ABSTRACT

AIM:

The main aim of this project is to design a robotic system with super intelligence which can automatically avoid obstacles in its way, which senses the fire and also detects bombs.

PURPOSE:

The purpose of the project is to implement a security system with robots for the detection of fire, bombs and obstacles and to provide an alert system when any of these are detected.

Description:

Now a day's every system is automated in order to face new challenges. In the present days Automated systems have less manual operations, flexibility, reliability and accurate. Due to this demand every field prefers automated control systems. Especially in the field of electronics automated systems are giving good performance. In the present scenario of war situations, unmanned systems plays very important role to minimize human losses. So this robot is very useful to do operations like detecting fire, obstacle, bombs and other things.

In this project we are controlling a robot with wire. In this system, a robot is fitted with motors. A micro controller is used to control all operations. According to the motor operations the ROBOT will operate in specified directions. If any obstacle is observed by the robot it will change its direction or it will stop. In this system we have fire sensor. Whenever the sensors sense the smoke, buzzer will ON and also we have used another sensor for detecting the bomb or any dangerous materials or land mines. A 12V battery is provided to power the robot to perform all functions. This kind of project is very useful where security is needed.

SOFTWARE: Embedded ‘C’

TOOLS: Small Device Cross Compiler & Keil uvision3.

TARGET DEVICE: 8051 Microcontroller

APPLICATIONS: Robot based application

ADVANTAGES: Low cost, automated operation, Low Power consumption.

Index

CHAPTER1. INTRODUCTION TO EMBEDDED SYSTEMS

1.1 Introduction to embedded systems

1.2 Applications of embedded system

1.3 Microcontrollers for embedded systems

CHAPTER2. 8051 MICRO CONTROLLER

2.1 8051 Architechture

2.2 Block diagram microcontroller

2.3 Logic symbol of 8051

2.4 Criteria for choosing a microcontroller

CHAPTER3. AT89S52 MICROCONTROLLERS

3.1 AT89S52 features

3.2 Block diagram

3.3 Memory organisation

3.4 Timers

3.5 Interrupts

CHAPTER4. WORKING FLOW OF THE PROJECT BLOCK DIAGRAM

4.1 Block diagram

4.2 Regulated power supply

4.3 Land mine sensor

**CHAPTER5. SOFTWAREDEVELOPEMENT**

5.1 Keil software

5.2 C51 Compiler & A51 Macro Assembler Software development

5.3µvision

5.4 Building an application in µvisionBUILDING AN APPLICATION IN µVISION

* CONCLUSION
* BIBILOGRAPHY

CHAPTER 1

INTRODUCTION TO EMBEDDED SYSTEMS

1.1 Introduction to Embedded Systems:

EMBEDDED SYSTEM

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, sometimes with real-time computing constraints. It is usually embedded as part of a complete device including hardware and mechanical parts. In contrast, a general-purpose computer, such as a personal computer, can do many different tasks depending on programming. Embedded systems have become very important today as they control many of the common devices we use.

Since the embedded system is dedicated to specific tasks, design engineers can optimize it, reducing the size and cost of the product, or increasing the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and MP3 players, to large stationary installations like traffic lights, factory controllers, or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not an exactly defined term, as many systems have some element of programmability. For example, Handheld computers share some elements with embedded systems — such as the operating systems and microprocessors which power them — but are not truly embedded systems, because they allow different applications to be loaded and peripherals to be connected.

An embedded system is some combination of computer hardware and software, either fixed in capability or programmable, that is specifically designed for a particular kind of application device. Industrial machines, automobiles, medical equipment, cameras, household appliances, airplanes, vending machines, and toys (as well as the more obvious cellular phone and PDA) are among the myriad possible hosts of an embedded system. Embedded systems that are programmable are provided with a programming interface, and embedded systems programming is a specialized occupation.

Certain operating systems or language platforms are tailored for the embedded market, such as Embedded Java and Windows XP Embedded. However, some low-end consumer products use very inexpensive microprocessors and limited storage, with the application and operating system both part of a single program. The program is written permanently into the system's memory in this case, rather than being loaded into RAM (random access memory), as programs on a personal computer are.

1.2 APPLICATIONS OF EMBEDDED SYSTEM

We are living in the Embedded World. You are surrounded with many embedded products and your daily life largely depends on the proper functioning of these gadgets. Television, Radio, CD player of your living room, Washing Machine or Microwave Oven in your kitchen, Card readers, Access Controllers, Palm devices of your work space enable you to do many of your tasks very effectively. Apart from all these, many controllers embedded in your car take care of car operations between the bumpers and most of the times you tend to ignore all these controllers.

In recent days, you are showered with variety of information about these embedded controllers in many places. All kinds of magazines and journals regularly dish out details about latest technologies, new devices; fast applications which make you believe that your basic survival is controlled by these embedded products. Now you can agree to the fact that these embedded products have successfully invaded into our world. You must be wondering about these embedded controllers or systems. What is this Embedded System?

The computer you use to compose your mails, or create a document or analyze the database is known as the standard desktop computer. These desktop computers are manufactured to serve many purposes and applications.

You need to install the relevant software to get the required processing facility. So, these desktop computers can do many things. In contrast, embedded controllers carryout a specific work for which they are designed. Most of the time, engineers design these embedded controllers with a specific goal in mind. So these controllers cannot be used in any other place.

Theoretically, an embedded controller is a combination of a piece of microprocessor based hardware and the suitable software to undertake a specific task.

These days designers have many choices in microprocessors/microcontrollers. Especially, in 8 bit and 32 bit, the available variety really may overwhelm even an experienced designer. Selecting a right microprocessor may turn out as a most difficult first step and it is getting complicated as new devices continue to pop-up very often.

In the 8 bit segment, the most popular and used architecture is Intel's 8031. Market acceptance of this particular family has driven many semiconductor manufacturers to develop something new based on this particular architecture. Even after 25 years of existence, semiconductor manufacturers still come out with some kind of device using this 8031 core.

* Military and aerospace software applications

From in-orbit embedded systems to jumbo jets to vital battlefield networks, designers of mission-critical aerospace and defense systems requiring real-time performance, scalability, and high-availability facilities consistently turn to the LynxOS® RTOS and the LynxOS-178 RTOS for software certification to DO-178B.

Rich in system resources and networking services, LynxOS provides an off-the-shelf software platform with hard real-time response backed by powerful distributed computing (CORBA), high reliability, software certification, and long-term support options.

The LynxOS-178 RTOS for software certification, based on the RTCA DO-178B standard, assists developers in gaining certification for their mission- and [safety-critical systems](http://www.lynuxworks.com/products/whitepapers/safety-critical.php3). Real-time systems programmers get a boost with LynuxWorks' [DO-178B RTOS training courses](http://www.lynuxworks.com/support/courses/do-178b.php).

LynxOS-178 is the first DO-178B and EUROCAE/ED-12B certifiable, POSIX®-compatible RTOS solution.

* [Communications applications](http://www.lynuxworks.com/products/jumpstart/communications.php3)

"Five-nines" availability, [CompactPCI](http://www.lynuxworks.com/board-support/cpci.php) hot swap support, and hard real-time response—[LynxOS](http://www.lynuxworks.com/rtos/rtos.php) delivers on these key requirements and more for today's carrier-class systems. Scalable kernel configurations, distributed computing capabilities, integrated communications stacks, and fault-management facilities make LynxOS the ideal choice for companies looking for a single operating system for all embedded telecommunications applications—from complex central controllers to simple line/trunk cards.

LynuxWorks [Jumpstart for Communications](http://www.lynuxworks.com/products/jumpstart/communications.php3) package enables OEMs to rapidly develop mission-critical communications equipment, with pre-integrated, state-of-the-art, data networking and porting software components—including source code for easy customization.

The Lynx Certifiable Stack (LCS) is a [secure TCP/IP protocol stack](http://www.lynuxworks.com/rtos/lcs.php3) designed especially for applications where standards certification is required.

* [Electronics applications and consumer devices](http://www.lynuxworks.com/solutions/electronics/index.php)

As the number of powerful embedded processors in consumer devices continues to rise, the [BlueCat® Linux®](http://www.lynuxworks.com/embedded-linux/embedded-linux.php) operating system provides a highly reliable and royalty-free option for systems designers.

And as the wireless appliance revolution rolls on, web-enabled navigation systems, radios, personal communication devices, phones and PDAs all benefit from the cost-effective dependability, proven stability and full product life-cycle support opportunities associated with BlueCat embedded Linux. BlueCat has teamed up with industry leaders to make it easier to build [Linux mobile phones](http://www.lynuxworks.com/solutions/electronics/linux-phone.php) with Java integration.

For makers of low-cost consumer electronic devices who wish to integrate the LynxOS real-time operating system into their products, we offer special [MSRP-based pricing](http://www.lynuxworks.com/rtos/msrp.php3) to reduce royalty fees to a negligible portion of the device's MSRP.

* [Industrial automation and process control software](http://www.lynuxworks.com/solutions/industrial/industrial.php)

Designers of [industrial and process control systems](http://www.lynuxworks.com/solutions/industrial/industrial.php) know from experience that LynuxWorks operating systems provide the security and reliability that their industrial applications require.

From [ISO 9001 certification](http://www.lynuxworks.com/products/iso9001.php3) to [fault-tolerance](http://www.lynuxworks.com/rtos/rtos-mmu-high-availability.php), [POSIX conformance](http://www.lynuxworks.com/products/posix/posix.php3), [secure partitioning](http://www.lynuxworks.com/products/whitepapers/partition.php) and [high availability](http://www.lynuxworks.com/rtos/high-availability.php3), we've got it all. Take advantage of our 20 years of experience

1.3 MICROCONTROLLERS FOR EMBEDDED SYSTEMS

In the Literature discussing microprocessors, we often see the term Embedded System. Microprocessors and Microcontrollers are widely used in embedded system products. An embedded system product uses a microprocessor (or Microcontroller) to do one task only. A printer is an example of embedded system since the processor inside it performs one task only; namely getting the data and printing it. Contrast this with a Pentium based PC. A PC can be used for any number of applications such as word processor, print-server, bank teller terminal, Video game, network server, or Internet terminal. Software for a variety of applications can be loaded and run. Of course the reason a pc can perform myriad tasks is that it has RAM memory and an operating system that loads the application software into RAM memory and lets the CPU run it.

In an Embedded system, there is only one application software that is typically burned into ROM. An x86 PC contains or is connected to various embedded products such as keyboard, printer, modem, disk controller, sound card, CD-ROM drives, mouse, and so on. Each one of these peripherals has a Microcontroller inside it that performs only one task. For example, inside every mouse there is a Microcontroller to perform the task of finding the mouse position and sending it to the PC. Table 1-1 lists some embedded products.

CHAPTER 2

8051 MICRO CONTROLLER

2.1 8051 ARCHITECTURE:

The generic 8051 architecture supports a Harvard architecture, which contains two separate buses for both program and data. So, it has two distinctive memory spaces of 64K X 8 size for both programmed and data. It is based on an 8 bit central processing unit with an 8 bit Accumulator and another 8 bit B register as main processing blocks. Other portions of the architecture include few 8 bit and 16 bit registers and 8 bit memory locations.

Each 8051 device has some amount of data RAM built in the device for internal processing. This area is used for stack operations and temporary storage of data.

This bus architecture is supported with on-chip peripheral functions like I/O ports, timers/counters, versatile serial communication port. So it is clear that this 8051 architecture was designed to cater many real time embedded needs.

FEATURES OF 8051 ARCHITECTURE

* Optimized 8 bit CPU for control applications and extensive Boolean processing capabilities.
* 64K Program Memory address space.
* 64K Data Memory address space.
* 128 bytes of on chip Data Memory.
* 32 Bi-directional and individually addressable I/O lines.
* Two 16 bit timer/counters.
* Full Duplex UART.
* 6-source / 5-vector interrupt structure with priority levels.
* On chip clock oscillator.

Now we may be wondering about the non-mentioning of memory space meant for the program storage, the most important part of any embedded controller. Originally this 8051 architecture was introduced with on-chip, ‘one time programmable’ version of Program Memory of size 4K X 8. Intel delivered all these microcontrollers (8051) with user’s program fused inside the device. The memory portion was mapped at the lower end of the Program Memory area. But, after getting devices, customers couldn’t change any thing in their program code, which was already made available inside during device fabrication.

2.2 BLOCK DIAGRAM OF 8051



Figure 4.1 - Block Diagram of the 8051 Core

So, very soon Intel introduced the 8051 devices with re-programmable type of Program Memory using built-in EPROM of size 4K X 8. Like a regular EPROM, this memory can be re-programmed many times. Later on Intel started manufacturing these 8031 devices without any on chip Program Memory.

2.3 MICROCONTROLLER LOGIC SYMBOL:



ALE/PROG*:* Address Latch Enable output pulse for latching the low byte of the address during accesses to external memory. ALE is emitted at a constant rate of 1/6 of the oscillator frequency, for external timing or clocking purposes, even when there are no accesses to external memory. (However, one ALE pulse is skipped during each access to external Data Memory.) This pin is also the program pulse input (PROG) during EPROM programming.

PSEN *:* Program Store Enable is the read strobe to external Program Memory. When the device is executing out of external Program Memory, PSEN is activated twice each machine cycle (except that two PSEN activations are skipped during accesses to external Data Memory). PSEN is not activated when the device is executing out of internal Program Memory.

EA/VPP*:*  When EA is held high the CPU executes out of internal Program Memory (unless the Program Counter exceeds 0FFFH in the 80C51). Holding EA low forces the CPU to execute out of external memory regardless of the Program Counter value. In the 80C31, EA must be externally wired low. In the EPROM devices, this pin also receives the programming supply voltage (VPP) during EPROM programming.

XTAL1: Input to the inverting oscillator amplifier.

XTAL2: Output from the inverting oscillator amplifier.

The 8051’s I/O port structure is extremely versatile and flexible. The device has 32 I/O pins configured as four eight bit parallel ports (P0, P1, P2 and P3). Each pin can be used as an input or as an output under the software control. These I/O pins can be accessed directly by memory instructions during program execution to get required flexibility.

These port lines can be operated in different modes and all the pins can be made to do many different tasks apart from their regular I/O function executions. Instructions, which access external memory, use port P0 as a multiplexed address/data bus. At the beginning of an external memory cycle, low order 8 bits of the address bus are output on P0. The same pins transfer data byte at the later stage of the instruction execution.

Also, any instruction that accesses external Program Memory will output the higher order byte on P2 during read cycle. Remaining ports, P1 and P3 are available for standard I/O functions. But all the 8 lines of P3 support special functions: Two external interrupt lines, two counter inputs, serial port’s two data lines and two timing control strobe lines are designed to use P3 port lines. When you don’t use these special functions, you can use corresponding port lines as a standard I/O. Even within a single port, I/O operations may be combined in many ways. Different pins can be configured as input or outputs independent of each other or the same pin can be used as an input or as output at different times. You can comfortably combine I/O operations and special operations for Port 3 lines.

All the Port 3 pins are multifunctional. They are not only port pins, but also serve the functions of various special features as listed below:

Port Pin Alternate Function

P3.0 RxD (serial input port)

P3.1 TxD (serial output port)

**MICROCONTROLLER**

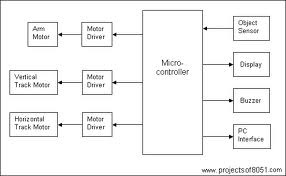
Micro controller is a true computer on a chip the design incorporates all of the features found in a microprocessor CPU: arithmetic and logic unit, stack pointer, program counter and registers. It has also had added additional features like RAM, ROM, serial I/O, counters and clock circuit.

Like the microprocessor, a microcontroller is a general purpose device, but one that is meant to read data, perform limited calculations on that data and control it’s environment based on those calculations. The prime use of a microcontroller is to control the operation of a machine using a fixed program that is stored in ROM and that does not change over the lifetime of the system.

The design approach of a microcontroller uses a more limited set of single byte and double byte instructions that are used to move code and data from internal memory to ALU. Many instructions are coupled with pins on the IC package; the pins are capable of having several different functions depending on the wishes of the programmer.

The microcontroller is concerned with getting the data from and on to its own pins; the architecture and instruction set are optimized to handle data in bit and byte size.

**FUNCTIONAL BLOCKS OF A MICROCONTROLLER**



2.4 CRITERIA FOR CHOOSING A MICROCONTROLLER

1. The first and foremost criterion for choosing a microcontroller is that it must meet task at hands efficiently and cost effectively. In analyzing the needs of a microcontroller based project we must first see whether it is an 8-bit, 16-bit or 32-bit microcontroller and how best it can handle the computing needs of the task most effectively. The other considerations in this category are:

(a) Speed: The highest speed that the microcontroller supports

(b) Packaging: Is it 40-pin DIP or QPF or some other packaging format?

This is important in terms of space, assembling and prototyping the

End product.

(c) Power Consumption: This is especially critical for battery-powered

Products.

(d) The amount of RAM and ROM on chip

(e) The number of I/O pins and timers on the chip.

1. Cost per unit: This is important in terms of final product in which a microcontroller is used.
2. The second criteria in choosing a microcontroller are how easy it is to develop products around it. Key considerations include the availability of an assembler, debugger, a code efficient ‘C’ language compiler, emulator, technical support and both in house and outside expertise. In many cases third party vendor support for chip is required.
3. The third criteria in choosing a microcontroller is it readily available in needed quantities both now and in future. For some designers this is even more important than first two criteria’s. Currently, of leading 8–bit microcontrollers, the 89C51 family has the largest number of diversified (multiple source) suppliers. By suppliers meant a producer besides the originator of microcontroller in the case of the 89C51, which was originated by Intel, several companies are also currently producing the 89C51. Viz: INTEL, PHILIPS, These companies include PHILIPS, SIEMENS, and DALLAS-SEMICONDUCTOR. It should be noted that Motorola, Zilog and Microchip Technologies have all dedicated massive resource as to ensure wide and timely availability of their product since their product is stable, mature and single sourced. In recent years they also have begun to sell the ASIC library cell of the microcontroller.

CHAPTER 3

AT89S52 MicrocontrolleR

3.1 AT89S52Features

• Compatible with MCS®-51 Products

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

– Endurance: 10,000 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 • Fast Programming Time

• Flexible ISP Programming (Byte and Page Mode)

• Green (Pb/Halide-free) Packaging Option

Description

The AT89S52 is a low-power, high-performance CMOS 8-bit microcontroller with 8K bytes of in-system programmable Flash memory. The device is manufactured using Atmel’s high-density nonvolatile memory technology and is compatible with the indus-try-standard 80C51 instruction set and pinout.

The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional nonvolatile memory pro-grammer. By combining a versatile 8-bit CPU with in-system programmable Flash on a monolithic chip, the Atmel AT89S52 is a powerful microcontroller which provides a highly-flexible and cost-effective solution to many embedded control applications.

The AT89S52 provides the following standard features: 8K bytes of Flash, 256 bytes of RAM, 32 I/O lines, Watchdog timer, two data pointers, three 16-bit timer/counters, a six-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 con-tents but freezes the oscillator, disabling all other chip functions until the next interrupt or hardware reset.

3.2BLOCK DIAGRAM:



Pin Description

VCC : Supply voltage.

GND : Ground.

Port 0

Port 0 is an 8-bit open drain bidirectional 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 program and data memory. In this mode, P0 has internal pull-ups. Port 0 also receives the code bytes during Flash programming and outputs the code bytes dur-ing program verification.

External pull-ups are required during program verification.

Port 1

Port 1 is an 8-bit bidirectional I/O port with internal pull-ups. 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 inter-nal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. 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 follow-ing table. Port 1 also receives the low-order address bytes during Flash programming and verification.



Port 2

Port 2 is an 8-bit bidirectional I/O port with internal pull-ups. 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 inter-nal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current (IIL) because of the internal pull-ups. Port 2 emits the high-order address byte during fetches from external program memory and dur-ing accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pull-ups 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 program-ming 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) P1.5 MOSI (used for In-System Programming) P1.6 MISO (used for In-System Programming) P1.7 SCK (used for In-System Programming)5 1919D–MICRO–6/

Port 3

Port 3 is an 8-bit bidirectional I/O port with internal pull-ups. 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 inter-nal pull-ups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current (IIL) because of the pull-ups. Port 3 receives some control signals for Flash programming and verification.

RST

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. This pin drives high for 98 oscillator periods after the Watchdog times out. The DISRTO bit in SFR AUXR (address 8EH) can be used to disable this feature. In the default state of bit DISRTO, the RESET HIGH out feature is enabled.

ALE/PROG

Address Latch Enable (ALE) is an output pulse for latching the low byte of the address during accesses to external memory. 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 purposes. Note, however, that one ALE pulse is skipped dur-ing 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 during 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

PSEN

Program Store Enable (PSEN) is the read strobe to external program memory. When the AT89S52 is executing code from external program memory, PSEN is activated twice each machine cycle, except that two PSEN activations are skipped during each access to exter-nal data memory.

EA/VPP

External Access Enable. EA must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed, EA will be internally latched on reset. EA should be strapped to VCC for internal program executions. This pin also receives the 12-volt programming enable voltage (VPP) during Flash programming.

XTAL1

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

XTAL2

Output from the inverting oscillator amplifier.

Special Function Registers

A map of the on-chip memory area called the Special Function Register (SFR) space is shown in Table 5-1. Note that not all of the addresses are occupied, and unoccupied addresses may not be imple-mented on the chip. Read accesses to these addresses will in general return random data, and write accesses will have an indeterminate effect. User software should not write 1s to these unlisted locations, since they may be used in future products to invoke new features. In that case, the reset or inactive values of the new bits will always be 0.

Timer 2 Registers:

Control and status bits are contained in registers T2CON (shown in Table 5- 2) and T2MOD (shown in Table 10-2) for Timer 2. The register pair (RCAP2H, RCAP2L) are the Capture/Reload registers for Timer 2 in 16-bit capture mode or 16-bit auto-reload mode.

Interrupt Registers:

The individual interrupt enable bits are in the IE register. Two priorities can be set for each of the six interrupt sources in the IP register.

Memory Organization MCS-51 devices have a separate address space for Program and Data Memory. Up to 64K bytes each of external Program and Data Memory can be addressed.

3.3Memory organisation

Program Memory

If the EA pin is connected to GND, all program fetches are directed to external memory. On the AT89S52, if EA is connected to VCC, program fetches to addresses 0000H through 1FFFH are directed to internal memory and fetches to addresses 2000H through FFFFH are to external memory.

Data Memory

The AT89S52 implements 256 bytes of on-chip RAM. The upper 128 bytes occupy a parallel address space to the Special Function Registers. This means that the upper 128 bytes have the same addresses as the SFR space but are physically separate from SFR space. When an instruction accesses an internal location above address 7FH, the address mode used in the instruction specifies whether the CPU accesses the upper 128 bytes of RAM or the SFR space. Instructions which use direct addressing access the SFR space. For example, the following direct addressing instruction accesses the SFR at location 0A0H (which is P2). MOV 0A0H, #data Instructions that use indirect addressing access the upper 128 bytes of RAM. For example, the following indirect addressing instruction, where R0 contains 0A0H, accesses the data byte at address 0A0H, rather than P2 (whose address is 0A0H). MOV @R0, #data Note that stack operations are examples of indirect addressing, so the upper 128 bytes of data RAM are available as stack space.

3.4 TIMERS

Watchdog Timer (One-time Enabled with Reset-out)

The WDT is intended as a recovery method in situations where the CPU may be subjected to software upsets. The WDT consists of a 14-bit counter and the Watchdog Timer Reset (WDTRST) SFR. The WDT is defaulted to disable from exiting reset. To enable the WDT, a user must write 01EH and 0E1H in sequence to the WDTRST register (SFR location 0A6H). When the WDT is enabled, it will increment every machine cycle while the oscillator is running. The WDT timeout period is dependent on the external clock frequency. There is no way to disable the WDT except through reset (either hardware reset or WDT overflow reset). When WDT over-flows, it will drive an output RESET HIGH pulse at the RST pin.

Using the WDT

To enable the WDT, a user must write 01EH and 0E1H in sequence to the WDTRST register (SFR location 0A6H). When the WDT is enabled, the user needs to service it by writing 01EH and 0E1H to WDTRST to avoid a WDT overflow. The 14-bit counter overflows when it reaches 16383 (3FFFH), and this will reset the device. When the WDT is enabled, it will increment every machine cycle while the oscillator is running. This means the user must reset the WDT at least every 16383 machine cycles. To reset the WDT the user must write 01EH and 0E1H to WDTRST. WDTRST is a write-only register. The WDT counter cannot be read or written. WhenWDT overflows, it will generate an output RESET pulse at the RST pin. The RESET pulse dura-tion is 98xTOSC, where TOSC = 1/FOSC. To make the best use of the WDT, it should be serviced in those sections of code that will periodically be executed within the time required to prevent a WDT reset.

WDT During Power-down and Idle

In Power-down mode the oscillator stops, which means the WDT also stops. While in Power-down mode, the user does not need to service the WDT. There are two methods of exiting Power-down mode: by a hardware reset or via a level-activated external interrupt which is enabled prior to entering Power-down mode.

When Power-down is exited with hardware reset, servicing the WDT should occur as it normally does whenever the AT89S52 is reset. Exiting Power-down with an interrupt is significantly different. The interrupt is held low long enough for the oscillator to stabilize. When the interrupt is brought high, the interrupt is serviced. To prevent the WDT from resetting the device while the interrupt pin is held low, the WDT is not started until the interrupt is pulled high. It is suggested that the WDT be reset during the interrupt service for the interrupt used to exit Power-down mode. To ensure that the WDT does not overflow within a few states of exiting Power-down, it is best to reset the WDT just before entering Power-down mode. Before going into the IDLE mode, the WDIDLE bit in SFR AUXR is used to determine whether the WDT continues to count if enabled. The WDT keeps counting during IDLE (WDIDLE bit = 0) as the default state. To prevent the WDT from resetting the AT89S52 while in IDLE mode, the user should always set up a timer that will periodically exit IDLE, service the WDT, and reenter IDLE mode. With WDIDLE bit enabled, the WDT will stop to count in IDLE mode and resumes the count upon exit from IDLE.

UART

The UART in the AT89S52 operates the same way as the UART in the AT89S52 and AT89C52.

Timer 0 and 1

Timer 0 and Timer 1 in the AT89S52 operate the same way as Timer 0 and Timer 1 in the AT89S52 and AT89C52.

Timer 2

Timer 2 is a 16-bit Timer/Counter that can operate as either a timer or an event counter. The type of operation is selected by bit C/T2 in the SFR T2CON (shown in Table 5-2). Timer 2 has three operating modes: capture, auto-reload (up or down counting), and baud rate generator. The modes are selected by bits in T2CON, as shown in Table 10-1. Timer 2 consists of two 8-bit registers, TH2 and TL2. In the Timer function, the TL2 register is incremented every machine cycle. Since a machine cycle consists of 12 oscillator periods, the count rate is 1/12 of the oscil-lator frequency.





In the Counter function, the register is incremented in response to a 1-to-0 transition at its corre-sponding external input pin, T2. In this function, the external input is sampled during S5P2 of every machine cycle. When the samples show a high in one cycle and a low in the next cycle, the count is incremented. The new count value appears in the register during S3P1 of the cycle following the one in which the transition was detected. Since two machine cycles (24 oscillator periods) are required to recognize a 1-to-0 transition, the maximum count rate is 1/24 of the oscillator frequency. To ensure that a given level is sampled at least once before it changes, the level should be held for at least one full machine cycle.

Capture Mode

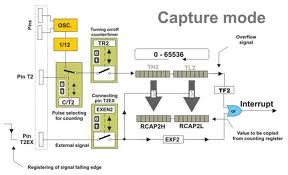
In the capture mode, two options are selected by bit EXEN2 in T2CON. If EXEN2 = 0, Timer 2 is a 16-bit timer or counter which upon overflow sets bit TF2 in T2CON. This bit can then be used to generate an interrupt. If EXEN2 = 1, Timer 2 performs the same operation, but a 1-to-0 transi-tion at external input T2EX also causes the current value in TH2 and TL2 to be captured into RCAP2H and RCAP2L, respectively. In addition, the transition at T2EX causes bit EXF2 in T2CON to be set. The EXF2 bit, like TF2, can generate an interrupt. The capture mode is illus-trated in Figure 10-1.

Auto-reload (Up or Down Counter)

Timer 2 can be programmed to count up or down when configured in its 16-bit auto-reload mode. This feature is invoked by the DCEN (Down Counter Enable) bit located in the SFR T2MOD (see Table 10-2). Upon reset, the DCEN bit is set to 0 so that timer 2 will default to count up. When DCEN is set, Timer 2 can count up or down, depending on the value of the T2EX pin.

Baud Rate Generator

Timer 2 is selected as the baud rate generator by setting TCLK and/or RCLK in T2CON (Table 5-2). Note that the baud rates for transmit and receive can be different if Timer 2 is used for the receiver or transmitter and Timer 1 is used for the other function. Setting RCLK and/or TCLK puts Timer 2 into its baud rate generator mode, as shown in Figure 11-1. The baud rate generator mode is similar to the auto-reload mode, in that a rollover in TH2 causes the Timer 2 registers to be reloaded with the 16-bit value in registers RCAP2H and RCAP2L, which are preset by software. The baud rates in Modes 1 and 3 are determined by Timer 2’s overflow rate according to the fol-lowing equation. The Timer can be configured for either timer or counter operation. In most applications, it is con-figured for timer operation (CP/T2 = 0). The timer operation is different for Timer 2 when it is used as a baud rate generator. Normally, as a timer, it increments every machine cycle (at 1/12 the oscillator frequency). As a baud rate generator, however, it increments every state time (at 1/2 the oscillator frequency). The baud rate formula is given below. where (RCAP2H, RCAP2L) is the content of RCAP2H and RCAP2L taken as a 16-bit unsigned integer. Timer 2 as a baud rate generator is shown in Figure 11-1. This figure is valid only if RCLK or TCLK = 1 in T2CON. Note that a rollover in TH2 does not set TF2 and will not generate an inter-rupt. Note too, that if EXEN2 is set, a 1-to-0 transition in T2EX will set EXF2 but will not cause a reload from (RCAP2H, RCAP2L) to (TH2, TL2). Thus, when Timer 2 is in use as a baud rate generator, T2EX can be used as an extra external interrupt. Note that when Timer 2 is running (TR2 = 1) as a timer in the baud rate generator mode, TH2 or TL2 should not be read from or written to. Under these conditions, the Timer is incremented every state time, and the results of a read or write may not be accurate. The RCAP2 registers may be read but should not be written to, because a write might overlap a reload and cause write and/or reload errors. The timer should be turned off (clear TR2) before accessing the Timer 2 or RCAP2 registers.



Programmable Clock Out

A 50% duty cycle clock can be programmed to come out on P1.0. This pin, besides being a regular I/O pin, has two alternate functions. It can be programmed to input the external clock for Timer/Counter 2 or to output a 50% duty cycle clock ranging from 61 Hz to 4 MHz (for a 16*-*MHz operating frequency). To configure the Timer/Counter 2 as a clock generator, bit C/T2 (T2CON.1) must be cleared and bit T2OE (T2MOD.1) must be set. Bit TR2 (T2CON.2) starts and stops the timer. The clock-out frequency depends on the oscillator frequency and the reload value of Timer 2 capture registers (RCAP2H, RCAP2L), as shown in the following equation. In the clock-out mode, Timer 2 roll-overs will not generate an interrupt. This behavior is similar to when Timer 2 is used as a baud-rate generator. It is possible to use Timer 2 as a baud-rate gen-erator and a clock generator simultaneously. Note, however, that the baud-rate and clock-out frequencies cannot be determined independently from one another since they both use RCAP2H and RCAP2L

3.5 INTERRUPTS

The AT89S52 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 13-1. 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 13-1 shows that bit position IE.6 is unimplemented. User software should not write a 1 to this bit position, since it may be used in future AT89 products. Timer 2 interrupt is generated by the logical OR of bits TF2 and EXF2 in register T2CON. Nei-ther 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.

Oscillator Characteristics

XTAL1 and XTAL2 are the input and output, respectively, of an inverting amplifier that can be configured for use as an on-chip oscillator, as shown in Figure 16-1. Either a quartz crystal or ceramic resonator may be used. To drive the device from an external clock source, XTAL2 should be left unconnected while XTAL1 is driven, as shown in Figure 16-2. There are no requirements on the duty cycle of the external clock signal, since the input to the internal clock-ing circuitry is through a divide-by-two flip-flop, but minimum and maximum voltage high and low time specifications must be observed.

Idle Mode

In idle mode, the CPU puts itself to sleep while all the on-chip peripherals remain active. The mode is invoked by software. The content of the on-chip RAM and all the special functions regis-ters remain unchanged during this mode. The idle mode can be terminated by any enabled interrupt or by a hardware reset. Note that when idle mode is terminated by a hardware reset, the device normally resumes pro-gram execution from where it left off, up to two machine cycles before the internal reset algorithm takes control. On-chip hardware inhibits access to internal RAM in this event, but access to the port pins is not inhibited. To eliminate the possibility of an unexpected write to a port pin when idle mode is terminated by a reset, the instruction following the one that invokes idle mode should not write to a port pin or to external memory

Power-down Mode

In the Power-down mode, the oscillator is stopped, and the instruction that invokes Power-down is the last instruction executed. The on-chip RAM and Special Function Registers retain their values until the Power-down mode is terminated. Exit from Power-down mode can be initiated either by a hardware reset or by an enabled external interrupt. Reset redefines the SFRs but does not change the on-chip RAM. The reset should not be activated before VCC is restored to its normal operating level and must be held active long enough to allow the oscillator to restart and stabilize.

Programming the Flash – Parallel Mode

The AT89S52 is shipped with the on-chip Flash memory array ready to be programmed. The programming interface needs a high-voltage (12-volt) program enable signal and is compatible with conventional third-party Flash or EPROM programmers. The AT89S52 code memory array is programmed byte-by-byte.

Programming Algorithm:

Before programming the AT89S52, the address, data, and control signals should be set up according to the “Flash Programming Modes” (Table 22-1) and Figure 22-1 and Figure 22-2. To program the AT89S52, take the following steps: 1. Input the desired memory location on the address lines. 2. Input the appropriate data byte on the data lines. 3. Activate the correct combination of control signals. 4. Raise EA/VPP to 12V. 5. Pulse ALE/PROG once to program a byte in the Flash array or the lock bits. The byte-write cycle is self-timed and typically takes no more than 50 μs. Repeat steps 1 through 5, changing the address and data for the entire array or until the end of the object file is reached.

Data Polling:

The AT89S52 features Data Polling to indicate the end of a byte write cycle. During a write cycle, an attempted read of the last byte written will result in the complement of the written data on P0.7. Once the write cycle has been completed, true data is valid on all outputs, and the next cycle may begin. Data Polling may begin any time after a write cycle has been initiated.

Ready/Busy:

The progress of byte programming can also be monitored by the RDY/BSY output signal. P3.0 is pulled low after ALE goes high during programming to indicate BUSY. P3.0 is pulled high again when programming is done to indicate READY.

Program Verify:

If lock bits LB1 and LB2 have not been programmed, the programmed code data can be read back via the address and data lines for verification.

The status of the individual lock bits can be verified directly by reading them back. Reading the Signature Bytes:

The signature bytes are read by the same procedure as a nor-mal verification of locations 000H, 100H, and 200H, except that P3.6 and P3.7 must be pulled to a logic low. The values returned are as follows. (000H) = 1EH indicates manufactured by Atmel (100H) = 52H indicates AT89S52 (200H) = 06H

Chip Erase:

In the parallel programming mode, a chip erase operation is initiated by using the proper combination of control signals and by pulsing ALE/PROG low for a duration of 200 ns - 500 ns. In the serial programming mode, a chip erase operation is initiated by issuing the Chip Erase instruction. In this mode, chip erase is self-timed and takes about 500 ms. During chip erase, a serial read from any address location will return 00H at the data output.Programming the Flash – Serial Mode The Code memory array can be programmed using the serial ISP interface while RST is pulled to VCC.

The serial interface consists of pins SCK, MOSI (input) and MISO (output). After RST is set high, the Programming Enable instruction needs to be executed first before other operations can be executed. Before a reprogramming sequence can occur, a Chip Erase operation is required. The Chip Erase operation turns the content of every memory location in the Code array into FFH. Either an external system clock can be supplied at pin XTAL1 or a crystal needs to be connected across pins XTAL1 and XTAL2. The maximum serial clock (SCK) frequency should be less than 1/16 of the crystal frequency. With a 33 MHz oscillator clock, the maximum SCK frequency is 2 MHz.

Serial Programming Algorithm

To program and verify the AT89S52 in the serial programming mode, the following sequence is recommended: 1. Power-up sequence: a. Apply power between VCC and GND pins. b. Set RST pin to “H”. If a crystal is not connected across pins XTAL1 and XTAL2, apply a 3 MHz to 33 MHz clock to XTAL1 pin and wait for at least 10 milliseconds. 2. Enable serial programming by sending the Programming Enable serial instruction to pin MOSI/P1.5. The frequency of the shift clock supplied at pin SCK/P1.7 needs to be less than the CPU clock at XTAL1 divided by 16. 3. The Code array is programmed one byte at a time in either the Byte or Page mode. The write cycle is self-timed and typically takes less than 0.5 ms at 5V. 4. Any memory location can be verified by using the Read instruction which returns the content at the selected address at serial output MISO/P1.6. 5. At the end of a programming session, RST can be set low to commence normal device operation. Power-off sequence (if needed): 1. Set XTAL1 to “L” (if a crystal is not used). 2. Set RST to “L”. 3. Turn VCC power off.

Data Polling:

The Data Polling feature is also available in the serial mode. In this mode, during a write cycle an attempted read of the last byte written will result in the complement of the MSB of the serial output byte on MISO.

CHAPTER 4

Working flow of the project

4.1BLOCK DIAGRAM

ROBOT

(H-BRIDGE)

LAND MINE SENSOR

FIRE SENSOR

BUZZER

MICRO CONTROLLER

POWER SUPPLY UNIT

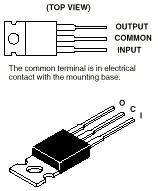
OBSTACLE SENSOR

4.2 REGULATED POWER SUPPLY

A variable regulated power supply, also called a variable bench power supply, is one where you can continuously adjust the output voltage to your requirements. Varying the output of the power supply is the recommended way to test a project after having double checked parts placement against circuit drawings and the parts placement guide.

This type of regulation is ideal for having a simple variable bench power supply. Actually this is quite important because one of the first projects a hobbyist should undertake is the construction of a variable regulated power supply. While a dedicated supply is quite handy e.g. 5V or 12V, it's much handier to have a variable supply on hand, especially for testing.

Most digital logic circuits and processors need a 5 volt power supply. To use these parts we need to build a regulated 5 volt source. Usually you start with an unregulated power To make a 5 volt power supply, we use a LM7805 voltage regulator IC (Integrated Circuit). The IC is shown below.



The LM7805 is simple to use. You simply connect the positive lead of your unregulated DC power supply (anything from 9VDC to 24VDC) to the Input pin, connect the negative lead to the Common pin and then when you turn on the power, you get a 5 volt supply from the Output pin.

CIRCUIT FEATURES

Brief description of operation: Gives out well regulated +5V output, output current capability of 100 mA

Circuit protection: Built-in overheating protection shuts down output when regulator IC gets too hot

Circuit complexity: Very simple and easy to build

Circuit performance: Very stable +5V output voltage, reliable operation

Availability of components: Easy to get, uses only very common basic components

Design testing: Based on datasheet example circuit, I have used this circuit successfully as part of many electronics projects

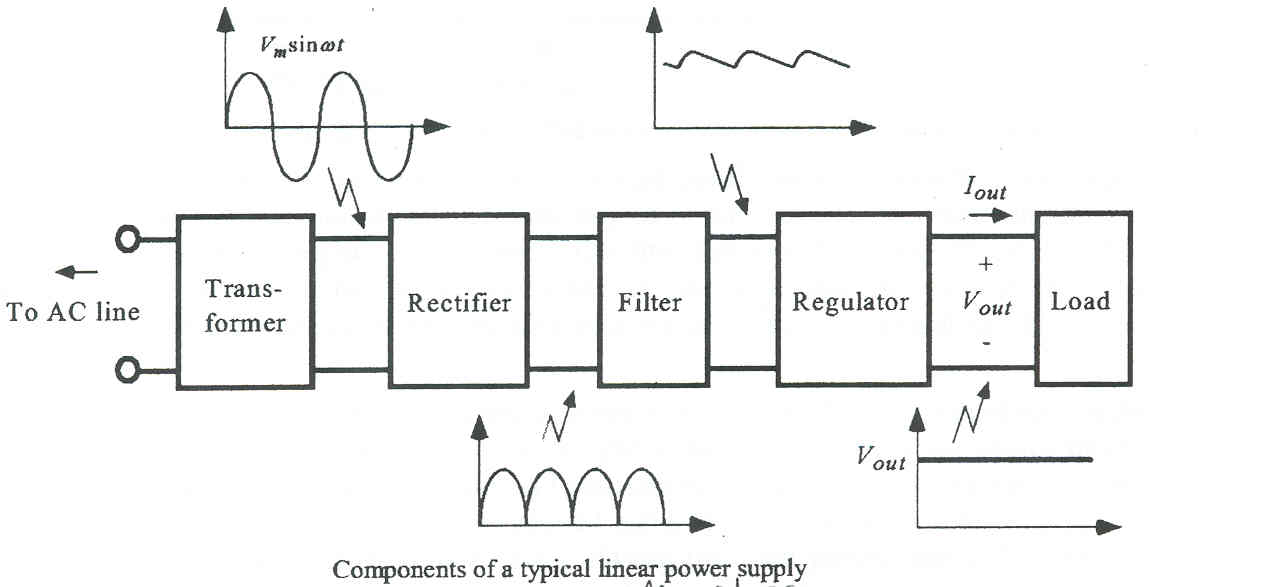
Applications: Part of electronics devices, small laboratory power supply

Power supply voltage: Unregulated DC 8-18V power supply

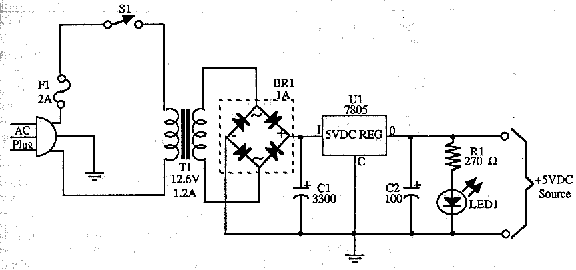
Power supply current: Needed output current + 5 mA

Component costs: Few dollars for the electronics components + the input transformer cost

BLOCK DIAGRAM



EXAMPLE CIRCUIT DIAGRAM:



WE CAN EVEN USE A USB CONNECTOR FOR THE REQUIRED SUPPLY INSTEAD OF THE ABOVE CIRCUIT

H-BRIDGE:

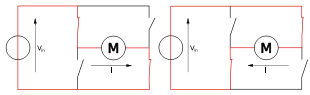
An H-bridge is an [electronic circuit](http://en.wikipedia.org/wiki/Electronic_circuit) which enables a voltage to be applied across a load in either direction. These circuits are often used in [robotics](http://en.wikipedia.org/wiki/Robotics) and other applications to allow DC motors to run forwards and backwards. H-bridges are available as [integrated circuits](http://en.wikipedia.org/wiki/Integrated_circuits), or can be built from [discrete components](http://en.wikipedia.org/wiki/Discrete_components).

## General

The term "H-bridge" is derived from the typical graphical representation of such a circuit. An H-bridge is built with four switches (solid-state or mechanical). When the switches S1 and S4 (according to the first figure) are closed (and S2 and S3 are open) a positive voltage will be applied across the motor. By opening S1 and S4 switches and closing S2 and S3 switches, this voltage is reversed, allowing reverse operation of the motor.

Using the nomenclature above, the switches S1 and S2 should never be closed at the same time, as this would cause a short circuit on the input voltage source. The same applies to the switches S3 and S4. This condition is known as shoot-through.

## [[Edit](http://en.wikipedia.org/w/index.php?title=H-bridge&action=edit&section=2)] Operation

[](http://en.wikipedia.org/wiki/File:H_bridge_operating.svg)

[magnify-clip](http://en.wikipedia.org/wiki/File:H_bridge_operating.svg)

The two basic states of an H-bridge.

The H-Bridge arrangement is generally used to reverse the polarity of the motor, but can also be used to 'brake' the motor, where the motor comes to a sudden stop, as the motor's terminals are shorted, or to let the motor 'free run' to a stop, as the motor is effectively disconnected from the circuit. The following table summarizes operation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S1 | S2 | S3 | S4 | Result |
| 1 | 0 | 0 | 1 | Motor moves right |
| 0 | 1 | 1 | 0 | Motor moves left |
| 0 | 0 | 0 | 0 | Motor free runs |
| 0 | 1 | 0 | 1 | Motor brakes |
| 1 | 0 | 1 | 0 | Motor brakes |

Buzzer

A buzzer or beeper is an [audio](http://en.wikipedia.org/wiki/Sound) signaling device, which may be [mechanical](http://en.wikipedia.org/wiki/Machine), [electromechanical](http://en.wikipedia.org/wiki/Electromechanics), or [electronic](http://en.wikipedia.org/wiki/Electronics). Typical uses of buzzers and beepers include [alarms](http://en.wikipedia.org/wiki/Alarm), [timers](http://en.wikipedia.org/wiki/Timer) and confirmation of user input such as a mouse click or keystroke.

|  |
| --- |
|  |

## Electromechanical

Early devices were based on an electromechanical system identical to an [electric bell](http://en.wikipedia.org/wiki/Electric_bell) without the metal gong. Similarly, a [relay](http://en.wikipedia.org/wiki/Relay) may be connected to interrupt its own actuating [current](http://en.wikipedia.org/wiki/Electric_current), causing the [contacts](http://en.wikipedia.org/wiki/Switch) to buzz. Often these units were anchored to a wall or ceiling to use it as a sounding board. The word "buzzer" comes from the rasping noise that electromechanical buzzers made.

**Electronic**

[](http://en.wikipedia.org/wiki/File:2007-07-24_Piezoelectric_buzzer.jpg)

[http://bits.wikimedia.org/skins-1.5/common/images/magnify-clip.png](http://en.wikipedia.org/wiki/File:2007-07-24_Piezoelectric_buzzer.jpg)

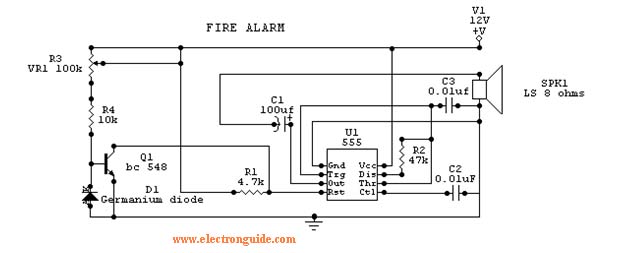
Piezoelectric disk beeper

A [piezoelectric](http://en.wikipedia.org/wiki/Piezoelectric) element may be driven by an [oscillating](http://en.wikipedia.org/wiki/Oscillation) electronic circuit or other audio signal source. Sounds commonly used to indicate that a button has been pressed are a click, a ring or a beep.

## Uses

* [Annunciator panels](http://en.wikipedia.org/wiki/Annunciator_panel)
* Electronic [metronomes](http://en.wikipedia.org/wiki/Metronome)
* [Game shows](http://en.wikipedia.org/wiki/Game_show)
* [Microwave ovens](http://en.wikipedia.org/wiki/Microwave_oven) and other [household appliances](http://en.wikipedia.org/wiki/Major_appliance)
* [Sporting](http://en.wikipedia.org/wiki/Sport) events such as [basketball](http://en.wikipedia.org/wiki/Basketball) games

Fire Alarm



An reverse biased germanium diode is used here as a heat sensor at normal room temperature the reverse resistance of the diode is very high in the order of over 10 kilo ohms so it produce no effects on the transistor Q1 which conducts and keeps the reset pin 4 of ic 555 at its ground level, and so the alarm doesn’t get activated.

When the temperature in the vicinity of the diode (the sensor) increases in case of fire, the reverse resistance of the germanium diode drops at about 70 degree its resistance drops to a value below 1 kilo ohms this stops Q1 conduction and the 555 ic pin 4 becomes positive through the resistor R1 which activates the alarm.

**4.3 Land mine sensor:**

Landmines would be ineffective weapons if they were easy to detect or if their presence was clearly marked. Therefore they often lie undetected and forgotten in the soil, years after a conflict has ceased, until they are usually tragically disturbed by civilians often children.

Additional complexity is caused by the variety of landmines which have been produced and planted in various conflicts. Some, of almost all plastic construction defeat metal detection, some are buried higher or lower in the soil while others feature varieties of detonation method or fusing which can foil mechanical destruction techniques.

The main problem with plastic landmines is that many contain very little metal, making it necessary to use extremely sensitive metal detectors, which then also detect all manner of scrap metallic objects and battlefield debris. The aim of multi-sensor systems is largely to reduce this false alarm rate and make detection and clearance faster and safer.

Despite various worthy conventions on their continuing use, landmines have been laid in conflicts around the world to such an extent that they continue to deny civilian populations access to their land, continue to cripple and kill innocent children and their clearance remains a slow, predominantly manual and costly operation around the world.

Sensatech is working on a project funded by QinetiQ to try to detect and possibly identify landmines within a soil without contacting or transmitting any pressure to the soil and without causing any type of landmine to be detonated.

Sensatech are approaching this task using arrays of capacitive sensors and non-contact 3d tomography techniques to try to map the soil under a sensor and detect objects embedded within the soil under the surface.

Assuming that sufficient definition and resolution could be obtained to the required depth within the soil, embedded objects could be compared to a database of known landmine types and a fit made if objects found in the soil match a known landmine type.

Additionally, it may be possible using capacitive and electric field techniques to try to identify the chemical composition of various elements at a range. This could allow the detection of materials within landmines such as various plastics or explosives

CHAPTER 5

SOFTWARE DEVELOPMENT

5.1 KEIL SOFTWARE

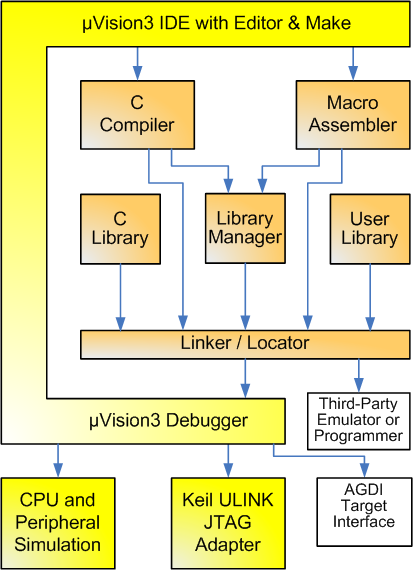


Figure 4.1 Keil Software- internal stages

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

5.2 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.

5.3 µVISIONµVISION2

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.

5.4 BUILDING AN APPLICATION IN µVISIONBUILDING 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:

Þ The “Debug Menu and Debug Commands” described below are available. The additional debug windows are discussed in the following.

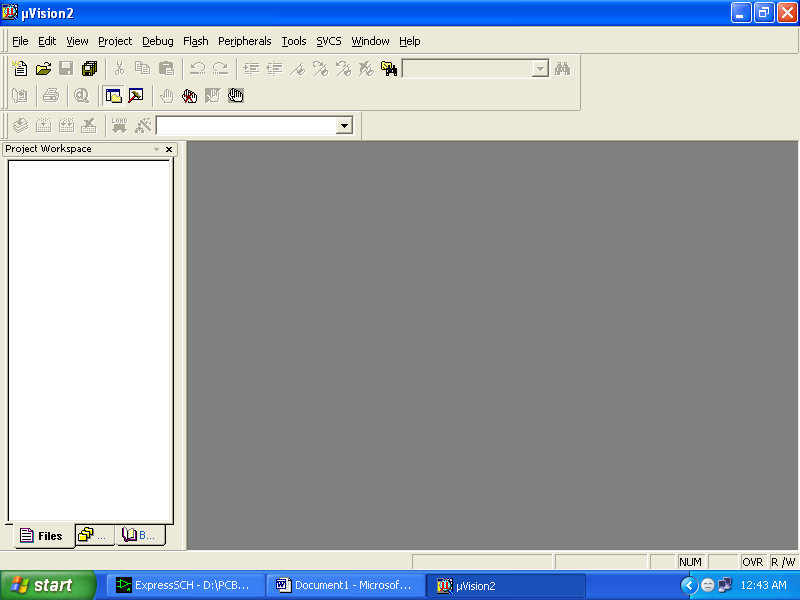
Þ The project structure or tool parameters cannot be modified. All build Commands are disabled.

Disassembly Window

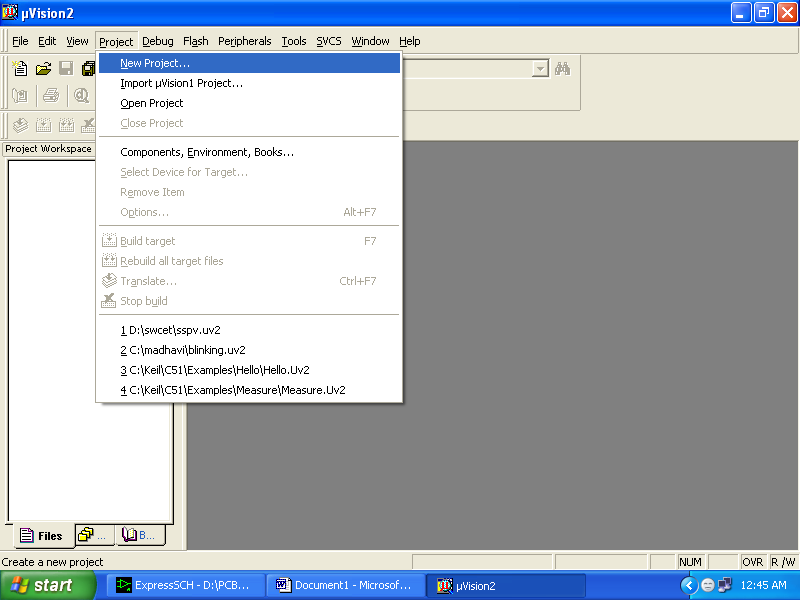
The Disassembly window shows your target program as mixed source and assembly program or just assembly code. A trace history of previously executed instructions may be displayed with Debug – View Trace Records. To enable the trace history, set Debug – Enable/Disable Trace Recording.

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.

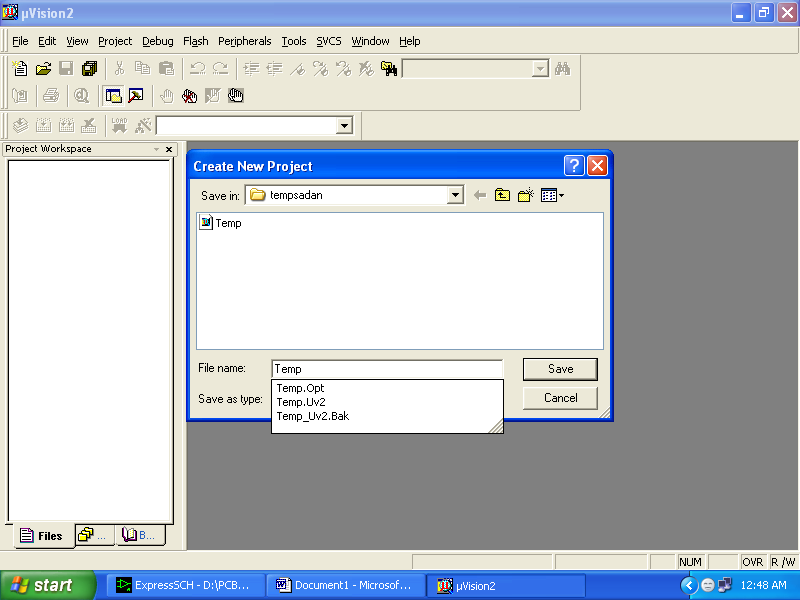
1. Click on the Keil uVision Icon on DeskTop
2. The following fig will appear



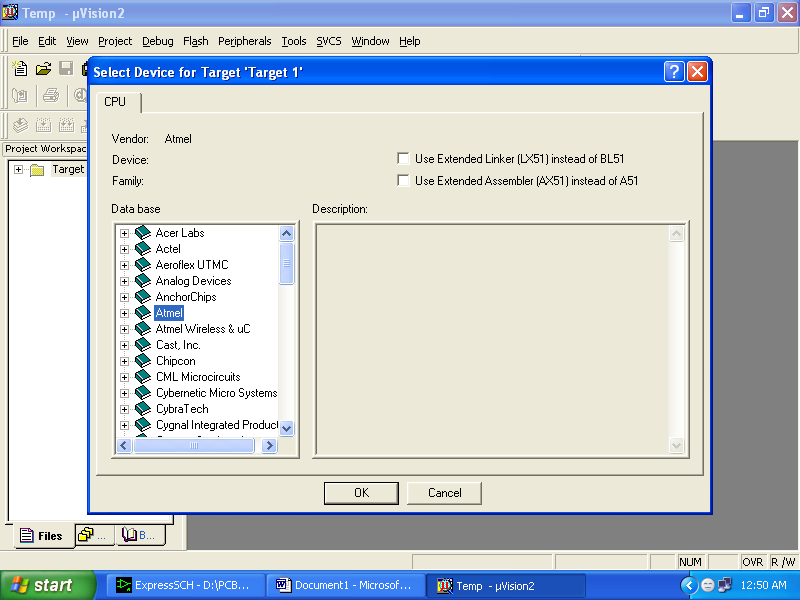
1. Click on the Project menu from the title bar
2. Then Click on New Project



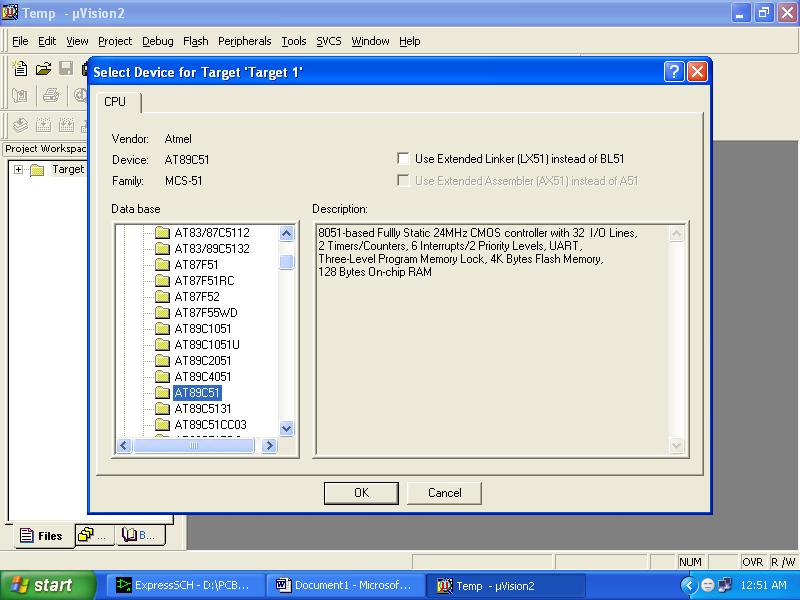
1. Save the Project by typing suitable project name with no extension in u r own folder sited in either C:\ or D:\



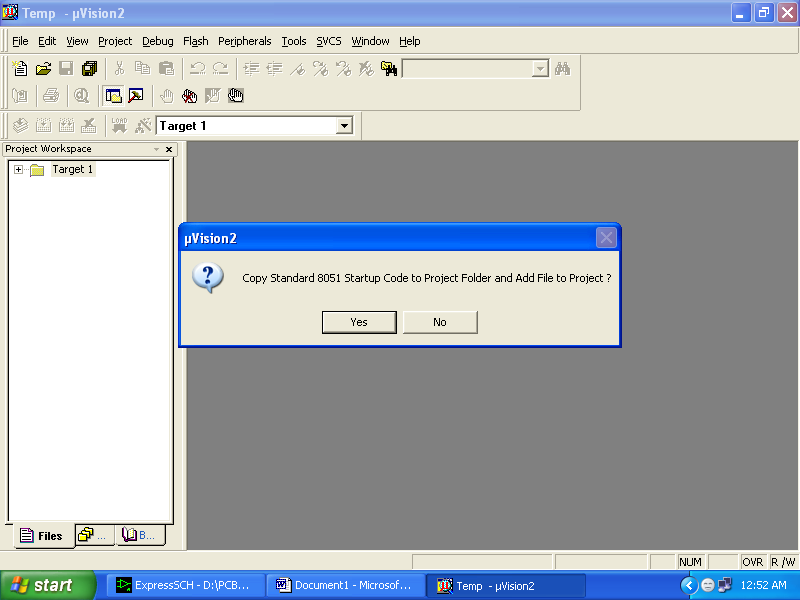
1. Then Click on Save button above.
2. Select the component for u r project. i.e. Philips……
3. Click on the + Symbol beside of Philips



1. Select AT89S52 as shown below



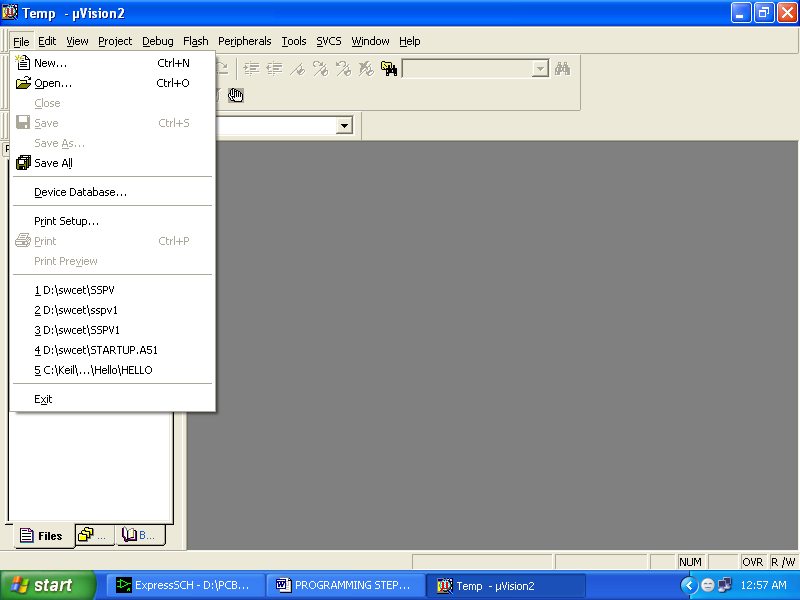
1. Then Click on “OK”
2. The Following fig will appear



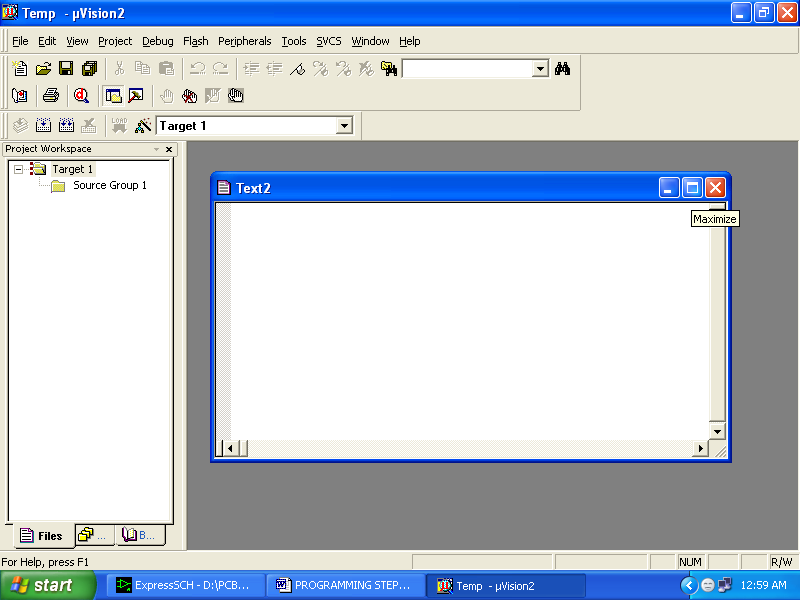
1. Then Click either YES or NO………mostly “NO”
2. Now your project is ready to USE
3. Now double click on the Target1, you would get another option “Source group 1” as shown in next page.



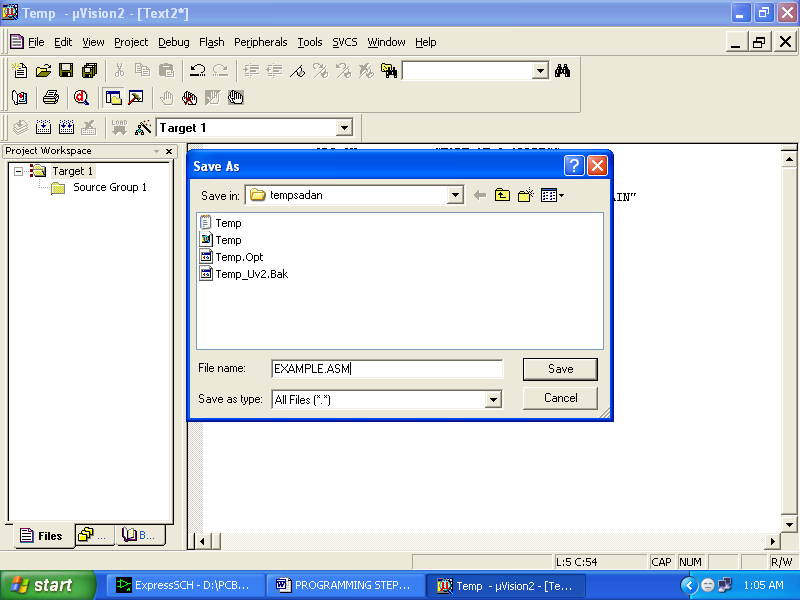
1. Click on the file option from menu bar and select “new”



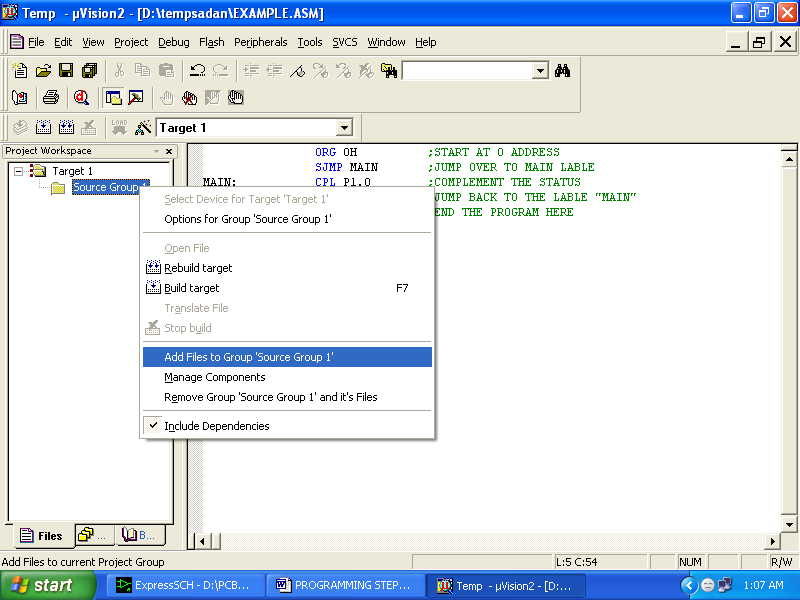
1. The next screen will be as shown in next page, and just maximize it by double clicking on its blue boarder.



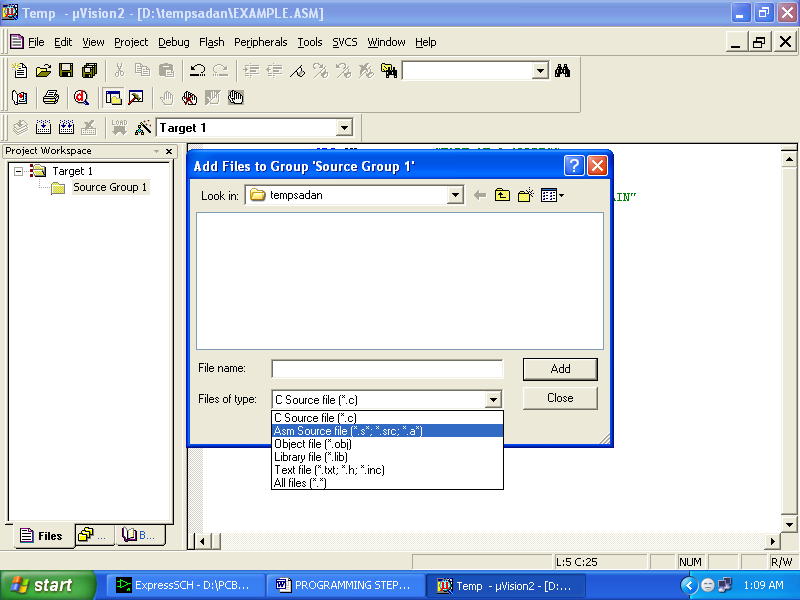
1. Now start writing program in either in “C” or “ASM”
2. For a program written in Assembly, then save it with extension “. asm” and for “C” based program save it with extension “ .C”



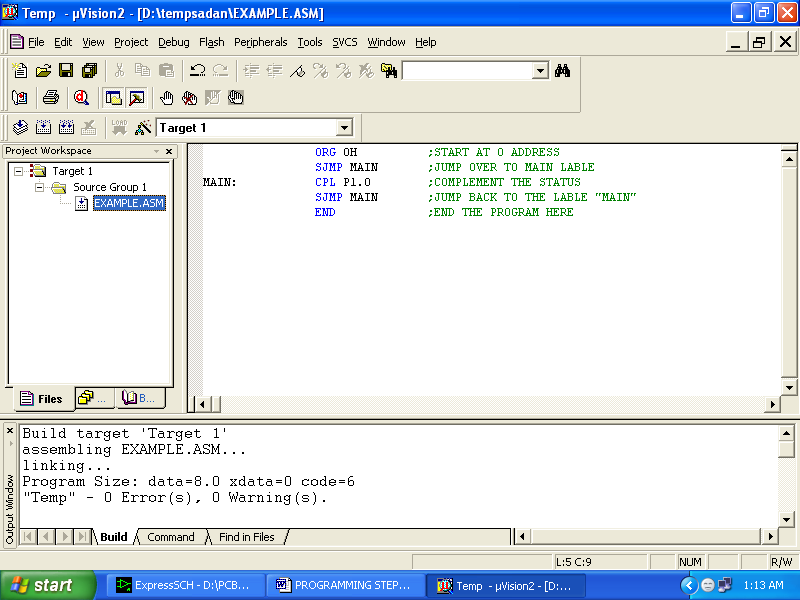
1. Now right click on Source group 1 and click on “Add files to Group Source”



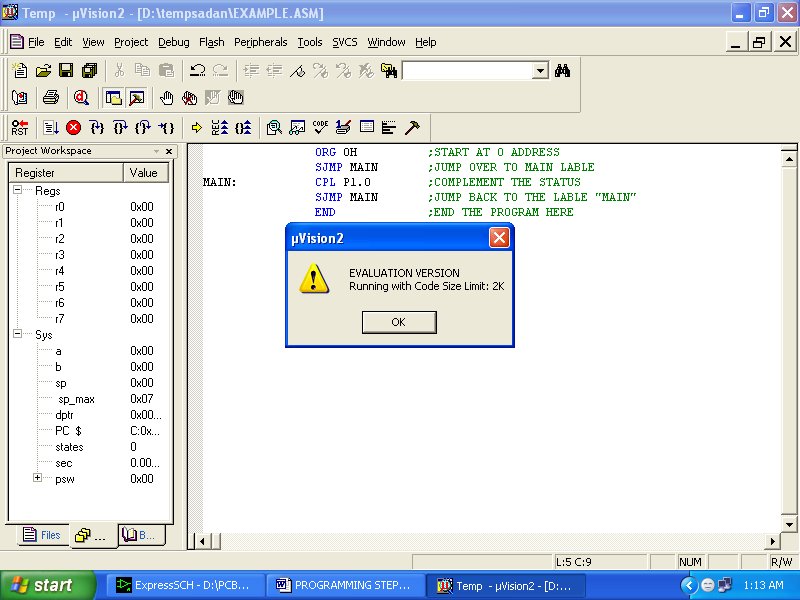
1. Now you will get another window, on which by default “C” files will appear.



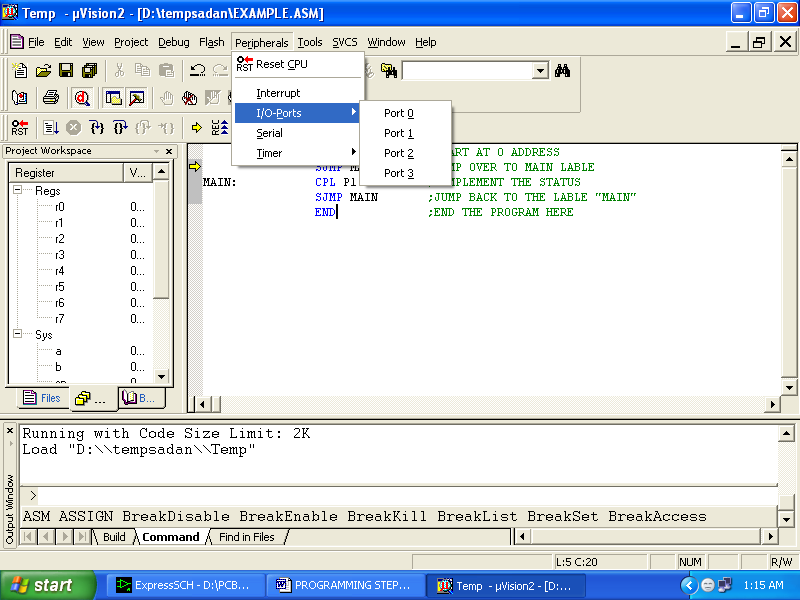
1. Now select as per your file extension given while saving the file
2. Click only one time on option “ADD”
3. Now Press function key F7 to compile. Any error will appear if so happen.



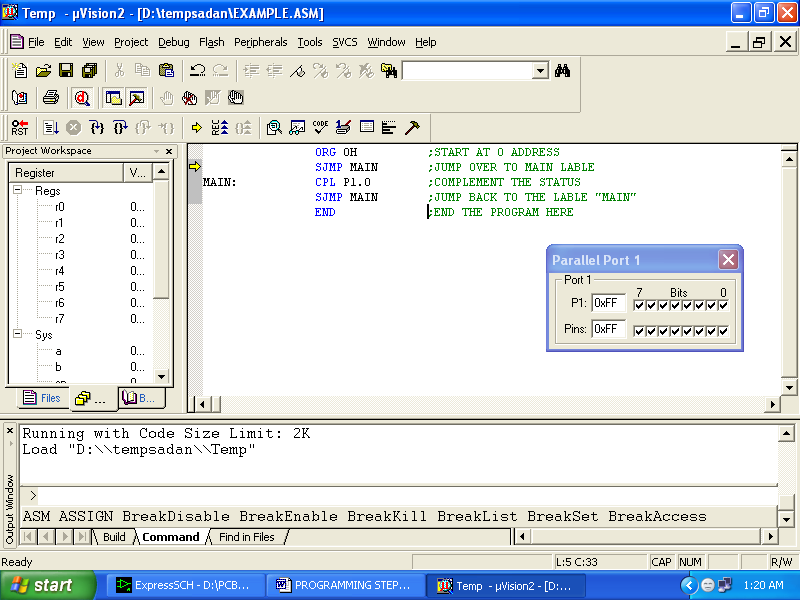
1. If the file contains no error, then press Control+F5 simultaneously.
2. The new window is as follows



1. Then Click “OK”
2. Now Click on the Peripherals from menu bar, and check your required port as shown in fig below



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



1. Now keep Pressing function key “F11” slowly and observe.
2. You are running your program successfully

CONCLUSION

The project “super intelligent robot” has been successfully designed and tested.

It has been developed by integrating features of all the hardware components used. 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.

**BIBLIOGRAPHY**

The 8051 Micro controller and Embedded

Systems

-**Muhammad Ali Mazidi**

**Janice Gillispie Mazidi**

The 8051 Micro controller Architecture,

Programming & Applications

**-Kenneth J.Ayala**

Fundamentals Of Micro processors and

Micro computers

**-B.Ram**

Micro processor Architecture, Programming

& Applications

**-Ramesh S.Gaonkar**

Electronic Components

-**D.V.Prasad**

Wireless Communications

- Theodore S. Rappaport

Mobile Tele Communications

- William C.Y. Lee

References on the Web:

[www.national.com](http://www.national.com)

[www.nxp.com](http://www.nxp.com)

[www.8052.com](http://www.8052.com)

[www.microsoftsearch.com](http://www.microsoftsearch.com)

[www.geocities.com](http://www.geocities.com)