Problem Statement

1. Initialize UART on both microcontrollers as before.

2. Make one of them as the receiver and the other as the transmitter with same parity and baud rate settings in both.
3. Connect the Rx pin of uC1 to Tx pin of uC2 and Tx pin of uC1 to Rx pin of uC2.
4. **Make sure that the ground connections of both the microcontrollers are same for proper communication**.
5. Write a program to send characters from one uC.
6. Receive them on the other uC and display them on a 16x2 LCD connected to PORT C.
7. Consider module 1A/1B for mcu UART connections.

Circuit Overview



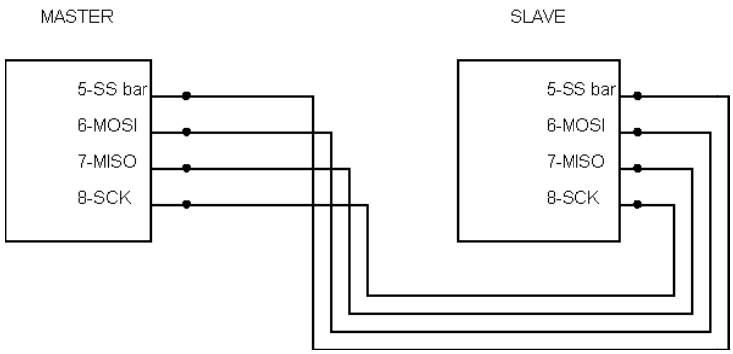
Experiment 8: Data transfer by SPI protocol

Problem Statement

1. In the graphical interface of CVAVR, enable SPI and do the following settings:
    - SPI clock rate- 2000 kHz
    - SPI type- Make one microcontroller the master and the other slave
    - Clock phase- Cycle half

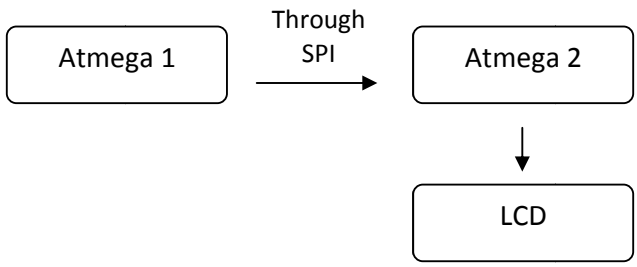
- Clock polarity- Low
- Data order- MSB first

2. Connect the following pin numbers of one uC to the corresponding pins of the other uC (As shown):
    - Pin number 5 (SS bar)
    - Pin number 6 (MOSI)
    - Pin number 7 (MISO)
    - Pin number 8 (SCK)



3. **Make sure that the ground connections of both the microcontrollers are same for proper communication**.
4. Write a program to send characters from the master.
5. Receive them on the slave and display the received data on a 16x2 LCD.
6. Consider module 1A/1B for mcu SPI connections, module 5 for LCD connections.
7. Now make another code to send ADC values from uC1 to uC2 and displaying it on LCD.

Circuit Overview

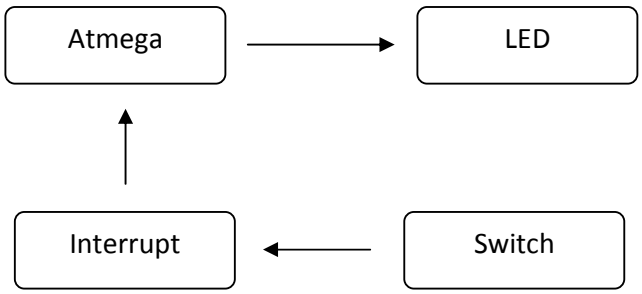


Experiment 9: Using External Interrupts

Problem Statement

1. Initialize INT0 in Falling Edge mode.
2. Write an LED blinking program in the Interrupt Service Routine.
3. Put a pull up resistance and a switch connecting to Gnd to provide interrupt.
4. Press the switch to blink the LED.
5. Consider module 1A for mcu interrupt pin connections, module6 for LED array connections and module 7 for switch pad.

Circuit Overview



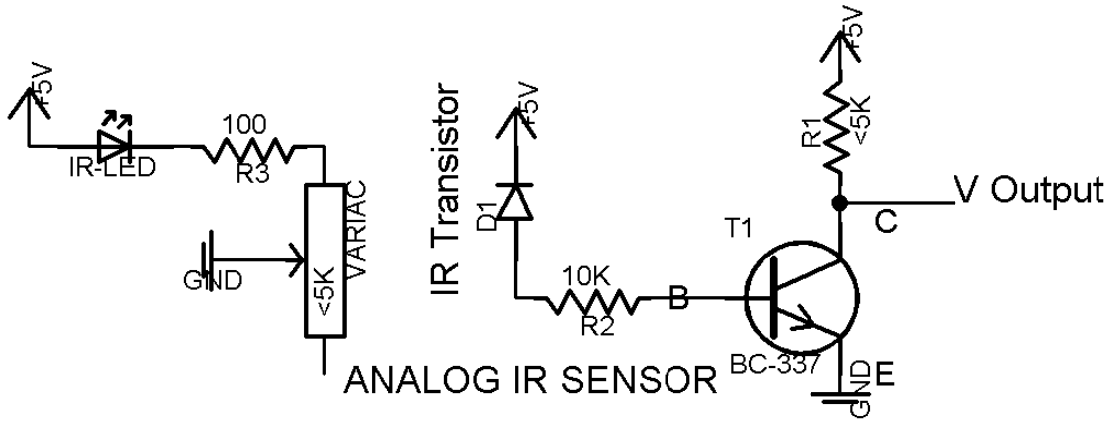
EXPERIMENTS IN ROBOTICS/AUTOMATION

Experiment 1: Analog IR Sensor

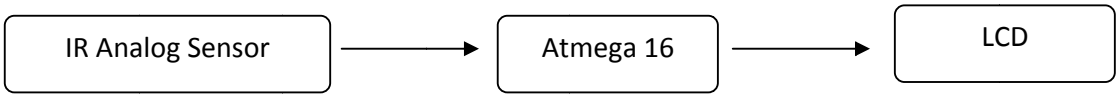
Part A: Interfacing Analog IR Sensor with microcontroller and to display variation in output on LCD.

Problem Statement

1. Make the circuit as shown in the figure.
2. The phototransistor D1 and IR-LED must be adjacent to each other (but not touching) and perpendicular to the breadboard.
3. After making the connections, measure the voltage V_{Output} by a multimeter.
4. Now bring a plane reflecting surface (white paper) over D1 and IR-LED and note the gradual change in the voltage.
5. Vary the resistance using variac and note the change in range of sensor.
6. Now enable ADC of Atmega 16 and check 'Use 8 bits'.
7. Bring the output of the analog sensor at one of the input pins for ADC (e.g., PINA.0) and write a code to read its digital value and display it on LCD.
8. Consider module 4A for Analog Sensor Module connections.



Circuit Overview



Part B: Analog IR Sensor used as a color sensor.

Problem Statement

1. Use the circuit connections from the previous part without any changes.
2. Now change the program code of the mcu to read ADC values for different colors viz take three-four colors for instance like RED, GREEN, BLACK , BLUE.
3. Take a pen and paper and note down the range of variations in ADC values when the sensor is exposed to these colors one by one.
4. Now alter the program code to check the range of ADC values and display corresponding color name on LCD.
5. Find out how many colors the mcu can differentiate.

Part C: Analog IR Sensor used for distance measuring.

Problem Statement

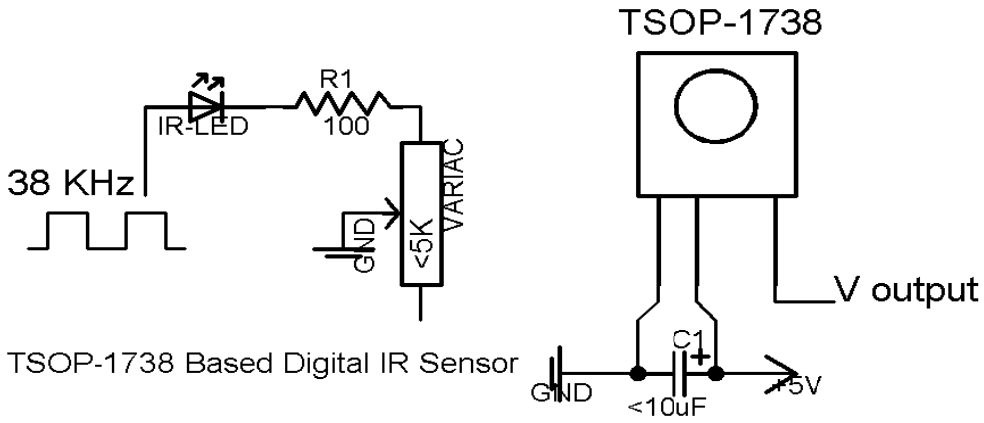
1. Use the circuit connections from the previous part without any changes.
2. Now change the program code of the mcu to read ADC values for distances of the order of centimeters
3. Take a pen and paper and note down the range of variations in ADC values cm by cm.
4. Now alter the program code to check the range of ADC values and display the distance on LCD.
5. Find out the minimum distance mcu can resolve using IR sonsor.

Experiment 2: Digital IR (TSOP) sensor

Part A: Interfacing Digital TSOP IR Sensor with microcontroller and to display variation in output on LCD / LED.

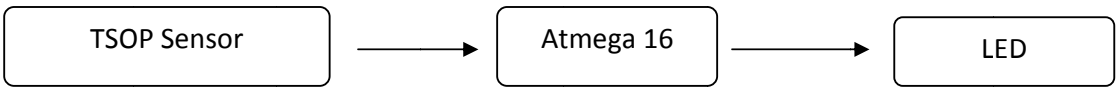
Problem Statement

1. Make the circuit as shown in the figure.
2. The TSOP sensor and IR-LED must be adjacent to each other (but not touching) and perpendicular to the breadboard.
3. To generate a pulse of 38kHz, initialize Timer 0 of Atmega-16 with following parameters:
   - Clock Source: System Clock
   - Clock Value: 1000.000 kHz
   - Mode: CTC top= OCR0
   - Output: Toggle on Compare Match
   - Set Compare Value as calculated for a pulse of 38 kHz (in hexadecimals).
4. Connect the output pin of Timer 0 with the positive terminal of IR-LED so as to give it a pulse of 38 kHz.
5. Check that the default output voltage of TSOP sensor is +5V (when no IR falls on it) and 0V on bringing a plane reflecting surface (white paper) over the sensor and the IR-LED.
6. Now connect the output pin of TSOP sensor to an I/O pin of Atmega 16 (e.g., PIND.0) and connect the positive terminal of an LED to another I/O pin of the uC (e.g., PIND.1). Connect negative terminal of the LED to gnd through a 220ohm resistance.
7. Now write a code that takes input from the TSOP sensor and glows the LED when the sensor receives IR.
8. Vary the resistance using variac and note the change in range of sensor.
9. Consider module 4B for Digital Sensor Module connections.



TSOP-1738 Based Digital IR Sensor

Circuit Overview



Part B: Digital TSOP IR Sensor used as contrast variation detection sensor.

Problem Statement

1. Use the same circuit connections.
2. Write a program code for detecting contrast variations, viz. black and white.
3. Contrast variation comes handy in designing line following robots.

Part C: Digital TSOP IR Sensor used as object detecting sensor.

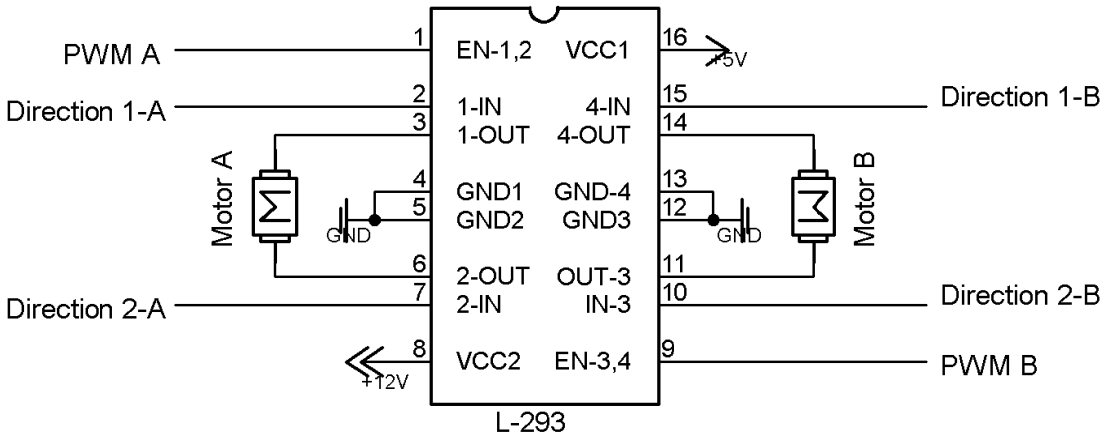
Problem Statement

1. Use the same circuit connections.
2. Write a program code for detecting obstacles in front of digital sensor.
3. Obstacle detection comes handy in obstacle detection robots.

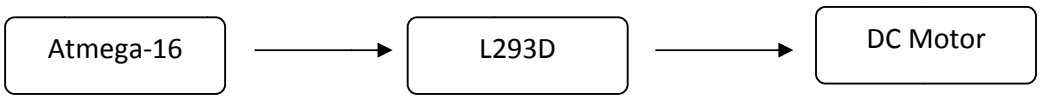
Experiment 3: Motor Control using driver L293

Problem Statement

1. Connect any two I/O pins of Atmega16 (pin no. 39 & 40) with the input pins of motor driver L293D i.e., Direction 1-A and Direction 2-A.
2. Connect the enable pin (PWM A) and VCC1 (pin no. 16) with constant 5V and ground as shown.
3. Connect the output pins of L293D with the terminals of the motor.
4. Give 12 V as VCC2 to L293D.
5. Write a code to so as to make pin 39 of uC high and pin 40 as low and note the direction of rotation of the motor.
6. Now interchange the outputs these pins of uC and note the change in direction of rotation.
7. Now make both the pins as high (or low) and observe the locking effect in the motor.
8. Consider module 1A for mcu connections, module 9 for Motor Driver Module connections.



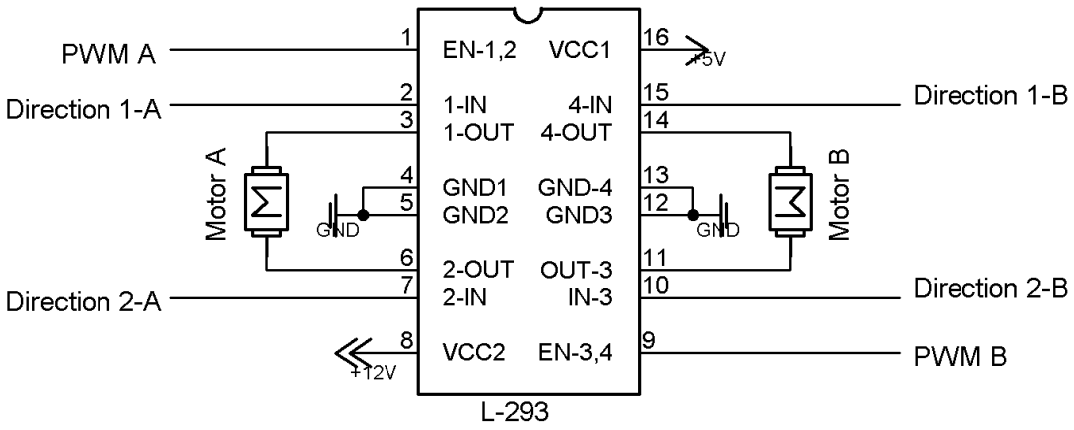
Circuit Overview



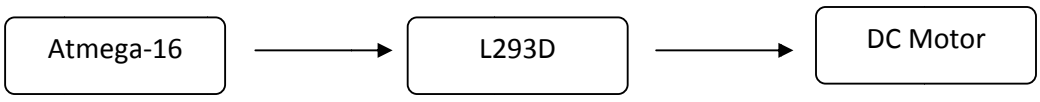
Experiment 4: Speed control of DC Motor

Problem Statement

1. Initialize Timer 0 of Atmega-16 with following parameters:
   - Clock Source: System Clock
   - Clock Value: 1000.000 kHz
   - Mode: Fast PWM top=FFh
   - Output: Non-Inverted PWM
   - Set Compare Value to 80 (decimal 128= hex 80)
2. Now Connect the Timer 0 output pin, i.e., PINB.3 (OCR0) to the enable pin of L293D, marked as PWM A.
3. All other connections are same as in Exp. 14.
4. Note the reduction in speed of motor as compared to that in Exp. 14.
5. To run the motor at full speed again, change OCR0 value to 255.
6. Consider module 9 for Motor driver connections.



Circuit Overview



Experiment 4: Automation

Part A: Controlling Motor through sensors.

Problem Statement

1. Interface Digital Sensor and Motor driver module with mcu.
2. Write a program code for the following:

- Motor 1 changes direction when obstacle comes in front of sensor.
- Motor 2 changes direction when obstacle comes in front of sensor.

Try to interface two sensors and four motors together. Easy..isn't it?

Circuit Overview

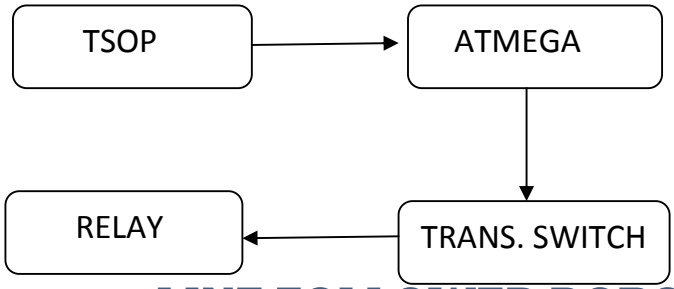
Part B: Switching Relay/Transistor through sensors.

Problem Statement

1. Interface Digital Sensor and Transistor module with the mcu.
2. Write a program code for the following:

- Relays switch when obstacle comes in front of sensor.

Circuit Overview



LINE FOLLOWER ROBOT

Task:

To design an autonomous robot to follow a line.

There can be two cases:

A: Bright strip over dark background (White over Black)

B: Dark strip over bright background (Black over White)

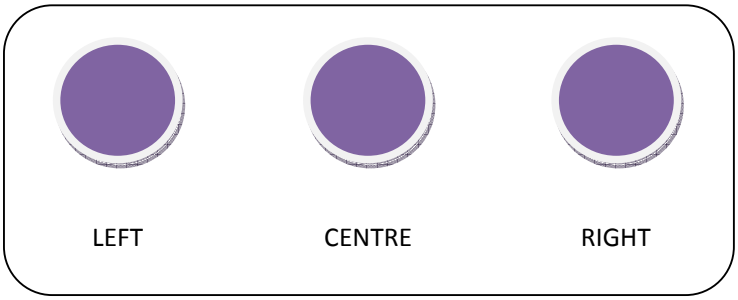
How to differentiate white and black?

Digital IR Sensor (TSOP) used for contrast detection as discussed.

How to follow a line?

We have sensors to differentiate between white and black surfaces. Hence we can use them in a pattern to detect line and move robot accordingly.

Arrangement of 3 sensors:

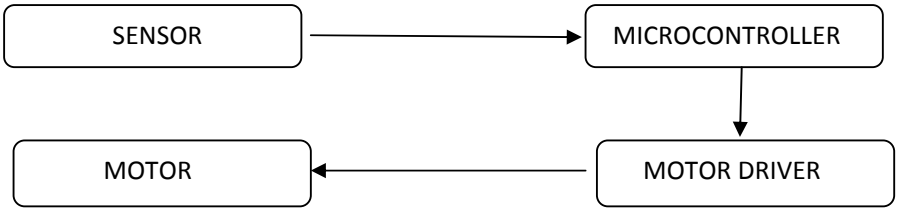


Each purple circle represents a Tx-Rx pair.

Sensor over white surface: output low (logic level 0)

Sensor over black surface: output high (logic level 1)

Design:

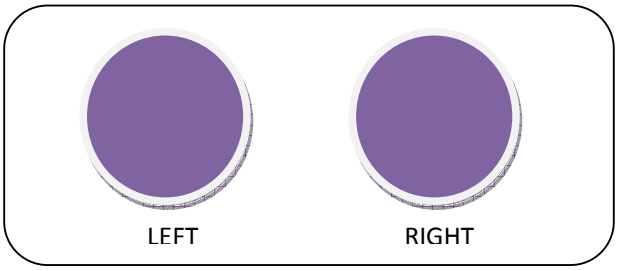


How to control robot:

Position of Line	Sensor (on)	Movement
Extreme Left	Left	Turn sharp left
Left	Centre and Right	Turn left
Centre	Centre	Move forward
Right	Centre and Right	Turn right
Extreme Right	Right	Turn sharp right

Simplifying Line Follower Robot:

Use two sensors instead, one for left and other for right.



How to control robot:

Position of line	Sensor (on)	Movement
Left	Left	Turn right
Centre	None	Move Forward
Right	Right	Turn left