Automatic railway gate control using 8051

Abstract:

* The objective of this paper is to provide an automatic railway gate at a level crossing replacing the gates operated by the gatekeeper. It deals with two things. Firstly, it deals with the reduction of time for which the gate is being kept closed, and secondly, to provide safety to the road users by reducing the accidents. By the presently existing system once the train leaves the station, the stationmaster informs the gatekeeper about the arrival of the train through the telephone. Once the gatekeeper receives the information, the closes the gate depending on the timing at which the train arrives. Hence, if the train is late due to certain reasons, then gate remain closed for a long time causing traffic near the gates. By employing the automatic railway gate control at the level crossing the arrival of the train is detected by the sensor placed near to the gate. Hence, the time for which it is closed is less compared to the manually operated gates and also reduces the human labour. This type of gates can be employed in an unmanned level crossing where the chances of accidents are higher and reliable operation is required. Since, the operation is automatic; error due to manual operation is prevented. Automatic railway gate control is highly economical microcontroller based arrangement, designed for use in almost all the unmanned level crossings in the country.

Introduction:

* A level crossing occurs where a railway line is intersected by a road or path on one level, without recourse to a bridge or tunnel. It is a type of at grade intersection. The term also applies when a light rail line with separate right-of-way in reserved track crosses a road in the same fashion. Other names include railway crossing, railroad crossing, road through railroad, train crossing or grade crossing. Early level crossings had a lay man in a nearby booth who would, on the approach of a train, wave a red flag or lantern to stop alralic and clear the tracks. Manual or electrical closable gates that barricaded the roadway were later introduced. The gates were intended to be a complete barrier against instruction of any road traffic onto the railway. In the early days of the railways much road traffic was horse drawn or included livestock. It was thus necessary to provide a real barrier. Thus, crossing gates, when closed to road traffic, crossed the entire width of the road. When opened to allow road users to cross the line, the gates were swung across the width of the railway, preventing any pedestrians or animals getting onto the line.

INTRODUCTION TO EMBEDDED SYSTEM:

An embedded system is described as combo of each programming and hardware. An extensively helpful importance of embedded structures is that they are gadgets used to control, show screen or help the errand of apparatus, hardware or plant. Embedded mirrors the way that they are a fundamental bit of the system. At the option ludicrous an all around useful PC may be connected to control the pastime of a major complex.An embedded structure is a system which will finish a predefined decided endeavor is the introduced structure and is even portrayed as mix of both programming and hardware. A comprehensively valuable importance of embedded systems is that they are contraptions used to control, screen or help the errand of apparatus, equipment or plant. Introduced mirrors how they are a fundamental bit of the structure. At the other remarkable a generally valuable PC may be used to control the action of a far reaching complex getting ready plant, and its embodiment will act naturally clear.

The extremely most straightforward inserted frameworks are equipped for performing just a solitary capacity or set of capacities to meet a solitary foreordained reason. In more mind boggling frameworks an application program that empowers the implanted framework to be utilized for a specific reason in a particular application decides the working of the installed framework. The capacity to have programs implies that the same implanted framework can be utilized for a wide range of purposes. Now and again a chip might be planned such that application programming for a specific reason can be added to the fundamental programming in a moment procedure, after which it isn't conceivable to roll out further improvements. The applications programming on such processors is now and again alluded to as firmware.

The least difficult gadgets comprise of a solitary microchip (frequently called a chip), which may itself be bundled with different chips in a cross breed framework or Application Specific Integrated Circuit (ASIC). Its information originates from a finder or sensor and its yield goes to a switch or activator which (for instance) may begin or stop the task of a machine or, by working a valve, may control the stream of fuel to a motor.

Embedded systems are set up in cell phones, digital cameras microwave ovens, answering machines, home security system, washing machines, lighting system, fax machines, copiers, printer, and scanners, cash registers, alarm system, automated teller machinces, and many other devices.

It consists of microprocessor, RAM and flash memory etc. programming in embedded system is not that easier like PC programming. It seems to be like programming in 15 years ago PCs. The hardware for the system is normally decided to make the gadget as low-priced as could be expected under the circumstances. Using an additional dollar and unit so as to make things simpler to the program may cost millions. When compared with buying a new device for a gadget design procuring a software engineer for an additional month is inexpensive. This implies the software engineer must make do with moderate processors and low memory, while in the meantime doing combating a requirement for effectiveness not seen in most PC application. The following is a rundown of issues particular to the embedded filed.

***TOOLS***

Embedded advancement makes up a little portion of aggregate encoding or programming. There’s likewise an expansive number of embedded architectures, not at all like the PC world where 1 instruction set rules, and the UNIX world where there’s just 3 or 4 real ones This implies that the tools are more extravagant. It likewise implies that they’re bringing down emphasized and less created. On a most embedded system design ,sooner or later you will quite often discover a complier bug or something to that affect

As we won’t generally run particular programs on our embedded processor, we can’t run generally a debugger on it. In this case the debugging devices are an alternate issue. This make altering your system troublesome. Unique hardware’s for, example, JTAG ports can conquer this issue to some degree. On the other hand, on the off chance that you stop on a breakpoint when your framework is controlling true fittings,(for example, an engine or motor),lasting supplies harm can happen. Subsequently, individuals doing installed programming rapidly get to be experts at utilizing serial IO channels and mistake message style debugging.

***RESOURCES***

The programs are needed to be written effectively as the embedded system regularly have the least expensive processors that can do the job to spare expense. This implies that your programs need to be composed as productively as could be allowed. At the point when managing substantial data sets, problems like memory cache misses that not at all issue in PC programming can harm you .fortunately, this will never occur time after time utilize sensibly proficient algorithms to begin, and upgrade just when vital. Debuggers don’t’ function admirably.

The algorithms designed by the software engineers need to be memory efficient (not at all like in PC programs, you will oftentimes relinquish processor time for memory, instead of the converse), since the for the same expense investment funds reasons, embedded system normally have the slightest memory they can escape with. So memory is additional an issue. It likewise implies you can’t bear to leak memory. Embedded applications by and large utilize deterministic memory procedures and maintain a strategic distance from the default ‘new’ and ‘malloc’ works, so breaks can be discovered and killed all the more effectively. Different resource programmers or software engineers expect may not in any case exist.

**NEED FOR EMBEDDED SYSTEMS**

In recent days the novel products brought in the market are using the embedded computers in different ways. The uses of embedded systems are virtually limitless, because every day new products are introduce to the market that utilizes embedded computer in novel ways of late, devices like chip, microcontrollers and FPGA chips have ended up much less expensive. A wiser thought is to purchase a generic chip and write own convention software or program for put into a practice a novel control

Creating uniquely designed chip to handle a specific undertaking a specific understanding or set of assignments expenses significantly more of an opportunity and cash. Numerous embedded machines even accompany broad libraries, with a goal that composition of your own particular programming turns into an extremely unimportant errand in reality. From a usage perspective, there is a real distinction between a computer and embedded systems. Inserted embedded systems are frequently needed to give Real-Time reaction. The fundamental components that make embedded frameworks remarkable are its reliability and simplicity in debugging

**EXPLAINATION FOR EMBEDDED SYSTEMS**

**Software architecture**

There are a few distinctions sorts of programming structural planning in as relatable point utilization.

* Simple control loop:

In this plan, the product just has a loop this plan. The loop calls subroutines, each of which deals a pieces of the hardware or programming.

* Interrupt control loop:

Some embedded systems are preventively intruding on controlled. This implies that undertaking carried out by the systems are activated by various type of occasions. In the presence of predefined timer an interrupt may occur or by a serial port controller accepting a byte. When the event handlers are simple and short these sorts of systems are utilized if event handlers need low latency.

* Micro kernels and Exo kernels

A logical setup from real-time operating systems is a micro kernel.The normal plan is that the logical setup from real-time operating systems is a microkernel. The normal plan is that the working framework kernel allots memory and switches the CPU to diverse strings of execution. Client mode methodologies actualize real capacitance for example, file systems, interface.

**Real-time embedded systems**

Embedded systems which are utilized to perform particular undertaken or operation in particular time period those frame works are known as real-time embedded systems.The embedded systems are of two types.

* Hard Real-time implanted frameworks:

These implanted frameworks take after a flat out dead line time period i.e., if the tasking is not done in a specific time period then there is a reason for harm to the whole system.

Example: Consider a framework in which we need to open a valve inside 30milli seconds. On the off chance that this valve is not opened in 30 ms this may cause harm to the whole supplies. So in such cases we utilize implanted frameworks for doing programmed operations.

* Soft Real-time implanted frameworks:

These embedded frameworks take after a relative dead line time period i.e., if the task is not done in a specific time that won’t result in harm to the supplies

Since these early applications in the 1960s, embedded systems have come down in price. In 1978 National Engineering Manufacturers Association released the standard for a programmable m

By the mid-1980s, many of the previously external system components had been integrated into the same chip as the processor, resulting in integrated circuits called microcontrollers, and widespread use of embedded systems became feasible.

**APPLICATION OF EMBEDDED SYSTEMS:**

**Consumer appliances:** A number of embedded systems are used at home which consists of microwave, ovens, remote control, DVD players and cameras and so on.

**Office automation**: Systems like fax machine, modem and printer and so on are used for office purposes

**Industrial automation:** Now-a-days more number of industries is using embedded systems for process control. In industries we design the embedded systems to perform a particular operation like monitoring temperature, voltage, current and so on later based on these monitored levels we perform control to other devices, we can also send information to a centralized monitoring station.

In significant manufacturing where human being there is let alone there we can employ robots which are programmed to do a particular function.

**Computer networking:** Embedded systems are used as bridges routers and son.

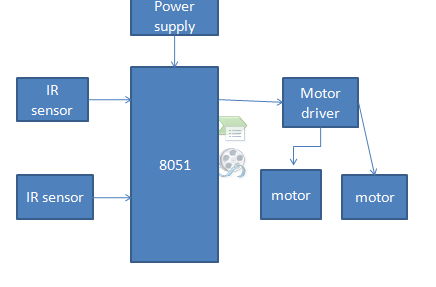
**Tele communication:** Cell phones, web cameras and so on.

Existing system:

* In the existing system, most doors are controlled by persons with the use of keys, security cards, password or pattern to open the door. we even can operate with the mobile through apps. The aim of this project is to help users for improvement of the door security. With the advancement in the technology, Wireless controlling technique is proposed with a fine combination of new technologies and embedded system.

Proposed system:

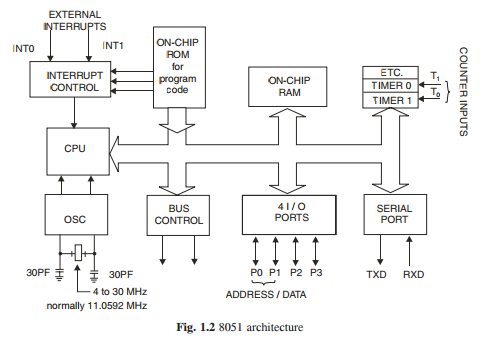
* In the proposed system, we are introducing 8051 module to access door lock. This signal is then sent to the microcontroller. In this project microcontroller act as a master and the remaining acts as slaves. Information from the micro controller is displayed on the LCD. Here the motor driver boards are connected to the circuit depending on the working of motor door will be opened or closed depending on the command given through mobile. Any unauthorized people cannot open the door it will be secured through our mobile.

Block Diagram: 

**Hardware requirements**

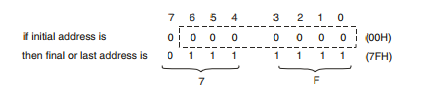
**8051 MICROCONTROLLER**

It is 8-bit microcontroller, means MC 8051 can Read, Write and Process 8 bit data. This is mostly used microcontroller in the robotics, home appliances like mp3 player, washing machines, electronic iron and industries. Mostly used blocks in the architecture of 8051 are as follows:



**128 Byte RAM for Data Storage**

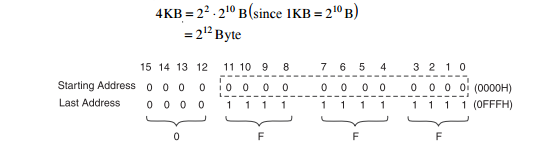
MC 8051 has 128 byte Random Access memory for data storage. Random access memory is non volatile memory. During execution for storing the data the RAM is used. RAM consists of the register banks, stack for temporary data storage. It also consists of some special function register (SFR) which are used for some specific purpose like timer, input output ports etc. Normally microcontroller has 256 byte RAM in which 128 byte is used for user space which is normally Register banks and stack. But other 128 byte RAM which consists of SFRs. We will discuss the RAM in detail in next section. Now what is the meaning of 128 byte RAM. What are address range which is provided for data storage.



This procedure of calculating the memory address is called as “memory mapping”. We can save data on memory locations from 00H to 7FH. Means total 128 byte space from 00H to 7FH is provided for data storage.

**4KB ROM**

* In 8051, 4KB read only memory (ROM) is available for program storage. This is used for permanent data storage. Or the data which is not changed during the processing like the program or algorithm for specific applications.
* This is volatile memory; the data saved in this memory does not disappear after power failure.
* We can interface up to 64KB ROM memory externally if the application is large. These sizes are specified different by their companies.
* Address Range of PC: Address range of PC means program counter (which points the next instruction to be executing) can be moved between these locations or we can save the program from this location to this location. The address range can be calculated in the same way just like the RAM which is discussed in previous section.



Address range of PC is 0000H to 0FFFH means total 4KB locations are available from 0000H to 0FFFH. At which we can save the program.

**Difference between RAM and ROM**

* RAM is used for data storage while ROM is used for program storage.
* Data of RAM can be changed during processing while data of ROM can’t be changed during processing. • We can take an example of calculator. If we want to perform addition of two numbers then we type the two numbers in calculator, this is saved in the RAM, but the Algorithms by which the calculation is performed is saved in the ROM. Data which is given by us to calculator can be changed but the algorithm or program by which calculation is performed can’t be changed.

**Timers and Counters**

Timer means which can give the delay of particular time between some events. For example on or off the lights after every 2 sec. This delay can be provided through some assembly program but in microcontroller two hardware pins are available for delay generation. These hardware pins can be also used for counting some external events. How much times a number is repeated in the given table is calculated by the counter.

* In MC8051, two timer pins are available T0 and T1, by these timers we can give the delay of particular time if we use these in timer mode.
* We can count external pulses at these pins if we use these pins in counter mode.
* 16 bits timers are available. Means we can generate delay between 0000H to FFFFH.
* Two special function registers are available.



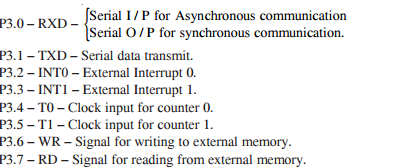
* If we want to load T0 with 16 bit data then we can load separate lower 8 bit in TL0 and higher 8 bit in TH0.
* In the same way for T1.
* TMOD, TCON registers are used for controlling timer operation.

**Serial Port**

* There are two pins available for serial communication TXD and RXD.
* Normally TXD is used for transmitting serial data which is in SBUF register, RXD is used for receiving the serial data.
* SCON register is used for controlling the operation.
* There are four modes of serial communication which has been discussed in next chapter.

**Input Output Ports**

* There are four input output ports available P0, P1, P2, P3.
* Each port is 8 bit wide and has special function register P0, P1, P2, P3 which are bit addressable means each bit can be set or reset by the Bit instructions (SETB for high, CLR for low) independently.
* The data at any port which is transmitting or receiving is in these registers.
* The port 0 can perform dual works. It is also used as Lower order address bus (A0 to A7) multiplexed with 8 bit data bus P0.0 to P0.7 is AD0 to AD7 respectively the address bus and data bus is demultiplex by the ALE signal and latch which is further discussed in details.
* Port 2 can be used as I/O port as well as higher order address bus A8 to A15.
* Port 3 also have dual functions it can be worked as I/O as well as each pin of P3 has specific function.



When external memory is interfaced with 8051 then P0 and P2 can’t be worked as I/O port they works as address bus and data bus, otherwise they can be accessed as I/O ports.

**Oscillator**

* It is used for providing the clock to MC8051 which decides the speed or baud rate of MC.
* We use crystal which frequency vary from 4MHz to 30 MHz, normally we use 11.0592 MHz frequency.

**Interrupts**

* Interrupts are defined as requests because they can be refused (masked) if they are not used, that is when an interrupt is acknowledged. A special set of events or routines are followed to handle the interrupts. These special routines are known as interrupt handler or interrupt service routines (ISR). These are located at a special location in memory.
* INT0 and INT1 are the pins for external interrupts.

**FEATURES OF 8051:**

|  |  |
| --- | --- |
| **Feature** | **Quantity** |
| ROM | 4k bytes |
| RAM | 128 bytes |
| Timer | 2 |
| I/O pins | 32A |
| Serial port | 1 |
| Interrupt sources | 6 |

* 4 KB on chip program memory.
* 128 bytes on chip data memory(RAM).
* 128 user defined software flags.
* 8-bit data bus
* 16-bit address bus
* 32 general purpose registers each of 8 bits
* 16 bit timers (usually 2, but may have more, or less).
* 3 internal and 2 external interrupts.
* Bit as well as byte addressable RAM area of 16 bytes.
* Four 8-bit ports, (short models have two 8-bit ports).
* 16-bit program counter and data pointer.
* 1 Microsecond instruction cycle with 12 MHz Crystal.

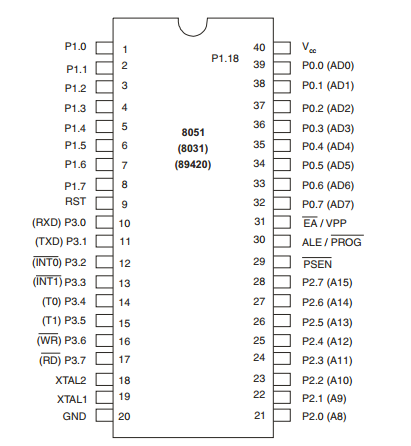
8051 models may also have a number of special, model-specific features, such as UARTs, ADC, OpAmps, etc…

**DIFFERENTIATE 8051 WITH 8052,8031 AND 8751**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **8051** | **8052** | **8031** | **8751** |
| **ROM(bytes)** | 4K | 8K | 0 | 4K (UV-EPROM) |
| **RAM(bytes)** | 128 | 256 | 128 | 128 |
| **Timers** | 2 | 3 | 2 | 2 |
| **I/O pins** | 32 | 32 | 32 | 32 |
| **Serial Port** | 1 | 1 | 1 | 1 |
| **Interrupts** | 6 | 8 | 6 | 6 |
| **Watchdog timer** | No | No | Yes | No |

**8051 PIN DESCRIPTION AND ARCHITECTURE:**

**PIN DIAGRAM OF 8051:**



Pin diagram of 8051

**Description of each pin is discussed here:**

• VCC → 5V supply

• VSS → GND

• XTAL2/XTALI are for oscillator input

• Port 0 – 32 to 39 – AD0/AD7 and P0.0 to P0.7

• Port 1 – 1 to 8 – P1.0 to P1.7

• Port 2 – 21 to 28 – P2.0 to P2.7 and A 8 to A15

• Port 3 – 10 to 17 – P3.0 to P3.7

• P 3.0 – RXD – Serial data input – SBUF

• P 3.1 – TXD – Serial data output – SBUF

• P 3.2 – INT0 – External interrupt 0 – TCON 0.1

• P 3.3 – INT1 – External interrupt 1 – TCON 0.3

• P 3.4 – T0 – External timer 0 input – TMOD

• P 3.5 – T1 – External timer 1 input – TMOD

• P 3.6 – WR – External memory write cycle – Active LOW

• P 3.7 – RD – External memory read cycle – Active LOW

• RST – for Restarting 8051

• ALE – Address latch enable 1 – Address on AD 0 to AD 7 0 – Data on AD 0 to AD 7

• PSEN – Program store enable

For describing pin diagram and pin configuration of 8051, we are taking into consideration a 40 pin DIP (Dual inline package). Now lets go through pin configuration in detail.

**Pin-40 :** Named as Vcc is the main power source. Usually its +5V DC.

You may note some pins are designated with two signals (shown in brackets).

**Pins 32-39:** Known as Port 0 (P0.0 to P0.7) – In addition to serving as I/O port, lower order address and data bus signals are multiplexed with this port (to serve the purpose of external memory interfacing). This is a bi directional I/O port (the only one in 8051) and external pull up resistors are required to function this port as I/O.

**Pin-31:-** ALE aka Address Latch Enable is used to demultiplex the address-data signal of port 0 (for external memory interfacing.)  2 ALE pulses are available for each machine cycle.

**Pin-30:-** EA/ External Access input is used to enable or disallow external memory interfacing. If there is no external memory requirement, this pin is pulled high by connecting it to Vcc.

**Pin- 29:-** PSEN or Program Store Enable is used to read signal from external program memory.

**Pins- 21-28:-** Known as Port 2 (P 2.0 to P 2.7) – in addition to serving as I/O port, higher order address bus signals are multiplexed with this quasi bi directional port.

**Pin 20:-** Named as Vss – it represents ground (0 V) connection.

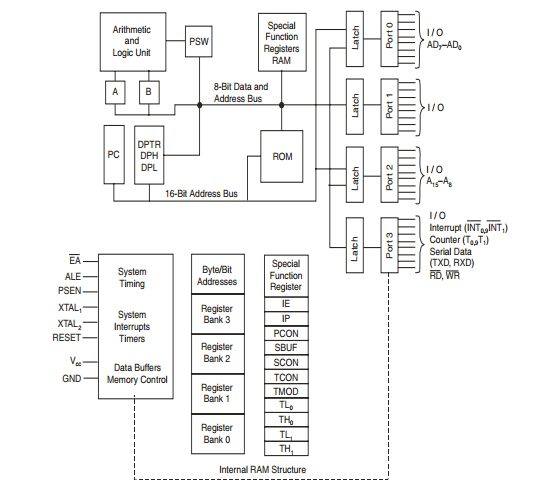
**Pins 18 and 19:-** Used for interfacing an external crystal to provide system clock.

**Pins 10 – 17:-** Known as Port 3. This port also serves some other functions like interrupts, timer input, control signals for external memory interfacing RD and WR , serial communication signals RxD and TxD etc. This is a quasi bi directional port with internal pull up.

**Pin 9:-** As explained before RESET pin is used to set the 8051 microcontroller to its initial values, while the microcontroller is working or at the initial start of application. The RESET pin must be set high for 2 machine cycles.

**Pins 1 – 8:-** Known as Port 1. Unlike other ports, this port does not serve any other functions. Port 1 is an internally pulled up, quasi bi directional I/O port.

**ARCHITECTURE OF 8051:**



Architecture block diagram of microcontroller 8051

Each block will be discussed step by step:

**ALU — Arithmetic Logical Unit**

This unit is used for the arithmetic calculations.

**A-Accumulator**

This register is used for arithmetic operations. This is also bit addressable and 8 bit register.

**B-Register**

This register is used in only two instructions MUL AB and DIV AB. This is also bit addressable and 8 bit register.

**PC-Program Counter**

• Points to the address of next instruction to be executed from ROM

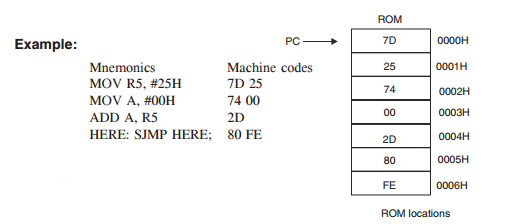
• It is 16 bit register means the 8051 can access program address from 0000H to FFFFH. A total of 64KB of code. 16 bit register means.



• Initially PC has 0000H

• ORG instruction is used to initialize the PC ORG 0000H means PC initialize by 0000H

• PC is incremented after each instruction.



• When 7D is accessed then PC locate the 0001H (next instruction to be executed)

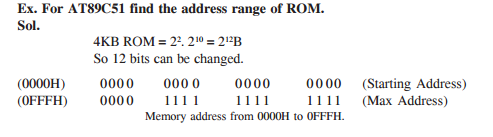
• When 00 is accessed then PC locate the 0004H (next instruction to be executed)

**ROM Memory Map in 8051**

→ 4KB, 8KB, 16KB, 32KB, 64KB on chip ROM is available.

→ Max ROM space is 64 KB because 16 bit address line is available in 8051.

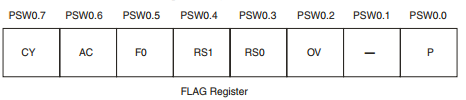
→ Starting address for ROM is 0000H (because PC which points the ROM is 16 bit wide).



**8051 Flag Bits and PSW Register**

→ Used to indicate the Arithmetic condition of ACC.

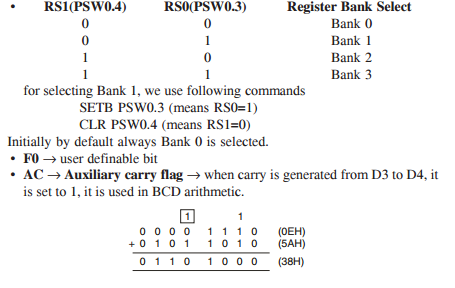
→ Flag register in 8051 is called as program status word (PSW). This special function register PSW is also bit addressable and 8 bit wide means each bit can be set or reset independently.



There are four flags in 8051

• P → Parity flag → PSW 0.0 1 – odd number of 1 in ACC 0 – even number of 1 in ACC

• OV(PSW 0.2) → overflow flag → this is used to detect error in signed arithmetic operation. This is similar to carry flag but difference is only that carry flag is used for unsigned operation.



Since carry is generated from D3 to D4, so AC is set.

• CY → carry flag → Affected after 8 bit addition and subtraction. It is used to detect error in unsigned arithmetic opr. We can also use it as single bit storage.

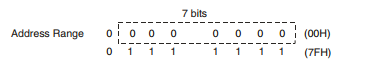
SETB C → for cy = 1

CLR C → for cy = 0

**Structure of RAM or 8051 Register Bank and Stack**

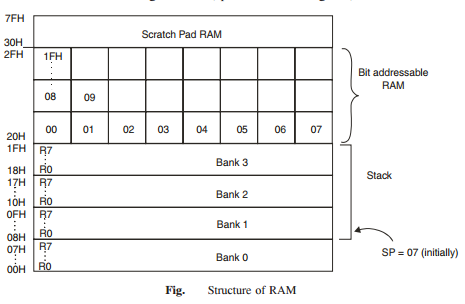
→ 128 byte RAM is available in 8051

→ 128 byte = 27 B



Address range of RAM is 00H to 7FH.

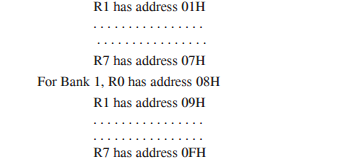
→ In MC8051, 128 byte visible or user accessible RAM is available which is shown in figure. Extra 128B RAM which is not user accessible. 80H to FFH used for storage of SFR (special function register)



→ Four Register Banks

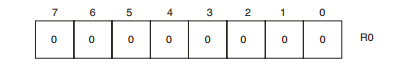
→ There are four register banks, in each register bank there are eight 8 bit register available from R0 to R7

→ By default Bank 0 is selected. For Bank 0, R0 has address 00H



For selecting banks we use RS0 and RS1 bit of PSW.

→ R0 to R7 registers are byte addressable means.



If we want to set the bit 3 of R0 then we can’t use SETB R0.3 We use MOV R0, #08H;

For changing single bit we can modify all the other bits of R0.

→ Locations 20H to 2FH is bit addressable RAM means each bit from 00H to FFH in this we can set or reset CF rather than changing whole byte.

→ Locations 30H to 7FH is used as scratch pad means we can use this space for data reading and writing or for data storage.

**Stack in 8051**

→ RAM locations from 08H to 1FH can be used as stack. Stack is used to store the data temporarily. Stack is last in first out (LIFO)

→ Stack pointer (SP) →

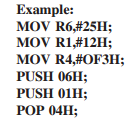
• 8bit register

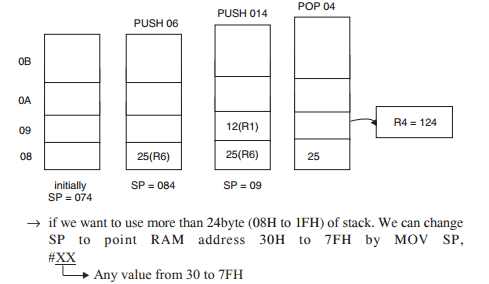
• It indicate current RAM address available for stack or it points the top of stack.

• Initially by default at 07H because first location of stack is 08H.

• After each PUSH instruction the SP is incremented by one while in MC after PUSH instruction SP is decremented.

• After each POP instruction the SP is decremented.





**Conflicting of Register Banks and Stack**

→ We know locations from 08H to 1FH is used as stack and it is also used as register bank.

→ If in the program, we use the Register Bank 1 to 3 and also use the stack then conflicts exist and error can be possible. For removing this situation we use the stack from location 30H to 7FH by shifting SP to 2FH.

MOV SP,#2FH;

**DPTR → Data Pointer in 8051**

→ 16 bit register, it is divided into two parts DPH and DPL.

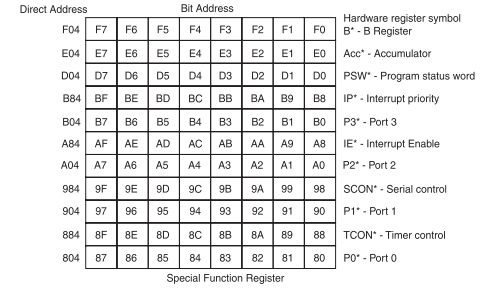
→ DPH for Higher order 8 bits, DPL for lower order 8 bits.

→ DPTR, DPH, DPL these all are SFRs in 8051.

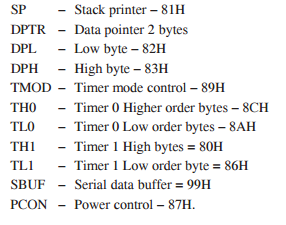
**Special Function Register**

→ (See Fig.) RAM scratch pad, there is extra 128 byte RAM which is used to store the SFRs

→ Following figure shows special function bit address, all access to the four I/O ports CPU register, interrupt control register, timer/counter, UART, power control are performed through registers between 80H and FFH.



**Byte Addressable SFR with byte address**



**Memory organization of 8051:**

Memory organization is depends on type of architecture used. There are two type of architectures are used in controller or processor generally: 1) Von Neumann architecture 2) Harvard architecture

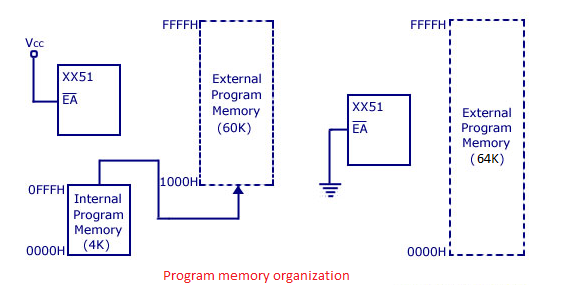
* **Von Neumann architecture**

The term Von Neumann architecture, also known as the Von Neumann model or the Princeton architecture. This architecture consist of address memory and data memory on a single unit. 8085 is based on Von Neumann architecture.

* **Harvard architecture**

Harvard architecture consists of program memory and data memory as separate unit. Thus for accessing Harvard architecture we need separate address bus, data bus and control bus. 8051 is based on Von Neumann architecture. Hence 8051 consist of two separate memory units, program memory as well as data memory.

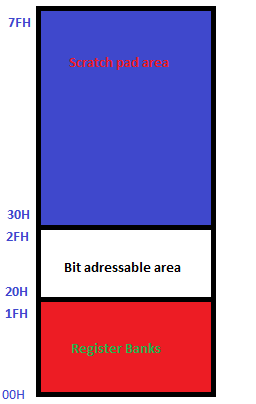
**Program memory organization:**

[](http://4.bp.blogspot.com/-e50yfAQqCBE/UibeZ9HxOBI/AAAAAAAAAYI/tSc2OUnovRQ/s1600/prom.png)

The program memory organization for 8051 family is as shown in fig. above. 8051 microcontroller has an on chip internal program ROM of 4K size and if needed we can add an external memory of size 60K maximum by interfacing. Hence total 64K size memory is available for 8051 microcontroller.  By default, the External Access (EA) pin should be connected Vcc its mean that instructions are fetched from internal memory initially.

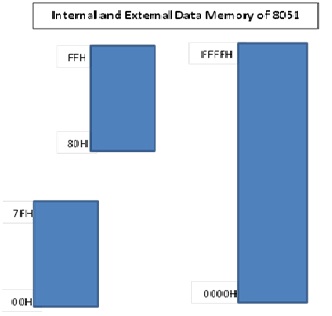
When we cross the internal limit of memory (4K), control of program will automatically goes to external memory and remaining instructions will fetch from external ROM. If we wants only external memory i.e. to fetch instruction from only external memory means if we want bypass internal program ROM, then we have to must connect External Access (EA) pin to ground  as shown in fig. above.

**Data memory organization:**

[](http://2.bp.blogspot.com/-wvrjbXgJLfQ/Uhu1dAe8L_I/AAAAAAAAAX0/CKJ4z2PrFwc/s1600/ram1.png)

In the 8051 family, 8051 has total 128 bytes of internal data RAM and we can interface external data memory up to 64K. Hence, total size of data memory in 8051 can be up to external 64K   + internal  128 bytes.  Internal RAM of 8051 is divided into 3 parts:- 1) Register banks 2) Bit addressable area 3) Scratch pad area.

There are 4 register banks in 8051 bank 0,1, 2 and 3. Each bank has 8 registers of 1 byte R0,R1…R7 respectively. Hence, register banks consist of the lowest 32 bytes of on chip RAM as shown in fig. above. At a time only one register bank can be selected for operations and bank registers are accessed using mnemonics R0..R1..etc.  By default register bank #0 is selected when we reset the system.



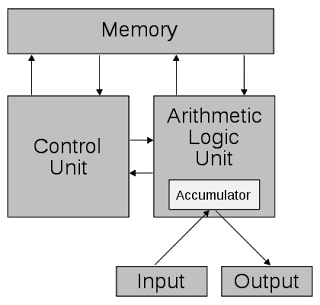
The bit addressable area is 16 bytes next to register banks. We can access each bit separately of bit addressable area, each bit have unique address of bit addressable area.The area of bit addressable space of 8051 is usually used to store bit variables address range 20H to 2FH (total 128 bits) is nothing but bit addressable area as shown in fig. Each bits can be accessed from 00H to 7FH within this 128 bits from 20H to 2FH.

Sometimes programming using bit addressable area saves wastage of memory. The upper 80 bytes are nothing but scratch pad area which is used for general purpose storing of data. Scratch pad area is in the address range 30H to 7FH. Scratch pad area can be used for stack memory also if default stack area is insufficient.

**HARVARD, VAN NEUMANN ARCHITECTURES:**

The Von Neumann and the Harvard Architecture is one important concept introduced in the basics of Computer Organization.

**The Von Neumann Architecture**

[](http://3.bp.blogspot.com/-MelXqxWSv9o/UBUxZ48PaeI/AAAAAAAABo8/4yboefCsi8U/s1600/von.png)

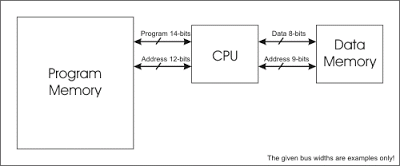
In the von Neumann Architecture, the computer consisted of a CPU, memory and I/O devices. The program is stored in the memory. The CPU fetches an instruction from the memory at a time and executes it.

Thus, the instructions are executed sequentially which is a slow process. Neumann m/c are called control flow computer because instruction are executed sequentially as controlled by a program counter. To increase the speed, parallel processing of computer have been developed in which serial CPU’s are connected in parallel to solve a problem. Even in parallel computers, the basic building blocks are Neumann processors.

The von Neumann architecture is a design model for a stored-program digital computer that uses a processing unit and a single separate storage structure to hold both instructions and data. It is named after mathematician and early computer scientist John von Neumann. Such a computer implements a universal Turing machine, and the common "referential model" of specifying sequential architectures, in contrast with parallel architectures.

One shared memory for instructions (program) and data with one data bus and one address bus between processor and memory. Instructions and data have to be fetched in sequential order (known as the Von Neuman Bottleneck), limiting the operation bandwidth. Its design is simpler than that of the Harvard architecture. It is mostly used to interface to external memory.

**HARVARD ARCHITECTURE**

[](http://3.bp.blogspot.com/-fQKx-Wcx4DA/UBUxu57HfeI/AAAAAAAABpE/tXLlrW5vI8Y/s1600/harvard.gif)

The term originated from the Harvard Mark 1 relay-based computer, which stored instructions on punched tape and data in relay latches.

Harvard Architecture: The Harvard architecture uses physically separate memories for their instructions and data, requiring dedicated buses for each of them. Instructions and operands can therefore be fetched simultaneously.

Different program and data bus widths are possible, allowing program and data memory to be better optimized to the architectural requirements. E.g.: If the instruction format requires 14 bits then program bus and memory can be made 14-bit wide, while the data bus and data memory remain 8-bit wide.

Harvard architecture is computer architecture with physically separate storage and signal pathways for instructions and data. The term originated from the Harvard Mark I relay-based computer, which stored instructions on punched tape (24 bits wide) and data in electro-mechanical counters (23 digits wide). These early machines had limited data storage, entirely contained within the data processing unit, and provided no access to the instruction storage as data, making loading and modifying programs an entirely offline process.

**Memory details**

In a Harvard architecture, there is no need to make the two memories share characteristics. In particular, the word width, timing, implementation technology, and  memory address structure can differ. Instruction memory is often wider than data memory. In some systems, instructions can be stored in read-only memory while data  memory generally requires read-write memory. In some systems, there is much more instruction memory than data memory so instruction addresses are much wider than data addresses.

**Contrast with other computer architectures**

In a computer with the contrasting von Neumann architecture (and no cache), the CPU can be either reading an instruction or reading/writing data from/to the memory. Both cannot occur at the same time since the instructions and data use the same bus system. In a computer using the Harvard architecture, the CPU can both read an instruction and reform a data memory access at the same time, even without a cache. A Harvard architecture computer can thus be faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.

The Modified Harvard architecture is very much like the Harvard architecture but provides a pathway between the instruction memory and the CPU that allows words  from the instruction memory to be treated as read-only data. This allows constant data, particularly text strings, to be accessed without first having to be copied into data memory, thus preserving more data memory for read/write variables. Special machine language instructions are provided to read data from the instruction memory. Most modern computers that are documented as Harvard Architecture are, in fact, ModifiedHarvard Architecture.

**Speed**

In recent years the speed of the CPU has grown many times in comparison to the access speed of the main memory. Care needs to be taken to reduce the number of times main memory is accessed in order to maintain performance. If, for instance, every instruction run in the CPU requires an access to memory, the computer gains nothing for increased CPU speed — a problem referred to as being memory bound.

It is possible to make extremely fast memory but this is only practical for small amounts of memory for both cost and signal routing reasons. The solution is to provide a small amount of very fast memory known as a CPU cache which holds recently accessed data. As long as the memory that the CPU needs is in the cache, the performance hit is much smaller than it is when the cache has to turn around and get the data from the main memory. Cache tuning is an important aspect of computer design.

Modern high performance CPU chip designs incorporate aspects of both Harvard and  von Neumann architecture. On-chip cache memory is divided into an instruction cache and a data cache. Harvard architecture is used as the CPU accesses the cache. In the case of a cache miss, however, the data is retrieved from the main memory, which is  not divided into separate instruction and data sections. Thus, while a von Neumann architecture is presented to the programmer, the hardware implementation gains the efficiencies of the Harvard architecture.

**Uses**

Harvard architectures are also frequently used in:

* Specialized digital signal processors, DSPs, commonly used in audio or video processing products. For example, Blackfin processors by Analog Devices, Inc. use a Harvard architecture.
* Most general purpose small microcontrollers used in many electronics applications, such as the PIC by Microchip Technology, Inc., and AVR by Atmel Corp. These processors are characterized by having small amounts of program and data memory, and take advantage of the Harvard architecture and reduced instruction sets (RISC) to ensure that most instructions can be executed within only one machine cycle, which is not necessarily one clock cycle. The separate storage means the program and data memories can have different bit depths.

**Example:**

PICs have an 8-bit data word but (depending on specific range of PICs) a 12-, 14-, or 16-bit program word. This allows a single instruction to contain a full-size data constant. Other RISC architectures, for example the ARM, typically must use at least two instructions to load a full-size constant.

**What is the difference between a von Neumann architecture and a Harvard architecture?**

**Harvard architecture** has separate data and instruction busses, allowing transfers to be performed simultaneously on both busses. Von **Neumann architecture** has only one bus which is used for both data transfers and instruction fetches, and therefore data transfers and instruction fetches must be scheduled - they cannot be performed at the same time.

It is possible to have two separate memory systems for Harvard **architecture**. As long as data and instructions can be fed in at the same time, then it doesn't matter whether it comes from a cache or memory. But there are problems with this. Compilers generally embed data (literal pools) within the code, and it is often also necessary to be able to write to the instruction memory space, for example in the case of self modifying code, or, if an ARM debugger is used, to set software breakpoints in memory. If there are two completely separate, isolated memory systems, this is not possible. There must be some kind of bridge between the memory systems to allow this.

Using a simple, unified memory system together with a Harvard architecture is highly inefficient. Unless it is possible to feed data into both busses at the same time, it might be better to use a von Neumann architecture processor.

**Use of caches**

At higher clock speeds, caches are useful as the memory speed is proportionally slower. **Harvard architectures** tend to be targeted at higher performance systems, and so caches are nearly always used in such systems.

**Von Neumann architectures** usually have a single unified cache, which stores both instructions and data. The proportion of each in the cache is variable, which may be a good thing. It would in principle be possible to have separate instruction and data caches, storing data and instructions separately. This probably would not be very useful as it would only be possible to ever access one cache at a time.

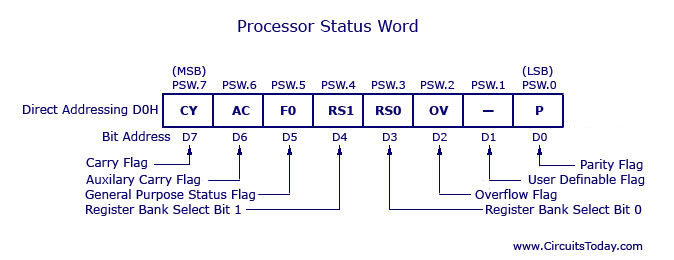
Caches for Harvard architectures are very useful. Such a system would have separate caches for each bus. Trying to use a shared cache on a Harvard architecture would be very inefficient since then only one bus can be fed at a time. Having two caches means it is possible to feed both buses simultaneously....exactly what is necessary for a Harvard architecture.

This also allows to have a very simple unified memory system, using the same address space for both instructions and data. This gets around the problem of literal pools and self modifying code. What it does mean, however, is that when starting with empty caches, it is necessary to fetch instructions and data from the single memory system, at the same time. Obviously, two memory accesses are needed therefore before the core has all the data needed. This performance will be no better than a von Neumann architecture. However, as the caches fill up, it is much more likely that the instruction or data value has already been cached, and so only one of the two has to be fetched from memory. The other can be supplied directly from the cache with no additional delay. The best performance is achieved when both instructions and data are supplied by the caches, with no need to access external memory at all.

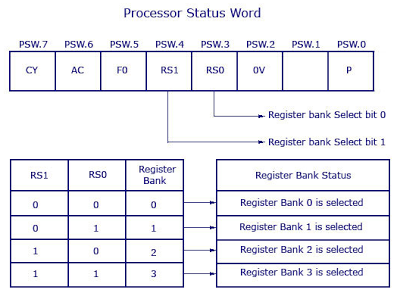
This is the most sensible compromise and the architecture used by ARMs Harvard processor cores. Two separate memory systems can perform better, but would be difficult to implement.

**PSW REGISTER FORMAT**

Commonly known as the PSW register, this is a vital SFR in the functioning of micro controller. This register reflects the status of the operation that is being carried out in the processor. The picture below shows PSW register and the way register banks are selected using PSW register bits – RS1 and RS0. PSW register is both bit and byte addressable. The physical address of PSW starts from D0H. The individual bits are then accessed using D1, D2 … D7.  The various individual bits are explained below.

[](http://www.circuitstoday.com/wp-content/uploads/2012/01/Processor-Status-Word.jpg)

The program status word (PSW) register is an 8-bit register.It is also referred to as the flag register. Although the PSW register is 8 bits wide, only 6 bits of it are used by the 8051.The two unused bits are user-definable flags. Four of the flags are called conditional flags, meaning that they indicate some conditions that result after an instruction is executed. These four are CY (carry), AC (auxiliary carry), P (parity), and OV (overflow).As seen from below figure, the bits PSW.3 and PSW.4 are designated as RS0 and RS1 as register selection bit, respectively, and are used to change the bank registers.The PSW.5 and PSW.l bits are general-purpose status flag bits and can be used by the programmer for any purpose. In other words, they are user definable. See below Figure for the bits of the PSW register.

[](http://4.bp.blogspot.com/-OxAlTPnQyQI/UdKLqduk4uI/AAAAAAAAANw/9lq_mWeTbrY/s528/psw-1.png)

**FORMAT OF PSW OF 8051**

The following is a brief explanation of four of the flag bits of the PSW register. The impact of instructions on these registers is then discussed.

**How the bits of PSW are change after arithmetic & logical operation?**  
  
**CY, the carry flag**

* This flag is set whenever there is a carry out from the D7 bit.
* This flag bit is affected after an 8-bit addition or subtraction.
* It can also be set to 1 or 0 directly by an instruction such as “SETB C” and “CLR C”where “SETB C” stands for “set bit carry” and “CLR C” for “clear carry”.

**AC, the auxiliary carry flag**

* If there is a carry from **D3 to D4 during an ADD or SUB** operation, this bit is set; otherwise, it is cleared.
* This flag is used by instructions that perform BCD (binary coded decimal) arithmetic.

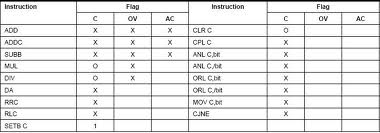
**P, the parity flag**

* The parity flag reflects the number of 1 s in the A (accumulator) register only.
* If the A register contains an odd number of Is, then P = 1. Therefore, P = 0 if A has an even number of 1s.

**OV, the overflow flag**

* This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.
* In general, the carry flag is used to detect errors in unsigned arithmetic operations.
* The overflow flag is only used to detect errors in signed arithmetic operations

**Instruction that affect flags of PSW**



**LCD:**

LCD (Liquid Crystal Display) is the innovation utilized in scratch pad shows and other littler PCs. Like innovation for light-producing diode (LED) and gas-plasma, LCDs permit presentations to be a lot more slender than innovation for cathode beam tube (CRT). LCDs expend considerably less power than LED shows and gas shows since they work as opposed to emanating it on the guideline of blocking light.

A LCD is either made with a uninvolved lattice or a showcase network for dynamic framework show. Likewise alluded to as a meager film transistor (TFT) show is the dynamic framework LCD. The uninvolved LCD lattice has a matrix of conductors at every crossing point of the network with pixels. Two conductors on the lattice send a current to control the light for any pixel. A functioning framework has a transistor situated at every pixel crossing point, requiring less current to control the luminance of a pixel.

Some aloof network LCD's have double filtering, which implies they examine the matrix twice with current in the meantime as the first innovation took one sweep. Dynamic lattice, be that as it may, is as yet a higher innovation.

A 16x2 LCD show is an essential module that is generally utilized in various gadgets and circuits. These modules more than seven sections and other multi fragment LEDs are liked. The reasons being: LCDs are affordable; effectively programmable; have no restriction of showing exceptional and even custom characters (not at all like in seven fragments), movements, etc.

A 16x2 LCD implies 16 characters can be shown per line and 2 such lines exist. Each character is shown in a lattice of 5x7 pixels in this LCD. There are two registers in this LCD, in particular Command and Data.

The directions given to the LCD are put away by the order register. An order is a direction given to LCD to play out a predefined assignment, for example, introducing it, clearing its screen, setting the situation of the cursor, controlling presentation, and so forth. The information register will store the information that will be shown on the LCD. The information is the character's ASCII incentive to show on the LCD.

**Data/Signals/Execution of LCD**

Now that was all about the signals and the hardware. Let us come to data, signals and execution.

Two types of signals are accepted by LCD, one is data and one is control. The LCD module recognizes these signals from the RS pin status. By pulling the R / W pin high, data can now also be read from the LCD display. Once the E pin has been pulsed, the LCD display reads and executes data at the falling edge of the pulse, the same for the transmission case.

It takes 39-43μS for the LCD display to place a character or execute a command. It takes 1.53ms to 1.64ms except for clearing display and searching for cursor to the home position.

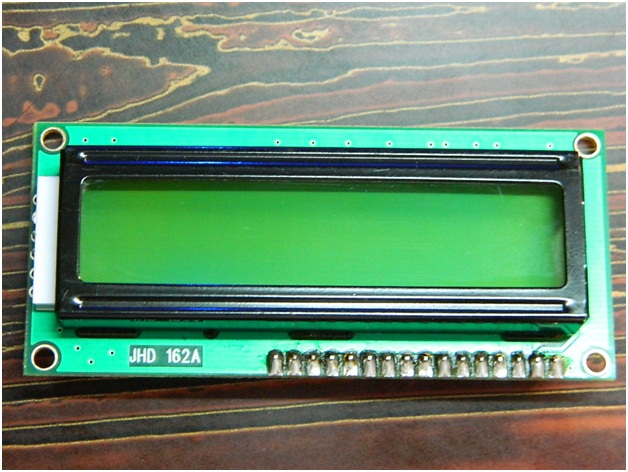
Any attempt to send data before this interval may result in failure in some devices to read data or execute the current data. Some devices compensate for the speed by storing some temporary registers with incoming data.

There are two RAMs for LCD displays, namely DDRAM and CGRAM. DDRAM registers the position in which the character would be displayed in the ASCII chart. Each DDRAM byte represents every single position on the display of the LCD.

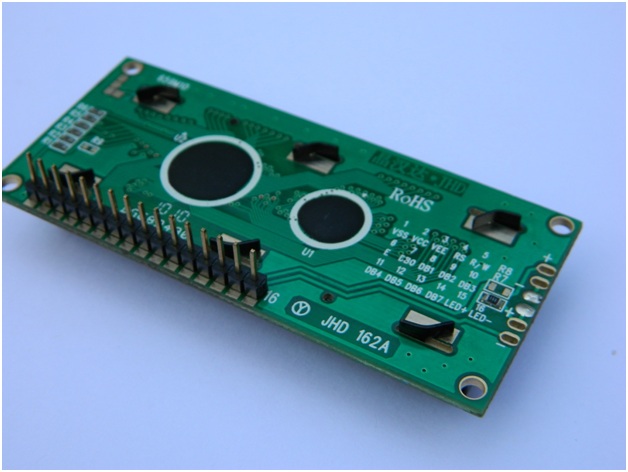
The DDRAM information is read by the LCD controller and displayed on the LCD screen. CGRAM enables users to define their personalized characters. Address space is reserved for users for the first 16 ASCII characters.

Users can easily display their custom characters on the LCD screen after CGRAM has been set up to display characters.

**Images of LCD Display:-**

[](http://www.circuitstoday.com/wp-content/uploads/2012/02/LCD-Display-Front-Side.jpg)

**LCD – Front View**

[](http://www.circuitstoday.com/wp-content/uploads/2012/02/lcd-display-back-side.jpg)

**LCD – Back View**

**Pin Diagram:**



**Pin Description:**

|  |  |  |
| --- | --- | --- |
| Pin No | Function | Name |
| 1 | Ground (0V) | Ground |
| 2 | Supply voltage; 5V (4.7V – 5.3V) | Vcc |
| 3 | Contrast adjustment; through a variable resistor | VEE |
| 4 | Selects command register when low; and data register when high | Register Select |
| 5 | Low to write to the register; High to read from the register | Read/write |
| 6 | Sends data to data pins when a high to low pulse is given | Enable |
| 7 | 8-bit data pins | DB0 |
| 8 | DB1 |
| 9 | DB2 |
| 10 | DB3 |
| 11 | DB4 |
| 12 | DB5 |
| 13 | DB6 |
| 14 | DB7 |
| 15 | Backlight VCC (5V) | Led+ |
| 16 | Backlight Ground (0V) | Led- |

**RS (Register select)**

A 16X2 LCD has two order and information registers. The determination of the register is utilized to change starting with one register then onto the next. RS=0 for the register of directions, while RS=1 for the register of information.

**Command Register**

The guidelines given to the LCD are put away by the direction register. An order is a direction given to LCD to play out a predefined assignment, for example, instating it, clearing its screen, setting the situation of the cursor, controlling showcase, and so on. Order preparing happens in the direction register.

**Data Register:**

The information register will store the information that will be shown on the LCD. The information is the character's ASCII incentive to show on the LCD. It goes to the information register and is prepared there when we send information to the LCD. While choosing RS=1, the information register.

**Read and Write Mode of LCD:**

As stated, the LCD itself comprises of an interface IC. This interface IC can be perused or composed by the MCU. A large portion of the occasions we're simply going to keep in touch with the IC since perusing will make it increasingly perplexing and situations like that are exceptionally uncommon. Information such as cursor position, status completion interrupts, etc. can be read if necessary.

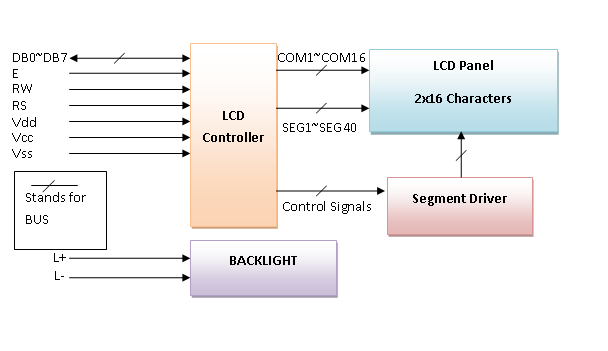
**LCD Commands:**

There are some preset commands in the LCD that we need to send to the LCD via some microcontroller. The following are some important command instructions:

|  |  |  |
| --- | --- | --- |
| **Sr.No.** | **Hex Code** | **Command to LCD instruction Register** |
| 1 | 01 | Clear display screen |
| 2 | 02 | Return home |
| 3 | 04 | Decrement cursor (shift cursor to left) |
| 4 | 06 | Increment cursor (shift cursor to right) |
| 5 | 05 | Shift display right |
| 6 | 07 | Shift display left |
| 7 | 08 | Display off, cursor off |
| 8 | 0A | Display off, cursor on |
| 9 | 0C | Display on, cursor off |
| 10 | 0E | Display on, cursor blinking |
| 11 | 0F | Display on, cursor blinking |
| 12 | 10 | Shift cursor position to left |
| 13 | 14 | Shift cursor position to right |
| 14 | 18 | Shift the entire display to the left |
| 15 | 1C | Shift the entire display to the right |
| 16 | 80 | Force cursor to beginning ( 1st line) |
| 17 | C0 | Force cursor to beginning ( 2nd line) |
| 18 | 38 | 2 lines and 5×7 matrix |

## Command codes for LCD

**Block Diagram of LCD Display:-**

**[](http://www.circuitstoday.com/wp-content/uploads/2012/02/LCD-Display-Block-Diagram.png)**

**Control and display commands**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Instruction Code** | | | | | | | | | | **Instruction Code Description** | **Execution time** |
| RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 |
| Read Data From RAM | 1 | 1 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Read data from internal RAM | 1.53-1.64ms |
| Write data to RAM | 1 | 0 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Write data into internal RAM (DDRAM/CGRAM) | 1.53-1.64ms |
| Busy flag & Address | 0 | 1 | BF | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Busy flag (BF: 1→ LCD Busy) and contents of address counter in bits AC6 AC0. | 39 µs |
| Set DDRAM Address | 0 | 0 | 1 | AC6 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Set DDRAM address in address counter. | 39 µs |
| Set CGRAM Address | 0 | 0 | 0 | 1 | AC5 | AC4 | AC3 | AC2 | AC1 | AC0 | Set CGRAM Address in address counter. | 39 µs |
| Function Set | 0 | 0 | 0 | 0 | 1 | DL | N | F | X | X | Set interface data length (DL: 4bit/8bit), Numbers of display line (N: 1-line/2-line) display font type (F:0→ 5×8 dots, F:1→ 5×11 dots) | 39 µs |
| Cursor or Display Shift | 0 | 0 | 0 | 0 | 0 | 1 | S/C | R/L | X | X | Set cursor moving and display shift control bit, and the direction without changing DDRAM data | 39 µs |
| Display & Cursor On/Off | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Set Display(D),Cursor(C) and cursor blink(b) on/off control | 39 µs |
| Entry Mode Set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | SH | Assign cursor moving direction and enable shift entire display. | 0µs |
| Return Home | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | X | Set DDRAM Address to “00H” from AC and return cursor to its original position if shifted. | 43µs |
| Clear Display | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Write “20H” to DDRAM and set DDRAM Address to “00H” from AC | 43µs |

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

The LCD can work in two striking modes, the 4-bit mode and the 8-bit mode. We send the information snack through snack in 4 bit mode, first upper chomp, by then lower snack. For those of you who don't have the foggiest idea what a goody is: a chomp is a four-piece gathering, so a byte's lower four bits (D0-D3) are the lower snack, while a byte's upper four bits (D4-D7) are the higher snack. This enables us to send 8 bit data. This connects with us to send 8 bit data. Whereas in 8 bit mode we can send the 8-bit information truly in one stroke since we utilize all the 8 information lines. You need to get it now; yes 8-bit mode is quicker and immaculate than 4-bit mode. In any case, the fundamental shortcoming is that it needs 8 microcontroller-related information lines. This will result in our MCU coming up short on I/O pins, so 4-bit mode is extensively utilized. To set these modes, no control pins are used.

**Software requirements**

**ABOUT KEIL:**

The µVision IDE from Keil joins venture administration, make offices, source code altering, system investigating, and finish recreation in one effective environment. The µVision advancement stage is anything but difficult to-utilize and helping you rapidly make inserted programs that work. The µVision editorial manager and debugger are coordinated in a solitary application that gives a consistent inserted venture advancement environment.

The Keil vision microcontroller Development Tools are intended to take care of the unpredictable issues confronting implanted programming designers.

* When beginning another undertaking, essentially select the microcontroller you use from the Device Database and the µVision IDE sets all compiler, constructing agent, linker, and memory alternatives for you.
* Numerous sample projects are incorporated to offer you some assistance with getting started with the most mainstream installed 8051 gadgets.
* The Keil µVision Debugger precisely reproduces on-chip peripherals (I²C, CAN, UART, SPI, Interrupts, I/O Ports, A/D Converter, D/A Converter, and PWM Modules) of your 8051 gadget.
* Simulation offers you some assistance with understanding equipment designs and stays away from time squandered on setup issues. Moreover, with reenactment, you can compose and test applications before target equipment is accessible.
* When you are prepared to start testing your product application with target equipment, utilize the MON51, MON390, MONADI, or FlashMON51 Target Monitors, the ISD51 In-System Debugger, or the ULINK USB-JTAG Adapter to download and test project code on your objective framework, streak enchantment is likewise an extremely surely understood for hex loader.

8051 microcontroller can be programmed in two languages

* Assembly language
* C language

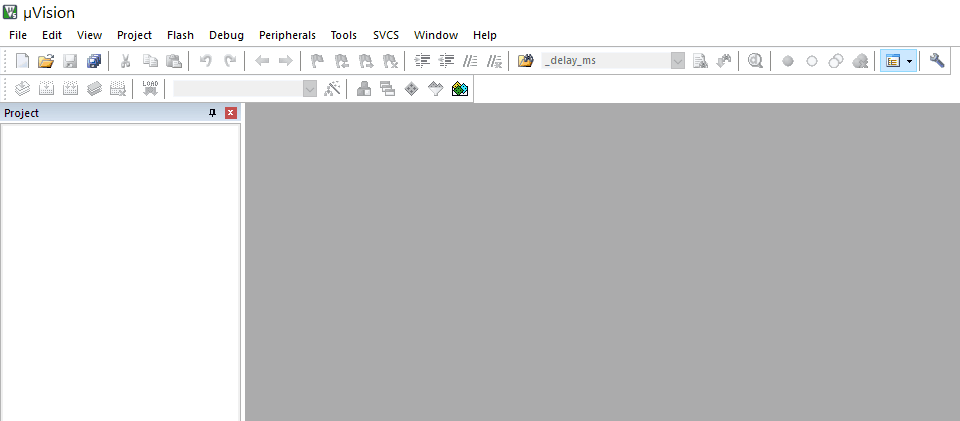
8051 microcontroller popular development IDE is MCU 8051 IDE and µVision to develop code.

Keil µVision IDE consists,

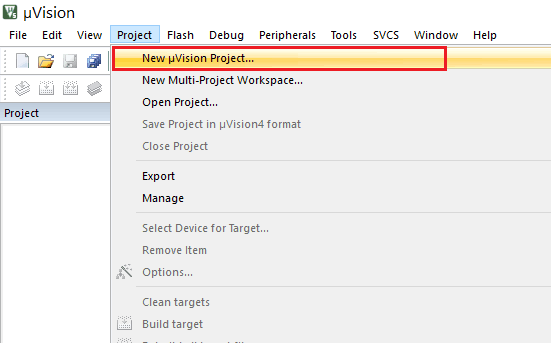
* C Compiler - C51.Exe
* Assembler - A51.Exe
* Linker/Locator - BL51.Exe
* Librarian - LIB51.Exe
* Hex Converter - OH51.Exe

Let’s develop simple LED blinking program using Keil µVision IDE with C51 compiler. here we are using AT89S52 microcontroller from 8051 family.

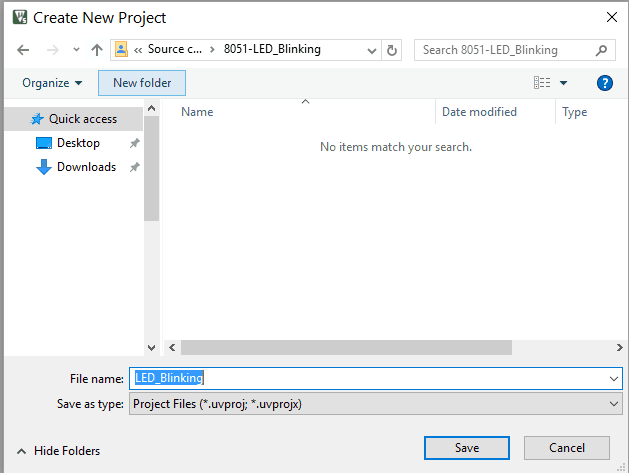
Now open Keil µVision



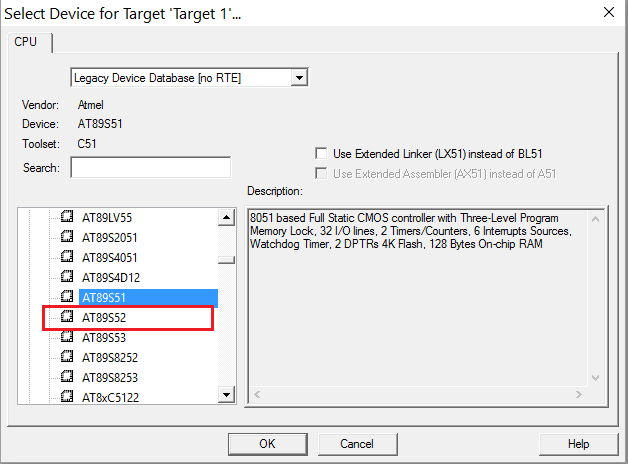
Click on Project menu and select New µVision Project…



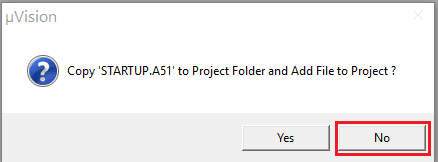
“Create New Project” window will pop up, type a project name and location for project and save.



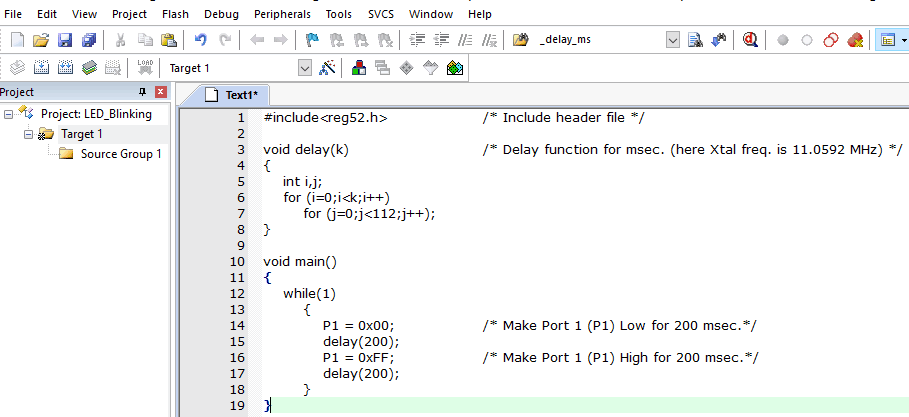
“Select Device for Target” window will pop up, select your device (here we selecting AT89S52)



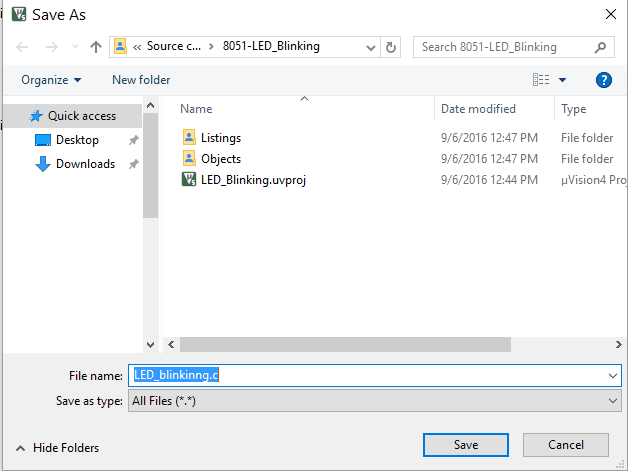
“µVision” window will ask for copy STARTUP.A51 to project folder and add file to project (here it not necessary so we have selected No)



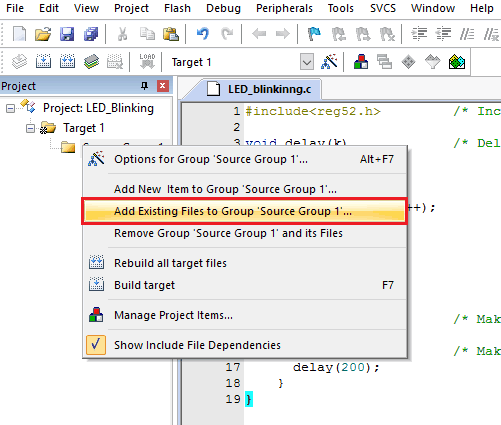
Now select New file from File menu and type your program code (here we have typed LED blinking program)



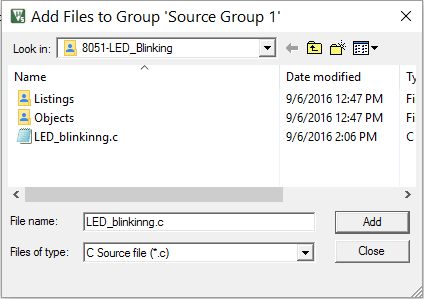
Save program code with “ .c “ extension (In case if you are using assembly language then save program code with “ .asm “ extension)



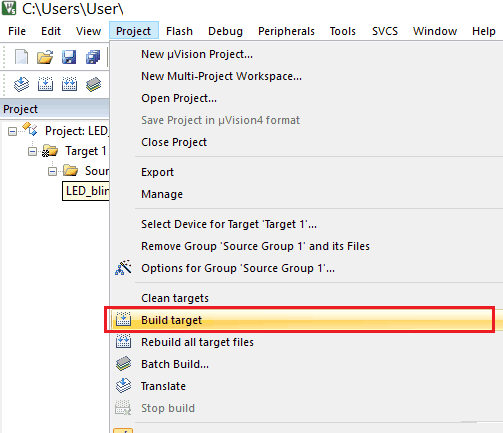
Right click on Source Group 1 folder from Target 1 and select “Add existing files to Group ‘Source Group 1’



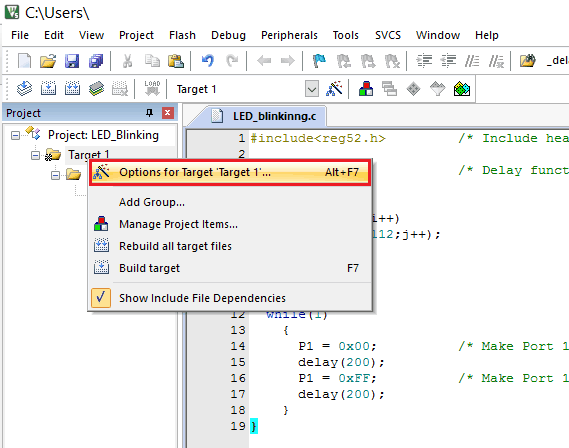
Select program file saved with “.c “or “.asm” (in case of assembly language) and add it. Then close that window. You can see added file in “Source Group 1” folder in left project window



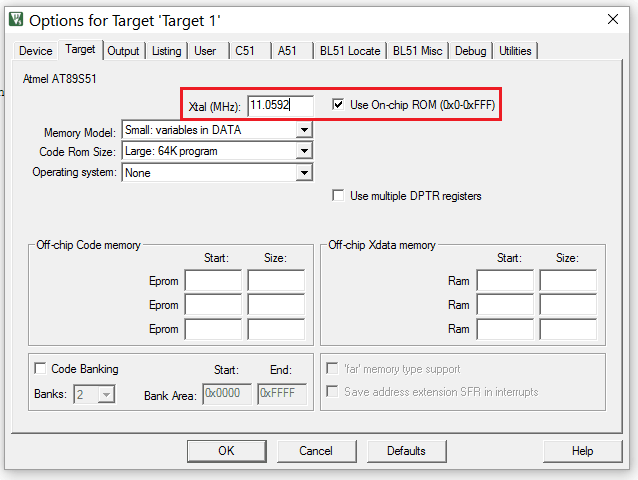
Now select Project menu and click on “Build target”, it will build project and give status in Build output window with Error and Warning count if any.



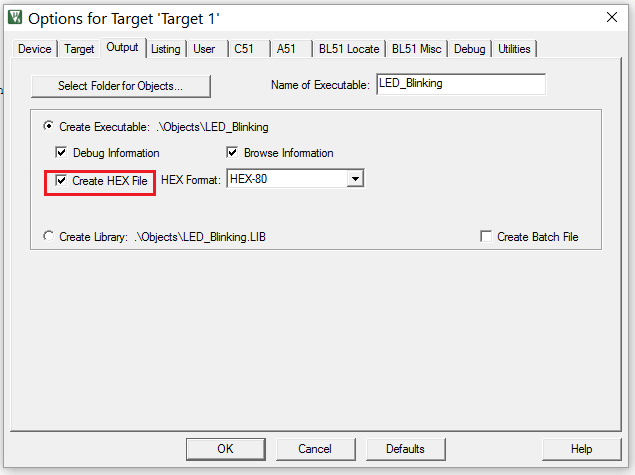
To create Hex file right click on Target 1 and select Option for Target ‘Target 1’



Target window will pop up, enter Xtal (MHz) frequency (here we used 11.0592 MHz) and tick mark in front of “Use On-chip ROM” tag.



Within same window select “Output” option and tick mark in front of “Create Hex File” tag and click on OK.



**Advantages:**

* The chance of loss of life and property due to drunken driving is minimized.
* Simple implementation leads to accurate results.
* Can be implemented on various types of vehicles.
* Less accidents, more safety.

**Applications:**

* This system can be implemented in vehicles to avoid accidents due to drunken driving.
* It can also be used by various organizations or authorities to monitor its employees and keep a check on them.

Conclusion:

* In this project, we have developed an efficient system to tackle the menace of drunken driving. Our main aim is to minimize the loss of lives and property which happen due to drunken driving. This system once implemented on a large scale will prove to be really helpful by shutting down the vehicle’s engine and alerting the nearby people before any mishap takes place.
* The sensor used in the project is very accurate and can be configured according to the requirements thereby increasing the efficiency.

Reference:

* Mugila J., Muthu lakshmi.M, Santhiya K, Prof. Dhivya. P [International Journal of Innovative Research in Science Engineering and Technology (IJIRTSE) ISSN: 2395-5619, Volume – 2, Issue – 7. July 2016] - “Smart helmet system using alcohol detection for vehicle protection”.
* Dhivya M and Kathiravan S, Dept. of ECE, Kalaignar Karunanidhi Institute of Technology [Smart Computing Review, vol. 5, no. 1, February 2015] - “Driver Authentication and Accident Avoidance System for Vehicles”.
* Nimmy James, Aparna C, Tenna P John, International Journal of Research in Computer and Communication Technology, Vol 3, Issue 1, January- 2014 – “Alcohol Detection System”.
* Prashanth K P1, Kishen Padiyar2, Naveen Kumar P H3, K Santhosh Kumar, Dept. of Mechanical Engineering, East West Institute of Technology, Bangalore, India [International Journal of Engineering Research & Technology (IJERT) ISSN: 2278-0181, IJERTV3IS100754, Vol. 3 Issue 10, October- 2014] - “Road Accident Avoiding System using Drunken Sensing Technique”.
* Ms. Subia Sayeed, Department of Electronics and communication, VVIET, Mysore, India [International Journal of Scientific & Engineering Research Volume 2, Issue 12, December-2011 1, ISSN 2229-5518] - “Drunken drive protection system”