CELL PHONE OPERATED LAND ROVER FOR MULTIPLE TASKS

Dyava Rama Krishna Reddy

A thesis submitted in partial fulfillment for the
Degree of Bachelor of Technology

in the Electrical and Electronic Engineering

Electrical and Electronic Engineering Department
Jawaharlal Nehru Technological University Hyderabad

Supervisor: Prof. Vidya Sagar

Jan 2013

INDEX

ABSTRACT

Robotic manipulators are widely used to replace human operators in tasks that are repetitive in nature. However, there are many tasks that are non-repetitive, unpredictable, or hazardous to the human operators. Clearing up a nuclear power plant leak or exploring the extreme depths of ocean are just some examples. The most developed robot in practical use today is the robotic arm and it is seen in applications throughout the world. Robotic are used to carry out work in outer space where man cannot survive and also used to do work in the medical field such as conducting experiments without exposing the researcher.

In early days, robotic manipulators have been implemented in different control techniques like mechanical control and the remote control or tele-opertation. But with the advent of high performance, a new way of control using mobile has been implemented which is introduced in this project.

All the above systems are controlled by the Microcontroller. In our project we are using the popular 8 bit microcontroller AT89S52. It is a 40 pin microcontroller. The Microcontroller AT89S52 is used to control the dc motors. It gets the signals from the DTMF decoder and it drives the motors according to the DTMF inputs. Two DC motors are used to drive the robot in four directions i.e. Front ,Back,Right,Left.

CHAPTER 1

1.1 INTRODUCTION

Now a day's every system is automated in order to face new challenges in the present day situation. Automated systems have less manual operations, so that the flexibility, reliabilities are high and accurate. Hence every field prefers automated control systems. Especially in the field of electronics automated systems are doing better performance.

Probably the most useful thing to know about the global system for mobile communication is that it is an international standard. If you travel in parts of world, GSM is only type of cellular service available. Instead of analog services, GSM was developed as a digital system using TDMA technology.

OBJECTIVE:

The goal of the project is to develop a system, which uses Mobile technology that keeps control of the various Home Appliances, which executes with respect to the signal sent by the mobile and all appliances are password protected. One can operate the appliances by entering the correct password, else no access to the device.

For utilization of appliances the new concept has been thought to manage them remotely by using GSM, which enables the user to remotely control switching of domestic appliances. Just by dialing keypad of remote telephone, from where you are calling you can perform ON / OFF operation of the appliances.

The ranges of appliances that can be controlled through tele remote systems are many in numbers. Some of them are as follows and this depends upon the usage priority of the appliances i.e. parking Lights, Music System or other electrical / electronic appliances.

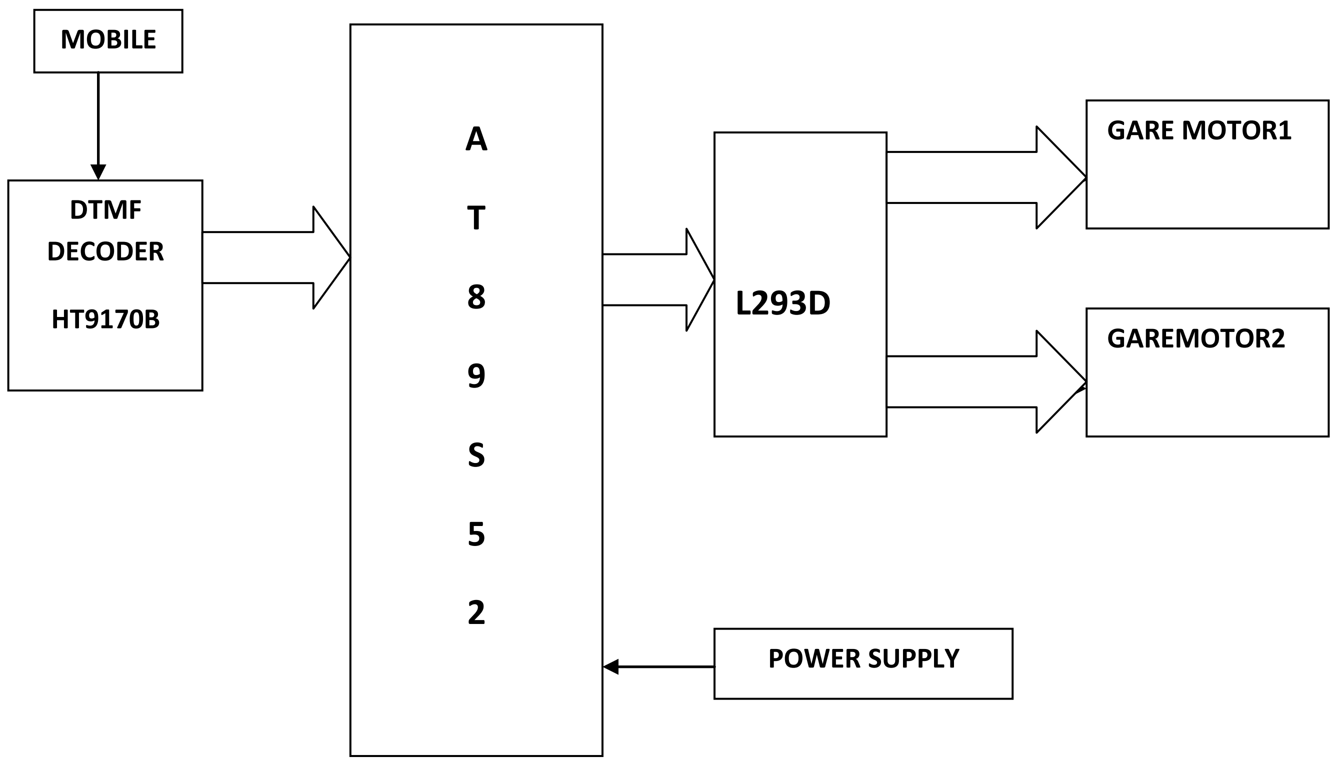
HARDWARE:

1. 8052 MICROCONTROLLER.
2. MOBILE.
3. DTMF DECODER IC.
4. RELAY
5. ULN2003

SOFTWARE:

1. KEIL MICROVISION
2. EMBEDDED C

BLOCK DIAGRAM:



CHAPTER 2

DESCRIPTION OF HARDWARE COMPONENTS

2.1 AT89S52

2.2.1 A BRIEF HISTORY OF 8051

In 1981, Intel corporation introduced an 8 bit microcontroller called 8051. this microcontroller had 128 bytes of RAM, 4K bytes of chip ROM, two timers, one serial port, and four ports all on a single chip. At the time it was also referred as " A SYSTEM ON A CHIP"

The 8051 is an 8-bit processor meaning that the CPU can work only on 8 bits data at a time. Data larger than 8 bits has to be broken into 8 bits pieces to be processed by the CPU. The 8051 has a total of four I\O ports each 8 bit wide.

There are many versions of 8051 with different speeds and amount of on-chip ROM and they are all compatible with the original 8051. this means that if you write a program for one it will run on any of them.

The 8051 is an original member of the 8051 family. There are two other members in the 8051 family of microcontrollers. They are 8052 and 8031. All the three microcontrollers will have the same internal architecture, but they differ in the following aspects.

- 8031 has 128 bytes of RAM, two timers and 6 interrupts.
- 8051 has 4K ROM, 128 bytes of RAM, two timers and 6 interrupts.
- 8052 has 8K ROM, 256 bytes of RAM, three timers and 8 interrupts.

Of the three microcontrollers, 8051 is the most preferable. Microcontroller supports both serial and parallel communication.

In the concerned project 8052 microcontroller is used. Here microcontroller used is AT89S52, which is manufactured by ATMEL laboratories.

NECESSITY OF MICROCONTROLLERS:

Microprocessors brought the concept of programmable devices and made many applications of intelligent equipment. Most applications, which do not need large amount of data and program memory, tended to be costly.

The microprocessor system had to satisfy the data and program requirements so, sufficient RAM and ROM are used to satisfy most applications .The peripheral control equipment also had to be satisfied. Therefore, almost all-peripheral chips were used in the design. Because of these additional peripherals cost will be comparatively high.

An example:
8085 chip needs:

An Address latch for separating address from multiplex address and data.32-KB RAM and 32-KB ROM to be able to satisfy most applications. As also Timer / Counter, Parallel programmable port, Serial port, and Interrupt controller are needed for its efficient applications.

In comparison a typical Micro controller 8051 chip has all that the 8051 board has except a reduced memory as follows.
4K bytes of ROM as compared to 32-KB, 128 Bytes of RAM as compared to 32-KB.

Bulky:

On comparing a board full of chips (Microprocessors) with one chip with all components in it (Microcontroller).

Debugging:

Lots of Microprocessor circuitry and program to debug. In Micro controller there is no Microprocessor circuitry to debug.

Slower Development time: As we have observed Microprocessors need a lot of debugging at board level and at program level, where as, Micro controller do not have the excessive circuitry and the built-in peripheral chips are easier to program for operation.

So peripheral devices like Timer/Counter, Parallel programmable port, Serial Communication Port, Interrupt controller and so on, which were most often used were integrated with the Microprocessor to present the Micro controller .RAM and ROM also were integrated in the same chip. The ROM size was anything from 256 bytes to 32Kb or more. RAM was optimized to minimum of 64 bytes to 256 bytes or more.

Microprocessor has following instructions to perform:

1. Reading instructions or data from program memory ROM.

2. Interpreting the instruction and executing it.

3. Microprocessor Program is a collection of instructions stored in a Nonvolatile memory.

4. Read Data from I/O device

5. Process the input read, as per the instructions read in program memory.

6. Read or write data to Data memory.

7. Write data to I/O device and output the result of processing to O/P device.

2.1.2 Introduction to AT89S52

The system requirements and control specifications clearly rule out the use of 16, 32 or 64 bit micro controllers or microprocessors. Systems using these may be earlier to implement due to large number of internal features. They are also faster and more reliable but, the above application is satisfactorily served by 8-bit micro controller. Using an inexpensive 8-bit Microcontroller will doom the 32-bit product failure in any competitive market place. Coming to the question of why to use 89S52 of all the 8-bit Microcontroller available in the market the main answer would be because it has 8kB Flash and 256 bytes of data RAM32 I/O lines, three 16-bit timer/counters, a Eight-vector two-level interrupt architecture, a full duplex serial port, on-chip oscillator, and clock circuitry.

In addition, the AT89S52 is designed with static logic for operation down to zero frequency and supports two software selectable power saving modes. The Idle Mode stops the CPU while allowing the RAM, timer/counters, serial port, and interrupt system to continue functioning. The Power Down Mode saves the RAM contents but freezes the oscillator, disabling all other chip functions until the next hardware reset. The Flash program memory supports both parallel programming and in Serial In-System Programming (ISP). The 89S52 is also In-Application Programmable (IAP), allowing the Flash program memory to be reconfigured even while the application is running.

By combining a versatile 8-bit CPU with Flash on a monolithic chip, the Atmel AT89S52 is a powerful microcomputer which provides a highly flexible and cost effective solution to many embedded control applications.

2.1.3 FEATURES

Compatible with MCS-51® Products

• 8K Bytes of In-System Programmable (ISP) Flash Memory

– Endurance: 1000 Write/Erase Cycles

• 4.0V to 5.5V Operating Range

• Fully Static Operation: 0 Hz to 33 MHz

• Three-level Program Memory Lock

• 256 x 8-bit Internal RAM

• 32 Programmable I/O Lines

• Three 16-bit Timer/Counters

• Eight Interrupt Sources

• Full Duplex UART Serial Channel

• Low-power Idle and Power-down Modes

• Interrupt Recovery from Power-down Mode

• Watchdog Timer

• Dual Data Pointer

-Power-off Flag

PIN DIAGRAM

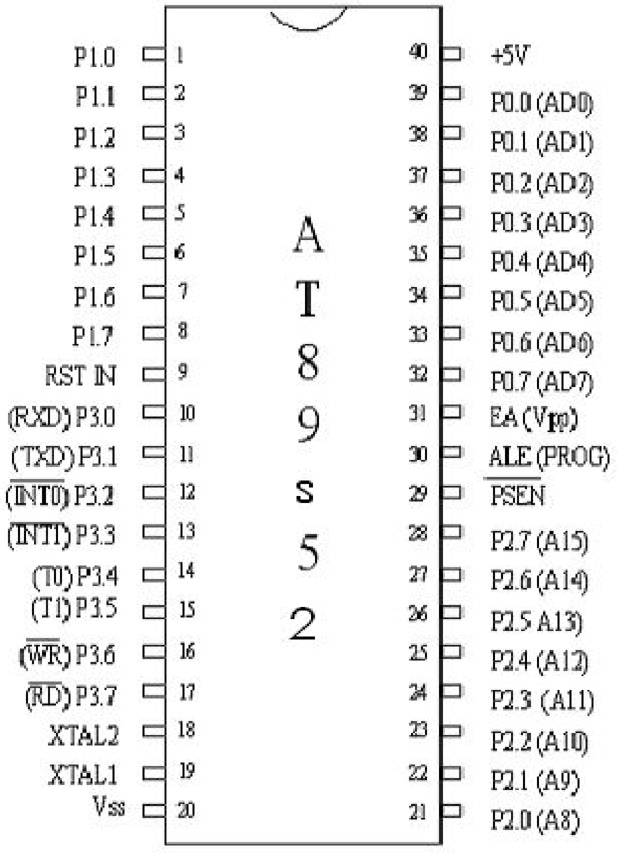


FIG-2 PIN DIAGRAM OF 89S52 IC

2.1.4 PIN DESCRIPTION

Pin Description

VCC: Supply voltage.

GND: Ground.

Port 0

Port 0 is an 8-bit open drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high- impedance inputs. Port 0 can also be configured to be the multiplexed low- order address/data bus during accesses to external pro-gram and data memory. In this mode, P0 has internal pullups

Port 0 also receives the code bytes during Flash program- ming and outputs the code bytes dur ing program verification. External pullups are required during program verification.

Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pullups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups. In addition, P1.0 and P1.1 can be configured to be the timer/counter 2 external count input (P1.0/T2) and the timer/counter 2 trigger input (P1.1/T2EX), respectively, as shown in the following table.

Port 1 also receives the low-order address bytes during

Flash programming and verification

Port Pin	Alternate Functions
P1.0	T2 (external count input to Timer/Counter 2), clock-out
P1.1	T2EX (Timer/Counter 2 capture/reload trigger and direction control)

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pullups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (I_{IL}) because of the internal pullups.Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pul- lups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

Port 3

Port 3 is an 8-bit bi-directional I/O port with internal pullups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins, they are pulled high by the internal pullups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (I_{IL}) because of the pullups. Port 3 also serves the functions of various special features of the AT89C51, as shown in the following table.

Port 3 also receives some control signals for Flash pro- gramming and verification.

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	INT0 (external interrupt 0)
P3.3	INT1 (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	WR (external data memory write
P3.7	RD (external data memory read strobe)

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device.

ALE/PROG

Address Latch Enable is an output pulse for latching the low byte of the address during accesses to external mem- ory. This pin is also the program pulse input (PROG) during ~~Flash~~ programming.

In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external timing or clocking Note, however, that one ALE pulse is skipped during each access to external data memory. If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only dur- ing a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the

ALE-disable bit has no effect if the microcontroller is in external execution mode.

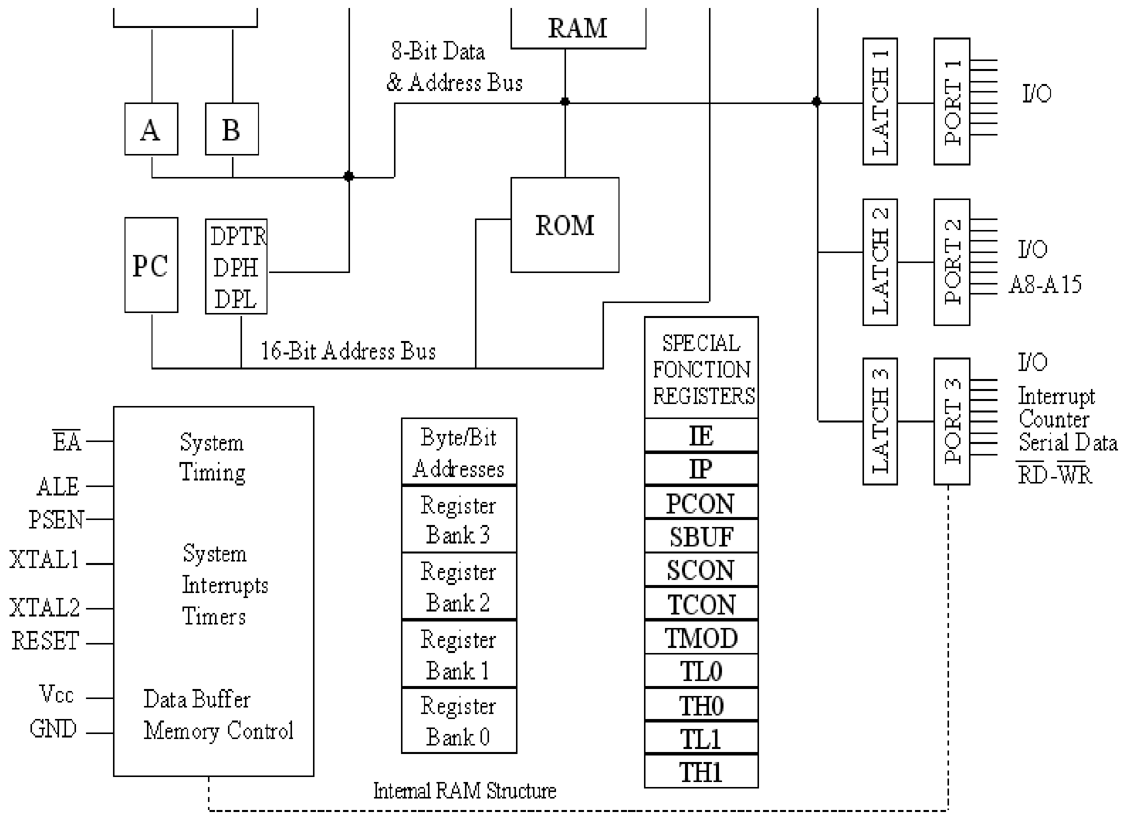
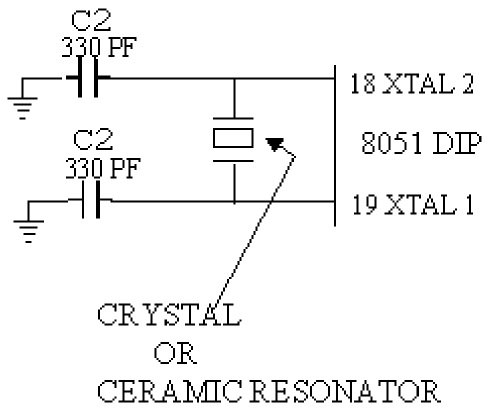


FIG-3 Functional block diagram of micro controller

The 8052 Oscillator and Clock:

The heart of the 8051 circuitry that generates the clock pulses by which all the internal all internal operations are synchronized. Pins XTAL1 And XTAL2 is provided for connecting a resonant network to form an oscillator. Typically a quartz crystal and capacitors are employed. The crystal frequency is the basic internal clock frequency of the microcontroller. The manufacturers make 8051 designs that run at specific minimum and maximum frequencies typically 1 to 16 MHz.



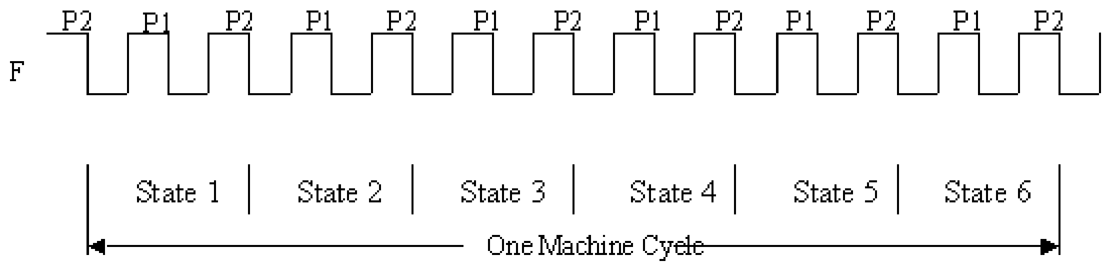


Fig-4 Oscillator and timing circuit

MEMORIES

Types of memory:

The 8052 have three general types of memory. They are on-chip memory, external Code memory and external Ram. On-Chip memory refers to physically existing memory on the micro controller itself. External code memory is the code memory that resides off chip. This is often in the form of an external EPROM. External RAM is the Ram that resides off chip. This often is in the form of standard static RAM or flash RAM.

a) Code memory

Code memory is the memory that holds the actual 8052 programs that is to be run. This memory is limited to 64K. Code memory may be found on-chip or off-chip. It is possible to have 8K of code memory on-chip and 60K off chip memory simultaneously. If only off-chip memory is available then there can be 64K of off chip ROM. This is controlled by pin provided as EA

b) Internal RAM

The 8052 have a bank of 256 bytes of internal RAM. The internal RAM is found on-chip. So it is the fastest Ram available. And also it is most flexible in terms of reading and writing. Internal Ram is volatile, so when 8051 is reset, this memory is cleared. 256 bytes of internal memory are subdivided. The first 32 bytes are divided into 4 register banks. Each bank contains 8 registers. Internal RAM also contains 256 bits, which are addressed from 20h to 2Fh. These bits are bit addressed i.e. each individual bit of a byte can be addressed by the user. They are numbered 00h to FFh. The user may make use of these variables with commands such as SETB and CLR.

Special Function registered memory:

Special function registers are the areas of memory that control specific functionality of the 8052 micro controller.

a) Accumulator (0E0h)

As its name suggests, it is used to accumulate the results of large no of instructions. It can hold 8 bit values.

b) B registers (0F0h)

The B register is very similar to accumulator. It may hold 8-bit value. The b register is only used by MUL AB and DIV AB instructions. In MUL AB the higher byte of the product gets stored in B register. In div AB the quotient gets stored in B with the remainder in A.

c) Stack pointer (81h)

The stack pointer holds 8-bit value. This is used to indicate where the

next value to be removed from the stack should be taken from. When a value is to be pushed onto the stack, the 8052 first store the value of SP and then store the value at the resulting memory location. When a value is to be popped from the stack, the 8052 returns the value from the memory location indicated by SP and then decrements the value of SP.

d) Data pointer

The SFRs DPL and DPH work together work together to represent a 16-bit value called the data pointer. The data pointer is used in operations regarding external RAM and some instructions code memory. It is a 16-bit SFR and also an addressable SFR.

e) Program counter

The program counter is a 16 bit register, which contains the 2 byte address, which tells the 8052 where the next instruction to execute to be found in memory. When the 8052 is initialized PC starts at 0000h. And is incremented each time an instruction is executes. It is not addressable SFR.

f) PCON (power control, 87h)

The power control SFR is used to control the 8051's power control modes. Certain operation modes of the 8051 allow the 8051 to go into a type of "sleep mode" which consumes much lee power.

SMOD	--	--	--	GF1	GF0	PD	IDL

g) TCON (timer control, 88h)

The timer control SFR is used to configure and modify the way in which the 8051's two timers operate. This SFR controls whether each of the two timers is running or stopped and contains a flag to indicate that each timer has overflowed. Additionally, some non-timer related bits are located in TCON SFR. These bits are used to configure the way in which the external interrupt flags are activated, which are set when an external interrupt occurs.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

h) TMOD (Timer Mode, 89h)

The timer mode SFR is used to configure the mode of operation of each of the two timers. Using this SFR your program may configure each timer to be a 16-bit timer, or 13 bit timer, 8-bit auto reload timer, or two separate timers. Additionally you may configure the timers to only count when an external pin is activated or to count "events" that are indicated on an external pin.

Gate	$C/\overline{T}$	M1	M0	Gate	$C/\overline{T}$	M1	M0
TIMER 1				TIMER 0			

i) TO (Timer 0 low/high, address 8A/8C h)

These two SFRs taken together represent timer 0. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up. What is configurable is how and when they increment in value.

j) T1 (Timer 1 Low/High, address 8B/ 8D h)

These two SFRs, taken together, represent timer 1. Their exact behavior depends on how the timer is configured in the TMOD SFR; however, these timers always count up..

k) P0 (Port 0, address 90h, bit addressable)

This is port 0 latch. Each bit of this SFR corresponds to one of the pins on a micro controller. Any data to be outputted to port 0 is first written on P0 register. For e.g., bit 0 of port 0 is pin P0.0, bit 7 is pin p0.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to low level.

l) P1 (port 1, address 90h, bit addressable)

This is port latch1. Each bit of this SFR corresponds to one of the pins on a micro controller. Any data to be outputted to port 0 is first written on P0 register. For e.g., bit 0 of port 0 is pin P1.0, bit 7 is pin P1.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to low level

m) P2 (port 2, address 0A0h, bit addressable):

This is a port latch2. Each bit of this SFR corresponds to one of the pins on a micro controller. Any data to be outputted to port 0 is first written on P0 register. For e.g., bit 0 of port 0 is pin P2.0, bit 7 is pin P2.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to low level.

n) P3 (port 3, address B0h, bit addressable)   :

This is a port latch3. Each bit of this SFR corresponds to one of the pins on a micro controller. Any data to be outputted to port 0 is first written on P0 register. For e.g., bit 0 of port 0 is pin P3.0, bit 7 is pin P3.7. Writing a value of 1 to a bit of this SFR will send a high level on the corresponding I/O pin whereas a value of 0 will bring it to low level.

o) IE (interrupt enable, 0A8h):

The Interrupt Enable SFR is used to enable and disable specific interrupts. The low 7 bits of the SFR are used to enable/disable the specific interrupts, where the MSB bit is used to enable or disable all the interrupts. Thus, if the high bit of IE is 0 all interrupts are disabled regardless of whether an individual interrupt is enabled by setting a lower bit.

EA	--	ET2	ES	ET1	EX1	ET0	EX0

p) IP (Interrupt Priority, 0B8h)

The interrupt priority SFR is used to specify the relative priority of each interrupt. On 8051, an interrupt maybe either low or high priority. An interrupt may interrupt interrupts. For e.g., if we configure all interrupts as low priority other than serial interrupt. The serial interrupt always interrupts the system, even if another interrupt is currently executing. However, if a serial interrupt is executing no other interrupt will be able to interrupt the serial interrupt routine since the serial interrupt routine has the highest priority.

--	--	PT2	PS	PT1	PX1	PT0	PX0

q) PSW (Program Status Word, 0D0h)

The program Status Word is used to store a number of important bits that are set and cleared by 8052 instructions. The PSW SFR contains the carry flag, the auxiliary carry flag, the parity flag and the overflow flag. Additionally, it also contains the register bank select flags, which are used to select, which of the “R” register banks currently in use.

CY	AC	F0	RS1	RS0	OV	--	P

r) SBUF (Serial Buffer, 99h)

SBUF is used to hold data in serial communication. It is physically two registers. One is writing only and is used to hold data to be transmitted out of 8052 via TXD. The other is read only and holds received data from external sources via RXD. Both mutually exclusive registers use address 99h.

I/O ports:

One major feature of a microcontroller is the versatility built into the input/output (I/O) circuits that connect the 8052 to the outside world. The main constraint that limits numerous functions is the number of pins available in the 8051 circuit. The DIP had 40 pins and the success of the design depends on the flexibility incorporated into use of these pins. For this reason, 24 of the pins may each used for one of the two entirely different functions which depend, first, on what is physically connected to it and, then, on what software programs are used to “program” the pins.

PORT 0

Port 0 pins may serve as inputs, outputs, or, when used together, as a bi directional low-order address and data bus for external memory. To configure a pin as input, 1 must be written into the corresponding port 0 latch by the program. When used for interfacing with the external memory, the lower byte of address is first sent via PORT0, latched using Address latch enable (ALE) pulse and then the bus is turned around to become the data bus for external memory.

PORT 1

Port 1 is exclusively used for input/output operations. PORTS 1 pin have no dual function. When a pin is to be configured as input, 1 is to be written into the corresponding Port 1 latch.

PORT 2

Port 2 maybe used as an input/output port. It may also be used to supply a high –order address byte in conjunction with Port 0 low-order byte to address external memory. Port 2 pins are momentarily changed by the address control signals when supplying the high byte a 16-bit address. Port 2 latches remain stable when external memory is addressed, as they do not have to be turned around (set to 1) for data input as in the case for Port 0.

PORT 3

Port 3 may be used to input /output port. The input and output functions can be programmed under the control of the P3 latches or under the control of various special function registers. Unlike Port 0 and Port 2, which can have external addressing functions and change all eight-port b se, each pin of port 3 maybe individually programmed to be used as I/O or as one of the alternate functions. The Port 3 alternate uses are:

Pin (SFR)	Alternate Use
P3.0-RXD (SBUF)	Serial data input
P3.1-TXD (SBUF)	Serial data output
P3.2-INTO 0 (TCON.1)	External interrupt 0
P3.3 - INTO 1 (TCON.3)	External interrupt 1
P3.4 - T0 (TMOD)	External Timer 0 input
P3.5 – T1 (TMOD)	External timer 1 input
P3.6 - $\overline{\text{WR}}$	External memory write pulse

P3.7 - $\overline{\text{RD}}$	External memory read pulse

INTERRUPTS:

The AT89S52 has a total of six interrupt vectors: two external interrupts (INT0 and INT1), three timer interrupts (Timers0, 1, and 2), and the serial port interrupt. These interrupts are all shown in Figure 10. Each of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. IE also contains a global disable bit, EA, which disables all interrupts at once. Note that Table 5 shows that bit position IE.6 is unimplemented. In the AT89S52, bit position IE.5 is also unimplemented. User software should not write 1s to these bit positions, since they may be used in future AT89 products.
Timer 2 interrupt is generated by the logical OR of bits TF2 and EXF2 in register T2CON. Neither of these flags is cleared by hardware when the service routine is vectored
to. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and that bit will have to be cleared in software.The Timer 0 and Timer 1 flags, TF0 and TF1, are set at S5P2 of the cycle in which the timers overflow. The values
are then polled by the circuitry in the next cycle. However, the Timer 2 flag, TF2, is set at S2P2 and is polled in the same cycle in which the timer overflows.

Table 5. Interrupt Enable (IE) Register

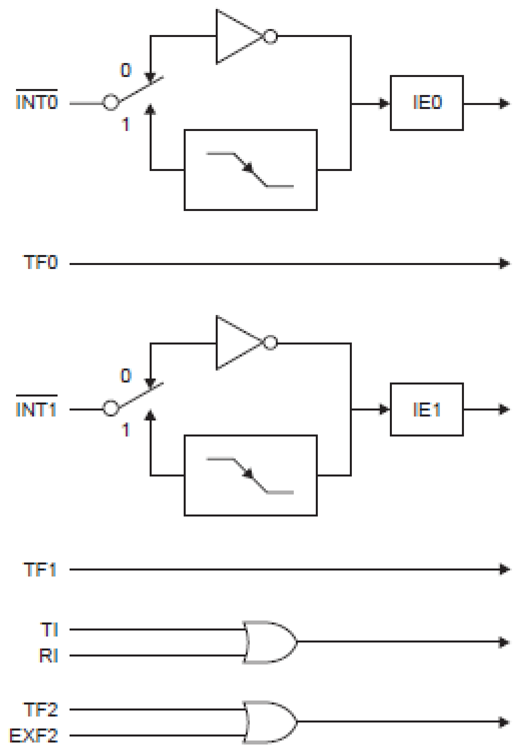
(MSB)							(LSB)
EA	–	ET2	ES	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt.

Enable Bit = 0 disables the interrupt.

Symbol	Position	Function
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
–	IE.6	Reserved.
ET2	IE.5	Timer 2 interrupt enable bit.
ES	IE.4	Serial Port interrupt enable bit.
ET1	IE.3	Timer 1 interrupt enable bit.
EX1	IE.2	External interrupt 1 enable bit.
ET0	IE.1	Timer 0 interrupt enable bit.
EX0	IE.0	External interrupt 0 enable bit.
User software should never write 1s to unimplemented bits, because they may be used in future AT89 products.		

Figure 10. Interrupt Sources



2.3Power Supply

2.3.1 INTRODUCTION

There are many types of power supply. Most are designed to convert high voltage AC mains electricity to a suitable low voltage supply for electronics circuits and other devices. A power supply can by broken down into a series of blocks, each of which performs a particular function. For example a 5V regulated supply can be shown as below

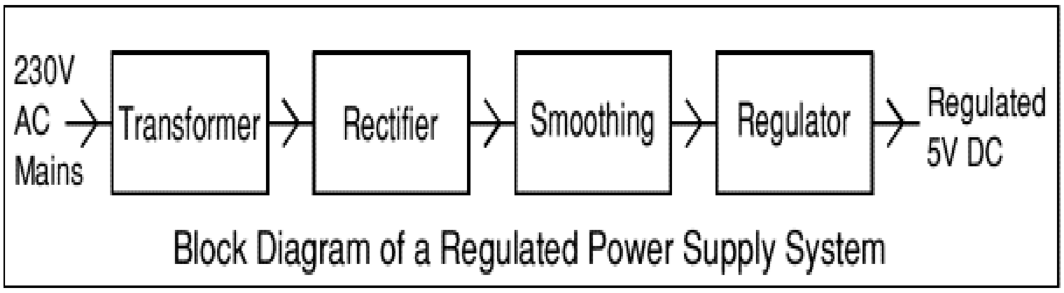


Fig 3.1: Block Diagram of a Regulated Power Supply System

Similarly, 12v regulated supply can also be produced by suitable selection of the individual elements. Each of the blocks is described in detail below and the power supplies made from these blocks are described below with a circuit diagram and a graph of their output:

2.3.2 Transformer:

A transformer steps down high voltage AC mains to low voltage AC. Here we are using a center-tap transformer whose output will be sinusoidal with 36volts peak to peak value.

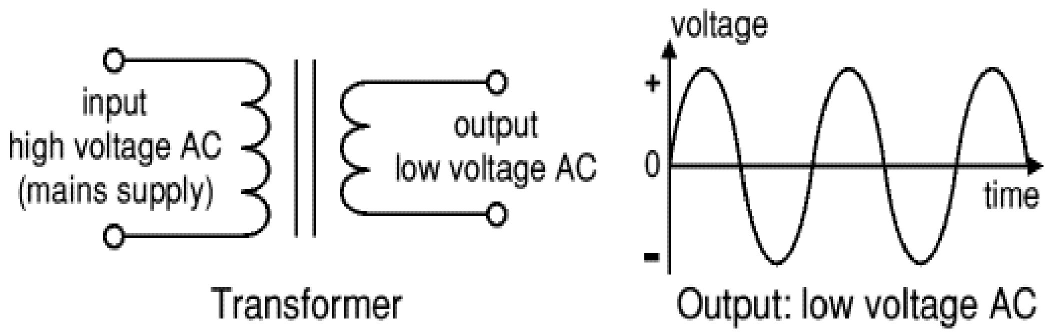


Fig: 2.3.1 Output Waveform of transformer

The low voltage AC output is suitable for lamps, heaters and special AC motors. It is not suitable for electronic circuits unless they include a rectifier and a smoothing capacitor. The transformer output is given to the rectifier circuit.

2.3.3 Rectifier:

A rectifier converts AC to DC, but the DC output is varying. There are several types of rectifiers; here we use a bridge rectifier.

The Bridge rectifier is a circuit, which converts an ac voltage to dc voltage using both half cycles of the input ac voltage. The Bridge rectifier circuit is shown in the figure. The circuit has four diodes connected to form a bridge. The ac input voltage is applied to the diagonally opposite ends of the bridge. The load resistance is connected between the other two ends of the bridge.

For the positive half cycle of the input ac voltage, diodes D1 and D3 conduct, whereas diodes D2 and D4 remain in the OFF state. The conducting diodes will be in series with the load resistance R_L and hence the load current flows through $R_{L.}$

For the negative half cycle of the input ac voltage, diodes D2 and D4 conduct whereas, D1 and D3 remain OFF. The conducting diodes D2 and D4 will be in series with the load resistance R_L and hence the current flows through R_L in the same direction as in the previous half cycle. Thus a bi-directional wave is converted into unidirectional.

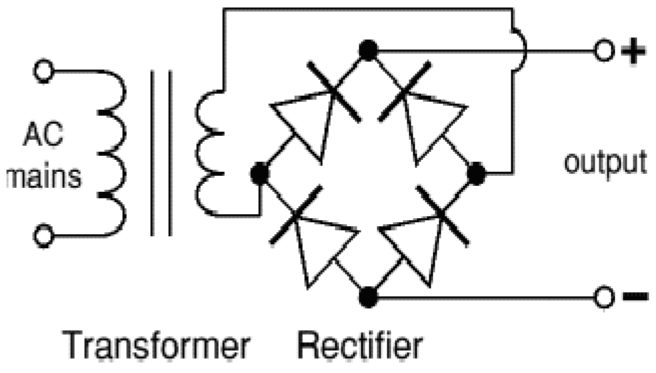


Figure 3.3 Rectifier circuit

Now the output of the rectifier shown in Figure 3.3 is shown below in Figure 3.4

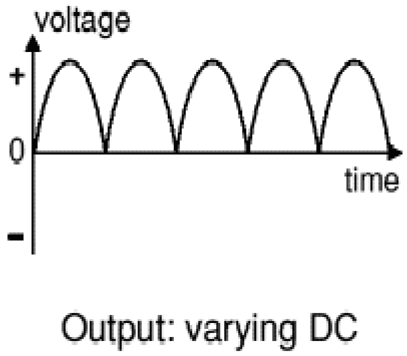


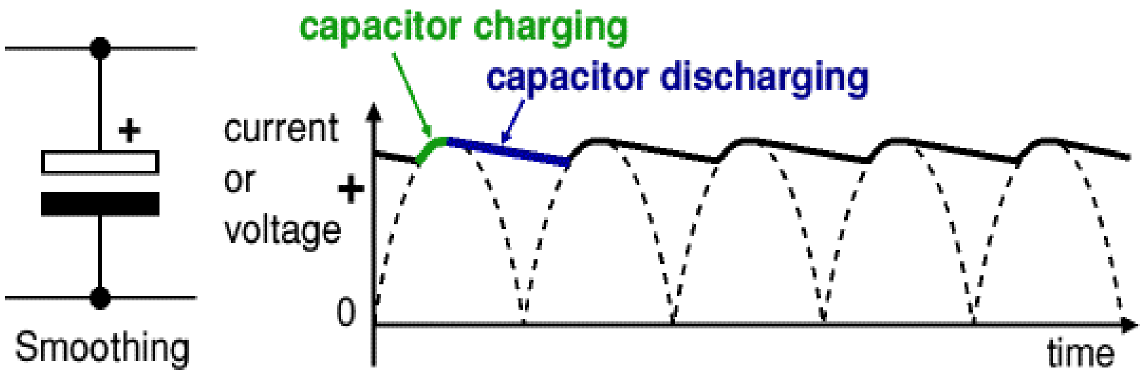
Figure 2.3.4 Output of the Rectifier

The varying DC output is suitable for lamps, heaters and standard motors. It is not suitable for lamps, heaters and standard motors. It is not suitable for electronic circuits unless they include a smoothing capacitor.

Smoothing:

The smoothing block smoothes the DC from varying greatly to a small ripple and the *ripple voltage* is defined as the deviation of the load voltage from its DC value. Smoothing is also named as filtering.

Filtering is frequently effected by shunting the load with a capacitor. The action of this system depends on the fact that the capacitor stores energy during the conduction period and delivers this energy to the loads during the no conducting period. In this way, the time during which the current passes through the load is prolonging Ted, and the ripple is considerably decreased. The action of the capacitor is shown with the help of waveform.



1) *Figure 2.3.5 Smoothing action of capacitor*

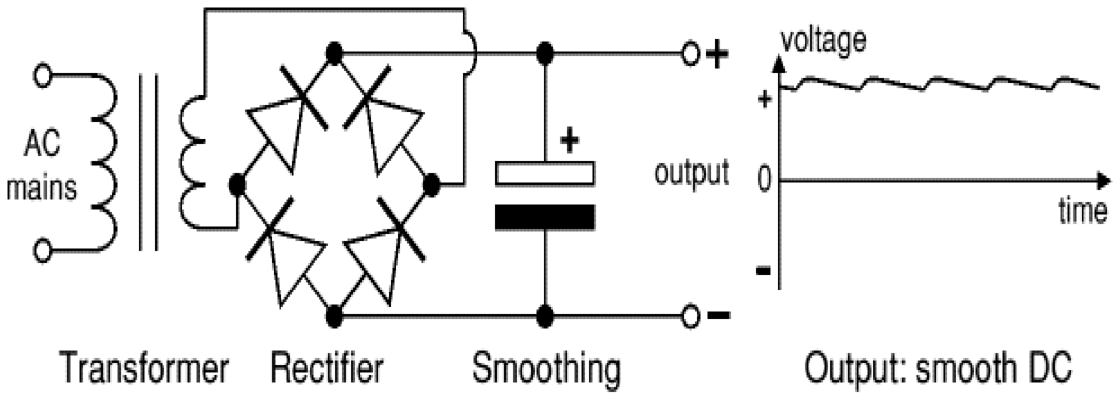
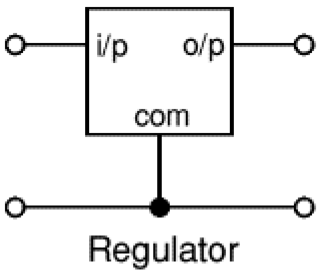


Figure2. 3.6 Waveform of the rectified output smoothing

2.3.4 Regulator:

Regulator eliminates ripple by setting DC output to a fixed voltage. Voltage regulator ICs are available with fixed (typically 5V, 12V and 15V) or variable output voltages. Negative voltage regulators are also available

Many of the fixed voltage regulator ICs has 3 leads (input, output and high impedance). They include a hole for attaching a heat sink if necessary. Zener diode is an example of fixed regulator which is shown here.



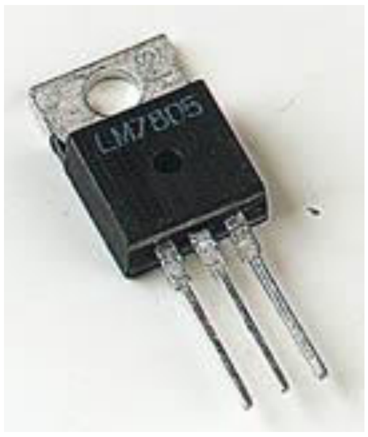
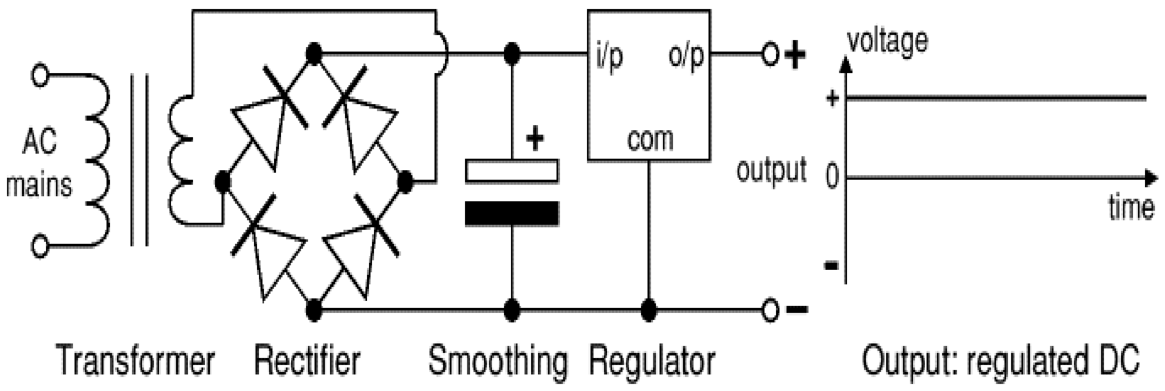


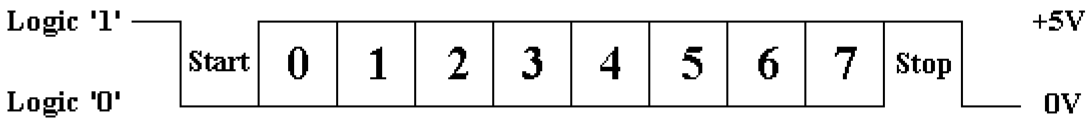
Figure 3.7 Regulator

Transformer + Rectifier + Smoothing + Regulator:



2.4 MAX 232

RS-232 WAVEFORM



TTL/CMOS Serial Logic Waveform

The diagram above shows the expected waveform from the UART when using the common 8N1 format. 8N1 signifies 8 Data bits, No Parity and 1 Stop Bit. The RS-232 line, when idle is in the Mark State (Logic 1). A transmission starts with a start bit which is (Logic 0). Then each bit is sent down the line, one at a time. The LSB (Least Significant Bit) is sent first. A Stop Bit (Logic 1) is then appended to the signal to make up the transmission.

The data sent using this method, is said to be framed. That is the data is framed between a Start and Stop Bit.

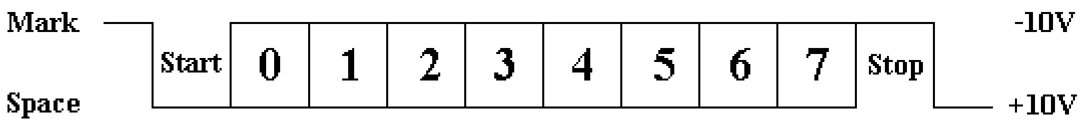
RS-232 Voltage levels

- +3 to +25 volts to signify a "Space" (Logic 0)
- -3 to -25 volts for a "Mark" (logic 1).
- Any voltage in between these regions (i.e. between +3 and -3 Volts) is undefined.

The data byte is always transmitted least-significant-bit first.

The bits are transmitted at specific time intervals determined by the baud rate of the serial signal.

This is the signal present on the RS-232 Port of your computer, shown below.

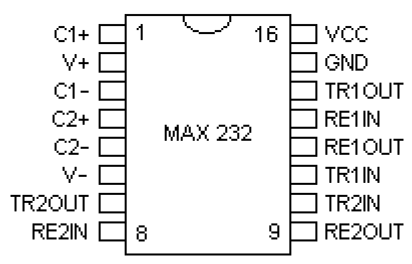


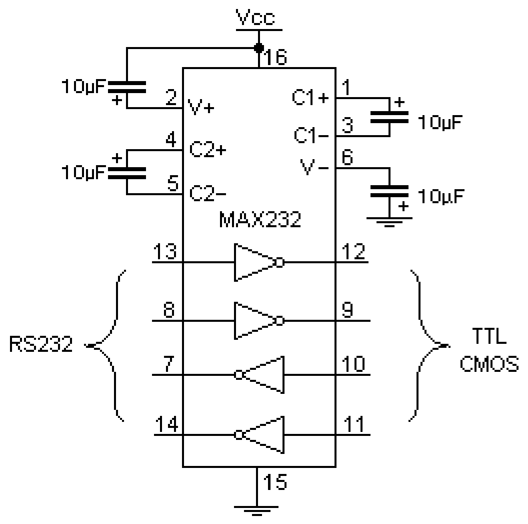
RS-232 Logic Waveform

RS-232 LEVEL CONVERTER

Standard serial interfacing of microcontroller (TTL) with PC or any RS232C Standard device , requires TTL to RS232 Level converter . A MAX232 is used for this purpose. It provides 2-channel RS232C port and requires external 10uF capacitors.

The driver requires a single supply of +5V.

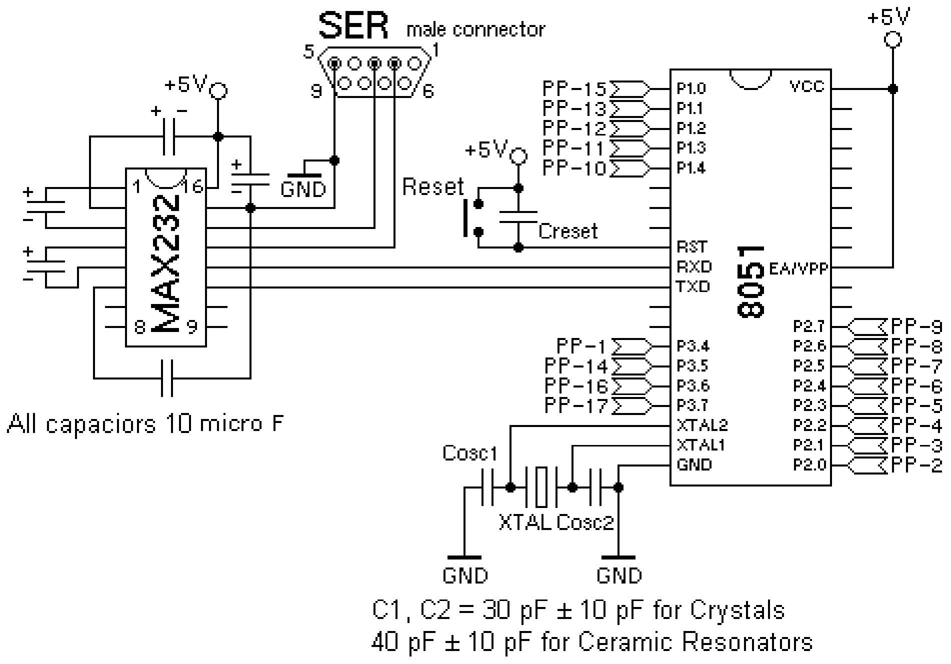




MAX-232 includes a Charge Pump, which generates +10V and -10V from a single 5v supply.

MICROCONTROLLER INTERFACING WITH RS-232 STANDARD DEVICES

- MAX232 (+5V -> +-12V converter)
- Serial port male 9 pin connector (SER)



SETTING SERIAL PORT

SCON

8 bit UART, RN enabled, TI & RI operated by program. - 50hex

Timer 1 Count

TH1 = 256 - ((Crystal / 384) / Baud) -PCON.7 is clear.

TH1 = 256 - ((Crystal / 192) / Baud)-PCON.7 is set.

So with PCON.7 is clear we get timer value = FDhex

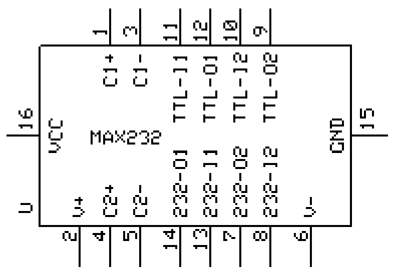
Serial communication between PC and microcontroller

When a processor communicates with the outside world, it provides data in byte sized chunks. Computers transfer data in two ways: parallel and serial. In parallel data transfers, often more lines are used to transfer data to a device and 8 bit data path is expensive. The serial communication transfer uses only a single data line instead of the 8 bit data line of parallel communication which makes the data transfer not only cheaper but also makes it possible for two computers located in two different cities to communicate over telephone.

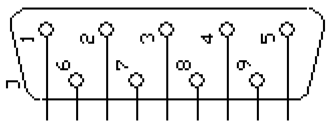
Serial data communication uses two methods, asynchronous and synchronous. The synchronous method transfers data at a time while the asynchronous transfers a single byte at a time. There are some special IC chips made by many manufacturers for data communications. These chips are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous asynchronous receiver transmitter). The AT89C51 chip has a built in UART.

In asynchronous method, each character is placed between start and stop bits. This is called framing. In data framing of asynchronous communications, the data, such as ASCII characters, are packed in between a start and stop bit. We have a total of 10 bits for a character: 8 bits for the ASCII code and 1 bit each for the start and stop bits. The rate of serial data transfer communication is stated in bps or it can be called as baud rate.

To allow the compatibility among data communication equipment made by various manufacturers, and interfacing standard called RS232 was set by the Electronics industries Association in 1960. Today RS232 is the most widely used I/O interfacing standard. This standard is used in PCs and numerous types of equipment. However, since the standard was set long before the advent of the TTL logic family, its input and output voltage levels are not TTL compatible. In RS232, a 1 bit is represented by -3 to -25V, while a 0 bit is represented +3 to +25 V, making -3 to +3 undefined. For this reason, to connect any RS232 to a microcontroller system we must use voltage converters such as MAX232 to connect the TTL logic levels to RS232 voltage levels and vice versa. MAX232 ICs are commonly referred to as line drivers.



The RS232 cables are generally referred to as DB-9 connector. In labeling, DB-9P refers to the plug connector (male) and DB-9S is for the socket connector (female). The simplest connection between a PC and microcontroller requires a minimum of three pin, TXD, RXD, and ground. Many of the pins of the RS232 connector are used for handshaking signals. They are bypassed since they are not supported by the 8051 UART chip.



IBM PC/ compatible computers based on x86(8086, 80286, 386, 486 and Pentium) microprocessors normally have two COM ports. Both COM ports have RS232 type connectors. Many PCs use one each of the DB-25 and DB-9 RS232 connectors. The COM ports are designated as COM1 and COM2. We can connect the serial port to the COM 2 port of a PC for serial communication experiments. We use a DB9 connector in our arrangement.

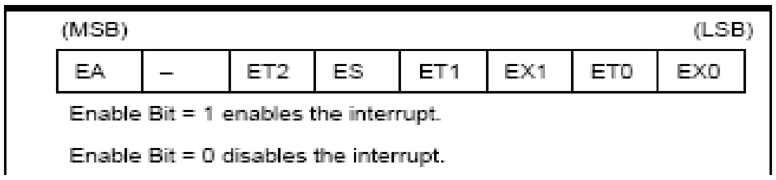
The AT89C51 has two pins that are used specifically for transferring and receiving data serially. These two pins are called TXD and RXD and are part of the port3 (P3.0 and P3.1). These pins are TTL compatible; therefore they require a line driver to make them RS232 compatible. One such line driver is the MAX232 chip. One advantage of MAX232 chip is that it uses a +5v power source which is the same as the source voltage for the at89c51. The MAX232 has two sets of line drivers for receiving and transferring data. The line drivers for TXD are called T1 and T2 while the line drivers for RXD are designated as R1 and R2. T1 and R1 are used for TXD and RXD of the 89c51 and the second set is left unused. In MAX232 that the TI line driver has a designation of T1 in and T1 out on pin numbers 11 and 14, respectively. The T1 in pin is the TTL side and is connected to TXD of the microcontroller, while TI out is the RS232 side that is connected to the RXD pin of the DB9 connector.

To allow data transfer between PC and the microcontroller system without any error, we must make sure that the baud rate of the 8051 system matches the baud rate of the PC's COM port.

Interrupts:-

The AT89C52 has a total of six interrupt vectors: two external interrupts (INT0 and INT1), three timer interrupts (Timers 0, 1, and 2), and the serial port interrupt. These interrupts are all shown in Figure 5.5. Of these interrupt sources can be individually enabled or disabled by setting or clearing a bit in Special Function Register IE. IE also contains a global disable bit, EA, which disables all interrupts at once.

Note that Table 5.3 shows that bit position IE.6 is unimplemented. In the AT89C51, bit position IE.5 is also unimplemented. User software should not write 1s to these bit positions, since they may be used in future AT89 products.

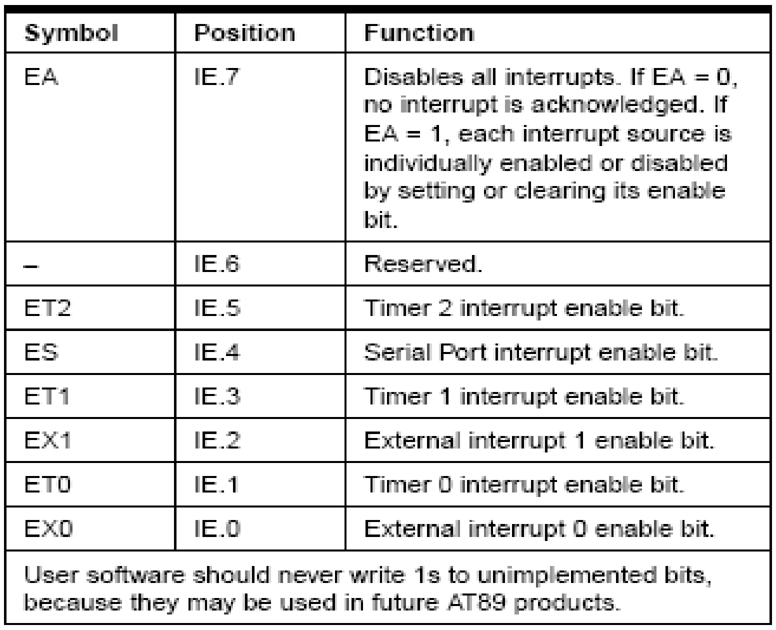


(MSB) (LSB)

EA	–	ET2	ES	ET1	EX1	ET0	EX0

Enable Bit = 1 enables the interrupt.

Enable Bit = 0 disables the interrupt.



Symbol	Position	Function
EA	IE.7	Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.
–	IE.6	Reserved.
ET2	IE.5	Timer 2 interrupt enable bit.
ES	IE.4	Serial Port interrupt enable bit.
ET1	IE.3	Timer 1 interrupt enable bit.
EX1	IE.2	External interrupt 1 enable bit.
ET0	IE.1	Timer 0 interrupt enable bit.
EX0	IE.0	External interrupt 0 enable bit.
User software should never write 1s to unimplemented bits, because they may be used in future AT89 products.		

Table: Interrupts Enable Register

Timer 2 interrupt is generated by the logical OR of bits TF2 and EXF2 in register T2CON. Neither of these flags is cleared by hardware when the service routine is vectored To. In fact, the service routine may have to determine whether it was TF2 or EXF2 that generated the interrupt, and that bit will have to be cleared in software.

The Timer 0 and Timer 1 flags, TF0 and TF1, are set at S5P2 of the cycle in which the timers overflow. The values are then polled by the circuitry in the next cycle. However, the Timer 2 flag, TF2, is set at S2P2 and is polled in the same cycle in which the timer overflows.

HT9170B-- DTMF RECIEVER

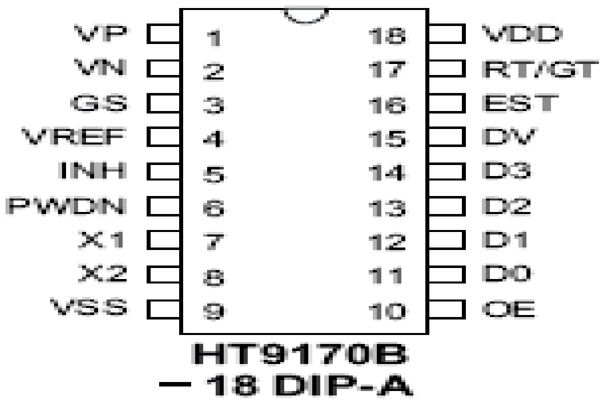
The HT9170B/D are Dual Tone Multi Frequency (DTMF) receivers integrated with digital decoder and band split filter functions as well as power-down mode and inhibit mode operations. Such devices use digital counting techniques to detect and decode all the 16 DTMF tone pairs into a 4-bit code output.

Highly accurate switched capacitor filters are implemented to divide tone signals into low and high group signals. A built-in dial tone rejection circuit is provided to eliminate the need for pre-filtering.

FEAUTURES:

- Operating voltage: 2.5V~5.5V
- Minimal external components
- No external filter is required
- Low standby current (on power down mode)
- Excellent performance
- Tristate data output for MCU interface
- 3.58MHz crystal or ceramic resonator
- 1633Hz can be inhibited by the INH pin
- HT9170B: 18-pin DIP package

PIN ASSIGMENT:



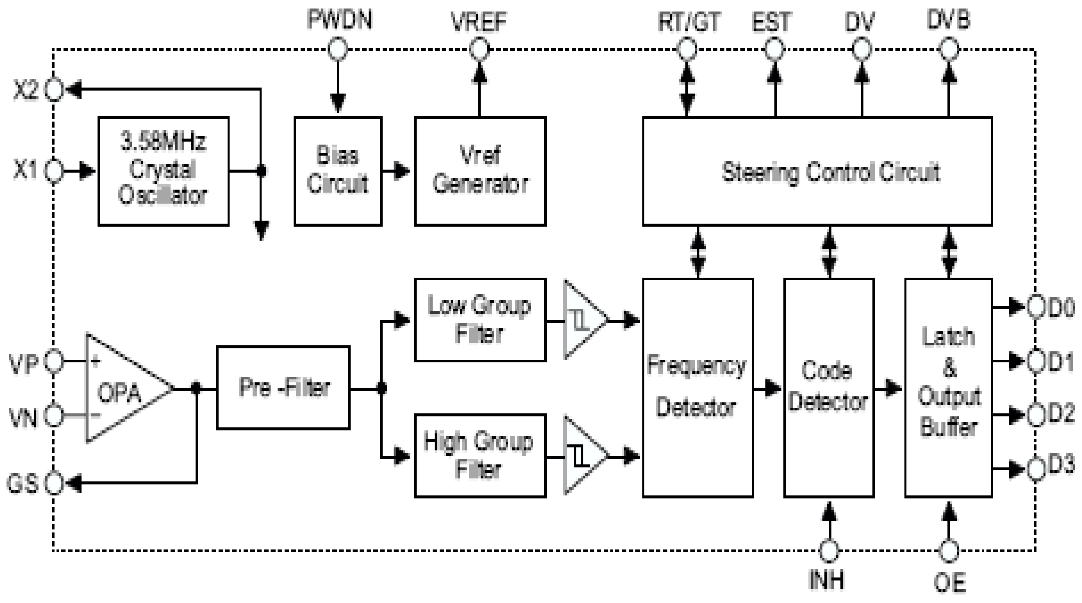
PIN DESCRIPTION:

Pin Name	I/O	Internal Connection	Description
VP	I	Operational Amplifier	Operational amplifier non-inverting input
VN	I		Operational amplifier inverting input
GS	O		Operational amplifier output terminal
VREEF	O	VREF	Reference voltage output, normally $V_{DD}/2$
X1	I	oscillator	The system oscillator consists of an inverter, a bias resistor and the necessary load capacitor on chip. A standard 3.579545MHz crystal connected to X1 and X2 terminals implements the oscillator function.
X2	O		
PWDN	I	CMOS IN Pull-low	Active high. This enables the device to go into power down mode and inhibits the oscillator. This pin input is internally pulled down.
INH	I	CMOS IN Pull-low	Logic high. This inhibits the detection of tones representing characters A, B, C and D. This pin input is internally pulled down.
VSS	—	—	Negative power supply, ground
OE	I	CMOS IN Pull-high	D0~D3 output enable, high active

D0~D3	O	CMOS OUT Tristate	Receiving data output terminals
			OE="H": Output enable
			OE="L": High impedance

DV	O	CMOS OUT	Data valid output When the chip receives a valid tone (DTMF) signal, the DV goes high; otherwise it remains low.
EST	O	CMOS OUT	Early steering output (see Functional Description)
RT/GT	I/O	CMOS IN/OUT	Tone acquisition time and release time can be set through connection with external resistor and capacitor.
VDD	—	—	Positive power supply, 2.5V~5.5V for normal operation

BLOCK DIAGRAM:



FUNCTIONAL DESCRIPTION:

The HT9170B/D tone decoders consist of three band pass filters and two digital decode circuits to convert a tone (DTMF) signal into digital code output.

An operational amplifier is built-in to adjust the input signal. The pre-filter is a band rejection filter, which reduces the dialing tone from 350Hz to 400Hz.

The low group filter filters low group frequency signal output whereas the high group filter filters high group Frequency signal output. A zero-crossing detector with follows each filters output hysteretic. When each signal amplitude at the output exceeds the specified level, it is transferred to full swing logic signal.

When input signals are recognized to be effective, DV becomes high, and the correct tone code (DTMF) digit is transferred.

(ii) Steering control circuit:

The steering control circuit is used for measuring the effective signal duration and for protecting against drop out of valid signals. It employs the analog delay by external RC time-constant controlled by EST.

The EST pin is normally low and draws the RT/GT pin to keep low through discharge of external RC. When a valid tone input is detected, EST goes high to charge RT/GT through RC.

When the voltage of RT/GT changes from 0 to VTRT (2.35V for 5V supply), the input signal is effective, and the code detector will create the correct code. After D0~D3 are completely latched, DV output becomes high.

When the voltage of RT/GT falls down from VDD to VTRT (i.e. when there is no input tone), DV output becomes

Low, and D0~D3 keeps data until a next valid tone input is produced. By selecting adequate external RC value, the minimum acceptable input tone duration (tACC) and the minimum acceptable inter-tone rejection (tIR) can be set. External

Components (R, C) are chosen by the formula.

tACC=tDP+tGTP;

tIR=tDA+tGTA;

Where tACC: Tone duration acceptable time

TDP: EST output delay time (_L__H_)

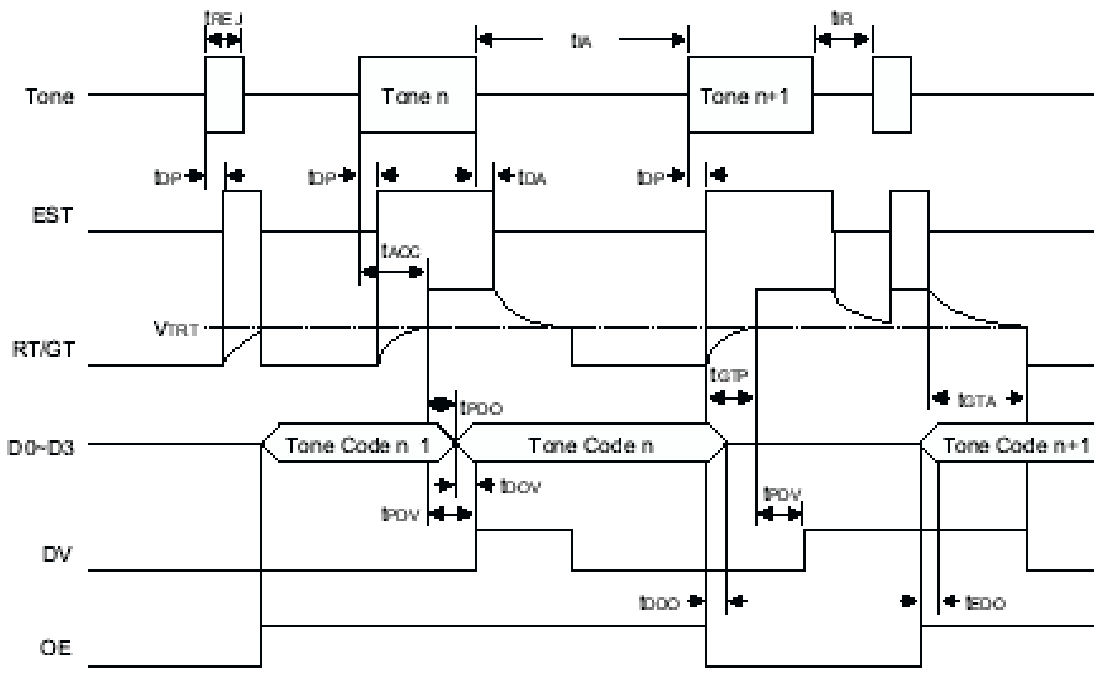
TGTP: Tone present time

TIR: Inter-digit pause rejection time

TDA: EST output delay time (_H__L_)

tGTA: Tone absent time

TIMING DIAGRAM



Applications

- PABX
- Central office
- Mobile radio
- Remote control
- Remote data entry
    - Call limiting
- Telephone answering systems

L293D

L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low-current control signal and provide a higher-current signal. This higher current signal is used to drive the motors.

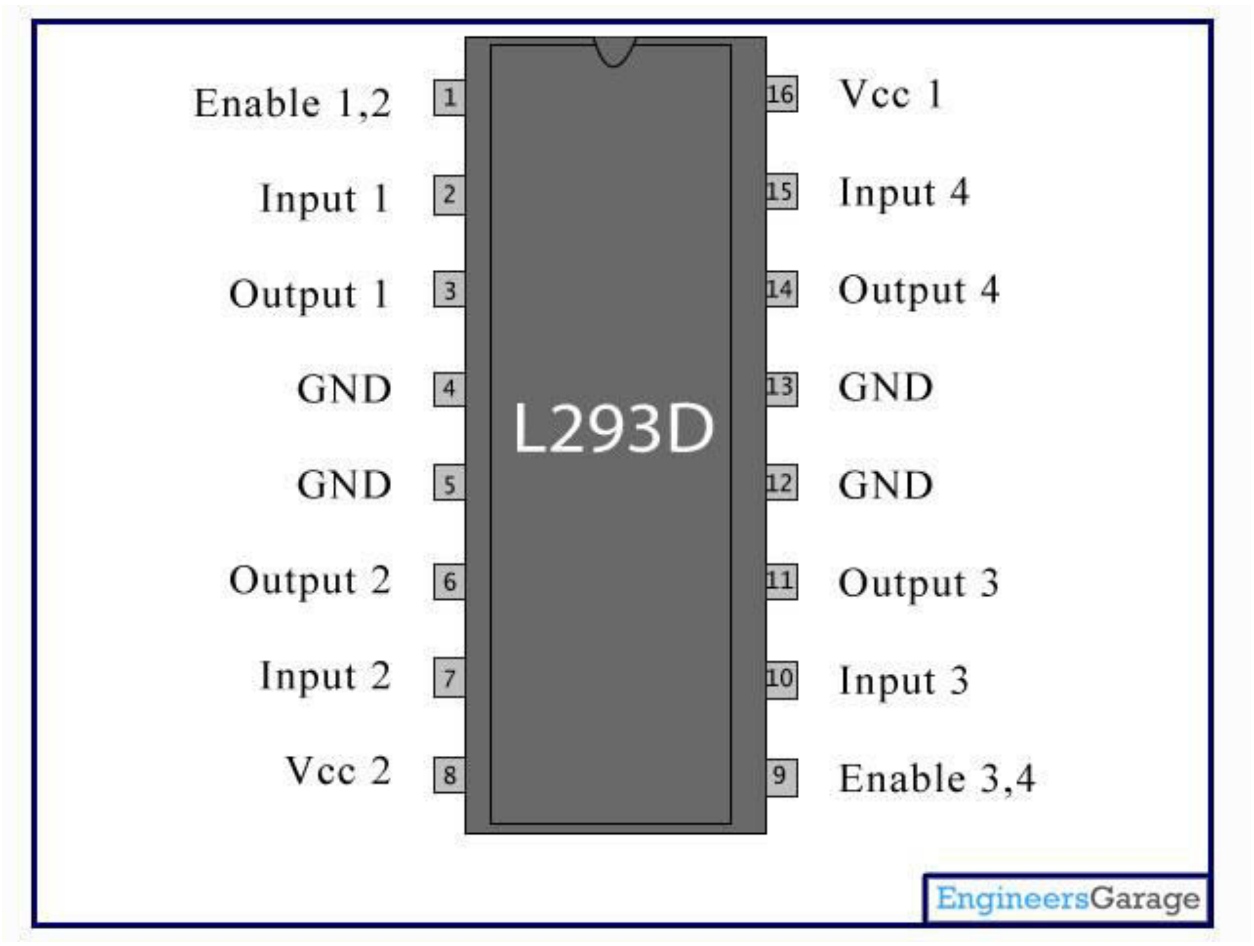
L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

Enable pins 1 and 9 (corresponding to the two motors) must be high for motors to start operating. When an enable input is high, the associated driver gets enabled. As a result, the outputs become active and work in phase with their inputs. Similarly, when the enable input is low, that driver is disabled, and their outputs are off and in the high-impedance state.

Description/ordering information (continued)

1.On the L293, external high-speed output clamp diodes should be used for inductive transient suppression.
2.A VCC1 terminal, separate from VCC2, is provided for the logic inputs to minimize device power dissipation.
3.The L293and L293D are characterized for operation from 0°C to 70°C.

Pin Diagram:



The L293 and L293D are quadruple high-currenthalf-H drivers. The L293 is designed to providebidirectional drive currents of up to 1 A at voltages from 4.5 V to 36 V. The L293D is designed toprovide bidirectional drive currents of up to600-mA at voltages from 4.5 V to 36 V. Bothdevices are designed to drive inductive loads suchas relays, solenoids, dc and bipolar stepping motors, as well as other high-current/high-voltage loads in positive-supply applications.

All inputs are TTL compatible. Each output is acomplete totem-pole drive circuit, with a Darlington transistor sink and a pseudo-Darlington source. Drivers are enabled in pairs, with drivers 1 and 2 enabled by 1,2EN and drivers 3 and 4 enabled by 3,4EN. When an enable input is high, the associated drivers are enabled, and their outputs are activeand in phase with their inputs. When the enable input is low, those drivers are disabled, and their outputs areoff and in the high-impedance state. With the proper data inputs, each pair of drivers forms a full-H (or bridge) reversible drive suitable for solenoid or motor applications.

Features

- Featuring Unitrode L293 and L293D Products Now From Texas Instruments
- Wide Supply-Voltage Range: 4.5 V to 36 V
- Separate Input-Logic Supply
- Internal ESD Protection
- Thermal Shutdown
- High-Noise-Immunity Inputs
- Functional Replacements for SGS L293 and SGS L293D
- Output Current 1 A Per Channel (600 mA for L293D)
- Peak Output Current 2 A Per Channel (1.2 A for L293D)
- Output Clamp Diodes for Inductive Transient Suppression (L293D)

Controlling Motors

While turning a motor on and off requires only one switch (or transistor) controlling the direction is
deceptively difficult. It requires no fewer than four switches (or transistors) arranged in a clever way.

H-Bridges

These four switches (or transistors) are arranged in a shape that resembles an 'H' and thus called an
H-Bridge. Each side of the motor has two transistors, one is responsible for pushing that side HIGH
the other for pulling it LOW. When one side is pulled HIGH and the other LOW the motor will spin in
one direction. When this is reversed (the first side LOW and the latter HIGH) it will spin the opposite
way.

DC Motor Example

Confused? that's alright it all starts making sense with an example. Cut out the breadboard layout
sheet below and download the example code from http://tinyurl.com/**qcpah9** and play around.

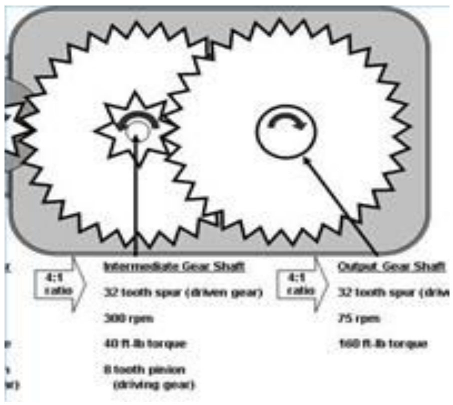
Stepper Motor Example (for use with 4, 5,6 & 8 wire motors)

The Arduino IDE has an included library for controlling stepper motors. To test it out with this setup,
plug the stepper motor in with coil A across OUT 1 & 2, and coil B across OUT 3 & 4. Then download
example code from http://tinyurl.com/**nyylun** and play around.

GEAR MOTOR

What Is a Gear Motor?

Gear motors are complete motive force systems consisting of an electric motor and a reduction gear train integrated into one easy-to-mount and -configure package. This greatly reduces the complexity and cost of designing and constructing power tools, machines and appliances calling for high torque at relatively low shaft speed or RPM. Gear motors allow the use of economical low-horsepower motors to provide great motive force at low speed such as in lifts, winches, medical tables, jacks and robotics. They can be large enough to lift a building or small enough to drive a tiny clock.



.

12V High Torque DC GEAR MOTOR

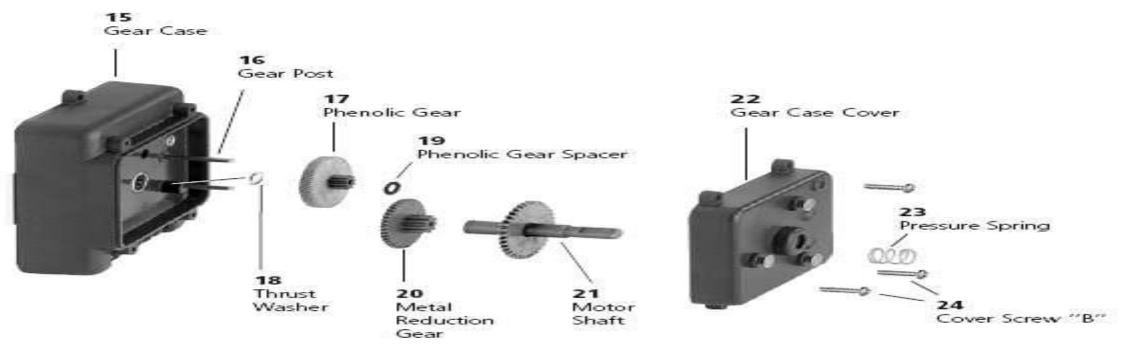
Operation Principle

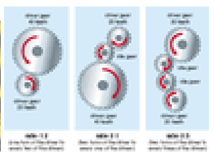
Most synchronous AC electric motors have output ranges of from 1,200 to 3,600 revolutions per minute. They also have both normal speed and stall-speed torque specifications. The reduction gear trains used in gear
motors are designed to reduce the output speed while increasing the torque. The increase in torque is inversely proportional to the reduction in speed. Reduction gearing allows small electric motors to move large driven loads, although more slowly than larger electric motors. Reduction gears consist of a small gear driving a larger gear. There may be several sets of these reduction gear sets in a reduction gear box.

Article II. Gear

Toothed wheel that transmits the turning movement of one shaft to another shaft. Gear wheels may be used in pairs or in threes if both shafts are to turn in the same direction. The gear ratio – the ratio of the number of teeth on the two wheels – determines the torque ratio, the turning force on the output shaft compared with the turning force on the input shaft. The ratio of the angular velocities of the shafts is the inverse of the gear ratio.

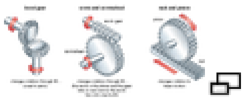
The common type of gear for parallel shafts is the **spur gear**, with straight teeth parallel to the shaft axis. The **helical gear** has teeth cut along sections of a helix or corkscrew shape; the double form of the helix gear is the most efficient for energy transfer. **Bevel gears**, with tapering teeth set on the base of a cone, are used to connect intersecting shafts.





The toothed and interlocking wheels which make up a typical gear movement.

Gear ratio is calculated by dividing the number of teeth on the driver gear by the number of teeth on the driven gear (gear ratio = driver/driven); the idler gears are ignored. Idler gears change the direction of rotation but do not affect speed. A high driven to driver ratio (middle) is a speed-reducing ratio.



Different gears are used to perform different engineering functions depending on the change in direction of motion that is needed. Rack and pinion gears are the commonest gears and are used in car steering mechanics.

Speed Reduction

- Sometimes the goal of using a gear motor is to reduce the rotating shaft speed of a motor in the device being driven, such as in a small electric clock where the tiny synchronous motor may be spinning at 1,200 rpm but is reduced to one rpm to drive the second hand, and further reduced in the clock mechanism to drive the minute and hour hands. Here the amount of driving force is irrelevant as long as it is sufficient to overcome the frictional effects of the clock mechanism.

Torque Multiplication

- Another goal achievable with a gear motor is to use a small motor to generate a very large force albeit at a low speed. These applications include the lifting mechanisms on hospital beds, power recliners, and heavy machine lifts where the great force at low speed is the goal.

Motor Varieties

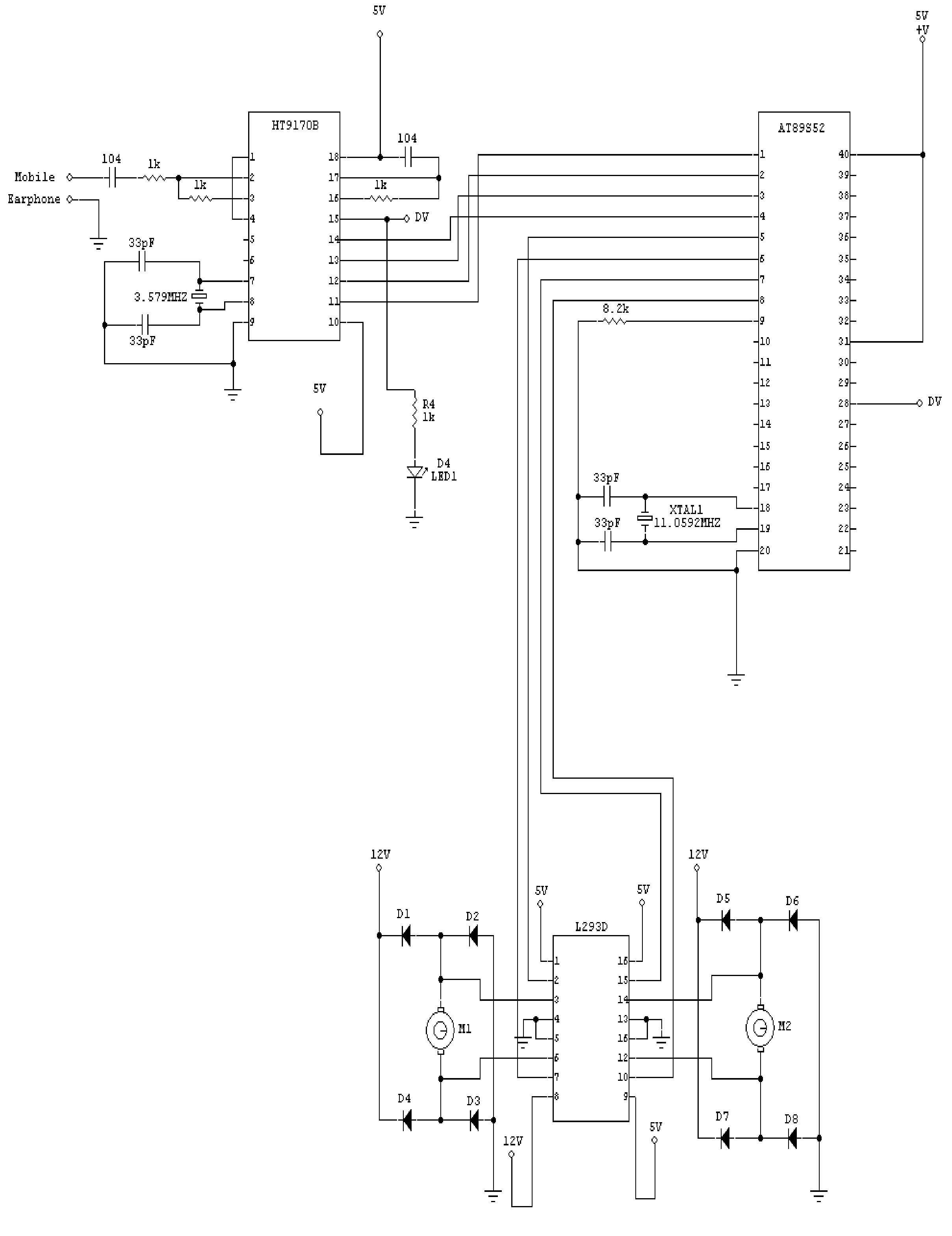
- Most industrial gear motors are AC-powered, fixed-speed devices, although there are fixed-gear-ratio, variable-speed motors that provide a greater degree of control. DC gear motors are used primarily in automotive applications such as power winches on trucks, windshield wiper motors and power seat or power window motors.

Many Applications

- What power can openers, garage door openers, stair lifts, rotisserie motors, timer cycle knobs on washing machines, power drills, cake mixers and electromechanical clocks have in common is that they all use various integrations of gear motors to derive a large force from a relatively small electric motor at a manageable speed. In industry, gear motor applications in jacks, cranes, lifts, clamping, robotics, conveyance and mixing are too numerous to count.

CHAPTER 3

CIRUIT DIAGRAM



Circuit operation

In early days, robotic manipulators have been implemented in different control techniques like mechanical control and the remote control or tele-opertation. But with the advent of high performance, a new way of control using mobile has been implemented which is introduced in this project. All the above systems are controlled by the Microcontroller. In our project we are using the popular 8 bit microcontroller AT89S52. It is a 40 pin microcontroller. The Microcontroller AT89S52 is used to control the dc motors. It gets the signals from the DTMF decoder and it drives the motors according to the DTMF inputs. Two DC motors are used to drive the robot in four directions i.e. Front ,Back,Right,Left.

Source code

```
#include<reg51.h>
delay(unsigned char);
//PORT 2
sbit L_MOTORFORWARD=P2^0;
sbit L_MOTORBACKWARD=P2^1;
sbit R_MOTORFORWARD=P2^6;
sbit R_MOTORBACKWARD=P2^7;
sbit obstacle=P0^0;
sbit buzzer=P0^7;
main()
63
{
int i;
L_MOTORFORWARD=0;
L_MOTORBACKWARD=0;
R_MOTORFORWARD=0;
R_MOTORBACKWARD=0;
smoke=1;
buzzer=0;
while(1)
{
if(P1==0xF2)
{
L_MOTORFORWARD=1;
R_MOTORFORWARD=1;
delay(150); //FORWARD
L_MOTORFORWARD=0;
R_MOTORFORWARD=0;
}
if(P1==0xF4)
{
L_MOTORFORWARD=1;
64
delay(150);
L_MOTORFORWARD=0; //LEFT
L_MOTORBACKWARD=0;
R_MOTORFORWARD=0;
R_MOTORBACKWARD=0;
}
if(P1==0xF6)
{
R_MOTORFORWARD=1;
delay(150);
R_MOTORFORWARD=0;
L_MOTORFORWARD=0;
L_MOTORBACKWARD=0; //RIGHT
R_MOTORBACKWARD=0;
}
if(P1==0xF8)
{
L_MOTORBACKWARD=1;
R_MOTORBACKWARD=1;
delay(150); //BACK
L_MOTORBACKWARD=0;
R_MOTORBACKWARD=0;
}
if(obstacle==0)
{
buzzer=1;
65
for(i=0;i<32000;i++);
buzzer=0;
}
if(usonic==0)
{
buzzer=1;
for(i=0;i<10000;i++);
buzzer=0;
}
} //while
} //main
```

```
delay(unsigned char time)
{
unsigned char i,j;
for(i=0;i<time;i++)
for(j=0;j<250;j++);
}
```

CHAPTER 4

SAMPLE PROGRAMS

Example 1:

```
              org 00h            // Starting  Of The Program From 00h memory
       back:  mov P1,#55h        //Move 55h to Port1
              acall delay        // Call Delay Function
              mov P1,#0AAh       //Move 55h to Port1
              lcall delay        // Call Delay Function
              sjmp back
       delay: mov r5,#30h
       again: djnz r5,again      // Generating delay
              ret                // Return Of Loop
              end           // End Of Program
```

Example 2:

```
          #include<reg51.h>
         void delay(unsigned int);  //Global Declaration  Of Delay
         void main()
          {
      P0=0x00;                  // Clearing Of Port O
     while(1)                   //Infinite Loop
{
P0=0xAA;
delay(30);
P0=0x55;
delay(30);
}
                    }
void delay(unsigned int x)          //Delay Main Function
{
unsigned int i,j;
for(i=0;i<=x;i++)
for(j=0;j<=1275;j++);
}
```

Example3:

```
                        #include<reg51.h>
                       sbit  SWITCH=P1^0;          // Input  to P1.0
                       sbit  LED =P2^5;              // Out  to P2.5
                       void main()
{
  while(1)                      //Infinite Loop
     {
             if (SWITCH==0)
               {
               LED=1;
               }
            else
              {
              LED=0;
              }
      }
 }
```

Example4:

```
#include<reg51.h>

unsigned char str[10]="MAGNI5";   // String Of Data

void main()

{

        unsigned int i=0;

        TMOD=0X20;                  // Timer1, Mode2

        SCON=0X50;                  //1 Start Bit And 1 Stop Bit

        TH1=-3;                     // Baud Rate 9600

        TR1=1;                      //Start Timer 1

While(1)

  {

                for(i=0;i<10;i++)

                {

                SBUF=str[i];

                while(TI==0);       // Wait  Data Till Bit Of Data

                TI=0;

                 }

        }

  }
```

SOFTWARE DEVELOPMENT

5.1 Introduction:

In this chapter the software used and the language in which the program code is defined is mentioned and the program code dumping tools are explained. The chapter also documents the development of the program for the application. This program has been termed as “Source code”. Before we look at the source code we define the two header files that we have used in the code.

5.2 Tools Used:

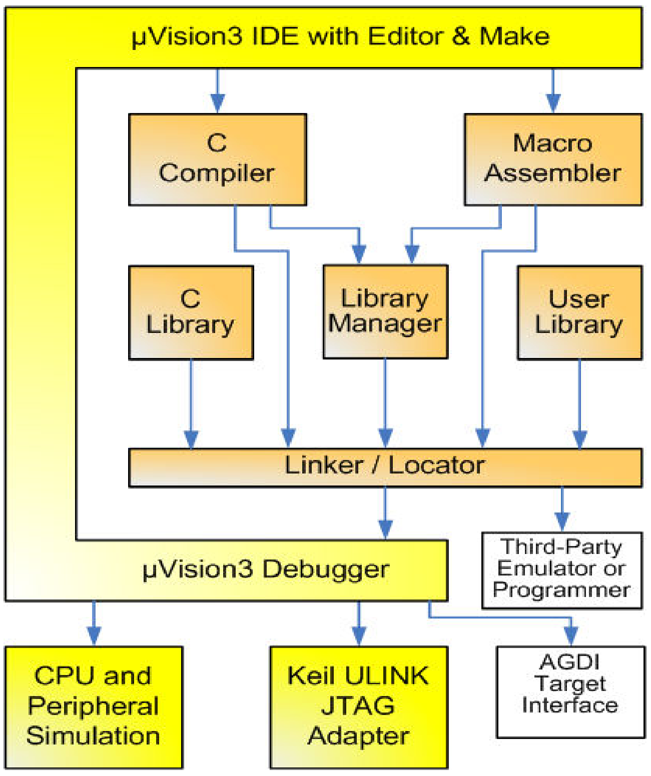


Figure 4.1 Keil Software- internal stages

Keil development tools for the 8051 Microcontroller Architecture support every level of software developer from the professional applications

5.3 C51 Compiler & A51 Macro Assembler:

Source files are created by the µVision IDE and are passed to the C51 Compiler or A51 Macro Assembler. The compiler and assembler process source files and create replaceable object files.

The Keil C51 Compiler is a full ANSI implementation of the C programming language that supports all standard features of the C language. In addition, numerous features for direct support of the 8051 architecture have been added.

5.4µVISION

What's New in µVision3?

µVision3 adds many new features to the Editor like Text Templates, Quick Function Navigation, and Syntax Coloring with brace high lighting Configuration Wizard for dialog based startup and debugger setup. µVision3 is fully compatible to µVision2 and can be used in parallel with µVision2.

What is µVision3?

µVision3 is an IDE (Integrated Development Environment) that helps you write, compile, and debug embedded programs. It encapsulates the following components:

- A project manager.
- A make facility.
- Tool configuration.
- Editor.
- A powerful debugger.

To help you get started, several example programs (located in the **\C51\Examples**, **\C251\Examples**, **\C166\Examples**, and **\ARM\...\Examples**) are provided.

- **HELLO** is a simple program that prints the string "Hello World" using the Serial Interface.
- **MEASURE** is a data acquisition system for analog and digital systems.
- **TRAFFIC** is a traffic light controller with the RTX Tiny operating system.
- **SIEVE** is the SIEVE Benchmark.
- **DHRY** is the Dhrystone Benchmark.
- **WHETS** is the Single-Precision Whetstone Benchmark.

Additional example programs not listed here are provided for each device architecture.

7.3 BUILDING AN APPLICATION IN µVISION

To build (compile, assemble, and link) an application in µVision2, you must:

1. Select Project -(forexample,**166\EXAMPLES\HELLO\HELLO.UV2**).
2. Select Project - Rebuild all target files or Build target.

    µVision2 compiles, assembles, and links the files in your project.

Creating Your Own Application in µVision2

To create a new project in µVision2, you must:

1. Select Project - New Project.
2. Select a directory and enter the name of the project file.
3. Select Project - Select Device and select an 8051, 251, or C16x/ST10 device from the Device Database™.
4. Create source files to add to the project.
5. Select Project - Targets, Groups, Files. Add/Files, select Source Group1, and add the source files to the project.
6. Select Project - Options and set the tool options. Note when you select the target device from the Device Database™ all special options are set automatically. You typically only need to configure the memory map of your target hardware. Default memory model settings are optimal for most applications.
7. Select Project - Rebuild all target files or Build target.

Debugging an Application in µVision2

To debug an application created using µVision2, you must:

1. Select Debug - Start/Stop Debug Session.
2. Use the Step toolbar buttons to single-step through your program. You may enter **G, main** in the Output Window to execute to the main C function.
3. Open the Serial Window using the **Serial #1** button on the toolbar.

    Debug your program using standard options like Step, Go, Break, and so on.

Starting µVision2 and Creating a Project

µVision2 is a standard Windows application and started by clicking on the program icon. To create a new project file select from the µVision2 menu

Project – New Project…. This opens a standard Windows dialog that asks you

for the new project file name.

We suggest that you use a separate folder for each project. You can simply use the icon Create New Folder in this dialog to get a new empty folder. Then

select this folder and enter the file name for the new project, i.e. Project1.

µVision2 creates a new project file with the name PROJECT1.UV2 which contains

a default target and file group name. You can see these names in the Project

Window – Files.

Now use from the menu Project – Select Device for Target and select a CPU

for your project. The Select Device dialog box shows the µVision2 device

database. Just select the microcontroller you use. We are using for our examples the Philips 80C51RD+ CPU. This selection sets necessary tool

options for the 80C51RD+ device and simplifies in this way the tool Configuration

Building Projects and Creating a HEX Files

Typical, the tool settings under Options – Target are all you need to start a new

application. You may translate all source files and line the application with a

click on the Build Target toolbar icon. When you build an application with

syntax errors, µVision2 will display errors and warning messages in the Output

Window – Build page. A double click on a message line opens the source file

on the correct location in a µVision2 editor window. Once you have successfully generated your application you can start debugging.

After you have tested your application, it is required to create an Intel HEX file to download the software into an EPROM programmer or simulator. µVision2 creates HEX files with each build process when Create HEX files under Options for Target – Output is enabled. You may start your PROM programming utility after the make process when you specify the program under the option Run User Program #1.

CPU Simulation

µVision2 simulates up to 16 Mbytes of memory from which areas can be

mapped for read, write, or code execution access. The µVision2 simulator traps

and reports illegal memory accesses.

In addition to memory mapping, the simulator also provides support for the

integrated peripherals of the various 8051 derivatives. The on-chip peripherals

of the CPU you have selected are configured from the Device

Database selection

You have made when you create your project target. Refer to page 58 for more

Information about selecting a device. You may select and display the on-chip peripheral components using the Debug menu. You can also change the aspects of each peripheral using the controls in the dialog boxes.

Start Debugging

You start the debug mode of µVision2 with the Debug – Start/Stop Debug

Session command. Depending on the Options for Target – Debug

Configuration, µVision2 will load the application program and run the startup

code µVision2 saves the editor screen layout and restores the screen layout of the last debug session. If the program execution stops, µVision2 opens an

editor window with the source text or shows CPU instructions in the disassembly window. The next executable statement is marked with a yellow arrow. During debugging, most editor features are still available.

For example, you can use the find command or correct program errors. Program source text of your application is shown in the same windows. The µVision2 debug mode differs from the edit mode in the following aspects:

⇒ The "Debug Menu and Debug Commands" described below are available. The additional debug windows are discussed in the following.

⇒ The project structure or tool parameters cannot be modified. All build Commands are disabled.

Disassembly Window

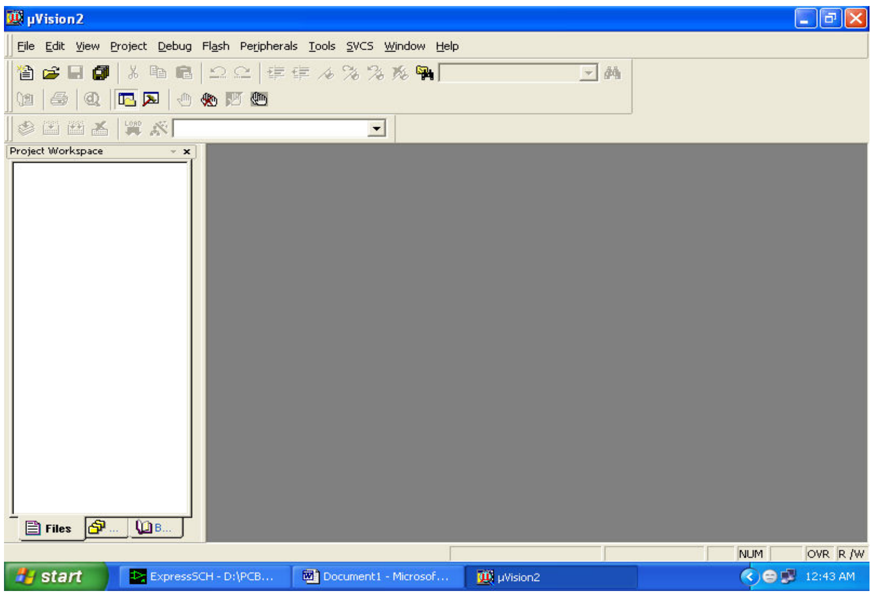
The Disassembly window shows your target program as mixed source and assembly program or just assembly code. A trace history of previously executed instructions may be displayed with Debug – View Trace Records. To enable the trace history, set Debug – Enable/Disable Trace Recording.

If you select the Disassembly Window as the active window all program step commands work on CPU instruction level rather than program source lines. You can select a text line and set or modify code breakpoints using toolbar buttons or the context menu commands.

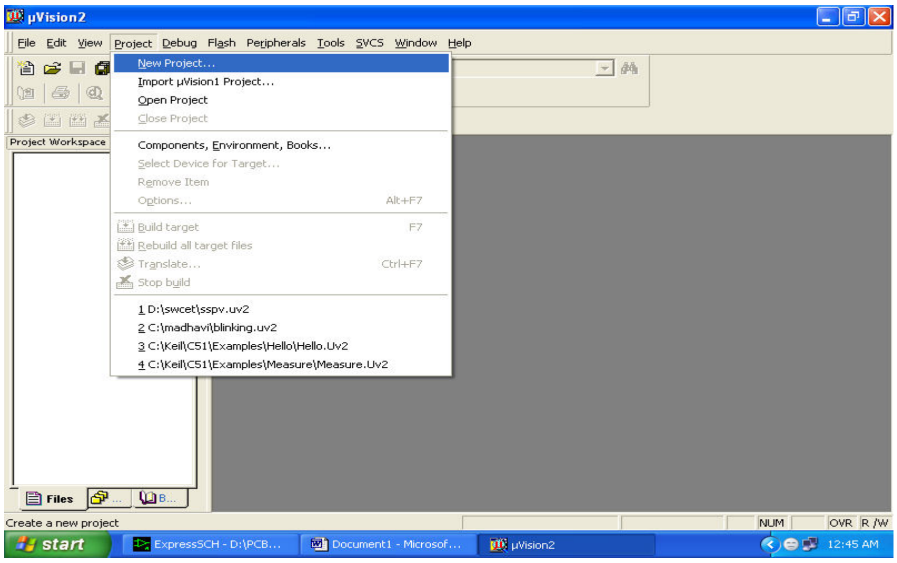
You may use the dialog Debug – Inline Assembly… to modify the CPU instructions. That allows you to correct mistakes or to make temporary changes to the target program you are debugging.

5.8 SOURCE CODE

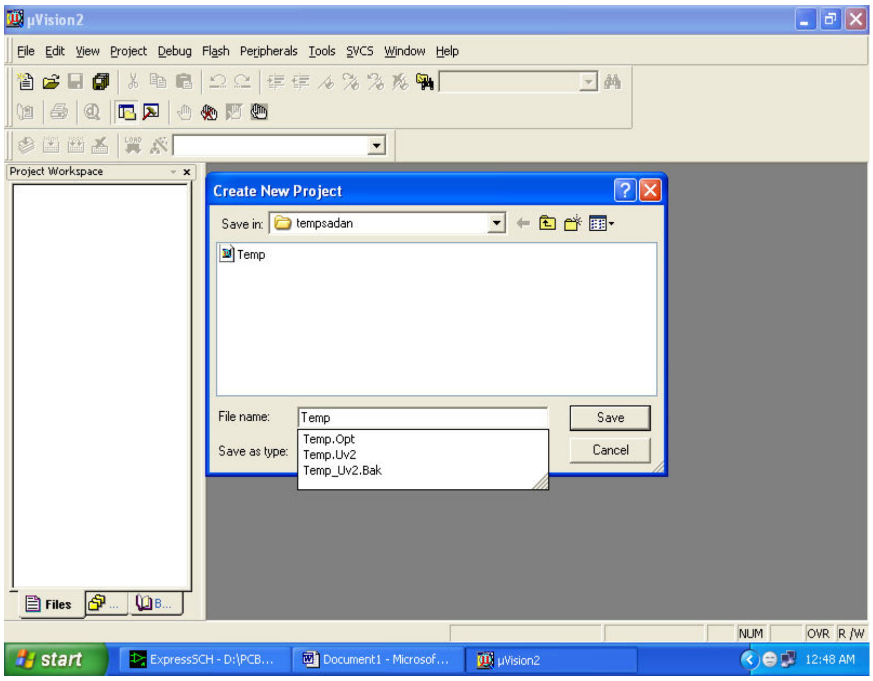
1. Click on the Keil uVision Icon on Desktop
2. The following fig will appear



3. Click on the Project menu from the title bar
4. Then Click on New Project



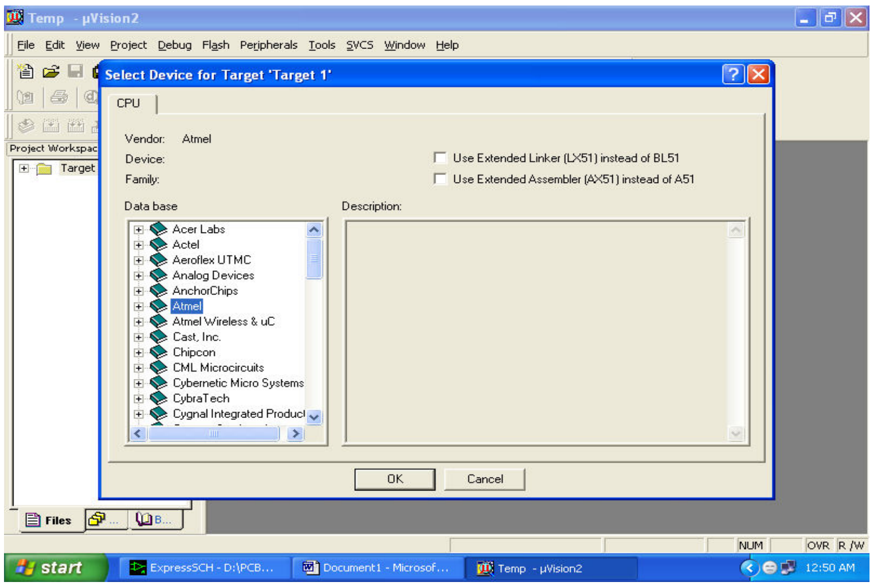
5. Save the Project by typing suitable project name with no extension in u r own folder sited in either C:\ or D:\



6. Then Click on Save button above.

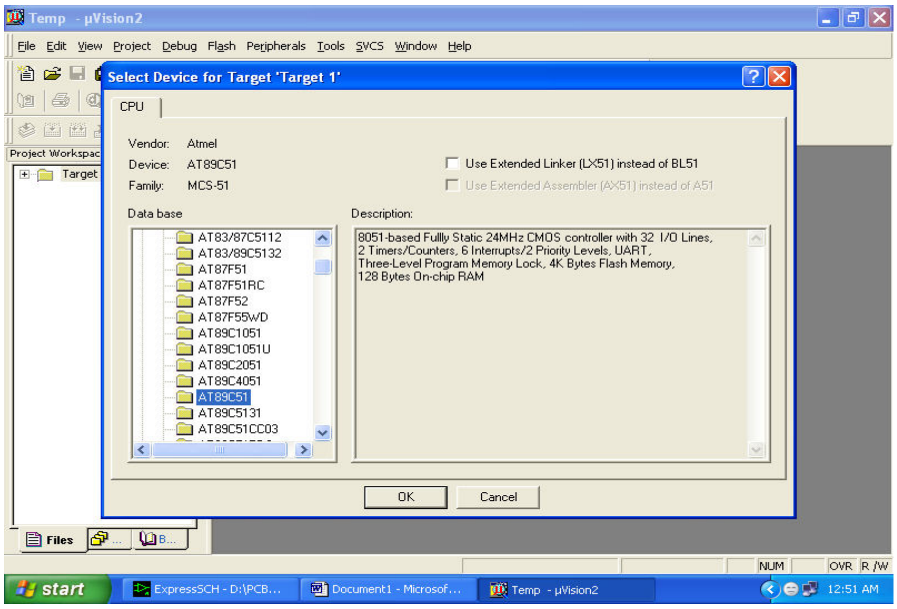
7. Select the component for u r project. i.e. Atmel……

8. Click on the + Symbol beside of Atmel



Vendor: Atmel
Device:
Family:
Use Extended Linker (LX51) instead of BL51
Use Extended Assembler (AX51) instead of A51
Data base
Description:
Acer Labs
Actel
Aeroflex UTMC
Analog Devices
AnchorChips
Atmel
Atmel Wireless & uC
Cast, Inc.
Chipcon
CML Microcircuits
Cybernetic Micro Systems
CybraTech
Cygnal Integrated Product

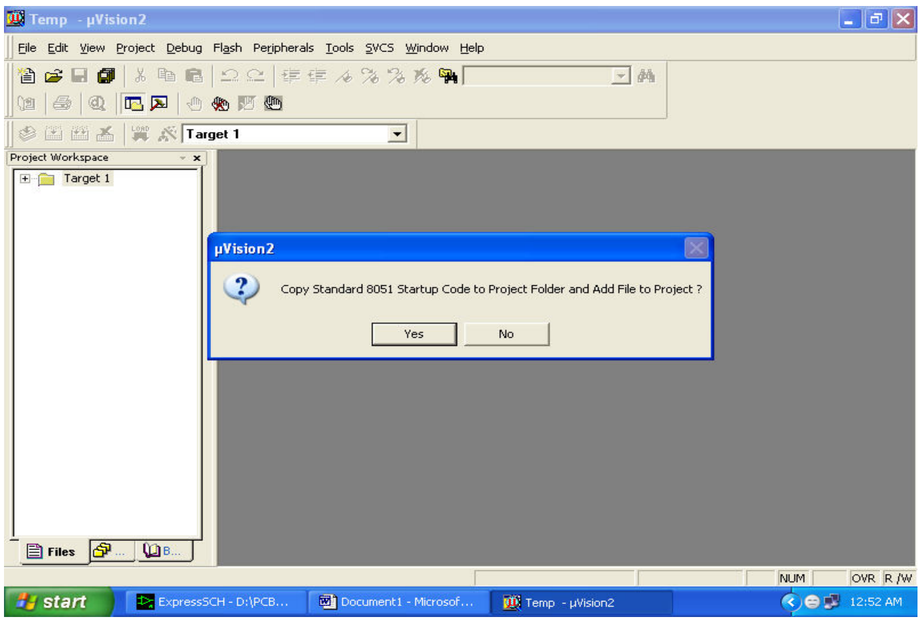
9. Select AT89C51 as shown below



Vendor: Atmel
Device: AT89C51
Family: MCS-51
Use Extended Linker (LX51) instead of BL51
Use Extended Assembler (AX51) instead of A51
Data base
AT83/87C5112
AT83/89C5132
AT87F51
AT87F51RC
AT87F52
AT87F55WD
AT89C1051
AT89C1051U
AT89C2051
AT89C4051
AT89C51
AT89C5131
AT89C51CC03
Description:
8051-based Fullly Static 24MHz CMOS controller with 32 I/O Lines,
2 Timers/Counters, 6 Interrupts/2 Priority Levels, UART,
Three-Level Program Memory Lock, 4K Bytes Flash Memory,
128 Bytes On-chip RAM

10. Then Click on "OK"

11. The Following fig will appear

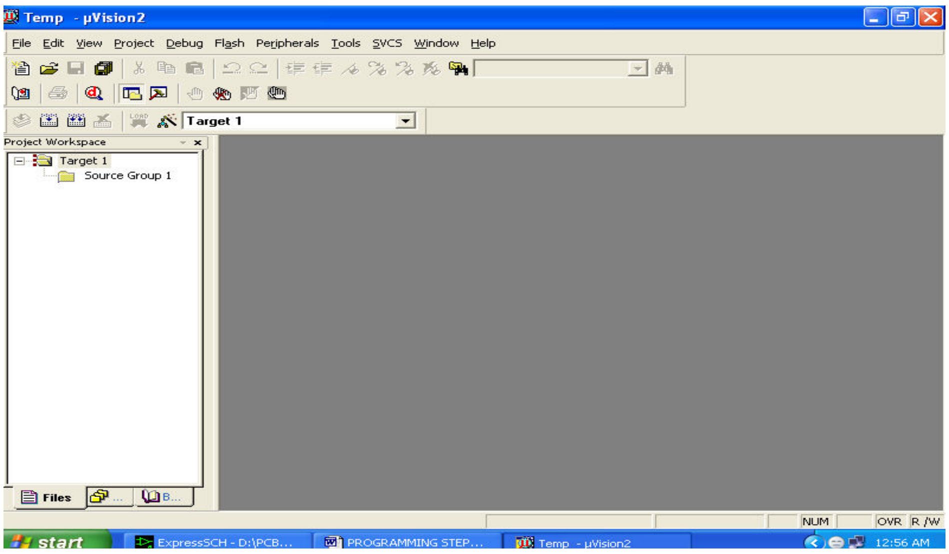


Copy Standard 8051 Startup Code to Project Folder and Add File to Project ?

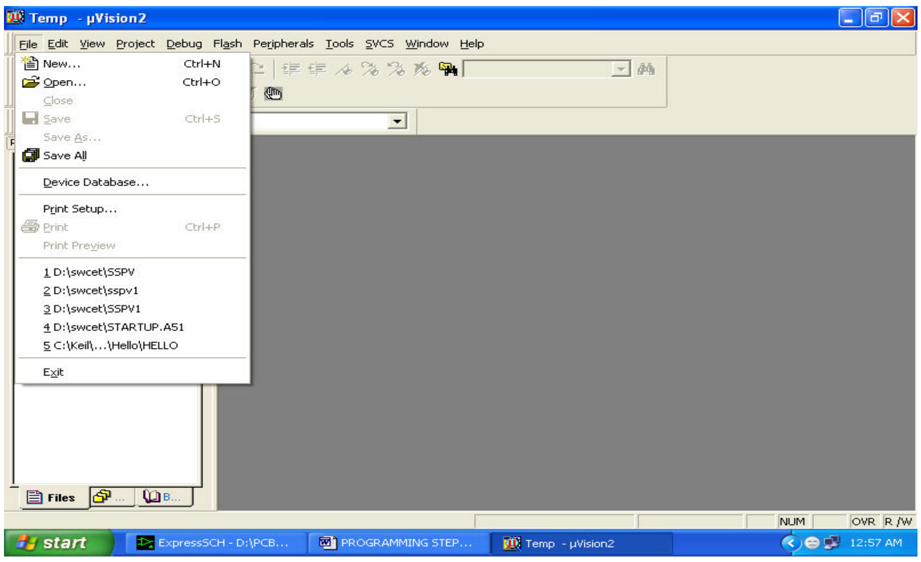
12. Then Click either YES or NO………mostly "NO"

13. Now your project is ready to USE

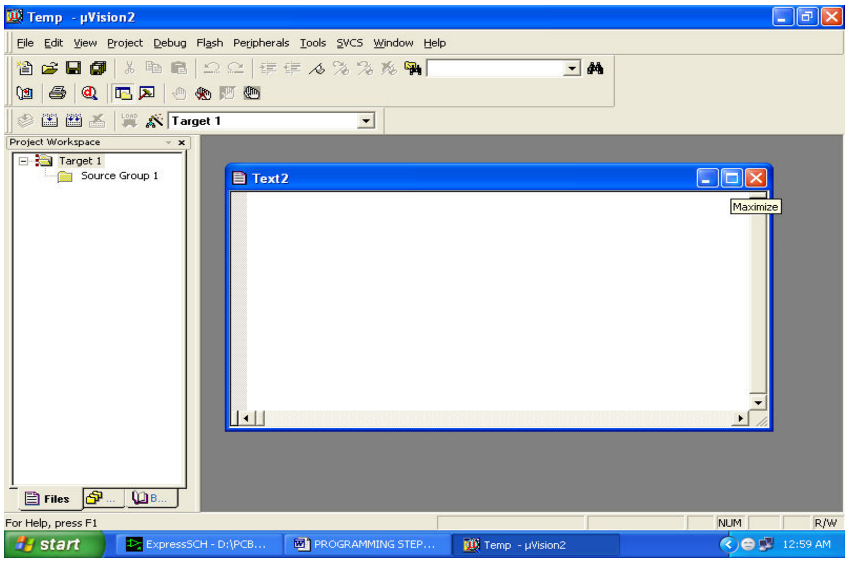
14. Now double click on the Target1, you would get another option "Source group 1" as shown in next page.



15. Click on the file option from menu bar and select "new"

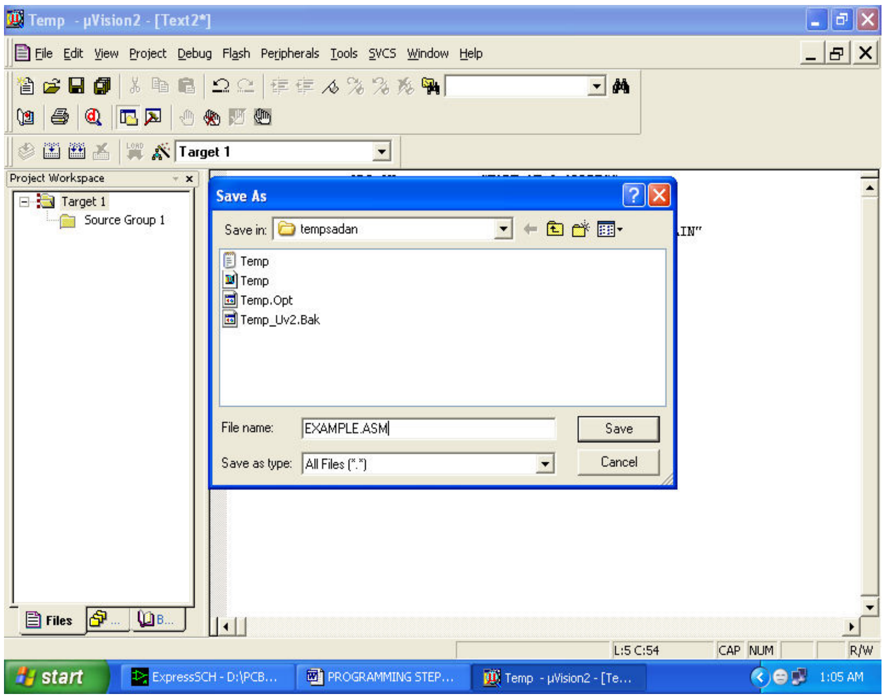


16. The next screen will be as shown in next page, and just maximize it by double clicking on its blue boarder.

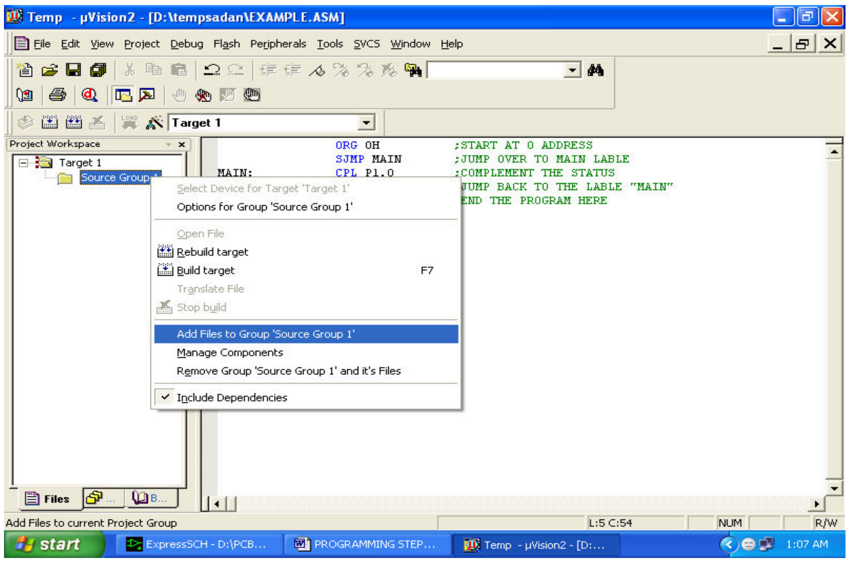


17. Now start writing program in either in “C” or “ASM”

18. For a program written in Assembly, then save it with extension “. asm” and for “C” based program save it with extension “ .C”

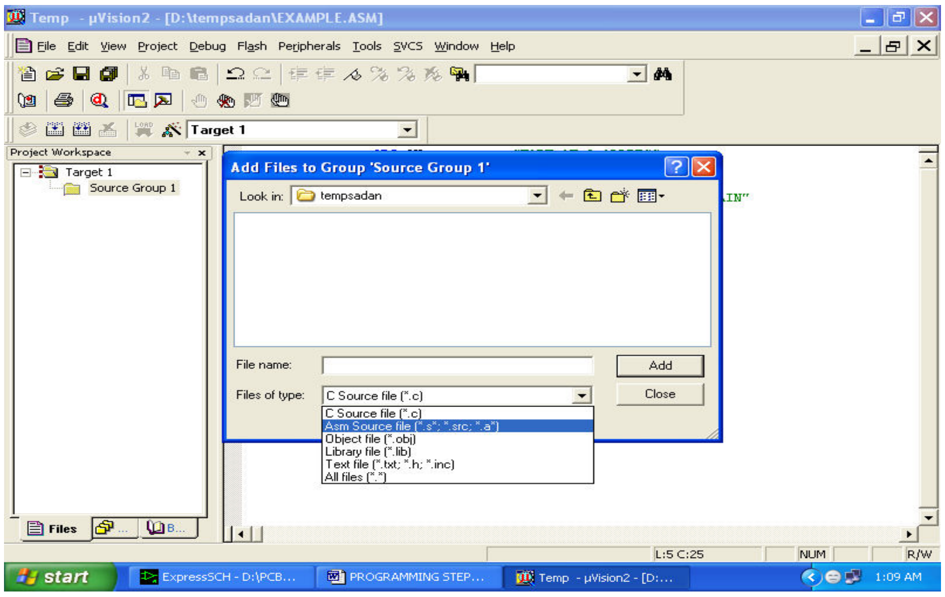


19. Now right click on Source group 1 and click on "**Add files to Group Source**"

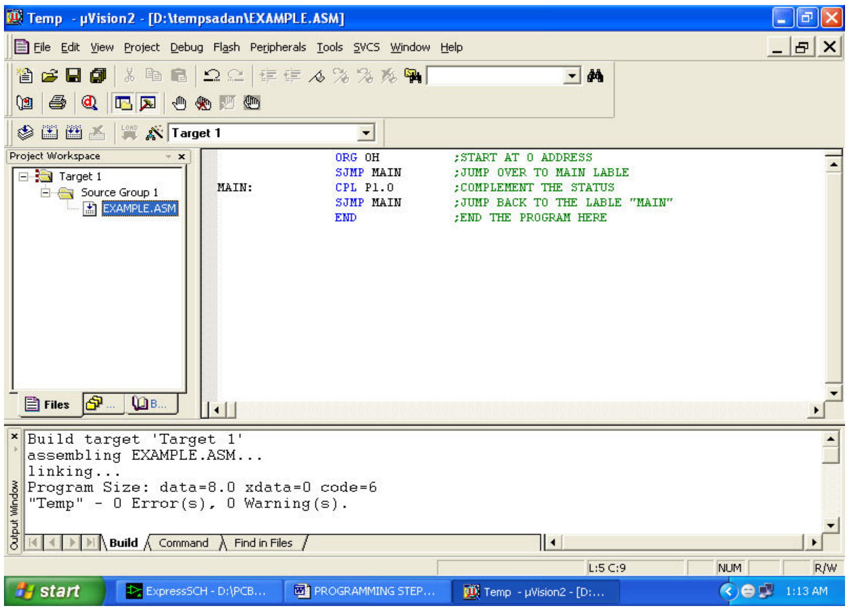

Temp - µVision2 - [D:\tempsadan\EXAMPLE.ASM]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Source Group 1
Select Device for Target 'Target 1'
Options for Group 'Source Group 1'
Open File
Rebuild target
Build target F7
Translate File
Stop build
Add Files to Group 'Source Group 1'
Manage Components
Remove Group 'Source Group 1' and it's Files
Include Dependencies
Files
Add Files to current Project Group
L:5 C:54
NUM
R/W
start
ExpressSCH - D:\PCB...
PROGRAMMING STEP...
Temp - µVision2 - [D:...
1:07 AM

```
                ORG OH              ;START AT 0 ADDRESS
                SJMP MAIN           ;JUMP OVER TO MAIN LABLE
MAIN:           CPL P1.0            ;COMPLEMENT THE STATUS
                                    JUMP BACK TO THE LABLE "MAIN"
                                    END THE PROGRAM HERE
```

20. Now you will get another window, on which by default "C" files will appear.



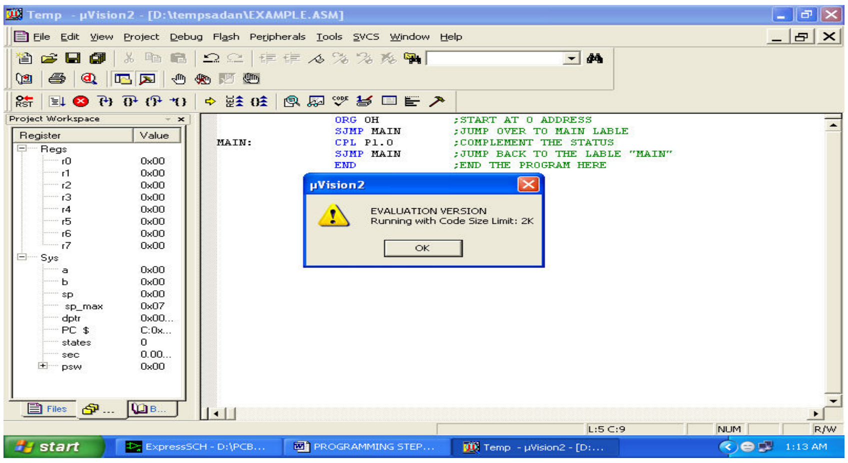
21. Now select as per your file extension given while saving the file
22. Click only one time on option “**ADD**”
23. Now Press function key F7 to compile. Any error will appear if so happen.



```
                ORG 0H          ;START AT 0 ADDRESS
                SJMP MAIN       ;JUMP OVER TO MAIN LABLE
MAIN:           CPL P1.0        ;COMPLEMENT THE STATUS
                SJMP MAIN       ;JUMP BACK TO THE LABLE "MAIN"
                END             ;END THE PROGRAM HERE
```

```
Build target 'Target 1'
assembling EXAMPLE.ASM...
linking...
Program Size: data=8.0 xdata=0 code=6
"Temp" - 0 Error(s), 0 Warning(s).
```

24. If the file contains no error, then press Control+F5 simultaneously.
25. The new window is as follows



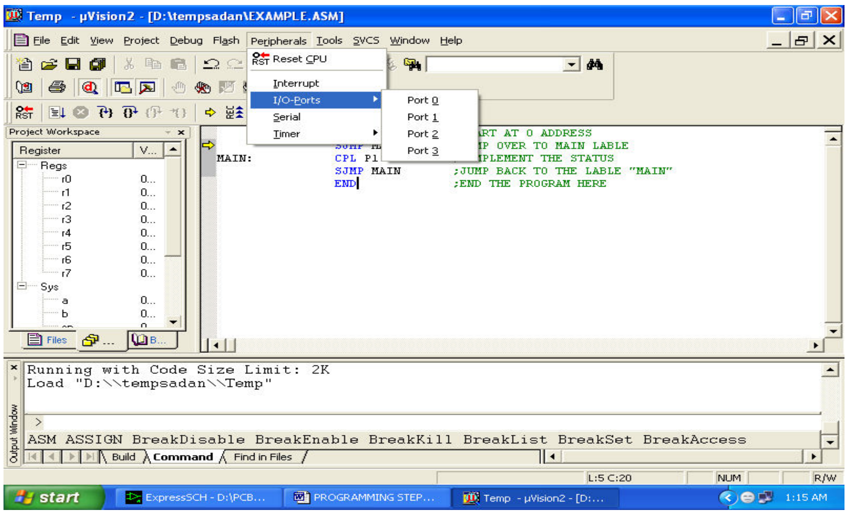
```
               ORG  0H           ;START AT 0 ADDRESS
               SJMP MAIN         ;JUMP OVER TO MAIN LABLE
MAIN:          CPL  P1.0         ;COMPLEMENT THE STATUS
               SJMP MAIN         ;JUMP BACK TO THE LABLE "MAIN"
               END               ;END THE PROGRAM HERE
```

EVALUATION VERSION
Running with Code Size Limit: 2K

Register	Value
Regs	
r0	0x00
r1	0x00
r2	0x00
r3	0x00
r4	0x00
r5	0x00
r6	0x00
r7	0x00
Sys	
a	0x00
b	0x00
sp	0x00
sp_max	0x07
dptr	0x00...
PC $	C:0x...
states	0
sec	0.00...
psw	0x00

26. Then Click “OK”

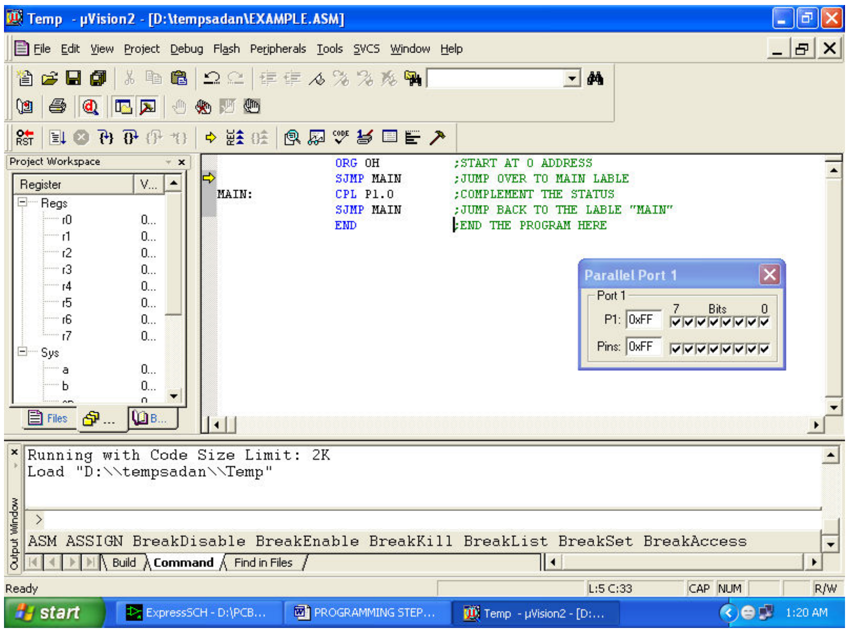
27. Now Click on the Peripherals from menu bar, and check your required port as shown in fig below



```
Running with Code Size Limit: 2K
Load "D:\\tempsadan\\Temp"

>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

28. Drag the port a side and click in the program file.



```
                ORG OH          ;START AT 0 ADDRESS
                SJMP MAIN       ;JUMP OVER TO MAIN LABLE
MAIN:           CPL P1.0        ;COMPLEMENT THE STATUS
                SJMP MAIN       ;JUMP BACK TO THE LABLE "MAIN"
                END             ;END THE PROGRAM HERE
```

```
Running with Code Size Limit: 2K
Load "D:\\tempsadan\\Temp"
>
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess
```

29. Now keep Pressing function key "F11" slowly and observe.
30. You are running your program successfully

5.6 Flash Magic:

Features:

- Straightforward and intuitive user interface
- Five simple steps to erasing and programming a device and setting any options desired
- Programs Intel Hex Files
- Automatic verifying after programming
- Fills unused flash to increase firmware security
- Ability to automatically program checksums. Using the supplied checksum calculation routine your firmware can easily verify the integrity of a Flash block, ensuring no unauthorized or corrupted code can ever be executed
- Program security bits
- Check which Flash blocks are blank or in use with the ability to easily erase all blocks in use
- Read the device signature
- Read any section of Flash and save as an Intel Hex File
- Reprogram the Boot Vector and Status Byte with the help of confirmation features that prevent accidentally programming incorrect values

- Displays the contents of Flash in ASCII and Hexadecimal formats
- Single-click access to the manual, Flash Magic home page and NXP Microcontrollers home page
- Ability to use high-speed serial communications on devices that support it. Flash Magic calculates the highest baud rate that both the device and your PC can use and switches to that baud rate transparently
- Command Line interface allowing Flash Magic to be used in IDEs and Batch Files
- Manual in PDF format
- supports half-duplex communications
- Verify Hex Files previously programmed
- Save and open settings
- Able to reset Rx2 and 66x devices (revision G or higher)
- Able to control the DTR and RTS RS232 signals when connected to RST and /PSEN to place the device into Boot ROM and Execute modes automatically. An example circuit diagram is included in the Manual. This is essential for ISP with target hardware that is hard to access.
- This enables us to send commands to place the device in Boot ROM mode, with support for command line interfaces. The installation includes an example project for the Keil and Raisonance 8051 compilers that show how to build support for this feature into applications.
- Able to play any Wave file when finished programming.
- built in automated version checker - helps ensure you always have the latest version.
- Powerful, flexible Just In Time Code feature. Write your own JIT Modules to generate last minute code for programming. Uses include:
    - Serial number generation
    - Copy protection and copy authorization
    - Storing program date and time - manufacture date
    - Storing program operator and location
    - Lookup table generation
    - Language tables or language selection
    - Centralized record keeping
- Obtaining latest firmware from the Corporate Web site or project intranet

Requirements:

Flash Magic works on any versions of Windows, except Windows 95. 10Mb of disk space is required. As mentioned earlier, we are automating two different routines in our project and hence we used the method of polling to continuously monitor those tasks and act accordingly

CHAPTER6

ADVANTAGES:

1. CONTROLLING THE DEVICE FROM REMOTE PLACE
2. LOW COST.
3. LESS COMPLXCITY

DIS ADVANTAGES

1. WITH OUT SIGNAL ITS NOT WORKING
2. HEAD PHONES SHOULD COMPATABLE WITH MOBILE PHONE

APPLICATIONS:

1.INDUSTRIAL APPLICATIONS

2. HOME APPLICATIONS

3. AGRICULTURE APPLICATIONS

CONCLUSION :

CONCLUSION

The project “MOBILE BASED LAND ROBOT” has been successfully designed and tested. Integrating features of all the hardware components used have developed it. Presence of every module has been reasoned out and placed carefully thus contributing to the best working of the unit. Secondly, using highly advanced IC’s and with the help of growing technology the project has been successfully implemented.

BIBILOGRAPHY

1. WWW.MITEL.DATABOOK.COM

2. WWW.ATMEL.DATABOOK.COM

3. WWW.FRANKLIN.COM

4. WWW.KEIL.COM

REFERENCES

1. "The 8051 Microcontroller Architecture, Programming & Applications"

   By Kenneth J Ayala.

2. "The 8051 Microcontroller & Embedded Systems" by Mohammed Ali Mazidi and Janice Gillispie Mazidi

3. "Power Electronics" by M D Singh and K B Khanchandan

4. "Linear Integrated Circuits" by D Roy Choudary & Shail Jain

5. "Electrical Machines" by S K Bhattacharya

6. "Electrical Machines II" by B L Thereja

7. www.8051freeprojectsinfo.com