INDEX

CHAPTER 1. INTRODUCTION

CHAPTER 2. DESCRIPTION OF HARDWARE COMPONENTS

2.1 AT89S52
   2.1.1 Introduction to AT89S52
   2.1.2 Features
   2.1.3 Architectural overview
   2.1.4 Pin description for AT89S52
   2.1.5 Memories

2.2 POWER SUPPLY
   2.2.1 Introduction
   2.2.2 Power Supply Circuit
   2.2.3 Transformer
   2.2.4 Rectifier
   2.2.5 Regulator

2.4 HARD WARE
   2.4.1 Power supply unit.
   2.4.2 GSM Modules.
   2.4.3 GPS MODULE
   2.4.4 MAX 232 IC.
   2.4.5 LCD DISPLAY.

   2.4.6 RELAY

   2.4.7 DC MOTOR

CHAPTER 3. CIRCUITS AND THEIR OPERATION

   3.1 Circuit diagram
CHAPTER 4. SOFTWARE DEVELOPMENT

4.1.1 Introduction
4.1.2 Tools used
4.1.3 C51 Compiler & A51 macro assembler
4.1.4 Start µ vision
4.1.5 overview of Keil cross compiler
4.1.6 Benefits of Keil compiler
4.1.7 Flashmagic

CHAPTER 5. RESULTS AND CONCLUSION

5.1 Advantages
5.2 Disadvantages
5.3 Applications
5.4 Conclusion

REFERENCES
INTRODUCTION

1.1 EMBEDDED SYSTEMS

Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real time performance constraints that must be met, for reason such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.

Wireless communication has become an important feature for commercial products and a popular research topic within the last ten years. There are now more mobile phone subscriptions than wired-line subscriptions. Lately, one area of commercial interest has been low-cost, low-power, and short-distance wireless communication used for personal wireless networks." Technology advancements are providing smaller and more cost effective devices for integrating computational processing, wireless communication, and a host of other functionalities. These embedded communications devices will be integrated into applications ranging from homeland security to industry automation and monitoring. They will also enable custom tailored engineering solutions, creating a revolutionary way of disseminating and processing information. With new technologies and devices come new business activities, and the need for employees in these technological areas. Engineers who have knowledge of embedded systems and wireless communications will be in high demand. Unfortunately, there are few adorable environments available for development and classroom use, so students often do not learn about these technologies during hands-on lab exercises. The communication mediums were twisted pair, optical fiber, infrared, and generally wireless radio.

OBJECTIVE OF THE PROJECT:
In recent years, vehicle thefts are increasing at an alarming rate around the world. People have started to use the theft control systems installed in their vehicles. The commercially available anti-theft vehicular systems are very expensive. This paper deals with the design & development of a theft control system for an automobile, which is being used to prevent / control the theft of a vehicle. The developed system makes Use of an embedded system based on GSM /GPS technology (GPS technology is further enhancement of the base paper). The designed & developed System is installed in the vehicle. An interfacing GSM MODEM (SIMCOM’s SIM300 GSM operates in the 900MHz and 1.8GHz bands GSM supports data transfer speeds of up to 9.6 kbps) and the GPS which are connected to the microcontroller, which is in turn, connected to the engine.

Once, the vehicle is being stolen, the information is being used by the vehicle owner for further processing. Where by sitting at a remote place, a particular number is dialed by them to the interfacing GSM MODEM that is with the hardware kit which is installed in the vehicle. By reading the signals received by the mobile, the engine is locked automatically and sends the SMS to the dialed no stating the exact position using GPS modem. Again it will come to the normal condition only after entering a secured password. The owner of the vehicle.

The main concept in this design is introducing the GSM and GPS Technologies into the embedded system. The designed unit is very simple & low cost. The entire designed unit is on a single chip. When the vehicle is stolen, owner will give a call to the kit placed in vehicle engine automatically stop which is indicated by the DC motor in turn sends the SMS to the authorized person showing the exact location using GPS. The car will unlock until the password is entered with help of 4x3 key pad interfaced to LPC 2148 by the authorized person.

**BLOCK DESIGN PROPOSAL FOR THE SYSTEM**
**HARDWARE REQUIREMENT FOR IMPLEMENTATION OF THE FUNCTION MODULE**

- CONTROLLER (ARM LPC 2148/AT89S52).
- GSM MODEM (SIMCOM’s SIM300).
- GPS MODEM (GPS 634R).
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”

AT89S52:
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 industry-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 programmer. 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 content but freezes the oscillator, disabling all other chip functions until the next interrupt
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.

The 8051 is the name of a big family of microcontrollers. The device which we are going to use along this tutorial is the 'AT89S52' which is a typical 8051 microcontroller manufactured by Atmel. Note that this part doesn't aim to explain the functioning of the different components of a 89S52 microcontroller, but rather to give you a general idea of the organization of the chip and the available features, which shall be explained in detail along this tutorial.

The block diagram provided by Atmel in their datasheet showing the architecture the 89S52 device can seem very complicated, and since we are going to use the C high level language to program it, a simpler architecture can be represented as the figure 1.2.A.
This figure shows the main features and components that the designer can interact with. You can notice that the 89S52 has 4 different ports, each one having 8 input/output lines providing a total of 32 I/O lines. Those ports can be used to output data and orders to other devices, or to read the state of a sensor, or a switch. Most of the ports of the 89S52 have 'dual function' meaning that they can be used for two different functions: the first one is to perform input/output operations and the second one is used to implement special features of the microcontroller like counting external pulses, interrupting the execution of the program according to external events, performing serial data transfer or connecting the chip to a computer to update the software.

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, whereas, 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.

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.

**FEATURES**

Compatible with MCS-51 Products
8K Bytes of In-System Reprogrammable Flash Memory
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
Programmable Serial Channel
Low-power Idle and Power-down Modes
4.0V to 5.5V Operating Range
Full Duplex UART Serial Channel
Interrupt Recovery from Power-down Mode
Watchdog Timer
Dual Data Pointer
Power-off Flag
Fast Programming Time
Flexible ISP Programming (Byte and Page Mode)

PIN DIAGRAM
2.1.4 PIN DESCRIPTION

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 pullups. Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification.
External pullups are required during program verification.

**Port 1**
Port 1 is an 8-bit bidirectional 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 (IIL) 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.

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1.0</td>
<td>T2 (external count input to Timer/Counter clock-out)</td>
</tr>
<tr>
<td>P1.1</td>
<td>T2EX (Timer/Counter 2 capture/reload trig and direction control)</td>
</tr>
<tr>
<td>P1.5</td>
<td>MOSI (used for In-System Programming)</td>
</tr>
<tr>
<td>P1.6</td>
<td>MISO (used for In-System Programming)</td>
</tr>
</tbody>
</table>

**Port 2**
Port 2 is an 8-bit bidirectional 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 (IIL) 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 @ D PTR). 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 programming and verification.

**Port 3**
Port 3 is an 8-bit bidirectional 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 (IIL) because of the pullups. Port 3 also serves the functions of various special features of the AT89S52, as shown in the following table. Port 3 also receives some control signals for Flash programming and verification.

<table>
<thead>
<tr>
<th>Port Pin</th>
<th>Alternate Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3.0</td>
<td>RXD (serial input port)</td>
</tr>
<tr>
<td>P3.1</td>
<td>TXD (serial output port)</td>
</tr>
<tr>
<td>P3.2</td>
<td>INT0 (external interrupt 0)</td>
</tr>
<tr>
<td>P3.3</td>
<td>INT1 (external interrupt 1)</td>
</tr>
<tr>
<td>P3.4</td>
<td>T0 (timer 0 external input)</td>
</tr>
<tr>
<td>P3.5</td>
<td>T1 (timer 1 external input)</td>
</tr>
<tr>
<td>P3.6</td>
<td>WR (external data memory write strobe)</td>
</tr>
</tbody>
</table>

**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 96 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 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 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 external 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.

---

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 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](image)

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.

1. 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.
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.

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.

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)

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 microcontroller. 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 microcontroller. 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 microcontroller. 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 microcontroller. 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 | 0 | 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.

```
| 0 | 0 | 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 | 0 | 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.

**POWER SUPPLY**

All digital circuits require regulated power supply. In this article we are going to learn how to get a regulated positive supply from the mains supply.

Figure 1 shows the basic block diagram of a fixed regulated power supply. Let us go through each block.

**TRANSFORMER**

A transformer consists of two coils also called as “WINDINGS” namely PRIMARY & SECONDARY.
They are linked together through inductively coupled electrical conductors also called as CORE. A changing current in the primary causes a change in the Magnetic Field in the core & this in turn induces an alternating voltage in the secondary coil. If load is applied to the secondary then an alternating current will flow through the load. If we consider an ideal condition then all the energy from the primary circuit will be transferred to the secondary circuit through the magnetic field.

\[ P_{\text{primary}} = P_{\text{secondary}} \]

So

\[ I_p V_p = I_s V_s \]

The secondary voltage of the transformer depends on the number of turns in the Primary as well as in the secondary.

\[ \frac{V_s}{V_p} = \frac{N_s}{N_p} \]

**Rectifier**

A rectifier is a device that converts an AC signal into DC signal. For rectification purpose we use a diode, a diode is a device that allows current to pass only in one direction i.e. when the anode of the diode is positive with respect to the cathode also called as forward biased condition & blocks current in the reversed biased condition.

Rectifier can be classified as follows:

1) **Half Wave rectifier.**
This is the simplest type of rectifier as you can see in the diagram a half wave rectifier consists of only one diode. When an AC signal is applied to it during the positive half cycle the diode is forward biased & current flows through it. But during the negative half cycle diode is reverse biased & no current flows through it. Since only one half of the input reaches the output, it is very inefficient to be used in power supplies.

2) **Full wave rectifier.**

Half wave rectifier is quite simple but it is very inefficient, for greater efficiency we would like to use both the half cycles of the AC signal. This can be achieved by using a center tapped transformer i.e. we would have to double the size of secondary winding & provide connection to the center. So during the positive half cycle diode D1 conducts & D2 is in reverse biased condition. During the negative half cycle diode D2 conducts & D1 is reverse biased. Thus we get both the half cycles across the load.
One of the disadvantages of Full Wave Rectifier design is the necessity of using a center tapped transformer, thus increasing the size & cost of the circuit. This can be avoided by using the Full Wave Bridge Rectifier.

3) **Bridge Rectifier.**

As the name suggests it converts the full wave i.e. both the positive & the negative half cycle into DC thus it is much more efficient than Half Wave Rectifier & that too without using a center tapped transformer thus much more cost effective than Full Wave Rectifier.

Full Bridge Wave Rectifier consists of four diodes namely D1, D2, D3 and D4. During the positive half cycle diodes D1 & D4 conduct whereas in the negative half cycle diodes D2 & D3 conduct thus the diodes keep switching the transformer connections so we get positive half cycles in the output.
If we use a center tapped transformer for a bridge rectifier we can get both positive & negative half cycles which can thus be used for generating fixed positive & fixed negative voltages.

**FILTER CAPACITOR**

Even though half wave & full wave rectifier give DC output, none of them provides a constant output voltage. For this we require to smoothen the waveform received from the rectifier. This can be done by using a capacitor at the output of the rectifier this capacitor is also called as “FILTER CAPACITOR” or “SMOOTHING CAPACITOR” or “RESERVOIR CAPACITOR”. Even after using this capacitor a small amount of ripple will remain.

We place the Filter Capacitor at the output of the rectifier the capacitor will charge to the peak voltage during each half cycle then will discharge its stored energy slowly through the load while the rectified voltage drops to zero, thus trying to keep the voltage as constant as possible.
If we go on increasing the value of the filter capacitor then the Ripple will decrease. But then the costing will increase. The value of the Filter capacitor depends on the current consumed by the circuit, the frequency of the waveform & the accepted ripple.

\[ C = \frac{V_r F}{I} \]

Where,

\( V_r \) = accepted ripple voltage. (should not be more than 10% of the voltage)

\( I \) = current consumed by the circuit in Amperes.

\( F \) = frequency of the waveform. A half wave rectifier has only one peak in one cycle so \( F = 25 \text{hz} \)

Whereas a full wave rectifier has Two peaks in one cycle so \( F = 100 \text{hz} \).

**VOLTAGE REGULATOR**

A Voltage regulator is a device which converts varying input voltage into a constant regulated output voltage. Voltage regulator can be of two types

1) Linear Voltage Regulator
   
   Also called as Resistive Voltage regulator because they dissipate the excessive voltage resistively as heat.

2) Switching Regulators.
   
   They regulate the output voltage by switching the Current ON/OFF very rapidly. Since their output is either ON or OFF it dissipates very low power thus achieving higher efficiency as compared to linear voltage regulators. But they are more complex & generate high noise due to their switching action. For low level of output power switching regulators tend to be costly but for higher output wattage they are much cheaper than linear regulators.

The most commonly available Linear Positive Voltage Regulators are the 78XX series where the XX indicates the output voltage. And 79XX series is for Negative Voltage Regulators.
After filtering the rectifier output the signal is given to a voltage regulator. The maximum input voltage that can be applied at the input is 35V. Normally there is a 2-3 Volts drop across the regulator so the input voltage should be at least 2-3 Volts higher than the output voltage. If the input voltage gets below the $V_{\text{min}}$ of the regulator due to the ripple voltage or due to any other reason the voltage regulator will not be able to produce the correct regulated voltage.

(i) Circuit diagram:

![Circuit Diagram of power supply](image)

Fig 2.3. Circuit Diagram of power supply

(ii) IC 7805:

7805 is an integrated three-terminal positive fixed linear voltage regulator. It supports an input voltage of 10 volts to 35 volts and output voltage of 5 volts. It has a current rating of 1 amp although lower current models are available. Its output voltage is fixed at 5.0V. The 7805 also has a built-in current limiter as a safety feature. 7805 is manufactured by many companies, including National Semiconductors and Fairchild Semiconductors.

The 7805 will automatically reduce output current if it gets too hot. The last two digits represent the voltage; for instance, the 7812 is a 12-volt regulator. The 78xx series of regulators is designed to work in complement with the 79xx series of negative voltage regulators in systems that provide both positive and negative regulated voltages, since the 78xx series can't regulate negative voltages in such a system.
The 7805 & 78 is one of the most common and well-known of the 78xx series regulators, as it's small component count and medium-power regulated 5V make it useful for powering TTL devices.

### Table 2.1. Specifications of IC7805

<table>
<thead>
<tr>
<th>SPECIFICATIONS</th>
<th>IC 7805</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_{out}$</td>
<td>5V</td>
</tr>
<tr>
<td>$V_{cin} - V_{out}$ Difference</td>
<td>5V - 20V</td>
</tr>
<tr>
<td>Operation Ambient Temp</td>
<td>0 - 125°C</td>
</tr>
<tr>
<td>Output $I_{max}$</td>
<td>1A</td>
</tr>
</tbody>
</table>

---

**GSM MODEM**

**Definitions**

The words, “Mobile Station” (MS) or “Mobile Equipment” (ME) are used for mobile terminals Supporting GSM services.

A call from a GSM mobile station to the PSTN is called a “mobile originated call” (MOC) or
“Outgoing call”, and a call from a fixed network to a GSM mobile station is called a “mobile Terminated call” (MTC) or “incoming call”.

What is GSM?
GSM (Global System for Mobile communications) is an open, digital cellular technology used for transmitting mobile voice and data services.

What does GSM offer?
GSM supports voice calls and data transfer speeds of up to 9.6 kbit/s, together with the transmission of SMS (Short Message Service).

GSM operates in the 900MHz and 1.8GHz bands in Europe and the 1.9GHz and 850MHz bands in the US. The 850MHz band is also used for GSM and 3G in Australia, Canada and many South American countries. By having harmonised spectrum across most of the globe, GSM’s international roaming capability allows users to access the same services when travelling abroad as at home. This gives consumers seamless and same number connectivity in more than 218 countries.
Terrestrial GSM networks now cover more than 80% of the world’s population. GSM satellite roaming has also extended service access to areas where terrestrial coverage is not available

HISTORY

In 1980’s the analog cellular telephone systems were growing rapidly all throughout Europe, France and Germany. Each country defined its own protocols and frequencies to work on. For example UK used the Total Access Communication System (TACS), USA used the
AMPS technology and Germany used the C-netz technology. None of these systems were interoperable and also they were analog in nature.

In 1982 the Conference of European Posts and Telegraphs (CEPT) formed a study group called the GROUPE SPECIAL MOBILE (GSM) The main area this focused on was to get the cellular system working throughout the world, and ISDN compatibility with the ability to incorporate any future enhancements. In 1989 the GSM transferred the work to the European Telecommunications Standards Institute (ETSI.) the ETS defined all the standards used in GSM.

3. BASICS OF WORKING AND SPECIFICATIONS OF GSM –

The GSM architecture is nothing but a network of computers. The system has to partition available frequency and assign only that part of the frequency spectrum to any base transceiver station and also has to reuse the scarce frequency as often as possible.

GSM uses TDMA and FDMA together. Graphically this can be shown below –

Fig 1. Representation of a GSM signal using TDMA & FDMA with respect to the transmitted power.

Some of the technical specifications of GSM are listed below –

<table>
<thead>
<tr>
<th>Multiple Access Method</th>
<th>TDMA / FDMA</th>
</tr>
</thead>
</table>


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uplink frequencies (MHz)</td>
<td>933-960 (basic GSM)</td>
</tr>
<tr>
<td>Downlink frequencies (MHz)</td>
<td>890-915 (basic GSM)</td>
</tr>
<tr>
<td>Duplexing</td>
<td>FDD</td>
</tr>
<tr>
<td>Channel spacing, kHz</td>
<td>200</td>
</tr>
<tr>
<td>Modulation</td>
<td>GMSK</td>
</tr>
<tr>
<td>Portable TX power, maximum / average (mW)</td>
<td>1000 / 125</td>
</tr>
<tr>
<td>Power control, handset and BSS</td>
<td>Yes</td>
</tr>
<tr>
<td>Speech coding and rate (kbps)</td>
<td>RPE-LTP / 13</td>
</tr>
<tr>
<td>Speech Channels per RF channel:</td>
<td>8</td>
</tr>
<tr>
<td>Channel rate (kbps)</td>
<td>270.833</td>
</tr>
<tr>
<td>Channel coding</td>
<td>Rate 1/2 convolutional</td>
</tr>
<tr>
<td>Frame duration (ms)</td>
<td>4.615</td>
</tr>
</tbody>
</table>

GSM was originally defined for the 900 Mhz range but after some time even the 1800 Mhz range was used for cellular technology. The 1800 MHz range has its architecture and specifications almost same to that of the 900 Mhz GSM technology but building the Mobile exchanges is easier and the high frequency Synergy effects add to the advantages of the 1800 Mhz range.

**4. ARCHITECTURE AND BUILDING BLOCKS –**

GSM is mainly built on 3 building blocks. (Ref Fig. 2)

- GSM Radio Network – This is concerned with the signaling of the system. Hand-overs occur in the radio network. Each BTS is allocated a set of frequency channels.
- GSM Mobile switching Network – This network is concerned with the storage of data required for routing and service provision.
GSM Operation and Maintenance – The task carried out by it include Administration and commercial operation, Security management, Network configuration, operation, performance management and maintenance tasks.

Fig. 2 The basic blocks of the whole GSM system

Explanations of some of the abbreviations used –

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Land Mobile Network (PLMN)</td>
<td>The whole GSM system</td>
</tr>
<tr>
<td>Mobile System (MS)</td>
<td>The actual cell phone that we use</td>
</tr>
<tr>
<td>Base Transceiver Station (BTS)</td>
<td>Provides connectivity between network and mobile station via the Air-interface</td>
</tr>
<tr>
<td><strong>BaseStationController (BSC)</strong></td>
<td>Controls the whole subsystem.</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Transcoding Rate &amp; Adaption Unit (TRAU)</td>
<td>This is instrumental in compressing the Data that is passed on to the network, is a part of the BSS.</td>
</tr>
<tr>
<td><strong>Mobile Services Switching Center (MSC)</strong></td>
<td>The BSC is connected to the MSC. The MSC routes the incoming and outgoing calls and assigns user channels on the A-interface.</td>
</tr>
<tr>
<td><strong>Home Location Register (HLR)</strong></td>
<td>This register stores data of large no of users. It is like a database that manages data of all the users. Every PLMN will have at least one HLR.</td>
</tr>
<tr>
<td><strong>Visitor Location Register (VLR)</strong></td>
<td>This contains part of data so that the HLR is not overloaded with inquiries. If a subscriber moves out of VLR area the HLR requests removal of data related to that user from the VLR.</td>
</tr>
<tr>
<td><strong>Equipment Identity Register (EIR)</strong></td>
<td>The IMEI no. is allocated by the manufacturer and is stored on the network in the EIR. A stolen phone can be made completely useless by the network/s if the IMEI no is known.</td>
</tr>
</tbody>
</table>
SIGNALLING SCHEMES AND CIPHERING CODES USED –

GSM is digital but voice is inherently analog. So the analog signal has to be converted and then transmitted. The coding scheme used by GSM is RPE-LTP (Rectangular pulse Excitation – Long Term Prediction)

![Transmitter for the voice signal](image1)

**Fig.3 Transmitter for the voice signal**

![Receiver for the Voice signal](image2)

**Fig.4 Receiver for the Voice signal**

The voice signal is sampled at 8000 bits/sec and is quantized to get a 13 bit resolution corresponding to a bit rate of 104 kbits/sec. This signal is given to a speech coder (codec) that
compresses this speech into a source-coded speech signal of 260 bit blocks at a bit rate of 13 kbit/sec. The codec achieves a compression ratio of 1:8. The coder also has a Voice activity detector (VAD) and comfort noise synthesizer. The VAD decides whether the current speech frame contains speech or pause, this is turn is used to decide whether to turn on or off the transmitter under the control of the Discontinuous Transmission (DTX). This transmission takes advantage of the fact that during a phone conversation both the parties rarely speak at the same time. Thus the DTX helps in reducing the power consumption and prolonging battery life. The missing speech frames are replaced by synthetic background noise generated by the comfort noise synthesize in a Silence Descriptor (SID) frame. Suppose a loss off speech frame occurs due to noisy transmission and it cannot be corrected by the channel coding protection mechanism then the decoder flags such frames with a bad frame indicator (BFI) In such a case the speech frame is discarded and using a technique called error concealment which calculates the next frame based on the previous frame.

**CIPHERING CODES**

MS Authentication algorithm’s –

These algorithms are stored in the SIM and the operator can decide which one it prefers using.

**A3/8**

The A3 generates the SRES response to the MSC’s random challenge, RAND which the MSC has received from the HLR. The A3 algorithm gets the RAND from the MSC and the secret key Ki from the SIM as input and generated a 32- bit output, the SRES response. The A8 has a 64 bit Kc output.

**A5/1 (Over the Air Voice Privacy Algorithm)**

The A5 algorithm is the stream cipher used to encrypt over the air transmissions. The stream cipher is initialized for every frame sent with the session key Kc and the no. of frames being decrypted / encrypted. The same Kc key is used throughout the call but different 22-bit frame is used.
TWO MAIN INTERFACES

The two main interfaces are the AIR and the ABIS interface. The figure shows the signaling between them.

AIR INTERFACE – signaling between MS and BTS

ABIS INTERFACE – signaling between BTS and BSC

AIR INTERFACE

The air interface is like the physical layer in the model. The signaling schemes used in the AIR interface are as follows –

- **BROADCAST CONTROL CHANNE (BCCH)**
  
  - **Broadcast Control Channel (BCCH)**

    This channel broadcasts a series of information elements to the MS, such as radio channel configuration, synchronization information etc.

- **FREQUENCY CORRECTION CHANNEL (FCCH)**

  This channel contains information about the correction in transmission frequency broadcasted to MS.
- **SYNCHRONIZATION CHANNEL (SCH)**

  It broadcasts data for the frame synchronization of a MS and information to identify a BSC.

- **COMMON CONTROL CHANNEL (BCH)**

  This is a point to multi-point signaling channel to deal with access management functions. Consists of 3 channels –

  - **RANDOM ACCESS CHANNEL (RACH)**

    It is the Uplink portion, accessed from the mobile stations in a cell to ask for a dedicated signaling channel for 1 transaction.

  - **ACCESS GRANT CHANNEL (AGCH)**

    It is the downlink portion used to assign a dedicated signaling channel.

  - **NOTIFICATION CHANNEL (NCH)**

    It is used to inform mobile stations about incoming calls and broadcast calls.

- **DEDICATED CONTROL CHANNEL (DCCH)**

  It is a Bi-directional point to point signaling channel. Consists of 3 channels –

  - **STAND ALONE DEDICATED CONTROL CHANNEL (SDDCH)** –

    Used for signaling between the BSS and MS when there is no active connection between them.

  - **SLOW ASSOCIATED CONTROL CHANNEL (SACCH)** –

    This channel had to continuously transfer data because it is considered as proof of existence of a physical radio connection.

  - **FAST ASSOCIATED CONTROL CHANNEL (FACCH)** –

    This channel is used to make additional band-width available for signaling
**Multi-Tech line settings**

A serial link handler is set with the following default values (factory settings): autobaud, 8 bits data, 1 stop bit, no parity, RTS/CTS flow control. Please use the +IPR, +IFC and +ICF Commands to change these settings.

Commands always start with AT (which means ATtention) and finish with a <CR> character.

Information responses and result codes

Responses start and end with <CR><LF>, except for the ATV0 DCE response format) and the ATQ1 (result code suppression) commands.

If command syntax is incorrect, an ERROR string is returned.

If command syntax is correct but with some incorrect parameters, the +CME ERROR: <Err> or +CMS ERROR: <SmsErr> strings are returned with different error codes.

If the command line has been performed successfully, an OK string is returned.

In some cases, such as “AT+CPIN?” or (unsolicited) incoming events, the product does not Return the OK string as a response.

**Product Serial Number +CGSN**

**Description:**

This command allows the user application to get the IMEI (International Mobile Equipment Identity) of the product.

**Syntax:**

Command syntax: AT+CGSN

<table>
<thead>
<tr>
<th>Command</th>
<th>Possible responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT+CGSN</td>
<td>012345678901234 OK</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Get the IMEI</td>
</tr>
<tr>
<td></td>
<td>Note: IMEI read from EEPROM</td>
</tr>
<tr>
<td>AT+CGSN</td>
<td>+CME ERROR: 22</td>
</tr>
<tr>
<td><strong>Note:</strong></td>
<td>Get the IMEI</td>
</tr>
<tr>
<td></td>
<td>Note: IMEI not found in EEPROM</td>
</tr>
</tbody>
</table>

**Repeat last command A/**
Description:
This command repeats the previous command. Only the A/ command itself cannot be repeated.

Syntax:
Command syntax: A/

<table>
<thead>
<tr>
<th>Command</th>
<th>Possible responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/</td>
<td>Note: Repeat last command</td>
</tr>
</tbody>
</table>

Signal Quality +CSQ

Description:
This command determines the received signal strength indication (<rssi>) and the channel bit error rate (<ber>) with or without a SIM card inserted.

Syntax:
Command syntax: AT+CSQ

<table>
<thead>
<tr>
<th>Command</th>
<th>Possible responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT+CSQ</td>
<td>+CSQ: &lt;rssi&gt;,&lt;ber&gt;</td>
</tr>
<tr>
<td></td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Note: &lt;rssi&gt; and &lt;ber&gt; as defined below</td>
</tr>
</tbody>
</table>

Defined values:
0: -113 dBm or less
1: -111 dBm
30: -109 to –53 dBm
31: -51dBm or greater
99: not known or not detectable

New message indication +CNMI

Description:
This command selects the procedure for message reception from the network.

Syntax:
Command syntax: AT+CNMI=<mode>,<mt>,<bm>,<ds>,<bfr>
Read message +CMGR

Description:
This command allows the application to read stored messages. The messages are read from the memory selected by +CPMS command.

Command syntax: AT+CMGR=<index>

<table>
<thead>
<tr>
<th>Command</th>
<th>Possible responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT+CNMI=2,1,0,0,0</td>
<td>OK</td>
</tr>
<tr>
<td>Note: &lt;mt&gt;=1</td>
<td></td>
</tr>
<tr>
<td>AT+CMTI: &quot;SM&quot;,1</td>
<td></td>
</tr>
<tr>
<td>Note: message received</td>
<td></td>
</tr>
<tr>
<td>AT+CNMI=2,2,0,0,0</td>
<td>OK</td>
</tr>
<tr>
<td>Note: &lt;mt&gt;=2</td>
<td></td>
</tr>
<tr>
<td>+CMT: “123456”,”98/10/01,12:30 00+00”,12!</td>
<td></td>
</tr>
<tr>
<td>,32,240, ”15379”,129,5&lt;CR&gt;&lt;LF&gt; message received</td>
<td></td>
</tr>
<tr>
<td>Note: message received</td>
<td></td>
</tr>
<tr>
<td>AT+CNMI=2,0,0,1,0</td>
<td>OK</td>
</tr>
<tr>
<td>Note: &lt;ds&gt;=1</td>
<td></td>
</tr>
<tr>
<td>Message to send &lt;ctrl-Z&gt;</td>
<td></td>
</tr>
<tr>
<td>Note: Send a message in text mode</td>
<td></td>
</tr>
<tr>
<td>+CMGS: 7</td>
<td></td>
</tr>
<tr>
<td>OK,</td>
<td></td>
</tr>
<tr>
<td>Note: Successful</td>
<td></td>
</tr>
<tr>
<td>AT+CMGS=&quot;+33146290800&quot;&lt;CR&gt; transmission</td>
<td></td>
</tr>
<tr>
<td>+CDS: 2, 116, ”+33146290800”, 145</td>
<td></td>
</tr>
<tr>
<td>“98/10/01,12:30:07+04”, “98/10/01 12:30:08-0</td>
<td></td>
</tr>
<tr>
<td>Note: message was correctly delivered</td>
<td></td>
</tr>
</tbody>
</table>

List message +CMGL
Description:
This command allows the application to read stored messages, by indicating the type of the Message to read. The messages are read from the memory selected by the +CPMS command.

Syntax: Command syntax: AT+CMGL=<stat>

<table>
<thead>
<tr>
<th>Command</th>
<th>Possible responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT+CMGL=&quot;REC UNREAD&quot;</td>
<td>+CMGL: 1,&quot;REC UNREAD&quot;,&quot;46290800&quot;,&lt;CR&gt;;&lt;LF&gt; Unread message!</td>
</tr>
<tr>
<td>Note: List unread messages in text mode</td>
<td>+CMGL: 3,&quot;REC UNREAD&quot;,&quot;46290800&quot;,&lt;CR&gt;;&lt;LF&gt; Another message unread!</td>
</tr>
<tr>
<td></td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Note: 2 messages are unread, these messages will then their status changed to &quot;REC READ&quot; (+CSDH:0)</td>
</tr>
<tr>
<td>AT+CMGL=&quot;REC READ&quot;</td>
<td>+CMGL: 2,&quot;REC READ&quot;,&quot;46290800&quot;,&lt;CR&gt;;&lt;LF&gt;</td>
</tr>
<tr>
<td>Note: List read messages in text mode</td>
<td>Keep cool</td>
</tr>
<tr>
<td></td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td>Note: No message found</td>
</tr>
<tr>
<td>AT+CMGL=&quot;STO SENT&quot;</td>
<td>OK</td>
</tr>
<tr>
<td>Note: List stored and sent messages in text mode</td>
<td></td>
</tr>
<tr>
<td>AT+CMGL=1</td>
<td>+CMGL: 1.1...26</td>
</tr>
<tr>
<td>Note: List read messages in PDU mode</td>
<td>0791336600030000F3040B913366920547F400130011904</td>
</tr>
<tr>
<td></td>
<td>0400741AA8E6A9C5201</td>
</tr>
<tr>
<td></td>
<td>OK</td>
</tr>
</tbody>
</table>

Defined values:

<stat> possible values (status of messages in memory):

<table>
<thead>
<tr>
<th>Text mode possible values</th>
<th>PDU mode possible values</th>
<th>Status of messages in memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;REC UNREAD&quot;</td>
<td>0</td>
<td>received unread messages</td>
</tr>
<tr>
<td>&quot;REC READ&quot;</td>
<td>1</td>
<td>received read messages</td>
</tr>
<tr>
<td>&quot;STO UNSENT&quot;</td>
<td>2</td>
<td>stored unsent messages</td>
</tr>
<tr>
<td>&quot;STO SENT&quot;</td>
<td>3</td>
<td>stored sent messages</td>
</tr>
<tr>
<td>&quot;ALL&quot;</td>
<td>4</td>
<td>all messages</td>
</tr>
</tbody>
</table>

Send message +CMGS

Description:
The <address> field is the address of the terminal to which the message is sent. To send the
Message, simply type, <ctrl-Z> character (ASCII 26). The text can contain all existing characters except <ctrl-Z> and <ESC> (ASCII 27). This command can be aborted using the <ESC> character when entering text. In PDU mode, only hexadecimal characters are used (‘0’…’9’, ’A’…’F’).

Syntax:
Command syntax in text mode:
AT+CMGS= <da> [ ,<toda> ] <CR>

The message reference, <mr>, which is returned to the application, is allocated by the product. This number begins with 0 and is incremented by one for each outgoing message (successful and failure cases); it is cyclic on one byte (0 follows 255).

Note: This number is not a storage number. Outgoing messages are not stored.

Delete message +CMGD

Description:
This command deletes one or several messages from preferred message storage (‘BM” SMS CB ‘RAM storage’, “SM” SMSPP storage ‘SIM storage’ or “SR” SMS Status-Report storage).

Syntax:
Command syntax: AT+CMGD=<Index> [,<DelFalg>]
Defines values

(1-20) when the preferred message storage is “BM”

Integer type values in the range of location numbers of SIM Message memory

When the preferred message storage is “SM” or “SR”.

<DelFlag>

0 Delete message at location <index>.
1 Delete All READ messages
2 Delete All READ and SENT messages
3 Delete All READ, SENT and UNSENT messages
4 Delete all messages.

MAX 232
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.
2.3.2 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.

Serial communication

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 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.

LCD DISPLAY:

(a) LCD MODULE

To display interactive messages we are using LCD Module. We examine an intelligent LCD display of two lines, 16 characters per line that is interfaced to the controllers. The protocol (handshaking) for the display is as shown. Whereas D0 to D7th bit is the Data lines, RS, RW and
EN pins are the control pins and remaining pins are +5V, -5V and GND to provide supply. Where RS is the Register Select, RW is the Read Write and EN is the Enable pin.

The display contains two internal byte-wide registers, one for commands (RS=0) and the second for characters to be displayed (RS=1). It also contains a user-programmed RAM area (the character RAM) that can be programmed to generate any desired character that can be formed using a dot matrix. To distinguish between these two data areas, the hex command byte 80 will be used to signify that the display RAM address 00h will be chosen. Port1 is used to furnish the command or data type, and ports 3.2 to3.4 furnish register select and read/write levels.

The display takes varying amounts of time to accomplish the functions as listed. LCD bit 7 is monitored for logic high (busy) to ensure the display is overwritten.

Liquid Crystal Display also called as LCD is very helpful in providing user interface as well as for debugging purpose. The most common type of LCD controller is HITACHI 44780 which provides a simple interface between the controller & an LCD. These LCD's are very simple to interface with the controller as well as are cost effective.

![2x16 Line Alphanumeric LCD Display](image)

2x16 Line Alphanumeric LCD Display

The most commonly used **ALPHANUMERIC** displays are 1x16 (Single Line & 16 characters), 2x16 (Double Line & 16 character per line) & 4x20 (four lines & Twenty characters per line). The LCD requires 3 control lines (RS, R/W & EN) & 8 (or 4) data lines. The number on data lines depends on the mode of operation. If operated in 8-bit mode then 8 data lines + 3 control lines i.e. total 11 lines are required. And if operated in 4-bit mode then 4 data lines + 3 control lines i.e. 7 lines are required. How do we decide which mode to use? It’s simple if you have sufficient data lines you can go for 8 bit mode & if there is a time constrain i.e. display should be faster then we have to use 8-bit mode because basically 4-bit mode takes twice as more time as compared to 8-bit mode.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Symbol</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Vss</td>
<td>Ground</td>
</tr>
<tr>
<td>2</td>
<td>Vdd</td>
<td>Supply Voltage</td>
</tr>
<tr>
<td>3</td>
<td>Vo</td>
<td>Contrast Setting</td>
</tr>
</tbody>
</table>
When $RS$ is low (0), the data is to be treated as a command. When $RS$ is high (1), the data being sent is considered as text data which should be displayed on the screen.

When $R/W$ is low (0), the information on the data bus is being written to the LCD. When $RW$ is high (1), the program is effectively reading from the LCD. Most of the times there is no need to read from the LCD so this line can directly be connected to Gnd thus saving one controller line.

The *ENABLE* pin is used to latch the data present on the data pins. A HIGH - LOW signal is required to latch the data. The LCD interprets and executes our command at the instant the EN line is brought low. If you never bring EN low, your instruction will never be executed.

**COMMANDS USED IN LCD**
Section I.1 RELAYS SPDT

Section I.2

Section I.3 Overview OF Relays

A relay is an electrically operated switch used to isolate one electrical circuit from another. In its simplest form, a relay consists of a coil used as an electromagnet to open and close switch contacts. Since the two circuits are isolated from one another, a lower voltage circuit can be used to trip a relay, which will control a separate circuit that requires a higher voltage or amperage. Relays can be found in early telephone exchange equipment, in industrial control circuits, in car audio systems, in automobiles, on water pumps, in high-power audio amplifiers and as protection devices.
Section I.4 Relay Switch Contacts
The switch contacts on a relay can be "normally open" (NO) or "normally closed" (NC)--that is, when the coil is at rest and not energized (no current flowing through it), the switch contacts are given the designation of being NO or NC. In an open circuit, no current flows, such as a wall light switch in your home in a position that the light is off. In a closed circuit, metal switch contacts touch each other to complete a circuit, and current flows, similar to turning a light switch to the "on" position. In the accompanying schematic diagram, points A and B connect to the coil. Points C and D connect to the switch. When you apply a voltage across the coil at points A and B, you create an electromagnetic field, which attracts a lever in the switch, causing it to make or break contact in the circuit at points C and D (depending if the design is NO or NC). The switch contacts remain in this state until you remove the voltage to the coil. Relays come in different switch configurations. The switches may have more than one "pole," or switch contact. The diagram shows a "single pole single throw" configuration, referred to as SPST. This is similar to a wall light switch in your home. With a single "throw" of the switch, you close the circuit.

Section I.5 The Single Pole Double Throw Relay
A single pole double throw (SPDT) relay configuration switches one common pole to two other poles, flipping between them. As shown in the schematic diagram, the common point E completes a circuit with C when the relay coil is at rest, that is, no voltage is applied to it.

This circuit is "closed." A gap between the contacts of point E and D creates an "open" circuit. When you apply power to the coil, a metal level is pulled down, closing the circuit between points E and D and opening the circuit between E and C. A single pole double throw relay can be used to alternate which circuit a voltage or signal will be sent to.
SPDT Relay:
(Single Pole Double Throw Relay) an electromagnetic switch, consist of a coil (terminals 85 & 86), 1 common terminal (30), 1 normally closed terminal (87a), and one normally open terminal (87) (Figure 1).

When the coil of an SPDT relay (Figure 1) is at rest (not energized), the common terminal (30) and the normally closed terminal (87a) have continuity. When the coil is energized, the common terminal (30) and the normally open terminal (87) have continuity.

The diagram below center (Figure 2) shows an SPDT relay at rest, with the coil not energized. The diagram below right (Figure 3) shows the relay with the coil energized. As you can see, the coil is an electromagnet that causes the arm that is always connected to the common (30) to pivot when energized whereby contact is broken from the normally closed terminal (87a) and made with the normally open terminal (87).

When energizing the coil of a relay, polarity of the coil does not matter unless there is a diode across the coil. If a diode is not present, you may attach positive voltage to either terminal of the coil and negative voltage to the other, otherwise you must connect positive to the side of the coil that the cathode side (side with stripe) of the diode is connected and negative to side of the coil that the anode side of the diode is connected.
Why do I want to use a relay and do I really need to?

Anytime you want to switch a device which draws more current than is provided by an output of a switch or component you'll need to use a relay. The coil of an SPDT or an SPST relay that we most commonly use draws very little current (less than 200 milliamps) and the amount of current that you can pass through a relay's common, normally closed, and normally open contacts will handle up to 30 or 40 amps. This allows you to switch devices such as headlights, parking lights, horns, etc., with low amperage outputs such as those found on keyless entry and alarm systems, and other components. In some cases you may need to switch multiple things at the same time using one output. A single output connected to multiple relays will allow you to open continuity and/or close continuity simultaneously on multiple wires.

There are far too many applications to list that require the use of a relay, but we do show many of the most popular applications in the pages that follow and many more in our Relay Diagrams - Quick Reference application. If you are still unclear about what a relay does or if you should use one after you browse through the rest of this section, please post a question in the 12volt's install bay. (We recommend Tyco (formerly Bosch) or Potter & Brumfield relays for all of the SPDT and SPST relay applications shown on this site.)
DC motors are configured in many types and sizes, including brushless, servo, and gear motor types. A motor consists of a rotor and a permanent magnetic field stator. The magnetic field is maintained using either permanent magnets or electromagnetic windings. DC motors
are most commonly used in variable speed and torque.

Motion and controls cover a wide range of components that in some way are used to generate and/or control motion. Areas within this category include bearings and bushings, clutches and brakes, controls and drives, drive components, encoders and resolves, Integrated motion control, limit switches, linear actuators, linear and rotary motion components, linear position sensing, motors (both AC and DC motors), orientation position sensing, pneumatics and pneumatic components, positioning stages, slides and guides, power transmission (mechanical), seals, slip rings, solenoids, springs.

Motors are the devices that provide the actual speed and torque in a drive system. This family includes AC motor types (single and multiphase motors, universal, servo motors, induction, synchronous, and gear motor) and DC motors (brush less, servo motor, and gear motor) as well as linear, stepper and air motors, and motor contactors and starters.

In any electric motor, operation is based on simple electromagnetism. A current-carrying conductor generates a magnetic field; when this is then placed in an external magnetic field, it will experience a force proportional to the current in the conductor, and to the strength of the external magnetic field. As you are well aware of from playing with magnets as a kid, opposite (North and South) polarities attract, while like polarities (North and North, South and South) repel. The internal configuration of a DC motor is designed to harness the magnetic interaction between a current-carrying conductor and an external magnetic field to generate rotational motion.

Let's start by looking at a simple 2-pole DC electric motor (here red represents a magnet or winding with a "North" polarization, while green represents a magnet or winding with a "South" polarization).
Every DC motor has six basic parts -- axle, rotor (a.k.a., armature), stator, commutator, field magnet(s), and brushes. In most common DC motors (and all that Beamers will see), the external magnetic field is produced by high-strength permanent magnets. The stator is the stationary part of the motor -- this includes the motor casing, as well as two or more permanent magnet pole pieces. The rotor (together with the axle and attached commutator) rotates with respect to the stator. The rotor consists of windings (generally on a core), the windings being electrically connected to the commutator. The above diagram shows a common motor layout -- with the rotor inside the stator (field) magnets.

The geometry of the brushes, commutator contacts, and rotor windings are such that when power is applied, the polarities of the energized winding and the stator magnet(s) are misaligned, and the rotor will rotate until it is almost aligned with the stator's field magnets. As the rotor reaches alignment, the brushes move to the next commutator contacts, and energize the next winding. Given our example two-pole motor, the rotation reverses the direction of current through the rotor winding, leading to a "flip" of the rotor's magnetic field, and driving it to continue rotating.

In real life, though, DC motors will always have more than two poles (three is a very common number). In particular, this avoids "dead spots" in the commutator. You can imagine how with our example two-pole motor, if the rotor is exactly at the middle of its rotation (perfectly aligned with the field magnets), it will get "stuck" there. Meanwhile, with a two-pole motor, there is a moment where the commutator shorts out the power supply (i.e., both brushes touch both commutator contacts simultaneously). This would be bad for the power supply, waste energy, and damage motor components as well. Yet another disadvantage of such a simple motor is that it would exhibit a high amount of torque" ripple" (the amount of torque it could produce is cyclic with the position of the rotor).
So since most small DC motors are of a three-pole design, let's tinker with the workings of one via an interactive animation (JavaScript required):

![Animation of three-pole DC motor](image)

You'll notice a few things from this -- namely, one pole is fully energized at a time (but two others are "partially" energized). As each brush transitions from one commutator contact to the next, one coil's field will rapidly collapse, as the next coil's field will rapidly charge up (this occurs within a few microsecond). We'll see more about the effects of this later, but in the meantime you can see that this is a direct result of the coil windings' series wiring:

![Circuit diagram of three-pole DC motor](image)

There's probably no better way to see how an average dc motor is put together, than by just opening one up. Unfortunately this is tedious work, as well as requiring the destruction of a perfectly good motor.

**CIRCUIT DIAGRAM:**
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:

![Diagram](Image)

*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.

5.4 µVISION

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 - (for example, 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.

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 your own folder situated in either C:\ or D:\
6. Then Click on Save button above.
7. Select the component for your project. i.e. Atmel……
8. Click on the + Symbol beside of Atmel

9. Select AT89C51 as shown below
10. Then Click on “OK”

11. The Following fig will appear
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 border.

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”

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.

24. If the file contains no error, then press Control+F5 simultaneously.
25. The new window is as follows
26. Then Click “OK”

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

28. Drag the port a side and click in the program file.
29. Now keep Pressing function key “F11” slowly and observe.
30. You are running your program successfully

CHAPTER 6.

Advantages:
1. Automatic engine locking
2. Intelligent security system

Disadvantages:
1. Without gsm network this system is not work.

Applications:
1. Vehicle tracking applications
CONCLUSION:

The project “GSM-GPS VEHICLE THEFT CONTROL SYSTEM” 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.
CHAPTER

BIBLIOGRAPHY, REFERENCES

i) BIBILOGRAPHY

1. WWW.MITEL.DATABOOK.COM
2. WWW.ATMEL.DATABOOK.COM
3. WWW.FRANKLIN.COM
4. WWW.KEIL.COM
5. WWW.KEIL.COM
6. WWW.ALLDATASHEETS.COM
7. WWW.WIKIPEDIA.COM
8. WWW.HOWSTUFFWORKS.COM
9. WWW.FLASHMAGICTOOL.COM

http://en.wikipedia.org/wiki/GSM
ii) REFERENCES

2. "Linear Integrated Circuits” by D Roy Choudary & Shail Jain.
3. "Electrical Machines” by S K Bhattacharya.
4. "Electrical Machines II” by B L Thereja.